Open-source toolset of an Ivanti CSA attacker

By Maxence Fossat

Archived: 2026-04-05 18:05:12 UTC

In recent incident responses where the root cause was an Ivanti CSA compromise, Synacktiv's CSIRT came across multiple open-source tools used by threat actors. This article dives into each of these tools, their functionalities and discusses efficient detection capabilities.

Introduction

In September and October 2024, Ivanti published multiple[1] security[2] advisories[3] regarding security policy bypasses and remote code execution vulnerabilities in their Cloud Services Appliance (CSA) product. It was later revealed by FortiGuard Labs Threat Research's work[4] that some threat actors had been actively chaining these vulnerabilities as early as September 9, 2024, before any security advisory or patch was publicly released by Ivanti.

In some compromise scenarios, even though the initial access stemmed from the exploitation of zero-day vulnerabilities, later stages were short of such proficient attacker tradecraft. Threat actors were seen using known malicious tools and noisy payloads for lateral movement, persistence and credential dumping.

Synacktiv's CSIRT was recently in charge of different forensic investigations where the root cause was a vulnerable CSA appliance exposed to the internet. During these engagements, we found a set of open-source tools used by the attacker to achieve its goals. In this article, we take a tour of the OSS toolset from an Ivanti CSA exploiter and discuss related detection capabilities.

suo5

After initial compromise of the Ivanti CSA appliance, the attacker managed to move to an internet-facing Exchange Server used for OWA connections. They planted an HTTP proxy tunnel going by the name of **suo5**. Available publicly on GitHub, with an initial commit dated February 2023, this tool is presented as a more performant alternative to other tunnelling tools such as reGeorg and Neo-reGeorg.

Interestingly enough, the GLASSTOKEN custom webshell, used during intrusions linked to the exploitation of 0-day vulnerabilities in Ivanti Connect Secure VPN[5] in January 2024, was based on the Neo-reGeorg tool. Additionally, the reGeorg tool was used as a commodity webshell in attacks linked to the exploitation of 0-day vulnerabilities in Microsoft Exchange[6] in March 2021.

The `suo5.aspx` webshell was dropped on the Exchange Server at the location `C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\OutlookEN.aspx`. This name could be a reference to HAFNIUM-related activities during post-exploitation of the Microsoft Exchange vulnerabilities[7] in March 2021.

Functionalities

For suo5 to work, an attacker-accessible web server (usually a server exposed to the internet) must run suo5 server-side code, available in .NET, Java or PHP (experimental). In our test case, the `OutlookEN.aspx` page run by the IIS component of the Exchange Server is the .NET-flavor of suo5 server-side code.

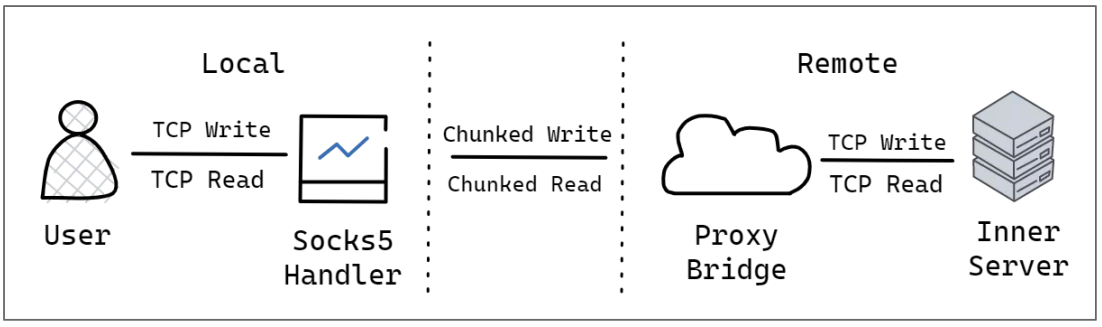
On the attacker's side, a SOCKS5 proxy must be set up and the suo5 binary communicates with the server-side code to transmit TCP data encapsulated in the HTTP(S) communication.

```
 $ ./suo5 -t https://mail.corporation.local/owa/auth/OutlookEN.aspx
[INFO] 02-27 13:28 header:
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
Referer: https://mail.corporation.local/owa/auth/
[INFO] 02-27 13:28 method: POST
[INFO] 02-27 13:28 connecting to target https://mail.corporation.local/owa/auth/OutlookEN.aspx
[INFO] 02-27 13:28 got data offset, 0
[WARN] 02-27 13:28 the target may behind a reverse proxy, fallback to HalfDuplex mode
[INFO] 02-27 13:28 starting tunnel at 127.0.0.1:1111

[Tunnel Info]
Target:  https://mail.corporation.local/owa/auth/OutlookEN.aspx
Proxy:   socks5://127.0.0.1:1111
Mode:    half

[INFO] 02-27 13:28 creating a test connection to the remote target
[INFO] 02-27 13:28 start connection to 127.0.0.1:0
[INFO] 02-27 13:28 successfully connected to 127.0.0.1:0
[INFO] 02-27 13:28 connection closed, 127.0.0.1:0
[INFO] 02-27 13:28 congratulations! everything works fine
[INFO] 02-27 13:28 start connection to 192.168.123.15:22
[INFO] 02-27 13:28 successfully connected to 192.168.123.15:22
```

