

Proxying to Compromise: SonicWall Secure Mobile Access 0-day Exploitation

By Steven Adair

Published: 2026-07-17 · Archived: 2026-07-28 02:01:03 UTC

In early July 2026, Volexity was engaged to perform an incident response investigation where it discovered a threat actor had successfully compromised SonicWall Secure Mobile Access (SMA) VPN appliances through a chain of multiple zero-day exploits in the devices. The initial compromise was discovered after suspect authentication and lateral movement attempts were observed from the SonicWall SMA appliances. Following [public disclosure by SonicWall on July 14, 2026](#), Volexity is now able to share details on the exploits used, when they were used, and what the threat actor did with their access.

The exploited vulnerabilities were found to affect SonicWall SMA 1000 series device models 6210, 7210, 8200v. The following vulnerabilities were patched through a hotfix earlier this week and included in versions 12.4.3-03453 and 12.5.0-02835 of the appliance:

- [CVE-2026-15409 \(SSRF\)](#)
- [CVE-2026-15410 \(Command Injection\)](#)

At the beginning of the investigation, Volexity initially worked with available security telemetry and logs exported from two impacted appliances. However, the organization was able to provide Volexity with administrative SSH access to the VPN appliances a short time later. Using SSH access to the devices, Volexity collected system memory (RAM) with [Volexity Surge Collect Pro](#), and gathered select files and full disk images. During the initial triage, Volexity identified disk artifacts that immediately confirmed both devices were compromised. The memory sample was then further analyzed with [Volexity Volcano](#).

Volexity's analysis of logs, disk images, and memory led to the discovery of a threat actor Volexity tracks as **UTA0533**. This threat actor was observed using multiple zero-day exploits, malware designed specifically for SonicWall SMA VPN appliances, as well as other attacker tradecraft. Volexity notes that **June 22, 2026**, was the earliest sign of compromise observed in the investigation.

Details of Volexity's analysis of UTA0533's operations, including the vulnerability workflow used to achieve remote code execution and the malware discovered, are detailed in the sections below.

Log Analysis and Key IOCs

Volexity initially worked with logs exported from the SonicWall SMA VPN appliances. These logs provided the full basis for investigating the intrusion to find most of the exploits used. However, using logs alone without a confirmed system-level compromise or accounting of on-system activity can make it difficult to connect all of the dots or prove what happened. Fortunately, Volexity was able to gather all the evidence needed and pieced together the entire workflow (and more) as described later in this post. There were various log files that provided clear

signs of suspicious activity and compromise. Examples of the key log sources and what indicators of compromise (IOCs) look like are noted below.

Note: Trivial details in the log files have been modified by Volexity for anonymization purposes.

/var/log/aventail/extraweb_access.log

The log captures the external web interaction with the SonicWall SMA VPN appliances. This includes the initial authentication bypass and SSRF via /wsproxy. Further, interactions with the observed backdoor/webshells was also recorded in this log. Checking logs for entries similar to those below are strong indicators of targeting and potential compromise.

Exploitation

Note the access to /wsproxy, with a bmID value starting with -3389, a set serviceType, and the host set to 0.0.0.0 (or similar value aimed at internal services). The use of ports 1050 (Couch DB) and 8188 (Control Service) were seen throughout the investigation. Also, note the status codes of 101, indicating a successful protocol upgrade (to WebSockets).

```
x.x.x.x - - [28/Jun/2026:23:19:17 -0000] "GET /wsproxy?  
bmID=-3389f61b21f4&serviceType=SSH&host=0.0.0.0&port=1050 HTTP/1.1" 101 - "-" -  
x.x.x.x - - [28/Jun/2026:23:19:48 -0000] "GET /wsproxy?  
bmID=-338944f133ea&serviceType=SSH&host=0.0.0.0&port=1050 HTTP/1.1" 101 - "-" -  
x.x.x.x - - [28/Jun/2026:23:20:41 -0000] "GET /wsproxy?  
bmID=-3389166d34d2&serviceType=SSH&host=0.0.0.0&port=1050 HTTP/1.1" 101 - "-" -  
x.x.x.x - - [28/Jun/2026:23:20:53 -0000] "GET /wsproxy?  
bmID=-3389fdv2401b&serviceType=SSH&host=0.0.0.0&port=8188 HTTP/1.1" 101 - "-" -  
x.x.x.x - - [28/Jun/2026:23:20:56 -0000] "GET /wsproxy?  
bmID=-3389ba1fdb2a&serviceType=SSH&host=0.0.0.0&port=8188 HTTP/1.1" 101 - "-" -
```

