Check Point
SOFTWARE TECHNOLOGIES LTD.

WE SECURE THE FUTURE

CHECK POINT
LOOKING INTO TESLACRYPT V3.0.1

BY STANISLAV SKURATOVICH, MALWARE REVERSE ENGINEER
MALWARE REVERSE ENGINEERING TEAM

OVERVIEW

TeslaCrypt is a very popular ransomware that first appeared in the wild at the beginning of 2015. Since its first appearance, TeslaCrypt has undergone several version changes, with each version introducing new abilities, fixing old bugs, adding new evasion techniques and using new technologies.

Several months ago, a new version of TeslaCrypt, v3.0.1 was released. It again introduces several notable changes, perhaps the most important of which is the ability to encrypt files while offline – i.e., C&C communication is not mandatory to initiate the encryption process. The new version also hides its activities by executing on very low priority, as well as closing applications that may lead to detection.

To provide a better understanding of the newly implemented mechanisms that support these new features, our research teams decided to closely examine this new version. This report details the inner workings of TelsaCrypt v3.0.1, as well as provides techniques that can be used to detect and block its malicious operations.

MALWARE FUNCTIONALITY AND PAYLOAD

The ransomware performs some initialization steps before starting the routines responsible for key generation and data decryption. First of all, the malware checks the `SECURITY_MANDATORY` level by calling the *`GetSidSubAuthority`* function and removes the `:ZoneIdentifier` stream from a binary file. If the integrity level is `SECURITY_MANDATORY_LOW_RID`, the ransomware uses the *`ShellExecuteEx`* function with a `runas` action for this command:

```
{WINDOWS_DIRECTORY}\system32\cmd.exe /c "" ${PATH_TO_EXE}
```

If the SECURITY_MANDATORY level is different, the ransomware checks if its path to the execution file contains the windir environment variable or CSIDL_MYDOCUMENTS (which depends on the user belonging to the Administrator group). If the path contains these specific directories, the ransomware continues execution. Otherwise, it copies the image file to one of the directories, depending on the user account rights. The name of the file is generated randomly and contains 0x0C lower ASCII symbols. It then executes the newly copied file and removes the original file with this command:

```
{WINDOWS_DIRECTORY}\system32\cmd.exe /c DEL ${PATH_TO_EXE}
```

The ransomware tries to stay persistent on the infected system by adding the following entry to the registry key:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    ${STRING_0x0C_RAND_LOWER}="C:\WINDOWS\system32\cmd.exe   /c   start   ""
"${PATH_TO_EXE}""
```

In order to allow higher level applications to access mapped network drives, the ransomware sets the following registry key value to 1:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Syste
m\EnableLinkedConnections
```

The ransomware reads the image checksum by accessing the field IMAGE_NT_HEADERS.Optional-Header.CheckSum. The strange thing is that it saves only the 2 middle bytes of that checksum in the process space. To run only one instance of the executable, the ransomware creates a mutex with the following name:

```
8765-123rvr4
```

At this point, the ransomware starts checking if it was already executed on the infected computer. First of all, it queries the following registry keys for the presence of the ID subkey:

```
HKEY_USERS\S-1-5-18\Software\zzzsys
HKEY_CURRENT_USER\Software\zzzsys
```

If such a subkey is present, the ransomware reads the value stored under it. This subkey specifies the *InstanceID* for the compromised machine. If the subkey is absent, the ransomware generates 0x08 random bytes (*InstanceID*) and stores them under the HKEY_CURRENT_USER\Software\zzzsys\ID registry key. The ID is converted to the hexlified string representation using the following format: *%X%X%X%X%X%X%X%X*. After generating or loading the ID, the ransomware checks the following registry keys for the presence of the data subkey:

```
HKEY_USERS\S-1-5-18\Software\zzzsys\hexlified_ID
HKEY_CURRENT_USER\Software\zzzsys\hexlified_ID
```

If the `data` subkey is present and the value length is equal to 0x100 bytes, the ransomware assumes that all global encryption keys are generated and initializes the following buffers:

```
struct reg_data {
  char EC_GLOB_BTN_ADDR[0x30]; // Bitcoin Address
  char G_EC_REC_PUB__AES_EC_GLOB_PRIV[0x80]; // Global recovery data
  char EC_GLOB_PUB_point2oct[0x48]; // Global public key converted to octet
string (see: OpenSSL EC_POINT_point2oct())
  uint64_t EncryptionKeysGenerationTime; // Time when encryption keys were
generated
};
```

If the `data` is valid, the ransomware generates only session ECDH keys (see **Session ECDH key generation**) that will be used for data encryption. If the `data` is absent or invalid, the ransomware performs the whole process of encryption key generation (see **Encryption key generation**). After the generation process, keys are saved to the following registry key using the same structure presented above:

```
HKEY_CURRENT_USER\Software\zzzsys\hexlified_ID\data
```

At this point, all initialization steps are performed, so the ransomware starts the following threads:

- Process killing thread (see **Process killing thread**).
- Remove Shadow Copy thread (see **Remove shadow copy thread**) if the `OSVERSIONINFO.dwBuildNumber` does not equal the `0xA28`, which refers to the Windows XP build number.