Using a tool like `proxychains`, the attacker can then communicate with any system accessible by the Exchange Server. This enables him to get a foothold inside the internal network of the victim. In our test case, using suo5 enables us to connect to a system (`192.168.123.15`) on another subnet, which would have been otherwise inaccessible:

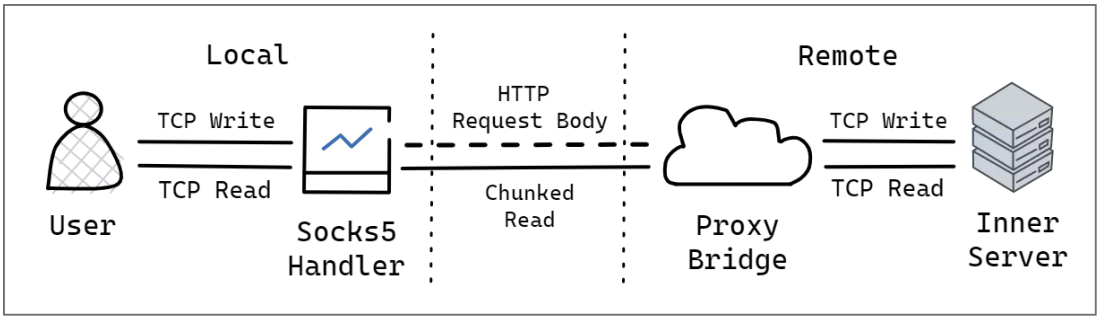
```
 $ proxychains ssh victim@192.168.123.15
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] Strict chain  ...  127.0.0.1:1111  ...  192.168.123.15:22  ...  OK
victim@192.168.123.15's password:
Linux lab-victim 6.1.0-28-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.119-1 (2024-11-22) x86_64
```

The blog post associated with the tool[8] explains how suo5 achieves better performance than its competitors: it uses the `chunked` directive for the `Transfer-Encoding` HTTP/1.1 header[9]. This essentially means that the request's sender can keep the connection open until it notifies the recipient that the entire message has been sent. In this mode, all encapsulated TCP packets are contained in a single HTTP connection, reducing overhead of multiple HTTP requests and responses. This is the "**full duplex**" mode advertised in the GitHub project's `README` :



suo5 full duplex mode (source: https://koalr.me/posts/suo5-a-hign-performace-http-socks/).

Due to a limitation in .NET HTTP request processing[10], the "full-duplex" mode cannot be used. The tool then falls back to "**half-duplex**" mode, where only the response is sent as a continuous data stream, but requests are sent as separate HTTP requests. The goal of "half-duplex" mode is also to bypass restrictions of an Nginx reverse proxy that would stand between the suo5 client and the server-side code. By default, Nginx buffers requests[11], a limitation that cannot be circumvented without modifying the reverse proxy's configuration. However, Nginx also buffers responses by default except if the `X-Accel-Buffering: no` header field is sent as part of the response[12] (which is the case for the suo5 webshell). This results in a connection still having better performance than Neo-reGeorg:



suo5 half duplex mode (source: https://koalr.me/posts/suo5-a-hign-performace-http-socks/).

Detection

To detect the server-side file (either on disk or loaded in memory), we want to create a YARA rule that avoids using easily changeable data like function and variable names. We want to focus on detecting functionalities of the webshell. One contextual IOC that we could use, however, is the `User-Agent` that acts as a kind of "password" so that only the client is authorized to connect to the webshell. Careless attackers might leave the default value:

```
Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.1.2.3
```

This is the User-Agent of a Chrome browser (version 109.1.2.3) on an LG Nexus 5 smartphone (Android 6.0 - Marshmallow).

Stealthier attackers will change the `User-Agent` checked by the server:

```
private bool checkAuth()
{
    string ua = Request.Headers.Get("User-Agent");
    if (ua == null || !ua.Equals("CUSTOM_USER_AGENT"))
    {
        return false;
    }
```

They will then pass that `User-Agent` string as an argument to the suo5 client, via the `--ua` flag:

```
$ ./suo5 --target https://mail.corporation.local/owa/auth/OutlookEN.aspx --ua "CUSTOM_USER_AGENT"
```

Due to the nature of this string, which goal is to discriminate against other suo5 clients or unintentional connections to the webshell, there is a high chance that it will be relatively unique, leaving an interesting IOC for DFIR and SOC analysts.

