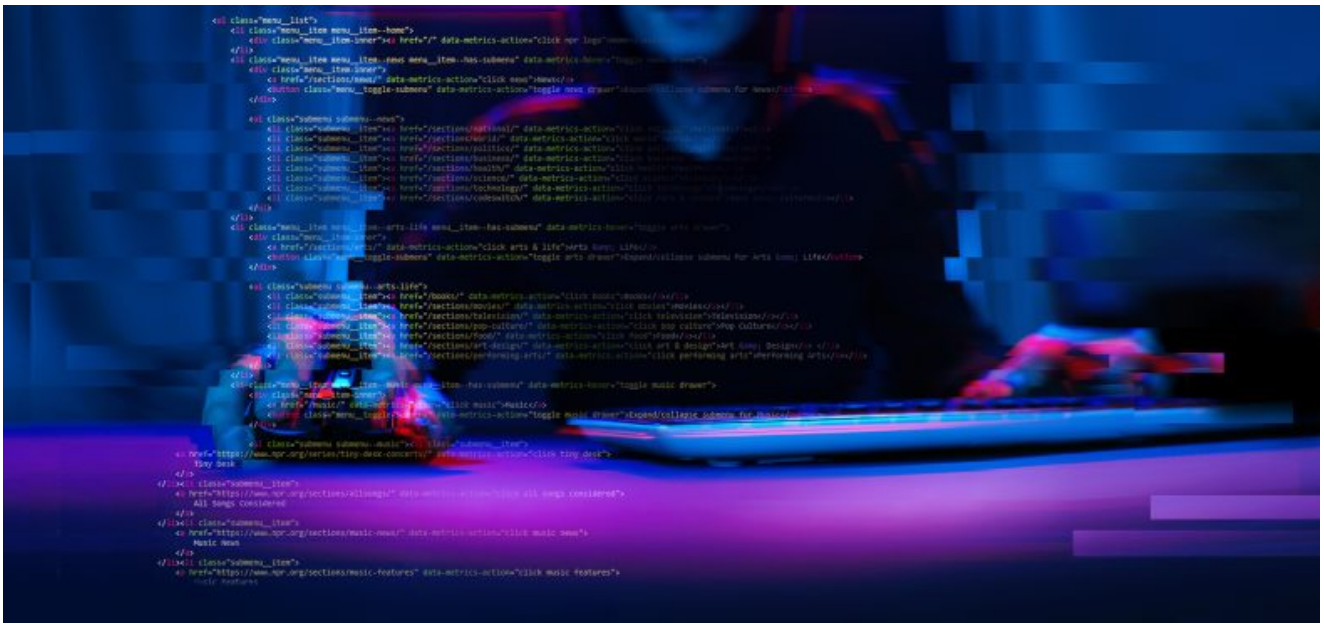
Auto-Color: An Emerging and Evasive Linux Backdoor

unit42.paloaltonetworks.com/new-linux-backdoor-auto-color/

February 24, 2025



Executive Summary

Between early November and December 2024, Palo Alto Networks researchers discovered new Linux malware called Auto-color. We chose this name based on the file name the initial payload renames itself after installation.

The malware employs several methods to avoid detection, such as:

- Using benign-looking file names for operating
- Hiding remote command and control (C2) connections using an advanced technique similar to the one used by the Symbiote malware family
- Deploying proprietary encryption algorithms to hide communication and configuration information

Once installed, Auto-color allows threat actors full remote access to compromised machines, making it very difficult to remove without specialized software.

This article will cover aspects of this new Linux malware, including installation, obfuscation and evasion features. We will also discuss its capabilities and indicators of compromise (IoCs), to help others identify this threat on their systems too.

Palo Alto Networks customers are better protected from the threats discussed in this article through the following products or services: Advanced WildFire machine-learning models, as well as Advanced URL Filtering and Advanced DNS Security, and Cortex XDR and XSIAM.

If you think you might have been compromised or have an urgent matter, contact the Unit 42 Incident Response team.

Related Unit 42 Topics **Backdoor**, **Linux**

Telemetry and Source Information

We received the first sample for this malware family on Nov. 5, 2024, and as of this writing, the most recent sample on Dec. 5, 2024. Our metadata analysis revealed that the malware family has primarily been used to target universities and government offices in North America and Asia.

Each time the malware deploys on a different target, it uses a different file name. The file name is usually a simple, ordinary word such as door or egg. We will discuss this feature further in Malware Startup and Installation.