Before starting disk encryption, the ransomware decrypts some additional information as file extensions that are normally encrypted: C&C server addresses and messages about the ransom. The decryption process starts with some preprocessing of encrypted internal data. After that, the preprocessed data is decrypted using a RC2-CBC-PKCS5 stream cipher with the key `kasdfgh283` and an IV specific for each decrypted string. See **Appendix G** for the list of decrypted file extensions. As the ransomware uses a specific key for data decryption, the technique described in **Appendix B** can be used to detect the presence of TeslaCrypt and prevent disk encryption (at least for the described version).

The ransomware generates a random string with a length of 0x09 bytes and uses it as part of the recovery file name:

```
CSIDL_MYDOCUMENTS\recover_file_${rand_name}.txt
```

The following data is saved to this file (each field is separated with a newline symbol `\n`):

Format	Data	Representation
%s	EC_GLOB_BTN_ADDR	Plaintext
%s	G_EC_REC_PUB__AES_EC_GLOB_PRIV	Hexlified data
%s	hexlified_id	Plaintext
%d	Image partial checksum	Number

Another thing the malware does is prepare 3 versions of the ransom message: as a web page (see **Appendix E**), as an image (see **Appendix D**), and as a plaintext message (see **Appendix F**). After all these actions, the ransomware starts the following threads:

- Internet communication thread that is responsible for sending the `Sub=Ping` command to the C&C server (see **Network and Communication**).
- Encryption thread, setting its priority to `IDLE` to avoid drastic increase of CPU usage.

The ransomware performs disk encryption operations using an AES-CBC-128 algorithm with the previously generated random keys. The file header that stores encryption-related information and encrypted data are stored in the original file. After these operations, the ransomware appends an `.mp3` extension to the original file name. Recovery files that notify the user about the encryption are stored in the directories. For more information, see **Encryption thread**.

The ransomware waits until the encryption thread finishes the disk encryption. After the encryption process is finished, the ransomware creates three files on the desktop and executes the *`ShellExecute`* function with the action `open` to show these files to the user:

Path	Content
${DESKTOP}\RECOVERY.TXT	Ransom message in plaintext
${DESKTOP}\RECOVERY.HTM	Ransom message as web-page
${DESKTOP}\RECOVERY.png	Ransom message as image

At this point, the encryption process is finished, so the ransomware starts the following threads:

- Internet communication thread that is responsible for sending the `Sub=Crypted` command to the C&C server (see **Network and Communication**).
- Remove Shadow Copy thread for the second time (see **Remove shadow copy thread**) if the `OSVERSIONINFO.dwBuildNumber` does not equal the `0xA28`, which refers to Windows XP build number.

At the end, the ransomware performs cleaning operations by removing the executable file from the disk using this command:

```
{WINDOWS_DIRECTORY}\system32\cmd.exe /c DEL ${PATH_TO_EXE}
```

RANDOM FUNCTION

To generate random data, the ransomware uses its own randomization function. When the randomization function is called, the ransomware checks if initialization buffers were previously generated. If not, the ransomware starts the process of buffer initialization. To randomize data in these buffers, it uses the following information:

- Statistical information about the `LanmanWorkstation` workstation, received by calling the *`NetGetStatistics`* function.
- Statistical information about the `LanmanServer` server, received by calling the *`NetGetStatistics`* function.
- Random buffer of size 0x40, generated by calling the *`CryptGenRandom`* function with a `PROV_RSA_FULL` provider.
- Random buffer of size 0x40, generated by calling the *`CryptGenRandom`* function with a `PROV_INTEL_SEC` provider, if such a provider is present in the system.
- Heap chunk information of the ransomware process.
- Process, module, and thread information.
- *`QueryPerformanceCounter`* function results.
- *`GlobalMemoryStatus`* function results.
- Current process' PID.

After the initialization buffers are filled, the ransomware uses them to generate random data. At the same time, it changes the buffers' initial state to avoid generating the same data. It should be emphasized that a SHA1 algorithm is used in this function.

