WHITEPAPER

Bitdefender

Security

FIN8 Threat Actor Goes Agile with New Sardonic Backdoor

www.bitdefender.com

Contents

Authors:

Eduard BUDACA – Security Researcher, Cyber Threat Intelligence Lab

Victor VRABIE - Security Researcher, Cyber Threat Intelligence Lab

Research coordinator:

Cristina VATAMANU - Senior Team Lead, Cyber Threat Intelligence Lab

Foreword

Since January 2016, FIN8 has been steadily building a reputation among financially motivated advanced threat actors. Bitdefender researchers are constantly monitoring this group's activity, and previous research released in early 2021 documented the use of a new, improved version of the BADHATCH backdoor.

This whitepaper focuses on the analysis of a new backdoor component discovered during a forensic investigation, described here.

As this backdoor has not been documented or referenced before, we named it “Sardonic”, given that artifacts led us to believe the threat actors use this name for an entire project including the backdoor itself, the loader and some additional scripts. We believe this project is still under development, and additional updates will likely follow.

Key facts about Sardonic:

- Sardonic is a new backdoor in the FIN8 ecosystem
- Sardonic is a project still under development and includes several components
- The new components were identified in a real-life attack and seems to be compiled just before the attack
- Sardonic backdoor is extremely potent and has a wide range of capabilities that help the threat actor leverage new malware on the fly without updating components

Attack Flow

The attack described in this whitepaper was uncovered after a FIN8 infection at a financial institution in the US. While we couldn't identify how the attackers gained initial access to the network, FIN8 is known for using social engineering and spear-phishing tactics to infiltrate target organizations.

As described in our previous threat intelligence report on FIN8, once in the network, the attackers began with network reconnaissance, obtaining information about the domain (users, domain controllers) and continued with lateral movement and privilege escalation. In addition to the use of WMIExec, which we reported earlier, we found traces of SMBExec from the same toolset (Impacket), along with, of course, the offensive features of their signature backdoor, BADHATCH.

The BADHATCH loader was deployed using PowerShell scripts downloaded from the 104.168.237[.]21 IP address using the legitimate sslip.io service. It was used during the reconnaissance, lateral movement, privilege escalation and possibly impact stages.

There were multiple attempts to deploy the Sardonic backdoor on domain controllers in order to continue with privilege escalation and lateral movement, but the malicious command lines were blocked. We saw no traces of BADHATCH on these high-value targets. However, we identified one SQL server where some artifacts indicate that the threat actors intended to deploy both backdoors.

Persistence

Deployment of this backdoor begins by running the Sardonic loader. First, a PowerShell script named “sldr.ps1” (SHA256 edfd3ae4def3ddffb37bad3424eb73c17e156ba5f63fd1d651df2f5b8e34a6c7) is copied to the victim machine and executed. We believe this was performed manually by the attackers, rather than being installed at one point as part of an automated loader process.

This script contains two base-64 strings, corresponding to two additional PowerShell scripts: the first is an RC4-encrypted second stage, and the second one implements the decryption algorithm for the former. In an improvement from earlier loaders used by FIN8, the decryption key “B4a0f3AE251b7689CFdDe1” is not included in the sample, but is given as a command-line argument.

The purpose of the second stage task is to install the persistence mechanism. Additionally, it can kill an existing process passed as a command-line argument, but we haven't seen this feature used.

Instead of persisting itself, this second stage abuses WMI (Windows Management Instrumentation) to execute a PowerShell command approximately 2-4 minutes after each startup, using the following event filter:

```
SELECT * FROM __InstanceModificationEvent WITHIN 60
WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System'
AND TargetInstance.SystemUpTime >= 140 AND TargetInstance.SystemUpTime < 240
```

That PowerShell command loads and executes a .NET DLL file from another WMI object:

```
powershell.exe -nop -c [System.Reflection.Assembly]::Load(([WmiClass] 'root\cimv2:Win32_
Base64Class').Properties['Prop'].Value);[MSDAC.PerfOSChecker]::StartCheck()
```

Additionally, to ensure that the third stage starts executing immediately, two more WMI objects are created which, combined, run it every four seconds, using the following event filter:

```
SELECT * FROM __InstanceModificationEvent WITHIN 4
WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System
```

These objects are deleted seven seconds later to ensure that this stage executes exactly once. The WMI objects used to start the next stage are depicted in figure 1.