Webshell Access

UTA0533 set up a method by which the URLs /__api__/login and /__api__/logout could be used to access malicious endpoints the threat actor installed on the system (detailed later). These paths are arbitrary and could be anything, as they were set up to re-write to specific threat actor destinations within nginx. However, presence of successful access to log entries similar to what appears below (or other non-expected endpoints) would be a strong indicator of compromise.

```
x.x.x.x - - [28/Jun/2026:17:21:16 -0000] "POST /__api__/logout HTTP/1.1" 200 211602 "-" -  
x.x.x.x - - [28/Jun/2026:17:21:18 -0000] "POST /__api__/logout HTTP/1.1" 200 291872 "-" -  
x.x.x.x - - [28/Jun/2026:17:21:20 -0000] "POST /__api__/logout HTTP/1.1" 200 263542 "-" -  
  
x.x.x.x - - [29/Jun/2026:00:35:49 -0000] "POST /__api__/login HTTP/1.1" 200 - "-" -  
x.x.x.x - - [29/Jun/2026:00:35:49 -0000] "POST /__api__/login HTTP/1.1" 200 - "-" -  
x.x.x.x - - [29/Jun/2026:00:35:50 -0000] "POST /__api__/login HTTP/1.1" 200 - "-" -
```

/var/log/aventail/access_servers.log

This log contains information related to the /wsproxy and WebSockets activity, as well. It contains details related to the attacking IP address and evidence of exploitation. A sample workflow from an attack is shown below:

```
[28/Jun/2026:23:50:30.857963 -0000] sw-sma-device 004841 ew 1000019d Internl Misc
::WEBSOCK::Websocket protocol use = BINARY Sec-WebSocket-Key = qcaHQ<snip>
[28/Jun/2026:23:50:30.858010 -0000] sw-sma-device 004841 ew 1000019d Info Misc
::WEBSOCK::Socket connected to backend success host = 0.0.0.0 and port =1050 Sec-WebSocket-
Key = qcaHQ<snip>
[28/Jun/2026:23:50:30.858011 -0000] sw-sma-device 004841 ew 1000019d Internl Misc
::WEBSOCK::Connection Request from Client IP = x.x.x.x Client Port = 29557 Sec-WebSocket-Key
= qcaHQ<snip>
[28/Jun/2026:23:50:30.858029 -0000] sw-sma-device 004841 ew 1000019d Internl Misc
::WEBSOCK::SSO Not Enable Sec-WebSocket-Key = qcaHQ<snip>
[28/Jun/2026:23:50:32.264308 -0000] sw-sma-device 004841 ew 1000019d Internl Misc
::WEBSOCK::Graceful shutdownwn by client Sec-WebSocket-Key = qcaHQ<snip>
[28/Jun/2026:23:50:32.264338 -0000] sw-sma-device 004841 ew 1000019d Internl Misc
::WEBSOCK::wsproxy_on_disconnect called Sec-WebSocket-Key = qcaHQ<snip>
```

/var/log/aventail/ctrl-service.log

This ctrl-service.log file contained key information related to command execution and privilege escalation (run as root). This log would capture the files that UTA0533 executed through CVE-2026-15410. Example log lines are shown below:

```
2026-06-28 22:34:34,527 - INFO - running hotfix removal for:../../../../tmp/1234.sh
2026-06-28 22:52:19,293 - ERROR - Command '['/usr/local/bin/remove_hotfix',
'../../../../tmp/1234.sh']' exited with status '1'
```

SonicWall SMA System Analysis

Volexity analyzed multiple SonicWALL SMA VPN appliances. Two were found to be compromised and were the focus of the investigation.