ENCRYPTION KEY GENERATION

The ransomware uses asymmetric cryptography during the key generation process. Asymmetric cryptography protocol is called Elliptic Curve Diffie-Hellman, which is a variety of Diffie-Hellman that uses elliptic curve cryptography. The standardized curve `secp256k1` is used during the key generation process. The following parameters for this elliptic curve are used (see OpenSSL source code):

```
static const struct {
    EC_CURVE_DATA h;
    unsigned char data[0 + 32 * 6];
} _EC_SECG_PRIME_256K1 = {
    {
        NID_X9_62_prime_field, 0, 32, 1
    },
    {
        /* p */
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0xFF, 0xFF, 0xFF, 0xFE, 0xFF, 0xFF, 0xFC, 0x2F,
```

```
        /* a */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        /* b */
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07,
        /* x */
        0x79, 0xBE, 0x66, 0x7E, 0xF9, 0xDC, 0xBB, 0xAC, 0x55, 0xA0, 0x62, 0x95,
        0xCE, 0x87, 0x0B, 0x07, 0x02, 0x9B, 0xFC, 0xDB, 0x2D, 0xCE, 0x28, 0xD9,
        0x59, 0xF2, 0x81, 0x5B, 0x16, 0xF8, 0x17, 0x98,
        /* y */
        0x48, 0x3a, 0xda, 0x77, 0x26, 0xa3, 0xc4, 0x65, 0x5d, 0xa4, 0xfb, 0xfc,
        0x0e, 0x11, 0x08, 0xa8, 0xfd, 0x17, 0xb4, 0x48, 0xa6, 0x85, 0x54, 0x19,
        0x9c, 0x47, 0xd0, 0x8f, 0xfb, 0x10, 0xd4, 0xb8,
        /* order */
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0xFF, 0xFF, 0xFF, 0xFE, 0xBA, 0xAE, 0xDC, 0xE6, 0xAF, 0x48, 0xA0, 0x3B,
        0xBF, 0xD2, 0x5E, 0x8C, 0xD0, 0x36, 0x41, 0x41
    }
};
```

The ransomware generates three sets of ECDH keys:

- Global ECDH keys that are generated only once, when the ransomware installs itself in the system (further `EC_GLOB_xxx`).
- ECDH keys that are used to encrypt global ECDH keys (further `EC_REC_xxx`).
- Session ECDH keys that are generated each time, when the ransomware is restarted (further `EC_SESS_xxx`). These keys are used to encrypt the user's data.

The ransomware uses AES-CBC-128 as a symmetric algorithm to encrypt the user's data.

The full process of key generation is described in the sections below. There are `EC_GENERATOR` states for the `secp256k1` object.

GLOBAL ECDH KEY GENERATION (EC_GLOB)

The ransomware starts with the global ECDH key generation. This process is described step by step:

1. Generate a random number of 0x20 byte size. This number is the global private key `EC_GLOB_PRIV`. It is deleted later to avoid data decryption without ransom payment.
2. Calculate the SHA256 hashsum of `EC_GLOB_PRIV`. This value is called `EC_GLOB_SHA_PRIV`.
3. Calculate the global public key `EC_GLOB_PUB` by performing the following operations:

```
EC_GLOB_PUB = EC_GENERATOR * EC_GLOB_SHA_PRIV
```

4. Calculate the global public BitCoin key by performing the following operations:

```
EC_GLOB_BTN_PUB = EC_GENERATOR * EC_GLOB_PRIV
```

After global keys are generated, the ransomware moves to the process of global recovery key generation. The ransomware calculates the BitCoin address `EC_GLOB_BTN_ADDR` using `EC_GLOB_BTN_PUB` as a public key.

RECOVERY ECDH KEY GENERATION (EC_REC)

The ransomware performs the following steps to generate the global recovery key:

1. Import the server public key `SRV_PUB` by setting hardcoded affine coordinates from the binary.

```
X=0x95667250209D992A05553BDF8CB0E1320B04B2E0FF9177FE88C32CF125FEA249
Y=0x198DCC6F24BCD3D38BE3A62A8F41F92E9D1A4690B54A2D8E4928A933312C643B
```

2. Generate a pair of private/public keys `EC_REC_PRIV` and

```
EC_REC_PUB = EC_GENERATOR * EC_REC_PRIV
```

   `EC_REC_PRIV` is deleted later to avoid data decryption without ransom payment.

3. Generate a global shared secret key using the SHA256 hashsum of:

```
GLOB_SHARED = EC_REC_PRIV * SRV_PUB
```

   This value is called `GLOB_SHA_SHARED`.

4. Encrypt `EC_GLOB_PRIV` using an AES-CBC-128 encryption algorithm. Concatenate `EC_REC_PUB` and encrypted `EC_GLOB_PRIV` to receive global recovery data `G_EC_REC_PUB__AES_EC_GLOB_PRIV`. The code is presented below:

```
def gen_global_rec_data(EC_GLOB_PRIV, GLOB_SHA_SHARED, EC_REC_PUB):
  IV = '\x00' * 16
  ciphertext = aes_encrypt(GLOB_SHA_SHARED, IV, EC_GLOB_PRIV, False)
  return EC_REC_PUB + ciphertext
```

SESSION ECDH KEY GENERATION (EC_SESS)

The ransomware performs these steps to generate a session AES-CBC-128 encryption key (used to encrypt the user's data):

1. Generate a random buffer of 0x20 byte size. The generated buffer is used as a key for the AES-CBC-128 algorithm (further `SESS_AES_KEY`).