The following YARA rule focuses on key characteristics of the webshell:

- Default User-Agent.
- `X-Accel-Buffering: no` response header.
- User-Agent-based authentication.
- Usage of TCP client to proxy communications.
- Binary serialization and de-serialization.
- Byte XORing and random key generation.

```
rule SYNACKTIV_WEBSHELL_ASPX_Suo5_May25 : WEBSHELL COMMODITY FILE
{
    meta:
        description = "Detects the .NET version of the suo5 webshell"
        author = "Synacktiv, Maxence Fossat [@cybiosity]"
        id = "d30a7232-f00b-45ab-9419-f43b1611445a"
        date = "2025-05-12"
        modified = "2025-05-12"
        reference = "https://www.synacktiv.com/en/publications/open-source-toolset-of-an-ivanti-csa-attacker"
        license = "DRL-1.1"
[...]
        score = 75
```

```
        tags = "WEBSHELL, COMMODITY, FILE"
        tlp = "TLP:CLEAR"
        pap = "PAP:CLEAR"

    strings:
        $user_agent = ".Equals(\"Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHT
        $header = "Response.AddHeader(\"X-Accel-Buffering\", \"no\")" ascii    // X-Accel-Buffering response hea
        $xor = /= \(byte\)\(\w{1,1023}\[\w{1,1023}\] \^ \w{1,1023}\);/    // XOR operation

        // suspicious functions
        $s1 = "Request.Headers.Get(\"User-Agent\")" ascii
        $s2 = "if (Request.ContentType.Equals(\"application/plain\"))" ascii
        $s3 = "Response.ContentType = \"application/octet-stream\";" ascii
        $s4 = "= Request.BinaryRead(Request.ContentLength);" ascii
        $s5 = "= Response.OutputStream;" ascii
        $s6 = "new TcpClient()" ascii
        $s7 = ".BeginConnect(" ascii
        $s8 = ".GetStream().Write(" ascii
        $s9 = "new BinaryWriter(" ascii
        $s10 = "new BinaryReader(" ascii
        $s11 = ".ReadBytes(4)" ascii
        $s12 = "BitConverter.GetBytes((Int32)" ascii
        $s13 = "BitConverter.ToInt32(" ascii
        $s14 = "Array.Reverse(" ascii
        $s15 = "new Random().NextBytes(" ascii

    condition:
        filesize < 100KB and ( $user_agent or ( ( $header or $xor ) and 8 of ( $s* ) ) or 12 of ( $s* ) )
}
```

Security professionals wishing to set up an IDS/IPS rule to detect HTTPS suo5 traffic are out of luck. Except for having TLS interception, their best bet would have been to detect specific client JA3[13]/JA4[14] fingerprints, but suo5 uses the [Randomized TLS Client Hello Fingerprint] functionality of the uTLS[15] Go library:

```
 uTlsConn := utls.UClient(conn, &utls.Config{
    InsecureSkipVerify: true,
    ServerName:         hostname,
    MinVersion:         utls.VersionTLS10,
    Renegotiation:      utls.RenegotiateOnceAsClient,
}, utls.HelloRandomizedNoALPN)
```

iox

The attacker wished to keep a foothold inside the victim's network that would last even after patches were applied on the Ivanti CSA appliance. They proceeded to deploy another tunnel named **iox**, directly on the internet-facing

appliance.

In its GitHub page, **iox** is presented as a better alternative to two other port forwarding tools: **lcx** (also known as **HTran**16 for **HUC Packet Transmit Tool**) and **ew** (also known as **Earthworm**17).

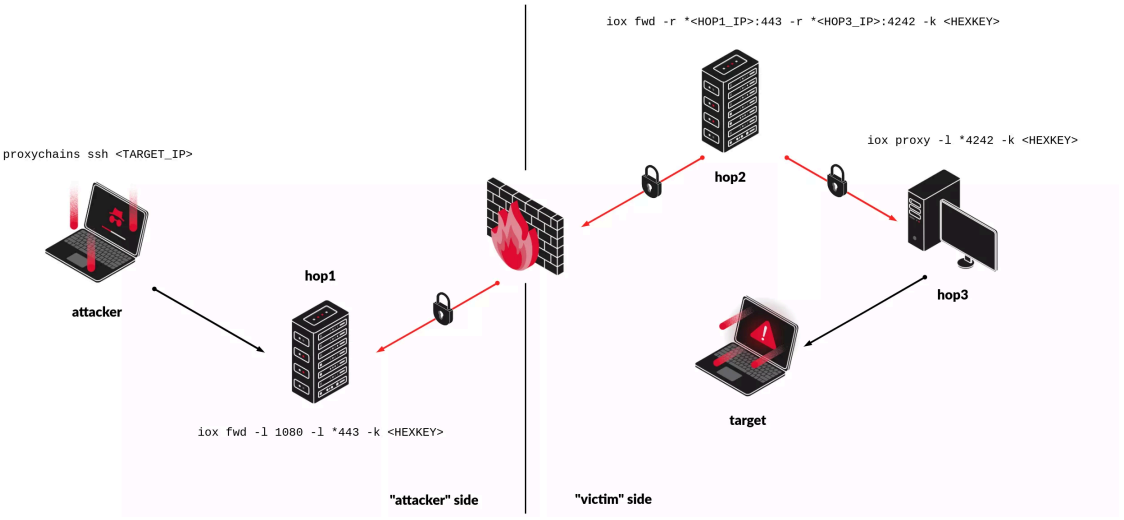
The **lcx** tool is a piece of history linked to Chinese hacking culture in the early 2000s. It is a port forwarding tool, created around 200318 by a Chinese hacker going by the name "Li0n" which is the founder of a nationalist hacking group called Honker's Union of China (HUC)19. It has been used20 in numerous21 attacks22 attributed to Chinese threat actors23 over the years.