Although the file sizes are always the same, the hashes are different. This is because the malware author statically compiled the encrypted C2 configuration payload into each malware sample, as we discuss in Target C2 Payload Information.

We do not currently know how the initial malware executable reaches its targets, but the file is intended to run explicitly by the victim on their Linux machine. Figure 1 shows the general flow after the malware starts execution.

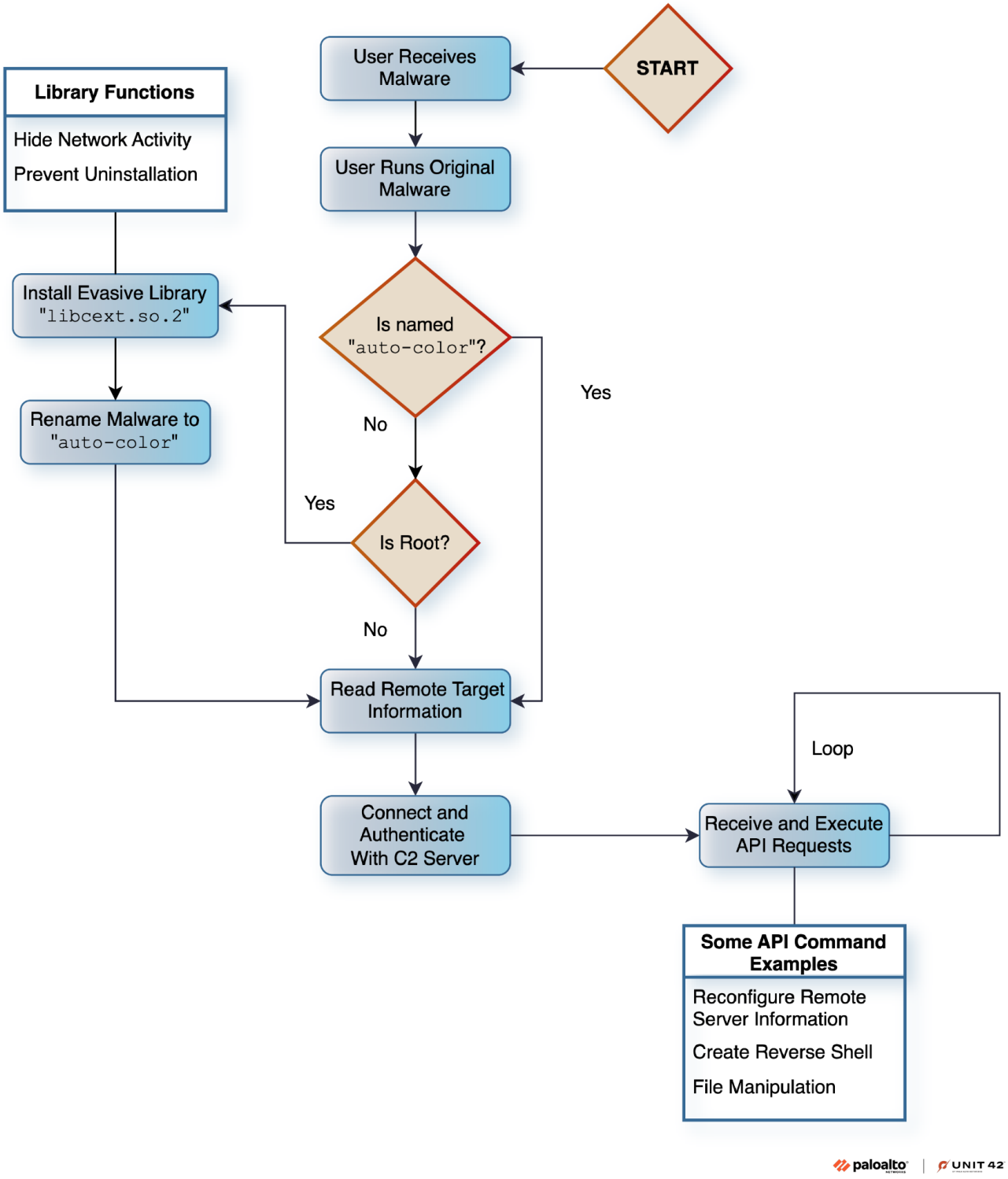


Figure 1. Flow diagram of Auto-color.

