Cl0p Ransomware Targets Linux Systems with Flawed Encryption | Decryptor Available

sentinelone.com/labs/cl0p-ransomware-targets-linux-systems-with-flawed-encryption-decryptor-available/

Antonis Terefos

Executive Summary

- SentinelLabs has observed the first Linux variant of Cl0p ransomware.
- The ELF executable contains a flawed encryption algorithm making it possible to decrypt locked files without paying the ransom.
- SentinelLabs has published a free decryptor for this variant here.

Background

SentinelLabs observed the first ELF variant of Cl0p (also known as Clop) ransomware variant targeting Linux systems on the 26th of December 2022. The new variant is similar to the Windows variant, using the same encryption method and similar process logic.

The mentioned sample appears to be part of a bigger attack that possibly occurred around the 24th of December against a University in Colombia (sample1, sample2, sample3, sample4, sample5). On the 5th of January the cybercrime group leaked victim's data on their onion page.

ELF Technical Analysis

The ELF Cl0p variant is developed in a similar logic to the Windows variant, though it contains small differences mostly attributed to OS differences such as API calls. It appears to be in its initial development phases as some functionalities present in the Windows versions do not currently exist in this new Linux version.

A reason for this could be that the threat actor has not needed to dedicate time and resources to improve obfuscation or evasiveness due to the fact that it is currently undetected by all 64 security engines on VirusTotal. SentinelOne Singularity detects Cl0p ransomware on both Linux and Windows devices.

SentinelOne Singularity detects Cl0p Linux ransomware

Initially, the ransomware creates a new process by calling fork and exits the parent-process. The child-process sets its file mode creation mask to any permission (read, write, execute) by calling umask(0). It then calls setsid, creates a session and sets the process group ID. It tries to access root by changing the working directory to "/" (chdir("/")). Once the permissions are set, the ransomware proceeds encrypting other directories.

Targeted Folders & Files

While the Windows versions contain a hashing algorithm in order to avoid encrypting specific folders and files, such functionality was not observed in the Linux variant. The ELF variant targets specific folders, subfolders and all files/types.

The discovered ELF sample targets files contained in the following directories for encryption, though we do not exclude the possibility of future versions including more directories.

Folder	Description
/opt	Contains subdirectories for optional software packages
/u01	Oracle Directory, mount point used for the Oracle software only.
/u02	Oracle Directory, used for the database files.
/u03	Oracle Directory, used for the database files.
/u04	Oracle Directory, used for the database files.
/home	Contains the home directory of each user.
/root	Contains the home directory of the root user.

Encryption Flaw

Windows versions of Cl0p ransomware use a Mersenne Twister PRNG (MT19937) to generate a **0x75** bytes size RC4 key for each file. This key is then validated (checks if the first five bytes are NULL) and used for file encryption. Then, by using the RSA public key, it encrypts the generated RC4 key and stores it to `$filename.$clop_extension`. Victims who pay the ransom demand receive a decryptor which decrypts the generated Cl0p file using the RSA private key, retrieves the generated RC4 key, and then decrypts the encrypted file.

This core functionality is missing in the Linux variant. Instead, we discovered a flawed ransomware-encryption logic which makes it possible to retrieve the original files without paying for a decryptor.

The Linux variant contains a hardcoded RC4 “master-key” which, during the execution of the main function, is copied into the global variable `szKeyKey`.

Sample’s RC4 “master-key”:

```
Jfkdskfku2ir32y7432uroduw8y7318i9018urewfdsZ2Oaifwuieh~~cudsffdsd
```

During the file encryption phase, the ransomware – similar to the Windows version – generates a **0x75** bytes size RC4 key, with the use of a lookup table and a PRNG byte. This generated RC4 key is used to encrypt the mappedAddress and write it back to the file.

Then by using the RC4 “master-key” the ransomware encrypts the generated RC4 key and stores it to `$filename.$clop_extension`. By using a symmetric algorithm (second RC4) to “encrypt” the file’s RC4 key, we were able to take advantage of this flaw and decrypt Cl0p-ELF encrypted files.