The **Earthworm** tool came as a more powerful replacement to the ageing **lcx/Htran** one. It incorporated **lcx**'s core concepts (different modes of port forwarding) and added SOCKS5 proxy and reverse SOCKS5 proxy functionalities. Its versatility made it an interesting tool to make complex forwarding chains, abstracting connection from a source to a destination host through multiple hops and taking advantage of the few open ports of each firewall in the way24. It was used as a commodity tunnelling tool in different attacks such as the active exploitation of the ProxyShell vulnerability by UNC298025 or the attack on US critical infrastructure by Volt Typhoon26.

Functionalities

iox has similar functionalities to **Earthworm**, with a cleaner syntax and some network code optimizations. It adds optional traffic encryption and UDP traffic forwarding.

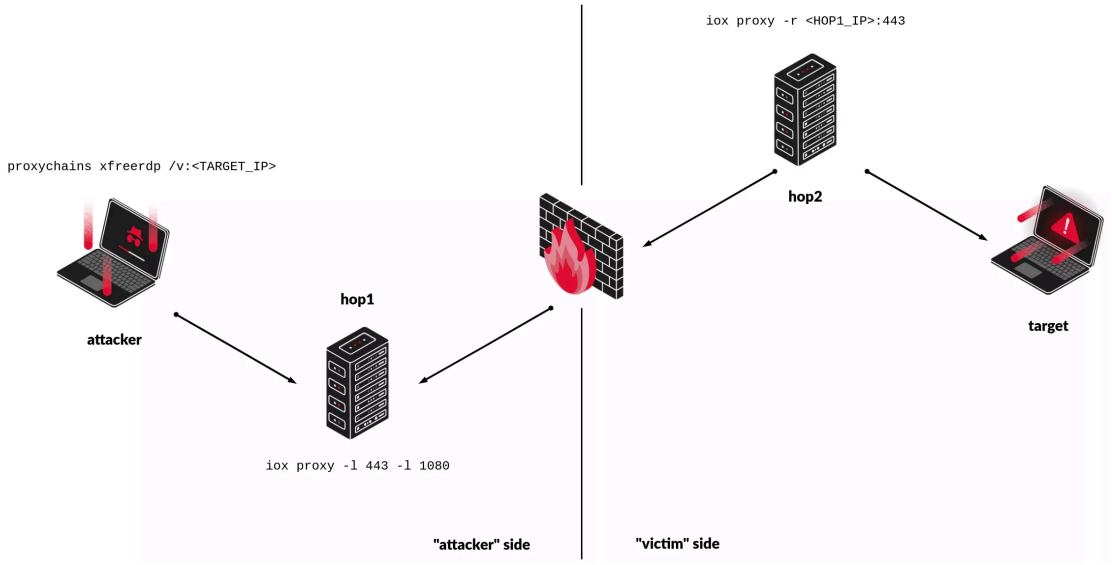
iox's port forwarding capabilities can be used to circumvent network flow restrictions (using well-known ports) and obfuscate the true source of network connections when a foothold has been established inside the victim's LAN. In the following example, the edge server (second hop — **hop2**) can communicate with outside hosts via port 443, so it establishes a connection to the first hop (**hop1**). The third hop in the chain (**hop3**) runs a SOCKS5 proxy. From the point of a view of the victim's network, requests that are coming from the attacker seem to originate from the third hop (**hop3**).



Example of iox's port forwarding and proxy capabilities.

Communications that are forwarded inside the victim's internal network are encrypted with symmetric encryption algorithm XChaCha20. This encryption is not meant to be a bullet-proof protection of the privacy of forwarded content, but an evasion technique against NIDS/NIPS.

A reverse SOCKS5 proxy can also be established. In the following scenario, traffic coming from the attacker machine seems to originate from the second hop (**hop2**).



Example of iox's reverse SOCKS5 proxy capabilities.

In this scenario, traffic encryption, which is an optional feature, is not activated.

Detection

Existing public detection rules rely on function and library names[27] or on arguments passed to the command line[28]. In the interest of adding complementary rules to the open-source community, we worked on detection rules relying on different identifiers.

During our engagement, we found an obfuscated sample of iox. Function names, library names, arguments and most of the strings had been removed or obfuscated. We then worked on a detection logic that would rely on discriminating patterns in the code. We needed homemade functions which logic would not rely on standard libraries. We focused on the homemade `iox/crypto` library.