Malware Startup and Installation

Once the malware initially runs on the victim machine, it will check whether the executable file name running is Auto-color. Initially, the original executables will all have different file names such as door or egg, and they will perform different logic if the name differs from

Auto-color. If its executable file name is not Auto-color, the malware will run its installation phase for an evasive library implant located within the executable itself.

If the current user lacks root privileges, the malware will not proceed with the installation of the evasive library implant on the system. It will proceed to do as much as possible in its later phases without this library.

If the current user has root privileges, the malware then installs a malicious library implant called libcext.so.2. This is to mimic the legitimate C utility library libcext.so.0 to evade detection.

The malware locates the base library directory path using the dladdr() function with a symbol used from the C standard library (libc). In this case, it used strerr(). If the symbol does not exist on the system, the malware will use the default base library path /lib instead.

After locating the library path, the executable copies and renames itself to /var/log/cross/auto-color. It then installs the library implant from memory to the base library path.

Finally, the malware writes the malicious library file name into /etc/ld.preload, which is a standard file on Linux systems. The OS' loader uses this file when loading executables on a Linux system. This means that all libraries referenced in this file will be loaded into the executable first by default, even if the loaded executable doesn't need it.

Since libraries in ld.preload are loaded first, a malicious library can override core libraries. This is accomplished by overwriting functions or other symbols (mainly libc functions), effectively intercepting and modifying behavior. This is also known as "hooking" any executable that tries to call libc functions.

Figure 2 below shows what happens whether or not the user has root privileges. The malware deletes its original executable in both cases. However, with root privileges, it preserves the Auto-color binary at /var/log/cross/auto-color.

```
rax_6.b = geteuid() == 0  // 1 if suid binary

if (rax_6.b == 0)
    // execute if NOT an suid binary for root
    // #green mode... (decrypted string)
    string_decrypt_and_cpy(dst: &buf, encrypted_string: &green_mode_encrypted)
    puts(str: getString(&buf))
    deallocate_resources(retval: &buf)
else
    // this execute if the binary is an suid binary
    int32_t errnum = install_lib()

    if (errnum == 0)
        // #install ok (decrypted string)
        string_decrypt_and_cpy(dst: &buf, encrypted_string: &_#install_ok)
        puts(str: getString(&buf))
        deallocate_resources(retval: &buf)
    else
        char* rax_8 = strerror(errnum)
        string_decrypt_and_cpy(dst: &buf, encrypted_string: &_#install_error)
        printf(format: "%s %d: %s\n", getString(&buf), zx.q(errnum), rax_8)
        deallocate_resources(retval: &buf)

int32_t var_158_1 = delete_exec()  // wipes itself out
```

Figure 2. Initial installation of Auto-color.

Malicious Library Implant Analysis

When the malicious library implant libcext.so.2 is installed, the actual library content is located within the original executable's memory, specifically the .rodata section.

This library has two main goals, for evasion and persistence:

- Hiding network activity between the malware and the remote target configured inside a global payload
- Preventing uninstallation by protecting /etc/ld.preload against modification or removal

Hiding Network Activity

On traditional Linux systems, the kernel holds a special file system called the proc file system, which contains information about the system as well as each running process. We will focus on one part of this file system, /proc/net/tcp, which contains information on all active network connections including source/destination IP addresses and port numbers.

As mentioned in the previous section, this library will hook functions used in libc for its own special purposes. In this case the malicious library is mainly hooking the open() family of functions.

For the most part, this hook will be passive in that it will just redirect the libc implementation of the function. However, when /proc/net/tcp is specifically passed in the function as a file, the malware's behavior changes.

When /proc/net/tcp is passed into the malicious library's open() function, it parses the file contents. The library checks each line to see whether certain local ports or remote IP addresses exist in a specific shared memory data structure. If so, the library will not write the specific entry containing the remote IP address or local port to a special file with the file path /tmp/cross/<user_id>/tcp. Otherwise, the line will be copied over as normal.

Finally, the malicious library's open() function returns a file descriptor for the modified file, concealing the manipulation from the victim.

Figure 3 shows what the /proc/net/tcp looks like before alteration.

Figure 3. Original contents of /proc/net/tcp.

Figure 4 shows the final result returned to the victim. The malware author did not format the output correctly, so the row numbers highlighted in red in Figure 3 and Figure 4 do not match.