Appliance 1

Volexity's review of the first appliance uncovered the following key findings:

- On June 22, 2026, a file was written to **/usr/bin/xzfind**. The file was a setuid binary named **rootrun** that allowed an unprivileged user to execute arbitrary commands as root.
- A few minutes after xzfind was written, another file was written to **/usr/lib/python3.11/site-packages/deploy_new.py**. This malware script, which Volexity calls **KNUCKLEBALL**, contained two embedded JAR archives that were injected into a legitimate SonicWall process.

- The first embedded JAR file was the open-source HTTP proxy-forwarding tool **Suo5**. The second was a Behinder-like webshell that Volexity calls **ORANGETAIL**.
- Once loaded, the JAR files enable the threat actor to interact with them via the following internet-accessible URI paths:
 - **/workplace/error.jsp**
 - **/workplace/dialogs/errorDialog.jsp**
- Access to the loaded implant required the following specific user-agent string:
 - `Mozilla/6.0 (Windows NT 11.0; Win64; x64) AppleWebKit/1537.136 (KHTML, like Gecko) Chrome/149.0.0.1 Safari/1537.136`
- Shortly after KNUCKLEBALL was written, persistence was established through the addition of the following line to the legitimate `/etc/init.d/workplace` startup script:
 - `python3 /usr/lib/python3.11/site-packages/deploy_new.py`
- Volexity identified additional files associated with exploitation and privilege escalation in `/tmp`. These artifacts were dated June 30, 2026, after the initial compromise.
- One file, `/tmp/hypdate.b64`, contained a Base64-encoded local privilege-escalation exploit that was later assigned [CVE-2026-15410](#).
- The files in `/tmp` were owned by the unprivileged account used by the appliance's internal database service. This indicated the threat actor could write and likely execute files through that service context.
- The NGINX Unit configuration file at `/var/lib/unit/conf.json` had been modified to add two routes leading to Suo5 and ORANGETAIL. The following inbound paths were rewritten and proxied to `http://127.0.0.1:8085`:
 - `/__api__/login` → `/workplace/error.jsp`
 - `/__api__/logout` → `/workplace/dialogs/errorDialog.jsp`

Appliance 2

The second appliance contained fewer artifacts but still revealed important evidence. The appliance had been rebooted on July 2, 2026. Volexity assesses that the reboot likely removed memory-resident backdoors and potentially other volatile artifacts.

Key findings from this system include the following:

- The same modifications to `/var/lib/unit/conf.json` identified on the first appliance were present. However, testing showed that the routes did not return valid responses.
- Memory and application testing indicated that Suo5 and ORANGETAIL were not actively loaded on this appliance at the time of collection.
- There were few persistent files associated with exploitation or privilege escalation. One exception was a zero-byte file named `txt`, which was owned by root and likely an artifact of post-exploitation activity.
- Multiple files created on July 2, 2026, were identified in `/var/tmp`. These files were associated with using `tcpdump` to capture unencrypted LDAP traffic. Notably, one file called **lib.sh** was a script to launch

tcpdump and save captured traffic in the same directory. Its functionality was equivalent to the following generalized command:

- ```
nohup tcpdump -i any '<traffic destined for internal directory servers on TCP port 389>' -w /var/tmp/<file_name> -C 100 -W 10 &
```

## Exploit Details

Volexity further analyzed logs, system memory, and other information to investigate and ultimately recreate a working exploit chain similar to the one used by UTA0533. Additional details on the exploits are detailed below.