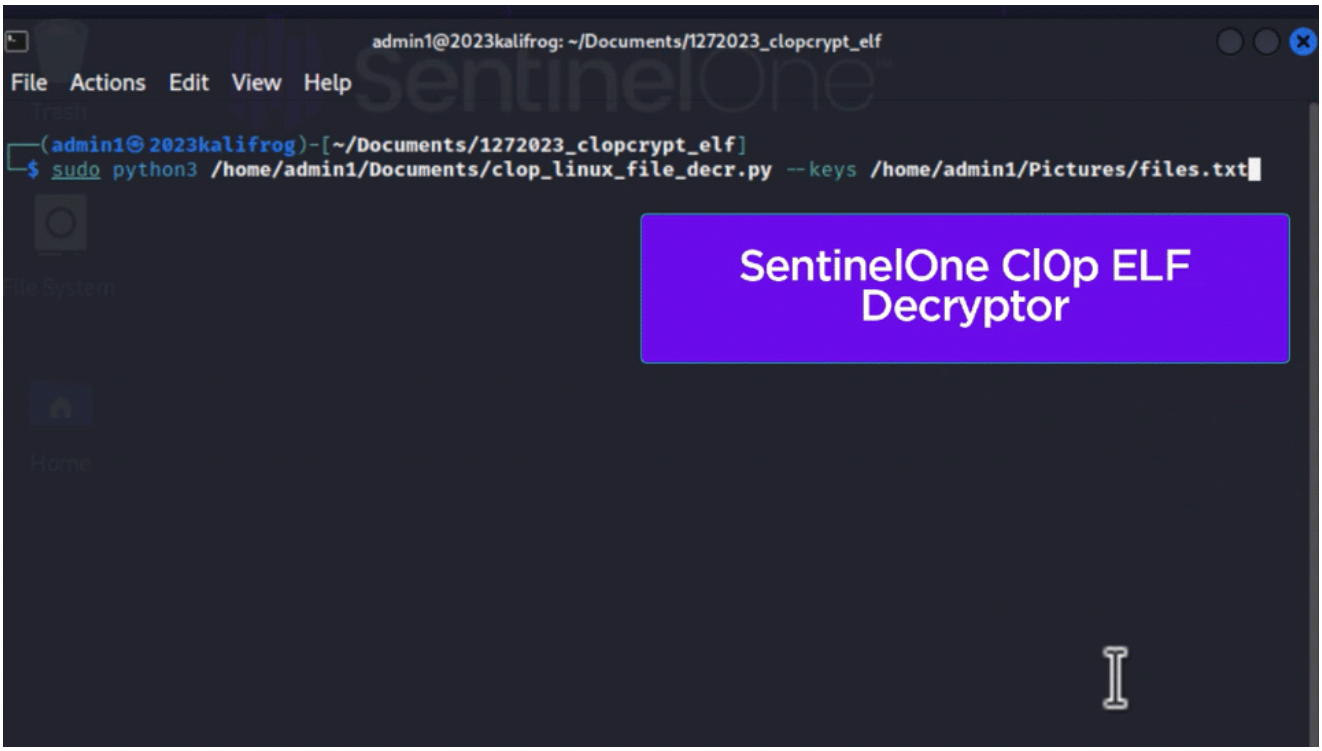
```
memset((int)rc4_key, 0, sizeof(rc4_key));
for ( index = 0; index <= 116; ++index )  // generate 0x75 RC4 key
  rc4_key[index] = lookup_table[randx()]; // Mersenne Twister PRNG
_rc4Full(rc4_key, 0x75u, mappedAddressPointer,, fileSize);// encrypt file using generated RC4 key
keyLength = strlen(szKeyKey) + 1;
keyLength_m1 = keyLength - 1;
_rc4Full(szKeyKey, keyLength - 1, rc4_key, 0x75u);// encrypt generated RC4 key using RC4 "master-key"
Createkey(pathname, (unsigned __int8 *)rc4_key);// Create key file, $filename.$clop_extension
return munmap(mappedAddressPointer,, fileSize);// "write" encrypted filebuffer
```

Cl0p-ELF encryption flaw

Cl0p-ELF Decryption Logic:

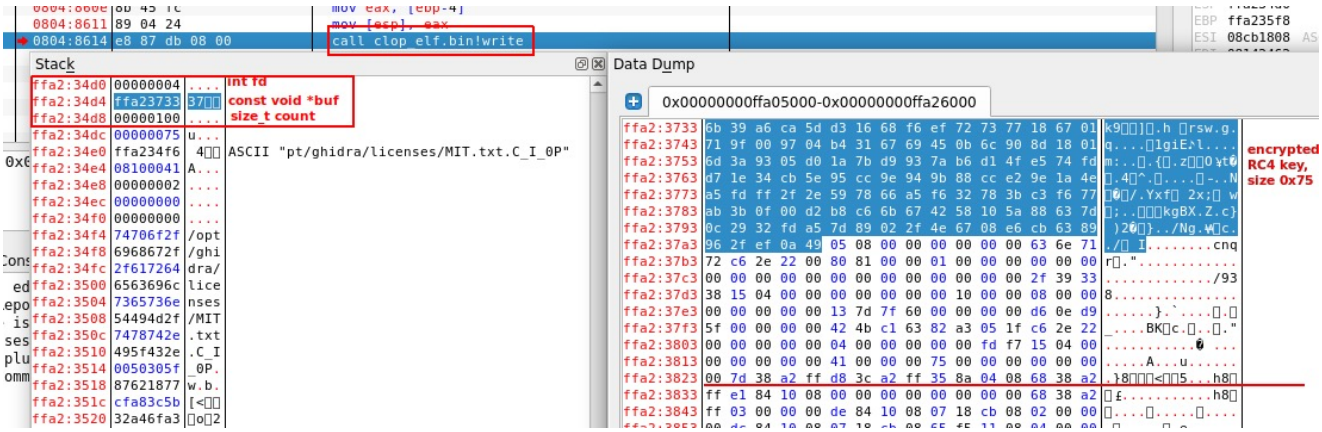
1. Retrieve RC4 “master-key”.
2. Read all `$filename.$clop_extension`.
3. Decrypt with RC4 using the RC4 “master-key”, the generated RC4 key.
4. Decrypt `$filename` with RC4 using the generated RC4 key.
5. Write decrypted to `$filename`.

We packed all this logic into the following Python script.



Cl0p File-Key Creation Flaw

The **0x75** bytes size PRNG RC4 key is encrypted with RC4 using the RC4 “master-key”. The encrypted RC4 output is **0x75** bytes size, though writes **0x100** bytes into the created Cl0p key `$filename.$clop_extension`. This results in writing memory data to the file and more specifically stack variables.



Cl0p-ELF file-key creation flaw.

This flaw provides some information regarding the file before encryption. This includes:

- File fstat64 result
  - total size, in bytes, file size (`st_size`)
  - time of last status change, exact time of file encryption (`st_ctime`)
  - and more forensics information regarding the file before the encryption.
- Size of buffer for file encryption (with check of `>= 0x5f5e100`)

- RC4 “master-key” size
- RC4 PRNG key size

```
struct  clopelfkeyfile  {
        byte encr_rc4key[117]; // encrypted RC4 PRNG key, size 0x75 bytes
        stat fdstat; // stat(fd, &fdstat), size 0x58 bytes
        long fdid; // file node unique id, size 0x8 bytes
        int fd; // file descriptor, size 0x4 bytes
        int fdmappedaddr; // file mapped address, size 0x4 bytes
        off_t fdsize; // file size, size 0x8 bytes
        int rc4_msize; // RC4 "master-key" size, size 0x4 bytes
        long rc4_fsize; // RC4 PRNG key size, size 0x8 bytes
        int fdnameaddr; // filename string address, size 0x4 bytes
        int frameaddr; // frame pointer address, size 0x4 bytes
        int retaddr; // function return address, size 0x4 bytes
        byte fdpathaddr[3]; // part of filepath strings address, size 0x3 bytes
}
```

Developed Functions & Names

In ELF binaries the `.symtab`, Symbol Table Section, holds information needed to locate and relocate a program’s symbolic definitions and references, allowing us to retrieve function and global variable names.