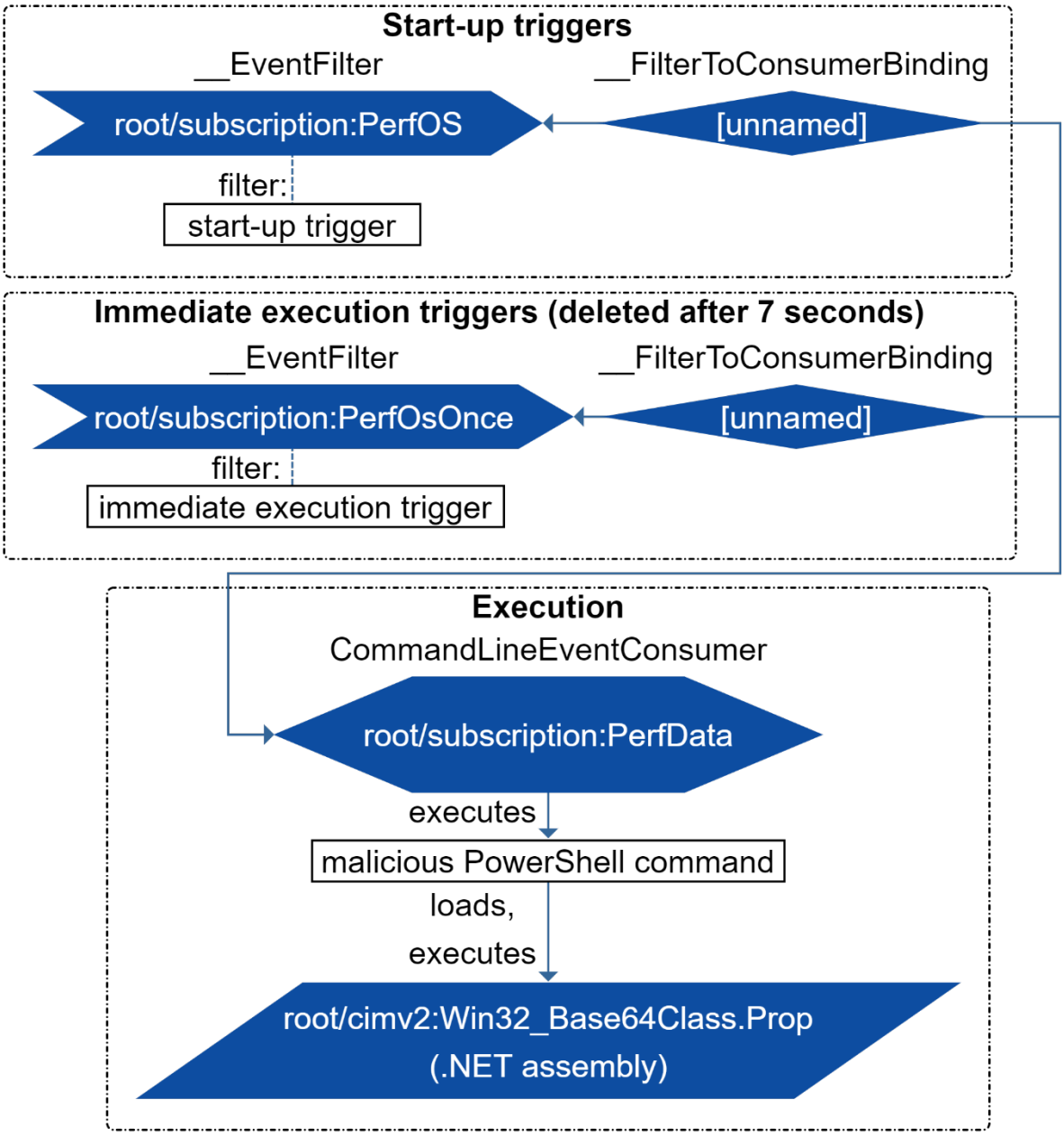

Fig. 1: WMI persistence objects

The third stage .NET DLL is contained in the second-stage PowerShell script, in 2 versions, one 32-bit and one 64-bit, each encoded in a base-64 string. The architecture-specific version is installed in a WMI "`root/cimv2:Win32_Base64Class`" object, in a "`Prop`" property. The PowerShell command mentioned above loads that assembly, then executes its "`MSDAC.PerfOSChecker::StartCheck`" method.

.NET loader and downloader shellcode

This stage consists of a .NET assembly that contains a piece of shellcode, compressed using GZip and RC4-encrypted, using the key "`CA0ac8F6655244d2E10e7819BD337bf9`", which is contained in the binary.

Since the shellcode is both self-modifying and contains a self-inject feature, an unmodified copy of itself is required. It is prepended by a null-terminated "`4BMARC2WKL`" marker and copied in the same buffer, immediately before the copy that is executed, as seen in fig. 2. The second copy of the shellcode is executed from the beginning, using the `Marshal.GetDelegateForFunctionPointer` method.

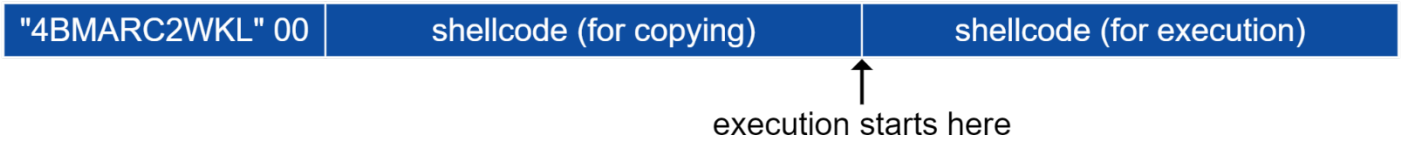

Fig. 2: Downloader shellcode buffer layout