2. Generate a pair of private/public keys `EC_SESS_PRIV` and

```
EC_SESS_PUB = EC_GENERATOR * EC_SESS_PRIV
```

`EC_SESS_PRIV` is deleted later to avoid data decryption without ransom payment.

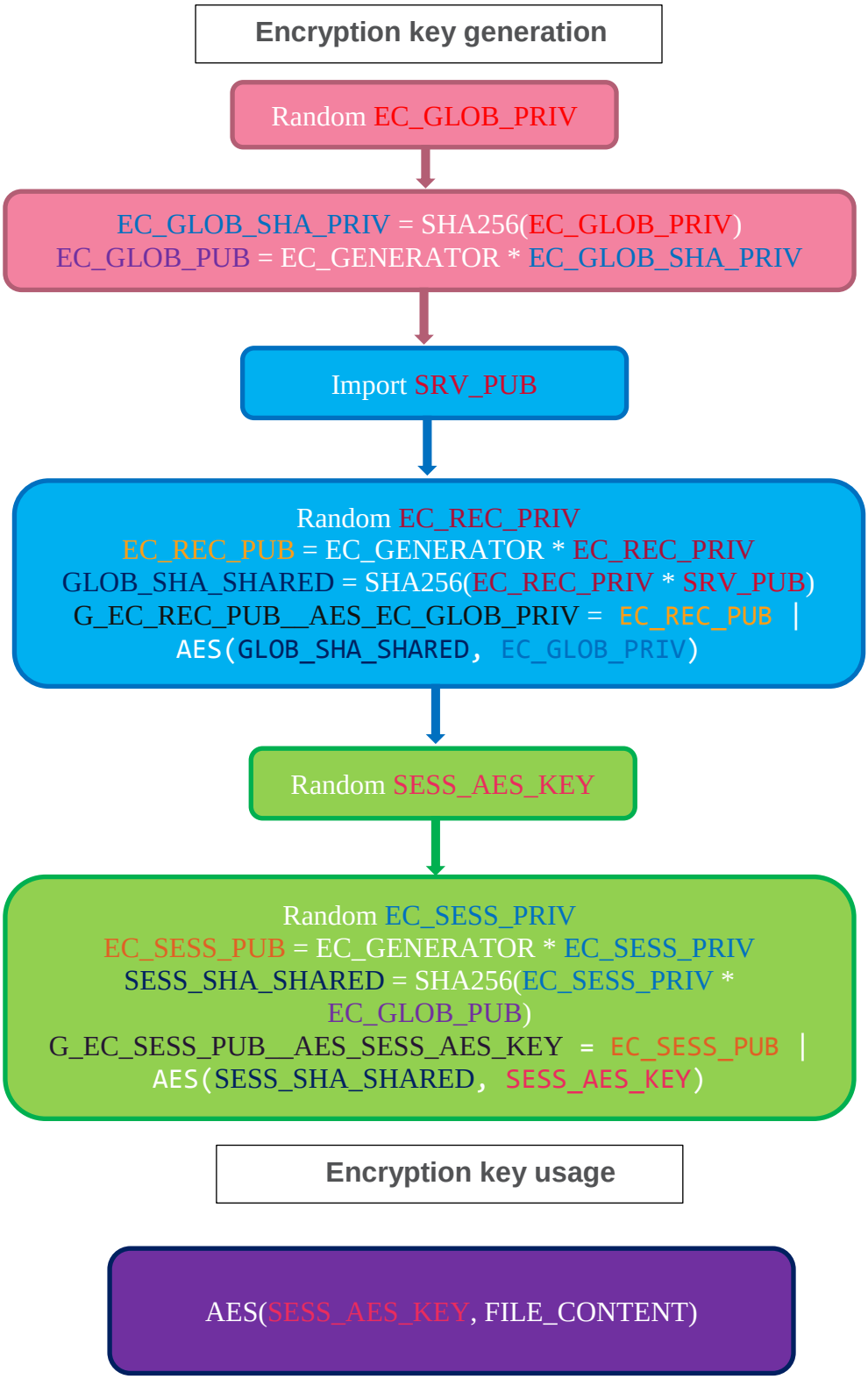
3. Generate a session shared secret key using the SHA256 hashsum of:

```
SESS_SHARED = EC_SESS_PRIV * EC_GLOB_PUB
```

This value is called `SESS_SHA_SHARED`.