### CVE-2026-15409

Volexity identified a pre-authentication /wsproxy bypass on SonicWall SMA appliances. The bypass allows an unauthenticated external request to establish a WebSocket tunnel to localhost-only services on the appliance. It works when the request uses a user-agent of **SMA Connect Agent** and a bmID value that begins with **-3389**, such as the following example (just key components dictated):

```
GET /wsproxy?bmID=-3389<suffix>&serviceType=SSH&host=0.0.0.0&port=<target-port>
User-Agent: SMA Connect Agent
```

Requests matching the example above will return the following:

```
HTTP/1.1 101 Switching Protocols
Sec-WebSocket-Protocol: binary
```

No valid SMA session cookie was required during this process.

Using the /wsproxy bypass, Volexity verified external access to localhost-only services, including the following:

```
127.0.0.1:1051 Erlang Port Mapper Daemon (EPMD)
127.0.0.1:1050 CouchDB Erlang distribution
127.0.0.1:8188 SMA control service / XML-RPC over TLS
```

A read-only [EPMD](#) names request returned the name couchdb at port 1050.

External access to the SMA control service allows the attacker to access methods in the sysCtrl endpoint, which were vulnerable to command injection, privilege escalation, and code execution. These are described in subsequent sections.

### Control-service Access

Volexity identified a separate issue where an attacker may be able to bypass the authentication to the SMA control service. This is a local management/control service used by the SonicWall SMA appliance to perform system-level operations on behalf of higher-level management components. The control service is accessible on **127.0.0.1:8188** and is reachable through the /wsproxy tunnel. It responds with the following:

```
HTTP/1.0 401 Unauthorized
Server: BaseHTTP/0.6 Python/3.11.12
WWW-Authenticate: Basic realm="XMLRPC"
```

The Basic authentication password is derived from the appliance-local hardware identifier:

```
/sys/class/dmi/id/product_uuid
```

An attacker who knows the value of this UUID can determine the password required to authenticate. Although the control service is intended to be localhost-only, the /wsproxy bypass exposes 127.0.0.1:8188 externally, thus this authentication mechanism becomes part of the external attack surface. The UUIDs value has its dashes (“-”) removed and is Base64-encoded to create a valid password.

Volexity found the product\_uuid file is world readable, so unprivileged users can obtain its value. On multiple appliances to which Volexity had access, the UUID was set to a common value seen across many different systems belonging to different customers. It is a default UUID that comes with various hardware providers. Volexity only observed this UUID in cases where the owner’s device was physical, meaning virtual appliances were unaffected. Using this known UUID, an attacker would likely succeed in breaching the appliances without any other exploitation.

It should be noted that this authentication bypass does not appear to have been used in the observed incident. Instead, the attacker abused a different vulnerability to read the product\_uuid file.

## CouchDB Exploitation

Based on forensic investigation, Volexity identified that the attacker’s route to compromise involved the use of CouchDB. Key evidence for this was a file named **1234.sh** located in the /tmp directory:

```
-rwx--x--x 1 couchdb daemon 123 Jun 29 06:50 1234.sh
```

CouchDB comes installed as part of the SMA appliance and is accessible via localhost only. Shipped CouchDB instances come with hardcoded credentials of **admin:admin**.

The exact operation that was used by UTA0553 for the CouchDB exploitation remains unknown to Volexity, but known facts are as follows:

- The CouchDB user was used to write and run a script that read /sys/class/dmi/id/product\_uuid.
- Volexity can timestamp match access to port 1050 to the write of the script in question.
- Knowing the value in product\_uuid would allow an attacker to access the vulnerable sysCtrl.
- All of the above occurred prior to any exploitation of endpoints under sysCtrl.

A proof-of-concept (PoC) has been [released by Rapid7](#) relating to this; however, it requires a hardcoded cookie value not present on systems analyzed by Volexity. The value from the POC appears to be derived from the static cookie set on the virtual SonicWall SMA appliance that is available for download.