Most of the shellcode is encrypted and begins with a simple decryption routine (fig. 3). After decryption, the shellcode resolves a long list of Win32 API functions and checks whether the current process (where the .NET assembly DLL and downloader shellcode is running) is "`powershell.exe`". This is expected when the DLL has been started using WMI, as the .NET assembly is loaded through PowerShell. In this case, the shellcode injects the unmodified copy mentioned above (located using the "`4BMARC2WKL`" marker) into a new thread in the parent process (in case of WMI persistence, the "`wmiprvse.exe`" process) and returns. This could be done to prevent a long-running PowerShell process, which could raise suspicions; the shellcode will restart executing in the parent process, decrypting itself and checking the process name again. This execution flow is explained in fig. 4.

```
                mov     r9b, 0D1h       ; initial key
                mov     rcx, 2BB4h      ; number of bytes to decrypt
                lea     r11, loc_19+1   ; address of piece to decrypt

decrypt_top:                            ; CODE XREF: seg000:loc_19↓j
                xor     [r11+rcx], r9b  ; decrypt a byte of code
                add     r9b, [r11+rcx]  ; update key

loc_19:                                 ; DATA XREF: seg000:0000000000
                loop    decrypt_top     ; decrypt a byte of code
; ---------------------------------------------------------------------------
                db  95h
```