Function Name	Description
do_heartbeat(void)	Main function which starts the encryption of various folders.
find(char *,char const*)	Multiple calls of this function are done by do_heartbeat; this function takes as parameter 1) the starting folder to encrypt (example, “/opt”) 2) regex of files to encrypt (example, “*.*”) and performs a recursive search from the starting folder until encrypts the “matching” regex files.
CreateRadMe(char *)	This function takes as parameter the folder to create the ransom note.
EncrFile(char *)	Encrypts given filepath.
existsFile(char *)	Checks if File exists, or if the process has the permissions to open.
_rc4Full(void const*,ushort,void *,ulong)	Wrapper function to _rc4Init and _rc4, which is used to encrypt a buffer with a given key.
Createkey(char *,uchar *)	Creates and writes into “%s.C_I_0P” the encrypted buffer.

Global Variable	Description
szKeyKey	Global variable of 0x64 bytes size, initialized during main function, containing RC4 “master-key” which encrypts the “randomly” generated 0x75 bytes size RC4 key.

Differences to Windows Variant

Rather than simply port the Windows version of Cl0p directly, the authors have chosen to build bespoke Linux payloads. We understand this to be the primary reason for the lack of feature parity between the new Linux version and the far more established Windows variant.

SentinelLabs expects future versions of the Linux variant to start eliminating those differences and for each updated functionality to be applied in both variants simultaneously.

Some of the differences worth highlighting are detailed below:

Differences	Description
Files/Folders exclusions	The Windows variant contains a hashing algorithm which excludes specific folders and files from encryption. This functionality was not observed in the Linux variant.
Extension exclusions	The Windows variant contains a hardcoded list of extensions to exclude from encryption. This functionality was not observed in the Linux variant.
Different methods of Reading/Writing depending on file size.	The Windows variant, depending on the size of the file, will choose different methods of reading a file and writing the encrypted buffer. Small files are ignored, medium-sized files will make use of ReadFile/WriteFile, large files will use CreateFileMappingW/MapViewOfFile/UnmapViewOfFile. The Linux variant encrypts all the files using mmap64/munmap. Both variants only encrypt the first 0x5f5e100 bytes of large files.
Ransom Note Decryption	The Windows variant stores the encrypted ransom note as a resource and decrypts it with a simple XOR algorithm. The Linux variant stores the note as plain text in “.rodata”.
Drive enumeration	The Windows variant initially enumerates through drives in order to “find” the starting point to recursively encrypt the folders. The Linux variant contains hardcoded “starting” folders.
RC4 default Key	Once the Windows variant generates a 0x75 size PRNG RC4 Key, it will check if the first 5 bytes are NULL; if so, it uses the default key for encryption. The Linux version does not perform this validation and does not contain a default RC4 key in case the first 5 bytes of the PRNG RC4 are NULL.

Command Line Parameters	The Windows variant can be executed in three ways: 1) without parameters encrypting all local and network drives, 2) with “runrun” parameter encrypting only network drives, 3) with a file as parameter which contains the folders to be encrypted (observed temp.ocx/temp.dat). The Linux variant does not accept command line parameters and recursively encrypts the specified hardcoded folders.
RC4 Key Encryption	The Windows variant encrypts the generated RC4 key responsible for the file encryption using the asymmetric algorithm RSA and a public key. In the Linux variant, the generated RC4 key is encrypted with a RC4 “master-key” (flawed logic).

Ransom Notes

The Linux variant of Clop ransomware drops a ransom note on victim machines with a `.txt` format.

```
CONTACT US BY EMAIL:
unlock@support-mult.com
or
unlock@rsv-box.com
OR WRITE TO THE CHAT AT->
http://6v4q5w7di74grj2vtmikzgx2tnq5eagyg2cubpcnqrvvee2ijpmprzqd.onion/remote0/938
68e77-1331-411a-9643-dc9ad26a5095?secret={UNI_NAME}
(use TOR browser)
```

ELF sample ransom note, “README_C_I_0P.TXT”.