Two constructs were chosen for our detection logic:

```
if len(key) < 0x20 {
   var c byte = 0x20 - byte(len(key)&0x1F)
```

in function `ExpandKey` and

```
bs[i] ^= byte(i) ^ bs[(i+1)%len(bs)]*((bs[len(bs)-1-i]*bs[i])%255)
```

in function `shuffle`.

Extracting small chunks of assembly code from the disassembled sample respectively gives the following patterns to detect:

```
 00000000004BFA73  48 8B 9C 24 88 00 00 00        mov    rbx, [rsp+88h]
00000000004BFA7B  48 83 FB 20                    cmp    rbx, 20h
00000000004BFA7F  0F 8D 87 02 00 00              jge    loc_4BFD0C
00000000004BFA85  48 89 DE                       mov    rsi, rbx
00000000004BFA88  48 83 E3 1F                    and    rbx, 1Fh
00000000004BFA8C  83 C3 E0                       add    ebx, 0FFFFFFE0h
00000000004BFA8F  F7 DB                          neg    ebx
```

and

```
 00000000004BF92C  44 0F B6 0C 07                 movzx  r9d, byte ptr [rdi+rax]
00000000004BF931  45 0F AF C8                    imul   r9d, r8d
00000000004BF935  41 BA FF FF FF FF              mov    r10d, 0FFFFFFFFh
00000000004BF93B  45 0F B6 DA                    movzx  r11d, r10b
00000000004BF93F  41 0F B6 C1                    movzx  eax, r9b
00000000004BF943  41 89 D1                       mov    r9d, edx
00000000004BF946  31 D2                          xor    edx, edx
00000000004BF948  66 41 F7 F3                    div    r11w
00000000004BF94C  41 0F AF D1                    imul   edx, r9d
00000000004BF950  31 CA                          xor    edx, ecx
00000000004BF952  41 31 D0                       xor    r8d, edx
00000000004BF955  44 88 04 0F                    mov    [rdi+rcx], r8b
```

In order to make our rule more effective while limiting false positives and maintaining sufficient performance, we structured it to detect patterns of assembly instructions regardless of registers used. For example, the instruction `cmp rbx, 20h` could use any register for integer arguments and results specified in the Go internal ABI specification[29]: `RAX, RBX, RCX, RDI, RSI, R8, R9, R10, R11`. While making sure to use 4-byte patterns as much as possible (to improve performance with the YARA/YARA-X engine), this results in the following hexadecimal string:

```
( 48 83 F8 20 | 48 83 FB 20 | 48 83 F9 20 | 48 83 FF 20 | 48 83 FE 20 | 49 83 F8 20 | 49 83 F9 20 | 49 83 FA 2(
```