Fig. 3: Shellcode decryption routine

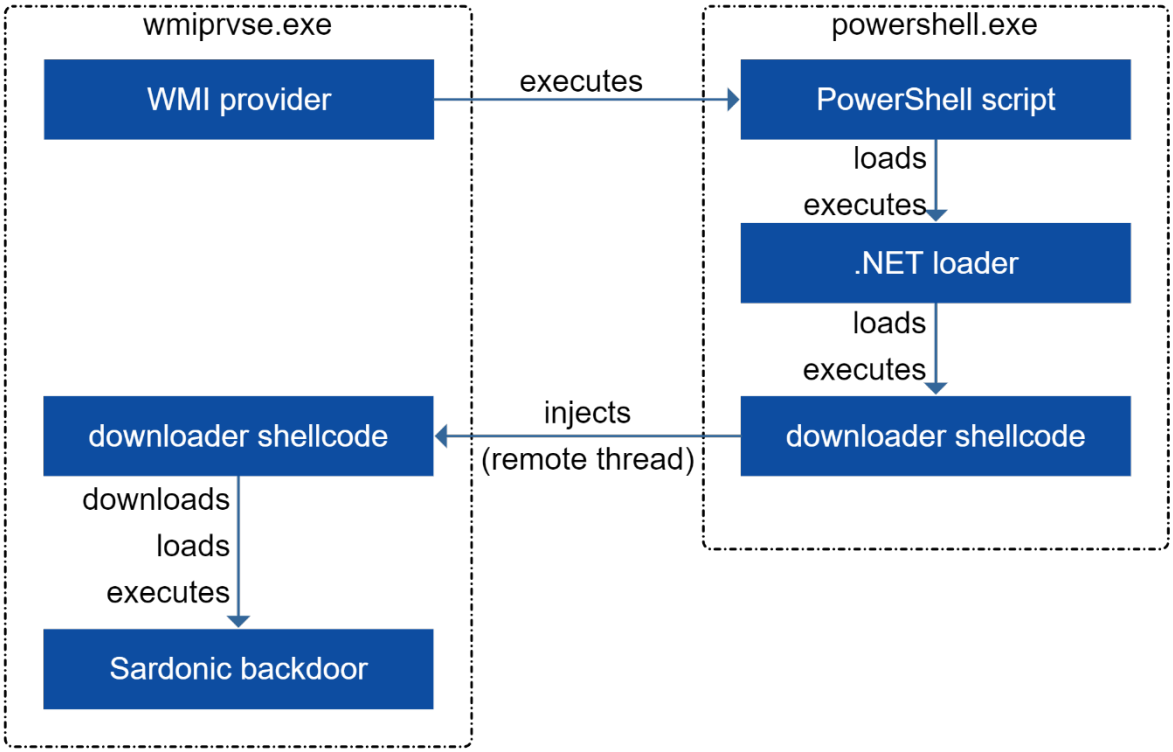

Fig. 4: Execution flow

When not running inside PowerShell the downloader routine starts: A 32-bit or 64-bit binary (depending on which .NET assembly was persisted by the PowerShell first stage) is downloaded from one of the following C&C servers, in order of priority: "`api-cdn[.]net`", "`git-api[.]com`", "`api-cdnw5[.]net`". A binary protocol over port 443 is used, identical to the one used by the final backdoor, described below. If a server has port 443 closed, the next server will be tried after 50 minutes (a much longer sleep time than normal, probably to avoid triggering any network detection).

The downloaded file is loaded in memory and its *Run* export is executed (all three argument values are zero). This is the final stage, the **Sardonic backdoor**. After the export function is executed, the shellcode exits. If it could not be loaded or executed, the download retries with the first C&C server.

Sardonic backdoor

The Sardonic backdoor is written in C++ and has the same C&C servers as the downloader shellcode ("`api-cdn[.]net`", "`git-api[.]com`", "`api-cdnw5[.]net`"), and talks over port 443 as well. It can obtain information about the system, execute commands, and has a plugin system that can load specially made DLLs and execute their functions. We believe that more than one person developed this project, as there are some differences in code style and level of use of the C++ standard library between functions.

Protocol

Communication with the C&C server, both by the downloader shellcode and the backdoor binary, is performed using a custom little-endian binary protocol with a 12-byte header. The first 4 bytes contain the size of the payload (excluding header), the next 4 bytes contain some flags (meanings are unknown, but a flag of 4 is used for messages from the client to the server), and the last 4 bytes of the header contain a 32-bit operation code (either a command or response type). The payload follows the 12-byte header, and its structure depends on the operation code.

