COMpfun successor Reductor infects files on the fly to compromise TLS traffic

By GReAT

Published: 2019-10-03 · Archived: 2026-04-05 23:09:52 UTC

In April 2019, we discovered new malware that compromises encrypted web communications in an impressive way. Analysis of the malware allowed us to confirm that the operators have some control over the target's network channel and could replace legitimate installers with infected ones on the fly. That places the actor in a very exclusive club, with capabilities that few other actors in the world have.

We called these new modules 'Reductor' after a .pdb path left in some samples. Besides typical RAT functions such as uploading, downloading and executing files, Reductor's authors put a lot of effort into manipulating digital certificates and marking outbound TLS traffic with unique host-related identifiers.

The Kaspersky Attribution Engine shows strong code similarities between this family and the COMPfun Trojan. Moreover, further research showed that the original COMpfun Trojan most probably is used as a downloader in one of the distribution schemes. Based on these similarities, we're quite sure the new malware was developed by the COMPfun authors.

The COMpfun malware was initially documented by G-DATA in 2014. Although G-DATA didn't identify which actor was using this malware, Kaspersky tentatively linked it to the Turla APT, based on the victimology. Our telemetry indicates that the current campaign using Reductor started at the end of April 2019 and remained active at the time of writing (August 2019). We identified targets in Russia and Belarus.

We registered two initial infection schemes: Reductor spreads by either infecting popular software distributions (Internet Downloader Manager, WinRAR, etc. and, for at least one victim, through a popular warez website over HTTP); or its decryptor/dropper is spread using COMpfun's ability to download files on already infected hosts.

How to mark the TLS handshake without even touching the traffic

The malware adds digital certificates from its data section to the target host and allows the operators to add additional certificates remotely through a named pipe. The solution that Reductor's developers found to mark TLS traffic is the most ingenious part. They don't touch the network packets at all; instead developers analyzed the Firefox source code and Chrome binary code to patch the corresponding pseudo random number generation (PRNG) functions in the process's memory.

Browsers use PRNG to generate the 'client random' sequence for the network packet at the very beginning of the TLS handshake. Reductor adds encrypted unique hardware- and software-based identifiers for the victims to this 'client random' field. In order to patch the system's PRNG functions, the developers used a small embedded Intel instruction length disassembler.

```
static SECStatus
ssl3_GetNewRandom(SSL3Random random)
{
    SECStatus rv;

    rv = PK11_GenerateRandom(random, SSL3_RANDOM_LENGTH);
    if (rv != SECSuccess) {
        ssl_MapLowLevelError(SSL_ERROR_GENERATE_RANDOM_FAILURE);
    }
    return rv;
}
```

```
/* Generate a new random if this is the first attempt. */
if (type == client_hello_initial) {
    rv = ssl3_GetNewRandom(ss->ssl3.hs.client_random);
    if (rv != SECSuccess) {
        goto loser; /* err set by GetNewRandom. */
    }
}

if (ss->vrange.max >= SSL_LIBRARY_VERSION_TLS_1_3) {
    rv = tls13_SetupClientHello(ss, type);
    if (rv != SECSuccess) {
        goto loser;
    }
}
```

In order to patch browser PRNG memory functions and add unique user IDs into the TLS handshake, the developers of Reductor had to analyze Firefox and Chrome code

Why we believe on-the-fly infection took place

As we don't know what happens on the 'server' side, we can only rely on 'client' analysis. In order to distinguish handshakes of interest from all the TLS traffic, the campaign operators firstly have to decrypt this 'client hello' field. This means the campaign operators somehow need to have access to the target's traffic.

The Reductor malware does not carry out a man-in-the-middle (MitM) attack itself. However, our initial thought was that the installed certificates may facilitate MitM attacks on TLS traffic; and the 'client random' field, with the unique ID in the handshake, would identify the traffic of interest. Later analysis provided even more basis for this idea.

We initially observed that infected installers were downloaded from HTTPS warez websites; but, as often happens, the files themselves were downloaded through unencrypted HTTP. This makes it technically possible to replace the files with malicious ones during the download process. Interestingly, the configuration data of some samples contained very popular legitimate websites. We really don't think they were compromised to serve as control servers.

In any case, we didn´t initially know how the installers were infected, because the original downloaded files were no longer available for analysis on the warez websites. And there was always the possibility that the installers were infected on the website from which they were originally downloaded.

Then more recent Reductor telemetry gave us a clue. This time samples were again being downloaded from warez websites, but we were able to confirm that in this new case the original installers were not infected. This allowed us to confirm that Reductor's operators have some control over the target's network channel and could replace legitimate installers with infected ones on the fly.