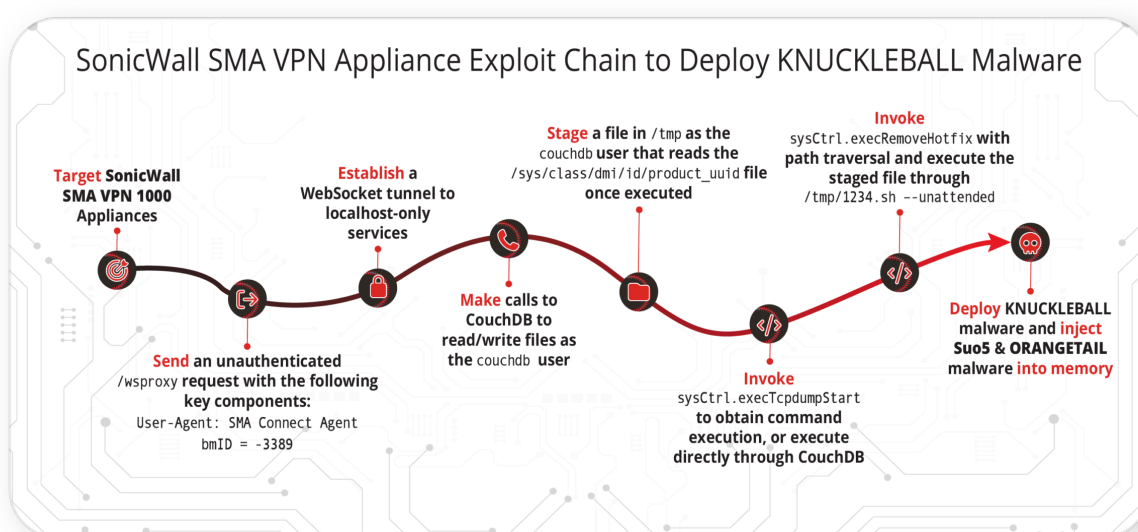
## CVE-2026-154010 – sysCtrl.execRemoveHotfix – Path Traversal and Execution

Volatility verified that sysCtrl.execRemoveHotfix ultimately invokes /usr/local/bin/remove\_hotfix. The helper builds a rollback path using caller-controlled input:

```
__rollback="/var/lib/aventail/avp/rollback/${__hotfix}"
chmod +x $__rollback
exec $__rollback --unattended
```

Because the input is not sufficiently constrained to the rollback directory, path traversal can resolve outside the intended path. For example: ../../../../tmp/1234.sh resolves to /tmp/1234.sh and causes the helper to execute /tmp/1234.sh --unattended.

## Exploitation Chain



The following summarizes the high-level workflow for exploitation based on Volatility's understanding of the forensic evidence is as follows:

1. Send an unauthenticated /wsproxy request with the following key components (User-Agent: SMA Connect Agent and URI parameter starting with bmID=-3389).
2. Establish a WebSocket tunnel to localhost-only services.
3. Make calls to CouchDB to read, write files as the couchdb user.
4. Stage a file in /tmp as the couchdb user that will read the /sys/class/dmi/id/product\_uuid file once executed.
5. Invoke execRemoveHotfix with path traversal and execute the staged file through /tmp/1234.sh --unattended to obtain command execution as root.

## Detecting Exploitation and Compromise

Volatility's findings clearly showed compromise and the timeline over which it occurred. The code for the privilege escalation exploit (CVE-2026-15410) gave exact blueprints to part of the attack. Other pieces gave a likely indication of how activity occurred (e.g., couchdb file owner). Volatility also found logs that served as excellent IOCs for the privilege escalation exploit and was able to fully recreate it.