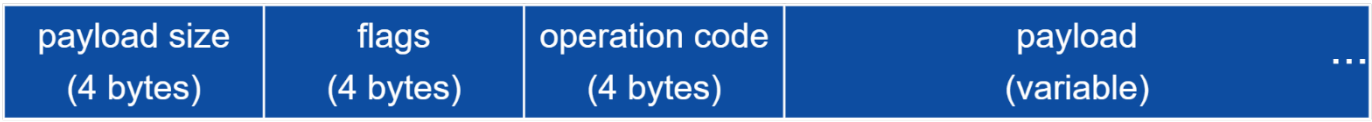

Fig. 5: Sardonic C&C communication binary protocol

Every packet sent to the C&C server contains a client ID made of 32 uppercase hex characters. This is constructed from the computer name, CPUID instruction results (number of fields and CPU manufacturer name), and C:\ drive serial number, processed down to 16 bytes that are then hex-encoded.

Only one connection with any of the three C&C servers can be open at any time. Additionally, the first server to have port 443 open will be used exclusively, even if it does not respond according to the protocol. The communication flow is as follows: immediately after connection, the client sends a packet containing its client ID. After that, until one of the parties closes the socket, repeatedly, the server sends a command and the client responds with a report that contains a maximum of 64KB (65536 bytes) of text data.

Despite running on port 443, no TLS encryption is used. Instead, the communication is initially plaintext, but the server can enable encryption at any time by sending a packet with the operation code `0x39`. The client generates a random 64-byte RC4 key at startup and, at the server's request, sends it encrypted using an RSA public key. All following communication in either direction will have the payload (but not the header) encrypted with that RC4 key.

Although the encryption features might seem comprehensive at first glance, using a mix of symmetric and asymmetric encryption, it can be easily defeated either through a man-in-the-middle attack, where a third party could communicate with the client in plain text and impersonate it to the server, or by using the fact that the private RSA key is publicly available. The public key comes from an open-source X.509 certificate used for testing in the OpenSSL repository, "`/C=AU/ST=Queensland/O=CryptSoft Pty Ltd/CN=Test CA (1024 bit)`", whose private key is also known. This is an unusual choice: the certificate is very old (issued on Dec. 5, 1999 and expired on June 10, 2005) and creating an RSA key pair is very easy. The result of this oversight is that, by identifying RSA-encrypted packets and extracting the RC4 key inside them, the entire communication can be decrypted from a traffic capture, which can be very useful in a forensic investigation.

Common backdoor commands

The Sardonic backdoor seems to be still under development; while analyzing the samples, we found a few commands for which the binary protocol parsing is implemented, but the execution of those commands is not implemented. However, Sardonic already provides enough features for the threat actors to complete their job.

A common feature in backdoors is the ability to execute a shell command and send its output back to the C&C server. For Sardonic, this behavior is triggered by a packet with operation code `0x4D`, with the command line as an argument. The backdoor runs it with the standard output and error streams redirected to memory using a pipe. After the command finishes, the output is sent to the C&C server.

Implemented in most backdoors is the ability to gather information about the system, which is then sent to the C&C server. When the server sends a packet with operation code 0x43, a JSON string is created, containing the following fields in this order:

- "`ver`": version string; we've seen "`v2.2.1r`", "`v2.2.4r`", and we have seen different samples with the same version string.
- "`modules`": space-separated list of installed plugins (which will be explained below)
- "`systeminfo`": output of "`cmd /c chcp 65001 & systeminfo`"
- "`netview`": output of "`cmd /c chcp 65001 & net view`"
- "`netstat`": output of "`cmd /c chcp 65001 & netstat /naop TCP`"

- "`tasklist`": output of "`cmd /c chcp 65001 & tasklist /v`"
- "`netstart`": output of "`cmd /c chcp 65001 & net start`"
- "`ipconfig`": output of "`cmd /c chcp 65001 & ipconfig`"