4. Encrypt `SESS_AES_KEY` using an AES-CBC-128 encryption algorithm. Concatenate `EC_SESS_PUB` and encrypted `SESS_AES_KEY` to receive the shared recovery data `G_EC_SESS_PUB__AES_SESS_AES_KEY`. The code is presented below:

```
def gen_shared_rec_data(SESS_AES_KEY, SESS_SHA_SHARED, EC_SESS_PUB):
  IV = '\x00' * 16
  ciphertext = aes_encrypt(SESS_SHA_SHARED, IV, SESS_AES_KEY, False)
  return EC_SESS_PUB + ciphertext
```



PROCESS KILLING THREAD

This thread is responsible for enumerating currently running processes in the operating system by using function *EnumProcesses*. The ransomware kills a process if one of the following substrings is found in the process' name:

Substrings	Blacklisted processes
Askmg	Task Manager
Rocex	Process Explorer
Egedi	Registry Editor
sconfi	System Configurator
Cmd	Command Line

REMOVE SHADOW COPY THREAD

This thread is responsible for removing shadow copies on the affected system to avoid restoring encrypted files from backup. The following steps are performed:

1. Disable file system redirection for the current thread, if the *Wow64DisableWow64FsRedirection* function is available.
2. Execute the *ShellExecuteEx* function with one the following commands:

   ```
   wmic.exe shadowcopy delete /noninteractive (verb=runas)
   wmic.exe shadowcopy delete /noninteractive (verb=open)
   ```

   The command depends on membership in the Local Administrator Group.
3. Revert file system redirection for the current thread, if the *Wow64RevertWow64FsRedirection* function is available.

ENCRYPTION THREAD

The ransomware generates a pseudo-random string (further `REC_FILENAME`) of 5 byte length using the *rand* function and the value returned by *GetTickCount* as a seed.

The ransomware gets all the drives in the system by calling the *GetLogicalDriveStrings* function. Drives `A:\` and `B:\` are skipped. Drives that are not of one of the following types are skipped as well:

```
DRIVE_FIXED
DRIVE_REMOTE
DRIVE_REMOVABLE
```

At the same time, all network resources are enumerated recursively using the `WNet` family functions. The ransomware looks for resources with the type `RESOURCETYPE_DISK`. If such a resource is found, its remote name is added to the array of paths to encrypt.

All drives and network disk resources are encrypted according to the rules described below.

The ransomware enumerates the directories recursively and performs the following actions inside each:

1. Encrypt files in the directory (see **Encrypt files**).
2. Check if the directory name belongs to the list presented below. If the name equals one of these, this directory is omitted:

```
CSIDL_COMMON_DESKTOPDIRECTORY
CSIDL_DESKTOPDIRECTORY
CSIDL_CDBURN_AREA
.*Desktop.*
.*${MODULE_NAME}.*
```

The following files are created in the affected directories:

File Name	File Content
"%s\\%s+%s.png" % (${CWD}, ""_RECoVERY_", ${REC_FILENAME})	Image with ransom payment information
"%s\\%s+%s.html" % (${CWD}, ""_RECoVERY_", ${REC_FILENAME})	Web browser page with ransom payment information
"%s\\%s+%s.txt" % (${CWD}, ""_RECoVERY_", ${REC_FILENAME})	Plaintext with ransom payment information

It should be emphasized that the following directories are skipped when performing the described operations:

```
.
..
CSIDL_COMMON_APPDATA
CSIDL_PROGRAM_FILES
CSIDL_WINDOWS
```

If the directory passes all the checks, the ransomware starts the process of file encryption. To check if a file is already encrypted or belongs to the infection meta-information file, the ransomware checks if the filename contains one of the following substrings case-insensitively (such a file is skipped):

```
.mp3
recove
```

The ransomware encrypts only files with specific extensions (see **Appendix G**). The encryption routine is fully described in the next subsection.

ENCRYPT FILES

The ransomware does not encrypt a file if its size is less than 0x20 bytes or larger than 0x13800000 bytes. If the 24th byte of a file equals the 0x04 value, the file is skipped as well.