The files and log entries detailed at the start of this post can be used as solid indicators of compromise. Further, Volexity has the following additional recommendations to search for signs of compromise outside of the log lines previously shared:

- Look for unexpected connections from SonicWall SMA appliances. This would include monitoring for both lateral movement and external connections.
- Examine SonicWall SMA appliances for unexpected files in /var/tmp and /tmp. Further, look for any unexpected file modifications or new files on the system.
- Examine the nginx configuration file /var/lib/unit/conf.json for any unexpected routes, such as those going to **http://127.0.0.1:8085**.
- Check the system for any unexpected setuid binaries, outside of those that are expected on the system, by running the command **find / -perm -4000**. The following binaries may show up legitimately:
  - /usr/lib/mysql/plugin/auth\_pam\_tool\_dir/auth\_pam\_tool
  - /usr/libexec/ssh-keysign
  - /usr/local/workplace/conf
  - /usr/bin/umount
  - /usr/bin/su
  - /usr/bin/mount
  - /usr/bin/newgrp
  - /usr/bin/at
  - /usr/bin/ping
  - /usr/bin/sudo
  - /usr/bin/ping6

Volexity has also released **yara** signatures to detect the malware families described in this blog. These signatures can be obtained from Volexity's GitHub repository [here](#).

## Discovered Malware Families

### ROOTRUN – xzfind

Volexity's disk analysis of the system revealed that the threat actor had dropped a file named xzfind. Volexity identified this binary as a privilege escalation utility that internally refers to itself as "rootrun". The table below details the analyzed file:

|                  |                                          |
|------------------|------------------------------------------|
| <b>Name(s)</b>   | xzfind                                   |
| <b>Size</b>      | 13.1KB (13464 Bytes)                     |
| <b>File Type</b> | ELF Executable                           |
| <b>MD5</b>       | 5cb00bbfe818ee3e85fb99ab1db1af7c         |
| <b>SHA1</b>      | 04d4a9fbb32e967200eb98be014ca914a03bfa6b |

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <b>SHA256</b> | 81a9af3846bad3a1107164ff7cf0a08e020b31a3b32fd17866e17d4c1565f7f2 |
|---------------|------------------------------------------------------------------|

This utility uses [setuid\(\)](#) function to elevate itself to root, then executes a provided command using the bash shell. The utility contains the following usage string showing its command-line syntax:

```
Usage: rootrun rootrun <command>
```

## KNUCKLEBALL – deploy\_new.py

The deploy\_new.py script that was dropped to disk and added to the workplace init.d script was designed to inject two embedded Java archive (JAR) files into the **workplace.startup.CommandStartup** process running on the SMA appliance. The table below details the script file:

|                  |                                                                  |
|------------------|------------------------------------------------------------------|
| <b>Name(s)</b>   | deploy_new.py                                                    |
| <b>Size</b>      | 79.6KB (81476 Bytes)                                             |
| <b>File Type</b> | Python                                                           |
| <b>MD5</b>       | b6df166291f80ee89032d769c99714f3                                 |
| <b>SHA1</b>      | b4ee1f50fbb49f0ff5fde3d026343bc23ee08d51                         |
| <b>SHA256</b>    | 8c470301dcb7278f73e622f1950073567b34011c64b60cdfbb0f89803923a5a3 |

The two embedded payloads are Base64-encoded and temporarily written to disk at the following paths during the injection process:

- /tmp/agent\_wp8.jar
- /tmp/agent\_wp9.jar

The process ID of the workplace.startup.CommandStartup process is located by enumerating the command-line pseudo-file for each PID listed in the [/proc filesystem](#). Once the process is identified, the script uses the [Java Attach API](#) (/tmp/.attach\_pid<PID> and /tmp/.java\_pid<PID>) to inject the dropped agent files into the JVM as [instrumentation agents](#) using the effective command line below:

```
load instrument false <path>
```

The internal log files used by both JAR files, /tmp/agent\_wp8.log and /tmp/agent\_wp9.log, are cleared prior to injection. The paths are linked to /dev/null to prevent them from appearing on the infected device. After injection, both JAR files are deleted, the overridden classes are tested, and the script modifies the nginx configuration via the local Unix socket interface, most notably adding the following two routes:

```
[
{
 "match": {
 "uri": "/__api__/login"
```

```

 },
 "action": {
 "rewrite": "/workplace/error.jsp",
 "proxy": "http://127.0.0.1:8085"
 }
 },
 {
 "match": {
 "uri": "/__api__/logout"
 },
 "action": {
 "rewrite": "/workplace/dialogs/errorDialog.jsp",
 "proxy": "http://127.0.0.1:8085"
 }
 }
],]

```

These nginx routes allow requests to a custom Java webshell, which Volexity calls ORANGETAIL, to be reached via external web requests.