The resulting YARA/YARA-X detection rule is as follows:

```
 rule SYNACKTIV_HKTL_Tunnel_X64_GO_Iox_May25 : COMMODITY FILE
{
    meta:
        description = "Detects the 64-bits version of the iox tunneling tool used for port forwarding and SOCKS5
        author = "Synacktiv, Maxence Fossat [@cybiosity]"
```

```
        id = "0b5a4689-58ea-45d5-aa14-a1455276352a"
        date = "2025-05-12"
        modified = "2025-05-12"
        reference = "https://www.synacktiv.com/en/publications/open-source-toolset-of-an-ivanti-csa-attacker"
        license = "DRL-1.1"
[...]
        score = 75
        tags = "COMMODITY, FILE"
        tlp = "TLP:CLEAR"
        pap = "PAP:CLEAR"

    strings:

        /*
        00000000004BFA73  48 8B 9C 24 88 00 00 00        mov    rbx, [rsp+88h]
        00000000004BFA7B  48 83 FB 20                     cmp    rbx, 20h
        00000000004BFA7F  0F 8D 87 02 00 00               jge    loc_4BFD0C
        00000000004BFA85  48 89 DE                        mov    rsi, rbx
        00000000004BFA88  48 83 E3 1F                     and    rbx, 1Fh
        00000000004BFA8C  83 C3 E0                        add    ebx, 0FFFFFFE0h
        00000000004BFA8F  F7 DB                           neg    ebx
        */
        $expand_key = {
            ( 48 8B 84 24 | 48 8B 9C 24 | 48 8B 8C 24 | 48 8B BC 24 | 48 8B B4 24 | 4C 8B 84 24 | 4C 8B 8C 24 |
            ( 48 83 F8 20 | 48 83 FB 20 | 48 83 F9 20 | 48 83 FF 20 | 48 83 FE 20 | 49 83 F8 20 | 49 83 F9 20 |
            ( 0F 8D ?? ?? ?? ?? | 7D ?? )
            ( 48 89 ?? | 49 89 ?? | 4C 89 ?? | 4D 89 ?? )
            ( 48 83 E0 1F | 48 83 E3 1F | 48 83 E1 1F | 48 83 E7 1F | 48 83 E6 1F | 49 83 E0 1F | 49 83 E1 1F |
            ( 83 C0 E0 | 83 C3 E0 | 83 C1 E0 | 83 C7 E0 | 83 C6 E0 | 41 83 C0 E0 | 41 83 C1 E0 | 41 83 C2 E0 | 4
            ( F7 D8 | F7 DB | F7 D9 | F7 DF | F7 DE | 41 F7 D8 | 41 F7 D9 | 41 F7 DA | 41 F7 DB )
        }

        /*
        00000000004BF92C  44 0F B6 0C 07                  movzx  r9d, byte ptr [rdi+rax]
        00000000004BF931  45 0F AF C8                     imul   r9d, r8d
        00000000004BF935  41 BA FF FF FF FF               mov    r10d, 0FFFFFFFFh
        00000000004BF93B  45 0F B6 DA                     movzx  r11d, r10b
        00000000004BF93F  41 0F B6 C1                     movzx  eax, r9b
        00000000004BF943  41 89 D1                        mov    r9d, edx
        00000000004BF946  31 D2                           xor    edx, edx
        00000000004BF948  66 41 F7 F3                     div    r11w
        00000000004BF94C  41 0F AF D1                     imul   edx, r9d
        00000000004BF950  31 CA                           xor    edx, ecx
        00000000004BF952  41 31 D0                        xor    r8d, edx
        00000000004BF955  44 88 04 0F                     mov    [rdi+rcx], r8b
        */
        $shuffle = {
```

```
            ( 44 0F B6 04 | 44 0F B6 0C | 44 0F B6 14 | 44 0F B6 1C | 44 0F B6 44 | 44 0F B6 4C | 44 0F B6 54 |
            ( 45 0F AF C0 | 45 0F AF C1 | 45 0F AF C2 | 45 0F AF C3 | 45 0F AF C8 | 45 0F AF C9 | 45 0F AF CA |
            [0-4]
            ( 41 B8 | 41 B9 | 41 BA | 41 BB ) FF FF FF FF
            ( 45 0F B6 C0 | 45 0F B6 C1 | 45 0F B6 C2 | 45 0F B6 C3 | 45 0F B6 C8 | 45 0F B6 C9 | 45 0F B6 CA |
            [0-4]
            ( 41 89 D0 | 41 89 D1 | 41 89 D2 | 41 89 D3 | 89 D0 | 89 D1 | 89 D2 | 89 D3 )
            31 D2
            ( 66 41 F7 F0 | 66 41 F7 F1 | 66 41 F7 F2 | 66 41 F7 F3 )
            ( 41 0F AF D0 | 41 0F AF D1 | 41 0F AF D2 | 41 0F AF D3 | 44 0F AF C2 | 44 0F AF CA | 44 0F AF D2 |
            ( 31 ?? | 41 31 ?? | 44 31 ?? | 45 31 ?? )
            ( 31 ?? | 41 31 ?? | 44 31 ?? | 45 31 ?? )
            ( 44 88 04 ?F | 44 88 0C ?F | 44 88 14 ?F | 44 88 1C ?F | 44 88 04 ?7 | 44 88 0C ?7 | 44 88 14 ?7 |
        }

    condition:
        ( uint16be( 0 ) == 0x4d5a or uint32be( 0 ) == 0x7f454c46 or uint32be( 0 ) == 0xcffaedfe ) and filesize <
}
```

This rule was tested with YARA, YARA-X and yaraQA to assess its performance. It helped us detect other in the wild obfuscated samples of the tool.

For the sake of completeness, the following rule, based only on strings found in the unobfuscated samples, was also created:

```
 rule SYNACKTIV_HKTL_Tunnel_GO_Iox_May25 : COMMODITY FILE
{
    meta:
        description = "Detects the iox tunneling tool used for port forwarding and SOCKS5 proxy"
        author = "Synacktiv, Maxence Fossat [@cybiosity]"
        id = "407d4f90-a281-4f0c-8d8e-ebe45217d3d9"
        date = "2025-05-12"
        modified = "2025-05-12"
        reference = "https://www.synacktiv.com/en/publications/open-source-toolset-of-an-ivanti-csa-attacker"
        license = "DRL-1.1"
[...]
        score = 75
        tags = "COMMODITY, FILE"
        tlp = "TLP:CLEAR"
        pap = "PAP:CLEAR"

    strings:
        $s1 = "Forward UDP traffic between %s (encrypted: %v) and %s (encrypted: %v)"
        $s2 = "Open pipe: %s <== FWD ==> %s"
        $s3 = "Reverse socks5 server handshake ok from %s (encrypted: %v)"
        $s4 = "Recv exit signal from remote, exit now"
```

```
        $s5 = "socks consult transfer mode or parse target: %s"

    condition:
        ( uint16be( 0 ) == 0x4d5a or uint32be( 0 ) == 0x7f454c46 or uint32be( 0 ) == 0xcffaedfe or uint32be(0) =
}
```

atexec-pro

The `atexec.py` 30 script is a well-known lateral movement tool leveraging Impacket31 classes to gain remote code execution on an endpoint through the Task Scheduler service.