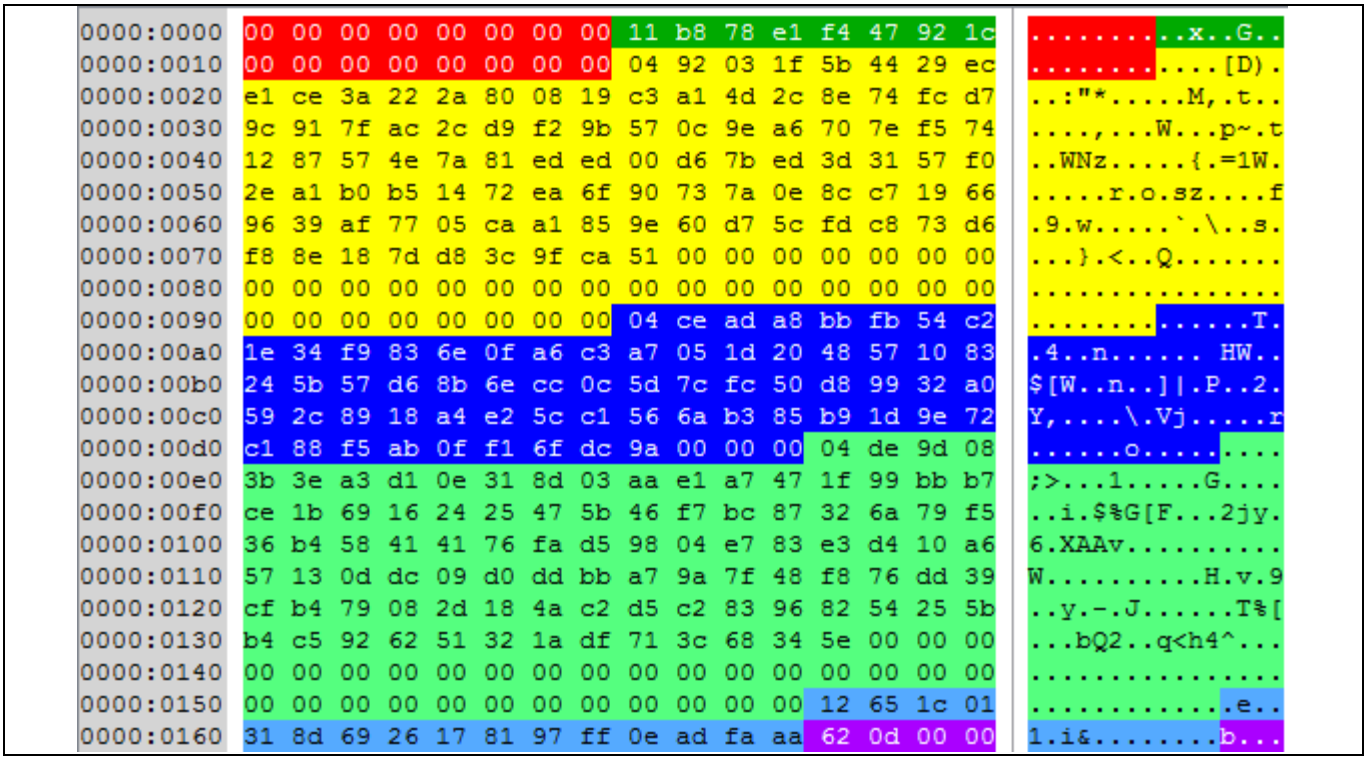
The ransomware encrypts the file content with an AES-CBC-128 encryption algorithm. It uses the previously generated `SESS_AES_KEY` as a key and the hardcoded buffer as an IV. The source code for file encryption routine is presented below:

```
def read_file(fname):
  with open(fname, 'rb') as f:
    return f.read()

def encrypt_file(fname, SESS_AES_KEY):
  IV = '\x12\x65\x1C\x01\x31\x8D\x69\x26\x17\x81\x97\xFF\x0E\xAD\xFA\xAA'
  return aes_encrypt(SESS_AES_KEY, IV, read_file(fname))
```

The ransomware saves specific meta information to the encrypted file, as well as ciphertext. The structure of an encrypted file is presented below:

```
struct enc_file {
  char zero_0[8]; // null bytes
  char inst_id[8]; // instance ID
  char zero_1[8]; // null bytes
  char G_EC_REC_PUB__AES_EC_GLOB_PRIV[0x80]; // Global recovery data
  char EC_GLOB_PUB_point2oct[0x44]; // Global public key converted to octet
string (see: OpenSSL EC_POINT_point2oct())
  char G_EC_SESS_PUB__AES_SESS_AES_KEY[0x80]; // Shared recovery data
  char aes_iv_file_enc[0x10]; // AES IV used during files encryption
  uint32_t orig_file_size; // original file size
  char enc_data[0]; // ciphertext
};
```

The example of encrypted file data is presented below:



```
0000:0000  00 00 00 00 00 00 00 00 11 b8 78 e1 f4 47 92 1c  ..........x..G..
0000:0010  00 00 00 00 00 00 00 00 04 92 03 1f 5b 44 29 ec  ............[D).
0000:0020  e1 ce 3a 22 2a 80 08 19 c3 a1 4d 2c 8e 74 fc d7  ..:"*.....M,.t..
0000:0030  9c 91 7f ac 2c d9 f2 9b 57 0c 9e a6 70 7e f5 74  ....,...W...p~.t
0000:0040  12 87 57 4e 7a 81 ed ed 00 d6 7b ed 3d 31 57 f0  ..WNz.....{.=1W.
0000:0050  2e a1 b0 b5 14 72 ea 6f 90 73 7a 0e 8c c7 19 66  .....r.o.sz....f
0000:0060  96 39 af 77 05 ca a1 85 9e 60 d7 5c fd c8 73 d6  .9.w.....`.\..s.
0000:0070  f8 8e 18 7d d8 3c 9f ca 51 00 00 00 00 00 00 00  ...}.<..Q.......
0000:0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0090  00 00 00 00 00 00 00 00 04 ce ad a8 bb fb 54 c2  ..............T.
0000:00a0  1e 34 f9 83 6e 0f a6 c3 a7 05 1d 20 48 57 10 83  .4..n...... HW..
0000:00b0  24 5b 57 d6 8b 6e cc 0c 5d 7c fc 50 d8 99 32 a0  $[W..n..]|.P..2.
0000:00c0  59 2c 89 18 a4 e2 5c c1 56 6a b3 85 b9 1d 9e 72  Y,....\.Vj.....r
0000:00d0  c1 88 f5 ab 0f f1 6f dc 9a 00 00 00 04 de 9d 08  ......o.........
0000:00e0  3b 3e a3 d1 0e 31 8d 03 aa e1 a7 47 1f 99 bb b7  ;>...1.....G....
0000:00f0  ce 1b 69 16 24 25 47 5b 46 f7 bc 87 32 6a 79 f5  ..i.$%G[F...2jy.
0000:0100  36 b4 58 41 41 76 fa d5 98 04 e7 83 e3 d4 10 a6  6.XAAv..........
0000:0110  57 13 0d dc 09 d0 dd bb a7 9a 7f 48 f8 76 dd 39  W..........H.v.9
0000:0120  cf b4 79 08 2d 18 4a c2 d5 c2 83 96 82 54 25 5b  ..y.-.J......T%[
0000:0130  b4 c5 92 62 51 32 1a df 71 3c 68 34 5e 00 00 00  ...bQ2..q<h4^...
0000:0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0150  00 00 00 00 00 00 00 00 00 00 00 00 12 65 1c 01  .............e..
0000:0160  31 8d 69 26 17 81 97 ff 0e ad fa aa 62 0d 00 00  1.i&........b...
```