Interestingly, the shell command outputs are not properly escaped. In version `v2.2.1r`, no escaping is done, and the abundance of newlines alone violates the JSON standard. Even if that can be ignored by a custom JSON parser, a stray double quote in one of the command outputs can cause the string to end prematurely, which almost certainly leads to unparseable JSON. This is partially fixed in version `v2.2.4r`, which converts double quote characters inside command outputs to single quote characters. This fixes the critical bug mentioned above, but still requires a non-standards-compliant JSON parser that allows newlines and other special characters unescaped inside strings.

The system information feature is the only one we saw developed between versions; fields "netstat", "tasklist", "netstart" have been added between v2.2.1r and v2.2.4r.

Finally, when a packet with operation code `0xB1` is received, the backdoor exits.

Plugin system

Sardonic plugins are MZPE DLLs with a *Run* exported function taking 3 arguments (argument count, array of null-terminated argument strings, and a 64KB output buffer), and an optional *Stop* exported function taking 2 arguments (reason integer and a 64KB output buffer). The output buffers are sent to the C&C as part of the report. The plugins are identified by a C&C-given name, and are saved and executed in memory (fileless operation). Command names in this section are obtained from Runtime Type Information (RTTI).

To install a plugin, the C&C sends an *Update* command (operation code `0x25`), which contains a plugin name and its image data. Counterintuitively, given the command name, the plugin name must not already exist. The DLL data is saved to memory unprocessed, and the plugin is not loaded or executed yet.

To execute an installed plugin, a *Run* command must be sent (operation code `0x1B`), specifying a plugin name and an arbitrary number of function parameters. The name is looked up in the list of installed plugins and loaded in memory (sections are copied, imports are resolved, relocations are applied, the TLS directory is loaded and the DLL entry point is executed). Then, the module's *Run* export is executed with the supplied arguments. After it finishes, the function's output is sent in the report.

An additional command, *RunModule* (operation code 0x9D), is implemented, which bundles the *Update* and *Run* commands, installing a new module and executing it in a single step.

When a plugin needs to be stopped, a *StopModule* command is issued, giving it a module name and stop reason integer. The plugin's *Stop* method is executed and its output is, as before, sent in the report. Then, the module is unloaded, but its DLL image data is kept and can be executed again with a subsequent *Run* command.

An overview of plugin commands is depicted in figure 6.