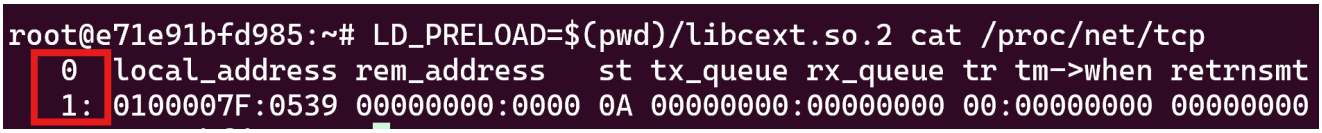


Figure 4. Modified contents of /proc/net/tcp from the malicious library.

The Symbiote malware family employed a similar, albeit simpler, technique to hide network connections. The Symbiote malware focused on concealment rather than manipulating or attempting to read socket information.

Target C2 Payload Information

Before the core part of the malware executes, it must decrypt the global target payload to find out which remote attacker servers it must connect to. It can obtain this global payload in two ways.

The first method is to read a specific file, /tmp/cross/config-err-XXXXXXXX or /var/log/cross/config-err-XXXXXXXX. The malware uses the first path if the user is non-root and the second path will exist if the user is root. The XXXXXXX part of the filenames are in hex and are generated dynamically.

These “config” files exist all over the malware and many of their purposes are different. However, the main config file manipulates the global payload information.

The threat actors can create the main config file and modify it to use later if they need to change the servers the malware connects to through the API mentioned later in this article. This file was not initially present on the system.

The second method will grab the payload data from the .data section if the file from the first method does not exist. This means that the threat actor must pre-compile each malware for each target if they want the remote target to be different.

The encryption in this target payload is the malware author’s own version of a stream cipher. A stream cipher is an encryption scheme in which the key interacts with each byte of the ciphertext.

The key, generated by a pseudorandom algorithm, continuously expands to match the ciphertext length. This contrasts with block ciphers like AES and DES, which operate on fixed-size blocks.

The format of the target payload we analyzed consists of three main parts: the size of the encrypted block, the ciphertext and the key. The size and key are 4-byte values but are originally in big-endian byte ordering, meaning that the most significant byte is ordered first rather than last.

Figure 5 shows how this encrypted format originally looked. The size and key are represented as arrays to emphasize their big-endian ordering. In this case, the ciphertext size is 0x8E, and the key is 0x51AF015D.