The ransomware appends an `.mp3` extension to the encrypted file. If a file with this name already exists, the ransomware removes it, sleeps for 400 milliseconds, and tries to save the encrypted data to a file once again.

NETWORK AND COMMUNICATION

The ransomware communicates with the C&C server to send information about the encryption keys, infected machine and the malware itself. The first communication attempt is performed before the encryption process; the second one is performed after all data on the compromised system is encrypted.

If the ransomware fails to establish communication with one of the C&C servers, it ends the thread without sending any information to the server. It should be emphasized that encryption is performed even with failed communication.

The ransomware sends packets to the C&C server using HTTP POST requests on the default TCP/80 port, with the following header values:

```
Accept: */*
Content-Type: application/x-www-form-urlencoded
UserAgent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch; rv:11.0)
like Gecko
```

CLIENT REQUEST

The HTTP POST request contains the following data:

```
Sub=%s&dh=%s&addr=%s&size=%lld&version=%s&OS=%ld&ID=%d&inst_id=%X%X%X%X%X%X%X%X
```

Each parameter is described below.

Field Name	Description
Sub	Command type
dh	Hexlified ECDH Global recovery key `G_EC_REC_PUB__AES_EC_GLOB_PRIV`
addr	Machine-specific bitcoin address, where ransom should be paid
size	Size of encrypted files in total (Mb)
version	Malware version (3.0.1 in researched sample)
OS	Operating system build number
ID	Part of malware image file checksum
inst_id	Machine-specific unique ID

When sending the initial packet to the C&C server, the ransomware uses the `Sub=Ping` command type. When sending the packet after whole disk encryption, the ransomware uses the `Sub=Crypted` command type.

The ransomware encrypts and encodes the data when it communicates with the C&C. It calculates the SHA256 checksum of the decrypted hardcoded string `032453242372394857237945324985 7`. The received hashsum is used as a key for the AES-CBC-128 encryption algorithm. The full code is presented below:

```
from binascii import hexlify
from hashlib import sha256

def inet_encrypt(data):
  data += '\x00'
  IV = '\xFF\xFF\xAA\xAA\x00\x00\xBE\xEF\xDE\xAD\x00\x00\xBE\xFF\xFF\xFF'
  key = sha256('032453242372394857237945324985 7')

  return aes_encrypt(key, IV, data)

def inet_http_prepare(data):
  return 'data=%s' % hexlify(inet_encrypt(data))
```

The ransomware uses one of the hardcoded C&C servers (see **Indicators of Compromise)** to notify about the current state.

It should be emphasized that there is a possibility to decrypt the packet sent to the C&C server and determine if it belongs to the TeslaCrypt process, thus breaking its execution (see **Appendix C**).

SERVER RESPONSE

The ransomware assumes that the C&C handled the request correctly if the response contains the `INSERTED` string.

DECRYPTION PROCESS

This section presents the details of the decryption process that may be performed after ransom payment:

1. The infected computer has the following meta information about encryption process: `G_EC_SESS_PUB__AES_SESS_AES_KEY`, `EC_GLOB_PUB`, `G_EC_REC_PUB__AES_EC_GLOB_PRIV`.

2. As `G_EC_REC_PUB__AES_EC_GLOB_PRIV` was previously transferred to the C&C server, the decrypter can extract the `EC_REC_PUB` key and calculate:

   ```
   GLOB_SHARED = SRV_PRIV * EC_REC_PUB
   ```

   It then calculates the SHA256 hashsum of the key `GLOB_SHA_SHARED`.