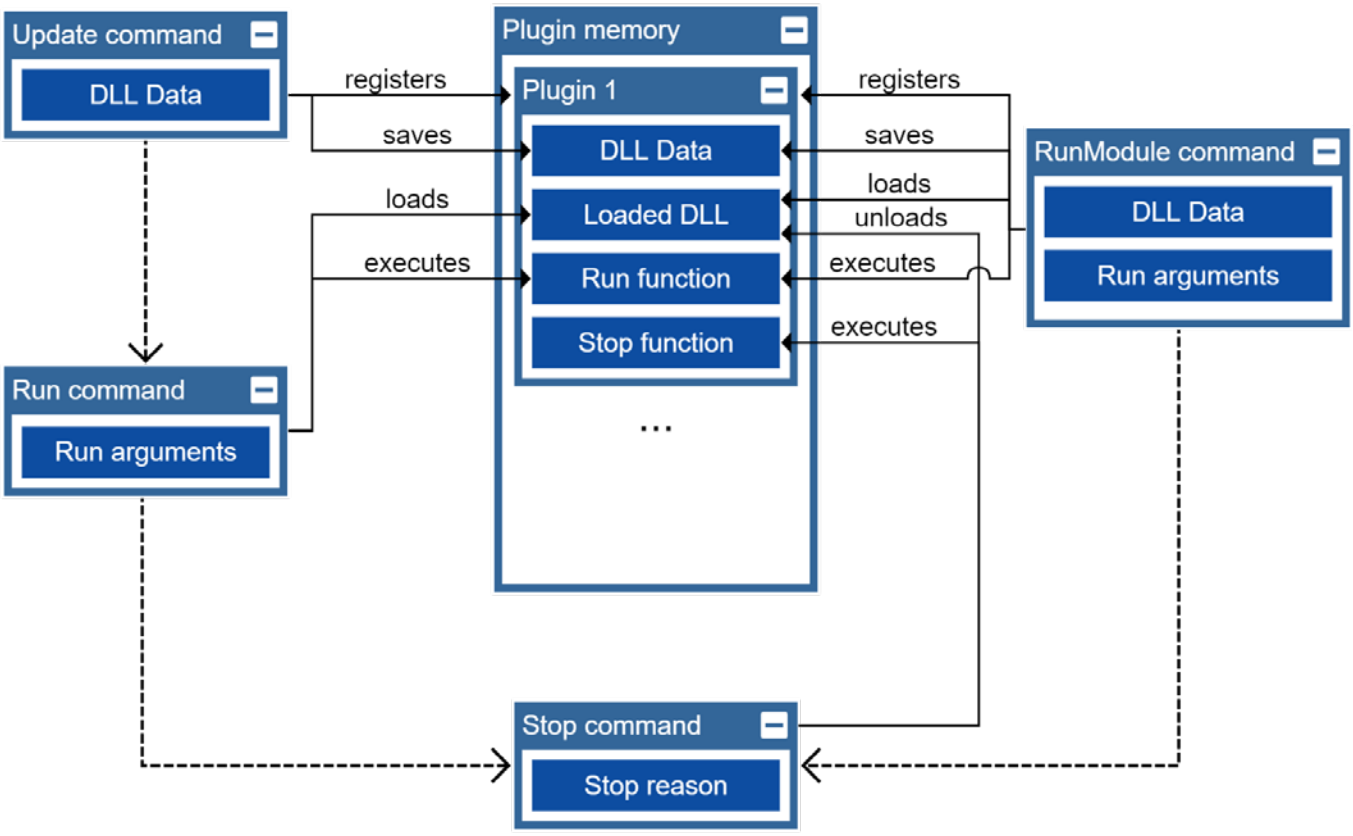

Fig. 6: Sardonic plugin commands

Unimplemented commands

From unused code and Runtime Type Information added by the compiler, we can infer a few features for which the parsing was implemented, but not the execution.

An *Install* command exists, having the same operation code as the one used by the downloader shellcode in the previous stage when requesting and receiving the backdoor DLL. The backdoor can apparently understand these packets but can't execute them yet.

Another interesting command name is *Multiplex*. This command's parsing in the binary protocol is assigned to multiple operation codes, which points to a large number of variations, but we can only speculate on its use.

Surprisingly, commands related to file manipulations are not implemented, but can still be performed by running shell commands or through a plugin. However, we found parsing for an *Upload* command, which we assume would be a file upload feature.

Naming and TTPs

Based on PDBs paths and in-the-wild filenames, we can infer the internal name of at least the PowerShell and .NET stages: Sardonic Loader, by combining the PDB path for the .NET assembly "`C:\Users\dev_win10_00\Documents\`**`Sardonic`**`\`**`SardonicUtility`**`\`**`Loader`**`Assembly\obj\x86\Release\MSDAC.pdb`" and PowerShell script name "`sldr.ps1`". Additionally, judging from the PDB parent directories and loader name, the name used by the attackers for the entire backdoor ecosystem is likely Sardonic.

In the attack we investigated, we saw a few interesting TTPs: Firstly, inspecting command lines used in the attack, we found evidence of both the *smbexec* and *wmiexec* tools from the Impacket suite, which are used for lateral movement. The usage of a "`C:\Windows\TEMP\execute.bat`" path is a smbexec IOC, and a command line ending with "`> \\127.0.0.1\ADMIN$\__<timestamp> 2>&1`" (with the timestamp in Unix format with fractional seconds) is an indicator of wmiexec usage, as can be seen in the source code for both tools.