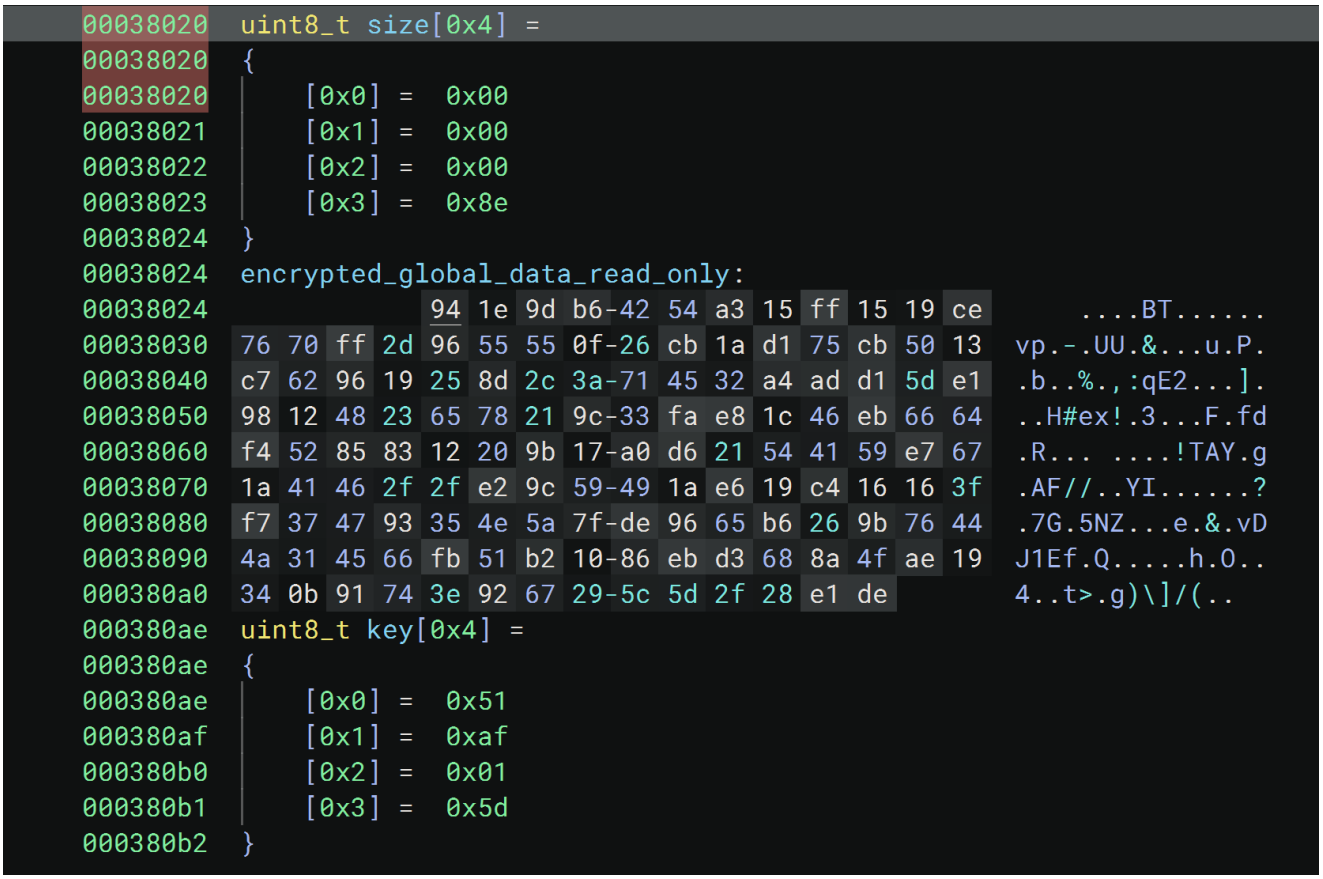


Figure 5. Encrypted format of the target payload.

The custom encryption algorithm does not use preexisting cryptographic standards like AES or DES. The key decrypts each byte of the ciphertext by performing a bitwise XOR and subtraction operations between the 4-byte key and a single byte of ciphertext.

After decrypting each byte, a new key is generated using the old key to operate on the next byte. This final payload contains the actual targets the malware will connect to when operating the main API discussed in the next section.

Core C2 Protocol and API Structure

Upon connecting to the threat actor's machine, the malware initiates a simple handshake with the remote server, with a simple random 16-byte value check.

If the server adheres to the protocol, it will echo the 16 bytes. After the handshake, the malware enters its main loop, awaiting commands from the remote target and following according to the metadata given.

Each message from the infected machine or the remote server follows a specific protocol structure unique to this malware family. One message consists of two main parts: a message header and a payload. The message header is then split into four main parts listed below:

- A 4-byte key that encrypts the rest of the metadata and payload
- A command ID that tells which specific operation is happening
- If the operation was successful, an error code value containing 0, or a value code representing the reason the error occurred
- A payload size

Keys in this protocol are dynamically generated using random values. Thus, the encryption in this protocol relies on the fact that it is secret rather than keeping the key secret within the program. Each message uses a unique, one-time key.

Once a message is given from the remote server to the infected machine, the malware will decrypt and parse the header and payload contents. The malware then reads the command ID value in the header to determine which functionality to execute based on a large switch statement. The next section includes a table that highlights the categories of functionality the malware can perform.

Each payload has a unique structure for the specific API command being run based on the command ID value due to the different types of arguments used. The payload structure uses a binary format rather than being sent in a human readable format like JSON or XML.

Before the arguments from the remote server can be used for an API command, the malware will need to convert arguments from network byte ordering to host byte ordering. This is needed because if the wrong byte ordering is used, a completely different value will be interpreted by the malware. The types of values used for arguments include C-style strings and integral values of potentially different byte lengths.

After a command has been received from the remote server, then parsed and executed, the malware will send back the result in a header-only message (a zero-length payload). This message gives the remote server information on what command was being executed as well as the error code that caused the command to fail, if any.

After a command finishes executing, the loop begins again waiting for the remote server to send another message to the infected machine. If the connection is broken, the malware will sleep before reconnecting to the remote server.

Malware C2 API Functionality

This section briefly describes the entire API and its main categories of capabilities. Table 1 describes each command ID value grouped together and the main functionality of each group of command IDs. Each command ID will be given in hex format, where XX is a placeholder value used to group the items together.