Reductor features

The malware authors are creative and sometimes even seem to be having a bit of fun. For instance, one of the web domains they use for COMpfun (the publicly known name) is compfun[.]net. The domain-user-password triad hardcoded into the decryptor-dropper was "uac is useless". Here's a summary of the different types of campaign artifacts found:

	Initial infection	**Escalation, detection avoidance**	**Main payload**
Malware	COMpfun Trojan	Reductor dropper-decryptor	Reductor Trojan
Process	One of the browsers	Same browser	lsass.exe
Persistence	COM CLSID hijacking	Auxiliary module, N/A	LSA notification package
Net encryption	AES 128	Local module, N/A	AES 128
Host encryption	Configuration data encrypted with one byte XOR and compressed with LZNT1	Reductor in resources encrypted with one byte XOR and compressed with LZNT1	Victims' unique IDs in TLS 'client hello' encrypted using XOR with changing round key

As we have already mentioned, there are two different methods used by the attackers to spread Reductor. In the first scenario, the attackers use infected software installers with 32- and 64-bit versions of Reductor included. These installers may be for popular Internet Download Manager, Office Activator, etc.

In the second scenario, the targets are already infected with the COMpfun Trojan, which uses COM CLSID for persistence. After getting into the browser's address space, the Trojan can receive the command to download additional modules from the C2. As a result, the target's browser downloaded Reductor's custom dropper-decryptor.

The coding style is quite distinctive throughout the modules. Take a look at them in the following table:

Feature	**Description**

Strings storage	All the strings in use, such as function names for resolving dynamic addresses, are returned by the small functions. The developers probably implemented them using the C preprocessor #define directive.
Function address dynamic resolution	For every dynamic linked library in use, the developers implemented a standalone function and a custom structure to store the addresses of its functions for further use.
Extensive use of custom structures	The developers used custom structures for every task: C2 communication, thread synchronization, resolving of system function addresses, etc.

System fingerprinting hashes inside TLS ‘client random’

As mentioned above, Reductor adds its own ‘victim id’ inside TLS packets. The first four-byte hash (cert_hash) is built using all of Reductor’s digital certificates. For each of them, the hash’s initial value is the X509 version number. Then they are sequentially XORed with all four-byte values from the serial number. All the counted hashes are XOR-ed with each other to build the final one. The operators know this value for every victim, because it’s built using their digital certificates.

The second four-byte hash (hwid_hash) is based on the target’s hardware properties: SMBIOS date and version, Video BIOS date and version and hard drive volume ID. The operators know this value for every victim because it’s used for the C2 communication protocol. The resulting custom 16-byte structure to spoof the originally PRNG-generated values looks like this:

```
1  struct client_hello_system_fingerprint {
2  DWORD initial_xor_key; // First four bytes generated by original system PRNG function
3  DWORD predefined_const; // Set to 0x45F2837D
4  DWORD cert_hash; // Reductor's digital certificates hash
5  DWORD hwid_hash // Target's hardware hash
6  };
```

The latter three fields are encrypted using the first four bytes – initial PRN XOR key. At every round, the XOR key changes with the MUL 0x48C27395 MOD 0x7FFFFFFF algorithm. As a result, the bytes remain pseudo random, but with the unique host ID encrypted inside.

PRNG patching

The table below enumerates the patched auxiliary and PRNG system functions.

Library	Patched function	Features

Auxiliary functions		
“ntdll.dll”	RtlReleaseResource()	Save auxiliary data like current thread ID and current tick count;
	memcpy()	If “client hello” has to be copied, then count cert_hash and hwid_hash, change source bytes to encrypted client_hello_system_fingerprint structure and call original memcpy();
One of the C runtime libraries	time64()	Save time passed since 1 January 1970;
“kernel32.dll” or “kernelbase.dll”	GetSystemTimeAsFileTime()	
PRNG functions		
“nss3.dll”	PK11_GenerateRandom()	Call original PRNG function and generate initial XOR key from its result. Change PRNG result: set seventh byte to 1, then save 0x45F2837D, hwid and cert hashes. Encrypt the result and return it instead of the original PRN. It will affect calls to ssl3_SendClientHello() -> ssl3_GetNewRandom(ss->ssl3.hs.client_random);
“advapi32.dll”	CryptGenRandom()	Spoof these system PRNG function results in similar way with some minor changes;
“bcrypt.dll”	BCryptGenRandom()	
“chrome.dll”	PRNG function	Find PRNG function by its binary code template and patch it like all the aforementioned.