The attacker executed commands on the Active Directory Domain Controller from another host inside the network using a variation of this script called `atexec-pro.py` 32.

Functionalities

The original `atexec.py` script first establishes an RPC connection over an SMB named pipe33 targeting the `\pipe\atsvc` 34 endpoint, and then uses this connection to specifically bind to the `ITaskSchedulerService` 35 interface for subsequent operations. It then creates a scheduled task, with a name made up of 8 random characters, which is configured to run a single command through the Command Prompt (`cmd.exe`) with the highest permissions available (`NT AUTHORITY\SYSTEM`) and to redirect the output to `%WINDIR%\Temp\<TASK_NAME>.tmp` .

The script triggers the task's execution, deletes the task, reads the content of the `.tmp` file through an SMB connection to the `ADMIN$` share and then deletes this file. This means the script only communicates with port 445 throughout its execution.

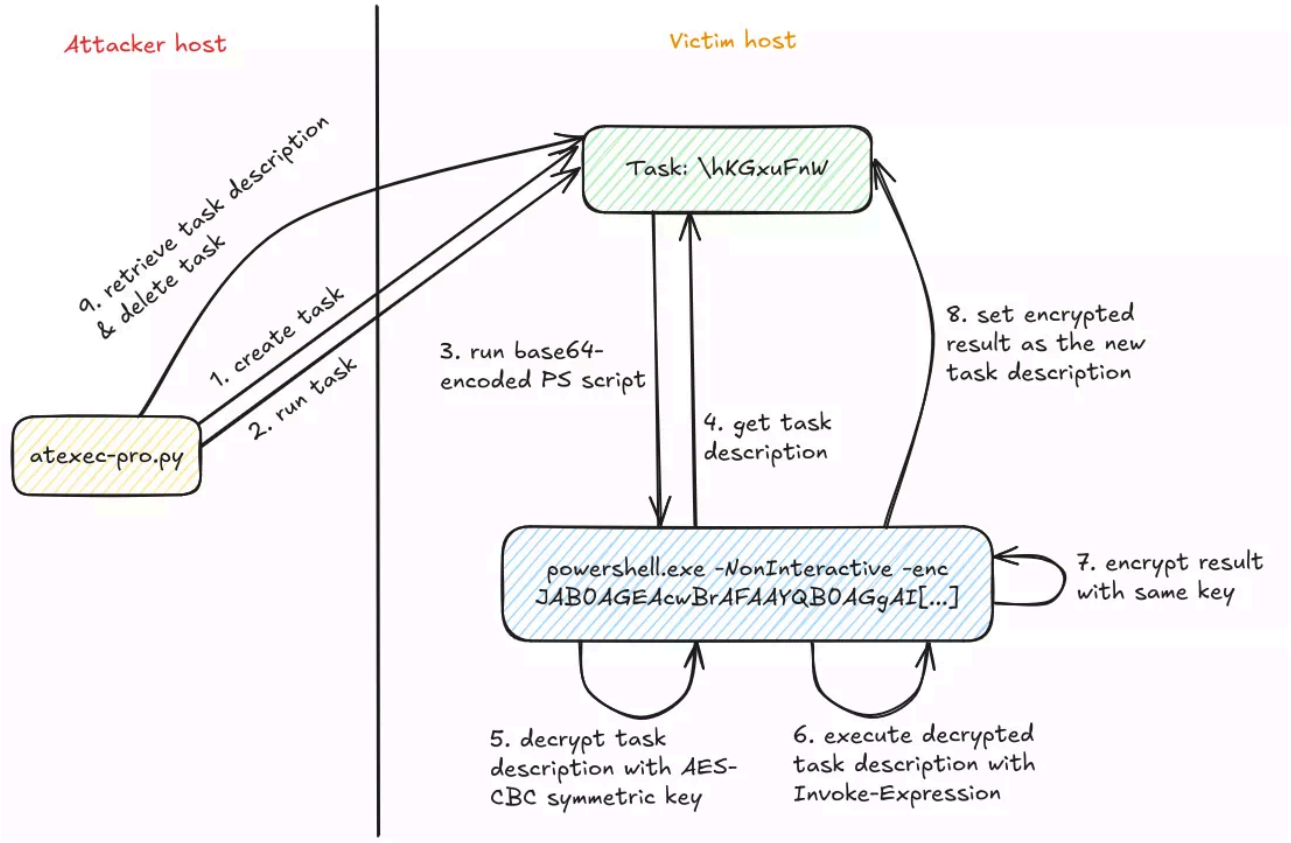
The `atexec-pro.py` script works quite differently. First of all, an alternative to RPC over SMB is offered, relying on RPC over TCP/IP36 for task manipulation. This protocol uses the RPC Endpoint Mapper, listening on port 135, to resolve the dynamic endpoint (high ports, ranging typically from 49152 to 6553537) of the `ITaskSchedulerService` interface. While it is interesting to have another option, most Windows Firewall configurations will, by default in AD domain networks, block incoming requests to dynamic ports, rendering this method of connection useless.