Command IDs	Category Name	Description

0, 1, 2, 3, 0xF	General options and kill switch	Sends host information and includes a kill switch to uninstall itself from the system
0x100	Reverse shell	Creates a reverse shell for the remote server to interact with the victim machine directly
0x2XX	File operations and manipulation	Create and/or modify files and execute programs locally
0x300	Network proxy	The infected machine will act as a middleman proxy for any connections between the remote target and the IP address given in the argument
0x4XX	Global payload manipulation	Sends and manipulates global configuration data mentioned previously

Table 1. API of Auto-color.

Conclusion

Auto-color is an emerging threat that Palo Alto Networks researchers discovered that does several things to avoid detection. The evasive actions range from trivial things such as renaming the malware to a benign-looking name like Auto-color, to more sophisticated methods such as hiding system network connections and preventing uninstallation through hooking libc functionality.

Upon execution, the malware attempts to receive remote instructions from a command server that can create reverse shell backdoors on the victim's system. The threat actors separately compile and encrypt each command server IP using a proprietary algorithm. IoCs are listed at the end to help readers identify whether their systems have been compromised by Auto-color.

Palo Alto Networks customers are better protected from the threats discussed in this article through the following products or services:

- The Advanced WildFire machine-learning models and analysis techniques have been reviewed and updated in light of the IoCs shared in this research.
- Cortex XDR and XSIAM block and alert on known behaviors and indicators associated with Auto-color.
- Advanced URL Filtering and Advanced DNS Security identify known URLs and domains associated with this activity as malicious.

If you think you may have been compromised or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America: Toll Free: +1 (866) 486-4842 (866.4.UNIT42)
- UK: +44.20.3743.3660
- Europe and Middle East: +31.20.299.3130
- Asia: +65.6983.8730
- Japan: +81.50.1790.0200
- Australia: +61.2.4062.7950
- India: 00080005045107

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

Malicious files from Auto-Color:

SHA256 hash: 270fc72074c697ba5921f7b61a6128b968ca6ccbf8906645e796cfc3072d4c43

- File size: 229,160 bytes
- File name: log
- File type: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked
- File description: Sample 1 malware from Auto-color

SHA256 hash: 65a84f6a9b4ccddcdae812ab8783938e3f4c12cfba670131b1a80395710c6fb4

- File size: 229,160 bytes
- File name: edus
- File type: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked
- File description: Sample 2 malware from Auto-color

SHA256 hash: 83d50fcf97b0c1ec3de25b11684ca8db6f159c212f7ff50c92083ec5fbd3a633

- File size: 229,160 bytes
- File name: egg
- File type: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked
- File description: Sample 3 malware from Auto-color

SHA256 hash:
a1b09720edcab4d396a53ec568fe6f4ab2851ad00c954255bf1a0c04a9d53d0a

- File size: 229,160 bytes

- File name: edu
- File type: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked
- File description: Sample 4 malware from Auto-color

SHA256 hash: bace40f886aac1bab03bf26f2f463ac418616bacc956ed97045b7c3072f02d6b

- File size: 229,160 bytes
- File name: door
- File type: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked
- File description: Sample 5 malware from Auto-color

SHA256 hash:
e1c86a578e8d0b272e2df2d6dd9033c842c7ab5b09cda72c588e0410dc3048f7

- File size: 229,160 bytes
- File name: exup
- File type: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked
- File description: Sample 6 malware from Auto-color

SHA256 hash: 85a77f08fd66aeabc887cb7d4eb8362259afa9c3699a70e3b81efac9042bb255

- File size: 229,160 bytes
- File name: law
- File type: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked
- File description: Sample 7 malware from Auto-color

SHA256 hash: bf503b5eb456f74187a17bb8c08bccc9b3d91a7f0f6fd50110540b051510d1ca

- File size: 35,160 bytes
- File name: libcext.so.2
- File type: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked
- File description: Library Implant from Auto-color

Malicious C2 IP Addresses from Auto-Color:

- 146[.]70[.]41[.]178:443 - log sample
- 216[.]245[.]184[.]214:443 - edus/egg sample
- 146[.]70[.]87[.]67:443 - edu/door sample
- 65[.]38[.]121[.]64:443 - exup sample
- 206[.]189[.]149[.]191:443 - law sample

Additional Resources

Symbiote Deep-Dive: Analysis of a New, Nearly-Impossible-to-Detect Linux Threat – Intezer

Copyright © 2025 Palo Alto Networks. All Rights Reserved