This differs somewhat from the Windows `.rtf` ransom note, although both use the email addresses `[email protected][.]com` and `[email protected][.]com` as ways for victims to contact the attackers.

```
 ___ {UNI_NAME} ___

 === DO NOT ATTEMPT TO RESTORE OR MOVE THE FILES YOURSELF. THIS MAY DESTROY
THEM ===

 Here are some of the files we downloaded from your network:
{UNI_PII}



If you refuse to cooperate, all data will be published
for free download on our portal:
http://santat7kpllt6iyvqbr7q4amdv6dzrh6paatvyrzl7ry3zm72zigf4ad.onion/ ->   TOR
browser

CONTACT US BY EMAIL->
unlock@support-mult.com
or
unlock@rsv-box.com
OR WRITE TO THE CHAT AT->
http://6v4q5w7di74grj2vtmikzgx2tnq5eagyg2cubpcnqrvvee2ijpmprzqd.onion/remote0/938
68e77-1331-411a-9643-dc9ad26a5095?secret={UNI_NAME}
(use TOR browser)
```

Window samples ransom note, “!_READ_ME.RTF”.

Conclusion

Over the last twelve months or so we have continued to observe the increased targeting of multiple platforms by individual ransomware operators or variants. The discovery of an ELF-variant of Cl0p adds to the growing list of the likes of Hive, Qilin, Snake, Smaug, Qyick and numerous others.

We know that Cl0p operations have shown little if no slow-down since the disruption in June 2021. While the Linux-flavored variation of Cl0p is, at this time, in its infancy, its development and the almost ubiquitous use of Linux in servers and cloud workloads suggests that defenders should expect to see more Linux-targeted ransomware campaigns going forward.

SentinelLabs continues to monitor the activity associated with Cl0p. SentinelOne Singularity protects against malicious artifacts and behaviors associated with Cl0p attacks including the ELF variant described in this post.

Indicators of Compromise

IOC Type	IOC Value

SHA1 ELF Cl0p	46b02cc186b85e11c3d59790c3a0bfd2ae1f82a5
SHA1 Win Cl0p	40b7b386c2c6944a6571c6dcfb23aaae026e8e82
SHA1 Win Cl0p	4fa2b95b7cde72ff81554cfbddc31bbf77530d4d
SHA1 Win Cl0p	a1a628cca993f9455d22ca2c248ddca7e743683e
SHA1 Win Cl0p	a6e940b1bd92864b742fbd5ed9b2ef763d788ea7
SHA1 Win Cl0p	ac71b646b0237b487c08478736b58f208a98eebf
SHA1 ELF Cl0p Note	ba5c5b5cbd6abdf64131722240703fb585ee8b56
SHA1 Win Cl0p Note	77ea0fd635a37194efc1f3e0f5012a4704992b0e
ELF Ransom Note	README_C_I_0P.TXT
Win Ransom Note	!_READ_ME.RTF
Cl0p Ransom Extension	.C_I_0P
Cl0p Contact Email	unlock[@]support-mult.com
Cl0p Contact Email	unlock[@]rsv-box.com
Cl0p Onion Leak Page	hxxp[:]//santat7kpllt6iyvqbr7q4amdv6dzrh6paatvyrzl7ry3zm72zigf4ad[.]onion

Cl0p Onion Chat Page	hxxp[:]//6v4q5w7di74grj2vtmikzgx2tnq5eagyg2cubpcnqrvvee2ijpmprzqd[.]onion

YARA Rule

```
rule ClopELF
{
    meta:
        author = "@Tera0017/@SentinelLabs"
        description = "Temp Clop ELF variant yara rule based on $hash"
        reference = "https://s1.ai/Clop-ELF”
        hash = "09d6dab9b70a74f61c41eaa485b37de9a40c86b6d2eae7413db11b4e6a8256ef"
    strings:
        $code1 = {C7 45 ?? 00 E1 F5 05}
        $code2 = {81 7D ?? 00 E1 F5 05}
        $code3 = {C7 44 24 ?? 75 00 00 00}
        $code4 = {C7 44 24 ?? 80 01 00 00}
        $code5 = {C7 00 2E [3] C7 40 04}
        $code6 = {25 00 F0 00 00 3D 00 40 00 00}
        $code7 = {C7 44 24 04 [4] C7 04 24 [4] E8 [4] C7 04 24 FF FF FF FF E8 [4] C9
C3}
    condition:
        uint32(0) == 0x464c457f and all of them
}
```