## Java Agents

The instrumentation agent wrapper of each JAR file locates a pre-existing, hardcoded target class and overrides it with an embedded payload class. The class manipulation is performed using another file, [javassist.jar](#), that is located by searching the following directories:

- /usr/local/java/lib
- /opt/java/lib
- /usr/share/java

Each payload class has been modified to use internal gating logic to prevent passive detection. Both gate access behind the following user-agent string:

```

Mozilla/6.0 (Windows NT 11.0; Win64; x64) AppleWebKit/1537.136 (KHTML, like Gecko)
Chrome/149.0.0.1 Safari/1537.136

```

This string resembles a legitimate user agent but uses version numbers that are inconsistent with any real-world user agent.

## suo5 – agent\_wp8.jar

The agent\_wp8.jar file is hardcoded to target the com/aventail/jsp/workplace/error\_jsp class.

The table below details the decoded JAR file:

| Name(s) | agent_wp8.jar |
|---------|---------------|
|---------|---------------|

|                  |                                                                  |
|------------------|------------------------------------------------------------------|
| <b>Size</b>      | 33.7KB (34520 Bytes)                                             |
| <b>File Type</b> | Java archive data (JAR)                                          |
| <b>MD5</b>       | 54d21399b8b52b48a0fef68450593e45                                 |
| <b>SHA1</b>      | c2b0ae0a1f42a139abe4dd612676066ec1426394                         |
| <b>SHA256</b>    | 1e1e68bbb899450a57274a8b12082ed4e2040a2aae77014f20431689d2b4edee |

The injected payload is a version of [suo5.jsp](#), an open-source HTTP forwarding proxy tool that has been modified to include the previously mentioned gating logic based on the user agent of the request. This logic is shown in the image below.

```
public void process(ServletRequest var1, ServletResponse var2) {
 HttpServletRequest var3 = (HttpServletRequest)var1;
 HttpServletResponse var4 = (HttpServletResponse)var2;
 String var5 = var3.getHeader("User-Agent");
 if (!"Mozilla/6.0 (Windows NT 11.0; Win64; x64) AppleWebKit/1537.136 (KHTML, like Gecko) Chrome/149.0.0.1 Safari/1537.136".equals(var5)) {
 try {
 var4.setStatus(404);
 } catch (Throwable var33) {}
 }
} else {
 String var6 = null;
 byte[] var7 = new byte[0];
 try {
 ServletInputStream var8 = var3.getInputStream();
 HashMap var9 = this.unmarshalBase64(var8);
 byte[] var10 = (byte[])var9.get("w");
 byte[] var11 = (byte[])var9.get("u");
 byte[] var12 = (byte[])var9.get("id");
 byte[] var13 = (byte[])var9.get("sid");
 if (var11 == null || var11.length != 1 || var12 == null || var12.length == 0 || var10 == null || var10.length == 0) {
 return;
 }
 if (var13 != null && var13.length > 0) {
 var6 = new String(var13);
 }
 }
}
```

## ORANGETAIL – agent\_wp9.jar

The agent\_wp9.jar file targets the com/aventail/jsp/workplace/dialogs/errorDialog\_jsp class. The table below details the decoded JAR file:

|                  |                                                                  |
|------------------|------------------------------------------------------------------|
| <b>Name(s)</b>   | agent_wp9.jar                                                    |
| <b>Size</b>      | 21.3KB (21800 Bytes)                                             |
| <b>File Type</b> | Java archive data (JAR)                                          |
| <b>MD5</b>       | 5f3a55201c511c9ff9be4c16c41028a2                                 |
| <b>SHA1</b>      | 5e5b716f2385c818ec61198be1a2a07a4560eac5                         |
| <b>SHA256</b>    | ea9154e374e4f77bc2cf54282e23543573980342a85bc888cb23f20b8bbba081 |