Once the connection is established, the tool presents itself as a shell:

```
 $ python3 atexec-pro.py -i TSCH corporation.local/Administrator@192.168.122.71
[!] This will work ONLY on Windows >= Vista
Password:
[*] Connecting to DCE/RPC as corporation.local\Administrator
[*] Successfully bound.
[+] Type help for list of commands. 🚀
ATShell (Administrator@192.168.122.71)> help

Documented commands (use 'help -v' for verbose/'help <topic>' for details):

Run Command
```

```
===========
cmd_exec  ps_exec

Post Exploitation
=================
download  execute_assembly  upload
```

Remote command execution works as follows:



It works similarly for .NET assembly execution. For upload and download, the file's content is encoded and read from or placed into the task's description.

Overall, the following functionalities are available:

- Command Prompt command execution.
- PowerShell command execution.
- File upload.
- File download.
- .NET assembly execution.
- Interaction with `ITaskSchedulerService` via RPC over SMB or RPC over TCP/IP.

Due to the way files are transferred (via the `Description` field of the task), file upload, file download and .NET assembly execution only support files up to 1 MB in size.

Detection

What `atexec-pro.py` makes up for in functionality, it lacks in stealth. Every command launched through the Task Scheduler first goes through a PowerShell script, as defined in the task XML definition:

```
 <Exec>
  <Command>powershell.exe</Command>
  <Arguments>-NonInteractive -enc {ps_command}</Arguments>
</Exec>
```

where `{ps_command}` is replaced by `atexec-pro.py` with a base64-encoded PowerShell script. The default scripts provided for each command of the tool can easily be flagged as suspicious:

- Base 64 encoding and decoding.
- AES encryption and decryption.
- Interaction with the Task Scheduler service via the Task Scheduler Scripting Objects38.

As such, even on default Windows Event Log configurations where Script Block Logging is not fully enabled and only suspicious scripts are logged, Event ID 4104 events are generated in Microsoft-Windows-PowerShell\Operational for each execution of an **atexec-pro**'s command.

This results in the following Sigma rule:

```
 title: atexec-pro - Suspicious PowerShell script
id: 8da0570e-adc3-4d2d-8acf-07f8cde5db3a
status: experimental
description: Suspicious PowerShell script contents related to execution of atexec-pro remote execution tool
license: DRL-1.1
references:
    - https://www.synacktiv.com/en/publications/open-source-toolset-of-an-ivanti-csa-attacker
author: Synacktiv, Maxence Fossat [@cybiosity]
date: 2025-05-12
modified: 2025-05-12
tags:
    - attack.execution
    - attack.t1053
    - tlp.clear
    - pap.clear
logsource:
    product: windows
    category: ps_script
    definition: Script Block Logging must be enabled
detection:
    selection_base:
        EventID: 4104
        ScriptBlockText|contains|all:
```

```
            - '[System.Convert]::ToBase64String('
            - '[System.Convert]::FromBase64String('
            - 'New-Object System.Security.Cryptography.AesManaged'
            - '[System.Security.Cryptography.CipherMode]::CBC'
            - '.CreateEncryptor()'
            - '.CreateDecryptor()'
            - 'New-Object -ComObject Schedule.Service'
            - '.GetTask('
            - '.RegistrationInfo.Description'
            - '.RegisterTaskDefinition('
    selection_script_cmd:
        ScriptBlockText|contains: 'iex'
    selection_script_upload:
        ScriptBlockText|contains: 'Set-Content -Path '
    selection_script_download:
        ScriptBlockText|contains: 'Get-Content -Path '
    selection_script_net:
        ScriptBlockText|contains|all:
            - '[System.Reflection.Assembly]::Load('
            - 'New-Object System.IO.StreamWriter'
            - '.Invoke('
            - 'New-Object System.IO.StreamReader('
    condition: selection_base and 1 of selection_script*
falsepositives:
    - Legitimate scripts using these cmdlets
level: high
```

A similar rule can be created based on command line content if process creation is logged.

As for forensic artefacts left by the execution of the original `atexec.py` script, the following article explores them (and many more remote code execution tools ;).

Conclusion

Throughout this article, we discovered three tools actively used by threat actors to tunnel traffic to and from the internal network (**suo5**, **iox**) and to execute code remotely (**atexec-pro**). We analysed their core capabilities and used them to create YARA and Sigma detection rules, focusing on broad detection, harder defence evasion and low false positive rates.

All detection rules created for this article will be maintained in the following GitHub repository.

If your organization needs assistance in removing doubt or responding to a security incident, please feel free to contact Synacktiv's CSIRT.