Firefox nss3.dll PK11_GenerateRandom() patching

Reductor patches nss3.dll for Firefox. This library’s source code is publicly available. PK11_GenerateRandom() is used in the /security/nss/lib/ssl/ssl3con.c in the ssl3_GetNewRandom() function. The SSL3_RANDOM_LENGTH constant is 32 bytes, so Reductor’s code changes all the results and the functions, which call to ssl3_GetNewRandom() will receive the modified random data with the encrypted target fingerprinting inside.

In this case, the caller function to ssl3_GetNewRandom(ss->ssl3.hs.client_random) is ssl3_SendClientHello() in order to generate the client random data for the initial communication handshake.

```
static SECStatus
ssl3_GetNewRandom(SSL3Random random)
{
    SECStatus rv;

    rv = PK11_GenerateRandom(random, SSL3_RANDOM_LENGTH);
    if (rv != SECSuccess) {
        ssl_MapLowLevelError(SSL_ERROR_GENERATE_RANDOM_FAILURE);
    }
    return rv;
}
```

```
/* Generate a new random if this is the first attempt. */
if (type == client_hello_initial) {
    rv = ssl3_GetNewRandom(ss->ssl3.hs.client_random);
    if (rv != SECSuccess) {
        goto loser; /* err set by GetNewRandom. */
    }
}

if (ss->vrange.max >= SSL_LIBRARY_VERSION_TLS_1_3) {
    rv = tls13_SetupClientHello(ss, type);
    if (rv != SECSuccess) {
        goto loser;
    }
}
```

To affect the TLS handshake malware authors patched PK11_GenerateRandom() inside the Firefox process memory

Patching PK11_GenerateRandom() would also affect the generation of any 256-bit (32 bytes) initialization vector (IV) generation, for example, for AES 256 in ssl_SelfEncryptProtect() or other crypto functions in NSS libraries used by Firefox. From our point of view, this would be a side effect of Reductor with no additional purpose.

Installed digital certificates

Reductor samples hold DER-encoded root X509v3 certificates in the .data section to add on the target hosts. The malware is also able to get additional certificates from the operators through a named pipe.

Certificate SHA1 fingerprint	CA for root cert	Valid till (GMT)
119B2BE9C17D8C7C5AB0FA1A17AAF69082BAB21D	ie-paypal	2031.11.17 22:56:10
546F7A565920AEB0021A1D05525FF0B3DF51D020	GeoTrust Rsa CA	2031.11.17 22:56:10
959EB6C7F45B7C5C761D5B758E65D9EF7EA20CF3	GeoTrust Rsa CA	2031.11.17 22:56:10
992BACE0BC815E43626D59D790CEF50907C6EA9B	VeriSign, Inc.	2031.11.17 22:56:10

One of the decoded CA X509v3 certificates inside the Reductor malware

C2 communication

All C2 communications are handled in a standalone malware thread. Reductor sends HTTP POST queries to the /query.php scripts on the C2s listed in its configuration. The POST query contains the target's unique hardware ID encrypted with AES 128. The C2 returns one of the following encrypted commands.

C2 command	**Features**
hostinfo	Get the host name
gettimeout	Get the timeout value from the corresponding registry value
options	Parse strings and set corresponding values in the system registries. So far only one option is supported – timeout
domainlist	Transmit the current C2 domains used by target
downfile	Download the file of interest
upfile	Upload the file of interest
execfile	Create the process that executes mentioned file
nop	Do nothing. Possibly used to check the connection with the host
kill	Delete installed digital certificates, files, cookies and system registry values including those related to COM CLSID or LSA notification package persistence

deletefile	Delete file at a specified path
certlist	Renew the digital certificates installed on target

Conclusions

Turla has in the past shown many innovative ways to accomplish its goals, such as using hijacked satellite infrastructure[2]. This time, if we're right that Turla is the actor behind this new wave of attacks, then with Reductor it has implemented a very interesting way to mark a host's encrypted TLS traffic by patching the browser without parsing network packets. The victimology for this new campaign aligns with previous Turla interests.

We didn't observe any MitM functionality in the analyzed malware samples. However, Reductor is able to install digital certificates and mark the targets' TLS traffic. It uses infected installers for initial infection through HTTP downloads from warez websites. The fact the original files on these sites are not infected also points to evidence of subsequent traffic manipulation.

File Hashes

- 27CE434AD1E240075C48A51722F8E87F
- 4E02B1B1D32E23975F496D1D1E0EB7A6
- 518AB503808E747C5D0DDE6BFB54B95A
- 7911F8D717DC9D7A78D99E687A12D7AD
- 9C7E50E7CE36C1B7D8CA2AF2082F4CD5
- A0387665FE7E006B5233C66F6BD5BB9D
- F6CAA1BFCCA872F0CBE2E7346B006AB4

Domains and IPs

- adstat.pw
- bill-tat.pw