The payload is a custom Java webshell that operates similarly to the open-source webshell [Behinder](#). The webshell accepts threat actor-supplied data through the find parameter of an HTTP POST request. The data is decrypted and dynamically loaded into a Java class for later use within the HTTP session. The payload is Base64-encoded and AES-128-ECB-encrypted using the hardcoded key 615[snipped]e4a4. Once decrypted and loaded, the Java class is associated with the hardcoded session key IG3L[snipped]y8ro1. Subsequent requests can instantiate and execute this class by referencing this session key. Responses from the webshell are encrypted and encoded identically to the initial request.

The table below summarizes the differences between Behinder and ORANGETAIL:

| Behinder 3.x                                 | ORANGETAIL                                                                                |
|----------------------------------------------|-------------------------------------------------------------------------------------------|
| Responds with raw AES-encrypted bytes        | Response uses JSON schema: {"message": "ok", "content": "<b64-aes>", "statuscode": "200"} |
| java.util.Base64 or sun.misc.BASE64Decoder   | Custom hand-rolled implementation (no standard library)                                   |
| Direct import of javax.crypto.Cipher         | Fully reflective: Class.forName → getMethod → invoke                                      |
| AES key derived from authentication password | Hardcoded AES key                                                                         |
| Page renders normally for all visitors       | Page returns 404 if user agent does not validate                                          |
| Configurable (commonly unnamed POST body)    | Parameter named find                                                                      |
| Plain string literals                        | All sensitive strings are built char-by-char via String.valueOf()                         |

## Subsequent Intrusion Activity

As mentioned in the [Appliance 2](#) section, following successful compromise of the device, UTA0533 used their access to sniff local traffic using tcpdump; inspecting unencrypted LDAP traffic to extract usernames and passwords. Volexity observed UTA0533 use both Suo5 and ORANGETAIL, attempting to pivot further into the customer network. This included authentication attempts from the SMA appliance to various systems on the network.

During the lateral movement phase of the attack, Volexity observed several hostnames belonging to the threat actor that were unintentionally leaked, including the following:

- DESKTOP-5P0TSCP
- DESKTOP-IC3C80F
- DESKTOP-KRLUI3J
- KALI

UTA0533's exploitation and web shell access came from over 200 different IP addresses over the intrusion. Several of the IP addresses were found to belong to ExpressVPN and MullvadVPN, based on lookups using [Spur](#). For the remainder of the IP addresses, Volexity did not find any discernable commonality between the source IP addresses observed. A few of the identified non-VPN IP addresses used in the attacks are listed below:

- 108.205.8.173
- 147.45.51.19
- 150.241.210.53
- 202.8.105.201
- 217.77.15.99
- 42.200.172.14
- 81.19.140.217
- 89.117.20.1

## Conclusion

UTA0533 combined multiple zero-day vulnerabilities to compromise SonicWall SMA VPN appliances and obtain root-level access. With root access, the threat actor could access stored or cached credentials, capture network traffic, and potentially intercept credentials processed by the appliances. Although UTA0533 demonstrated significant capability in compromising the SonicWall appliances, available evidence suggests the threat actor was less successful moving laterally or gaining access to other systems.

Volexity has released YARA signatures to detect the malware families described in this blog. These signatures are available [here](#).

## Acknowledgements

Volexity would like to thank its customer for working closely together and permitting public sharing of the investigation details.

## Investigative Assistance

If any organization or individual believes they may have been targeted by a similar attack, please feel free to reach out to Volexity via our [contact form](#). We would be glad to assess any potential targeting and assist in determining if such an attack may have succeeded.

---

*If you are interested in learning more about Volexity's services, including [Threat Intelligence](#), [Network Security Monitoring](#) and [Incident Response](#), or our leading memory forensics solutions, [Volexity Surge Collect Pro](#) for memory acquisition and [Volexity Volcano](#) for memory analysis, please do not hesitate to [contact us](#).*

---