Another interesting technique is the use of the ping command to check connectivity to the C&C servers after running the first-stage PowerShell script. This indicates the attackers did not observe the expected C&C connection from the machine, which is because one of the links in the loader chain was blocked by the security solution. This technique is used to ensure that the failure is not due to network connectivity issues or firewalls.

Thirdly, the attackers tried to evade detection by renaming the PowerShell script from "`sldr.ps1`" to "`s.txt`" and "`s.ps1`". This did not succeed in bypassing the security solution.

Finally, from previous versions we saw earlier this year, the BADHATCH loader has been improved with an RC4 layer of encryption, whose key is not contained in the loader script, but received as a command-line argument. This is a defense evasion tactic aimed at sample repositories (either public, like VirusTotal, or private): without the separately stored key, it is impossible to obtain the payload of these scripts, even if such a sample is found. However, we found at least one loader file from a different attack that reused this password (hex-encoded: `b640a9c0e64704e1e202a07774613a29971fe5aa`). We do not know yet if this key holds for multiple samples or if this is a one-time error on their part.

Recommendations

FIN8 continues to strengthen its capabilities and malware delivery infrastructure. The highly skilled financial threat actor is known to take long breaks to refine tools and tactics to avoid detection before it strikes viable targets.

Bitdefender recommends that companies in target verticals (retail, hospitality, finance) check for potential compromise by applying the following IoCs to their EDR, XDR and other security defenses.

To further minimize the impact of financial malware, companies should:

- 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 Managed Detection and Response providers.

Indicators Of Compromise

Domains
api-cdn[.]net
git-api[.]com
api-cdnw5[.]net
104-168-237-21.sslip[.]io
89.45.4[.]192

URLs
https://104-168-237-21.sslip[.]io/134af6
https://104-168-237-21.sslip[.]io/edaea0

Hashes
ede6ca7c3c3aedeb70e8504e1df70988263aab60ac664d03995bce645dff0935
5b8b732d0bb708aa51ac7f8a4ff5ca5ea99a84112b8b22d13674da7a8ca18c28
4e73e9a546e334f0aee8da7d191c56d25e6360ba7a79dc02fe93efbd41ff7aa4
05236172591d843b15987de2243ff1bfb41c7b959d7c917949a7533ed60aafd9
edfd3ae4def3ddffb37bad3424eb73c17e156ba5f63fd1d651df2f5b8e34a6c7
827448cf3c7ddc67dca6618f4c8b1197ee2abe3526e27052d09948da2bc500ea
0e11a050369010683a7ed6a51f5ec320cd885128804713bb9df0e056e29dc3b0
0980aa80e52cc18e7b3909a0173a9efb60f9d406993d26fe3af35870ef1604d0
64f8ac7b3b28d763f0a8f6cdb4ce1e5e3892b0338c9240f27057dd9e087e3111
2d39a58887026b99176eb16c1bba4f6971c985ac9acbd9e2747dd0620548aaf3
8cfb05cde6af3cf4e0cb025faa597c2641a4ab372268823a29baef37c6c45946
72fd2f51f36ba6c842fdc801464a49dce28bd851589c7401f64bbc4f1a468b1a
6cba6d8a1a73572a1a49372c9b7adfa471a3a1302dc71c4547685bcbb1eda432

Filenames
sldr.ps1

WMI objects
root/cimv2:Win32_Base64Class
root/subscription:PerfOs
root/subscription:PerfData
root/subscription:PerfOsOnce
Unnamed object in root/subscription, class “__FilterToConsumerBinding”, filter value “__eventfilter.name='PerfOs'”
Unnamed object in root/subscription, class “__FilterToConsumerBinding”, filter value “__eventfilter.name='PerfOsOnce'”

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

CRN AVTEST AV Gartner 451 Research FORRESTER IDC GLOBAL

TECHNOLOGY ALLIANCES

Microsoft NUTANIX aws Pivotal Cloud Foundry CITRIX

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.

FIN_
08/24/21August 20, 2021 1:05 pm(08/24/21)