3. It can decrypt `EC_GLOB_PRIV` using an AES-CBC-128 algorithm with `GLOB_SHA_SHARED` as a key, and then calculate the SHA256 hashsum of `EC_GLOB_SHA_PRIV`. It sends this key with a local decrypter to the infected machine.

4. The local decrypter on the infected machine enumerates all encrypted files and uses `EC_GLOB_SHA_PRIV` to restore the session shared key:

   ```
   SESS_SHARED = EC_SESS_PUB * EC_GLOB_SHA_PRIV
   ```

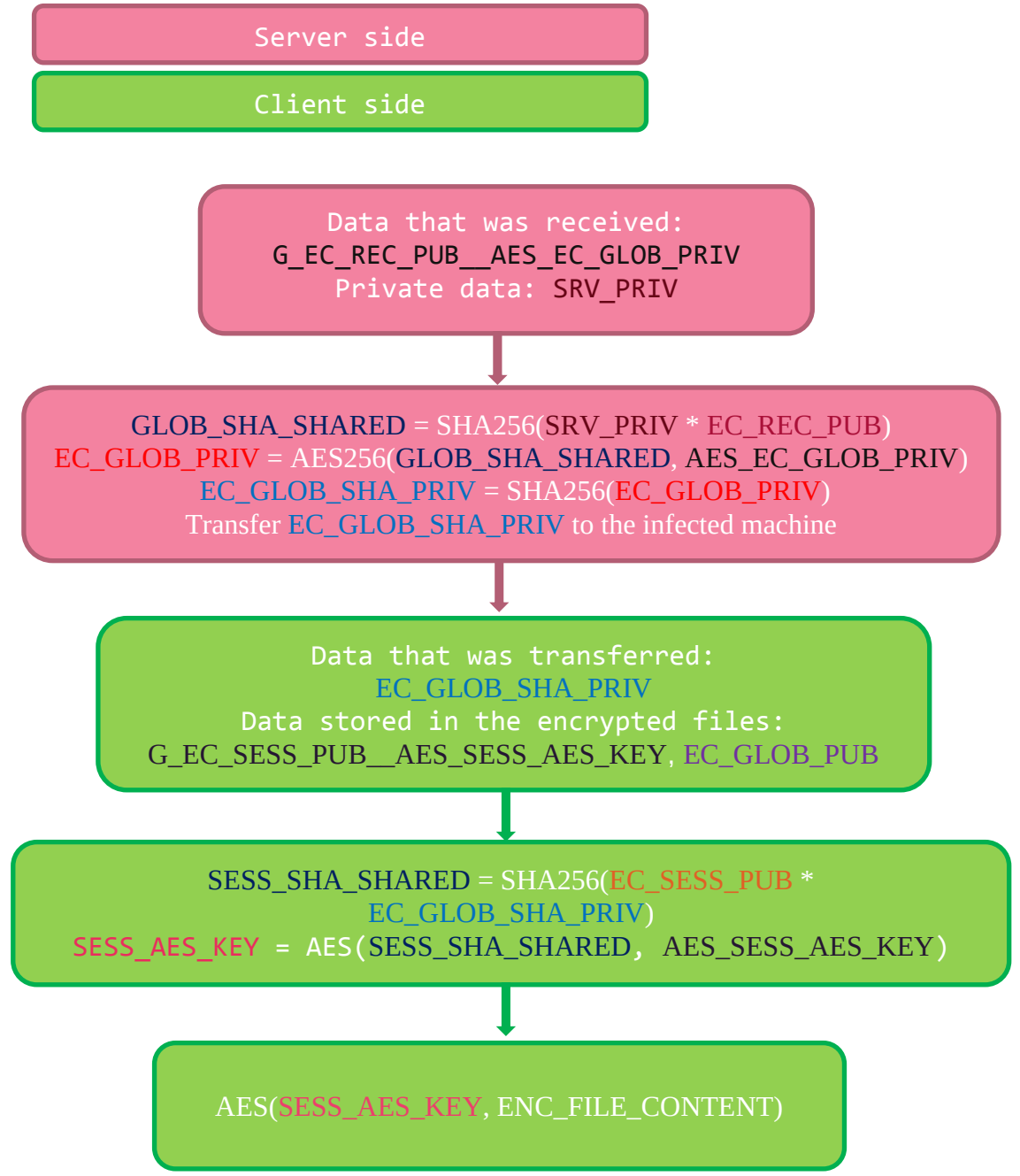
   It then calculates the SHA256 hashsum of `SESS_SHA_SHARED`. `EC_SESS_PUB` is taken from session recovery data `G_EC_SESS_PUB__AES_SESS_AES_KEY` that is saved in the encrypted file header.

5. The local decrypter on the infected machine decrypts `SESS_AES_KEY` key using an AES-CBC-128 algorithm with the key `SESS_SHA_SHARED`.

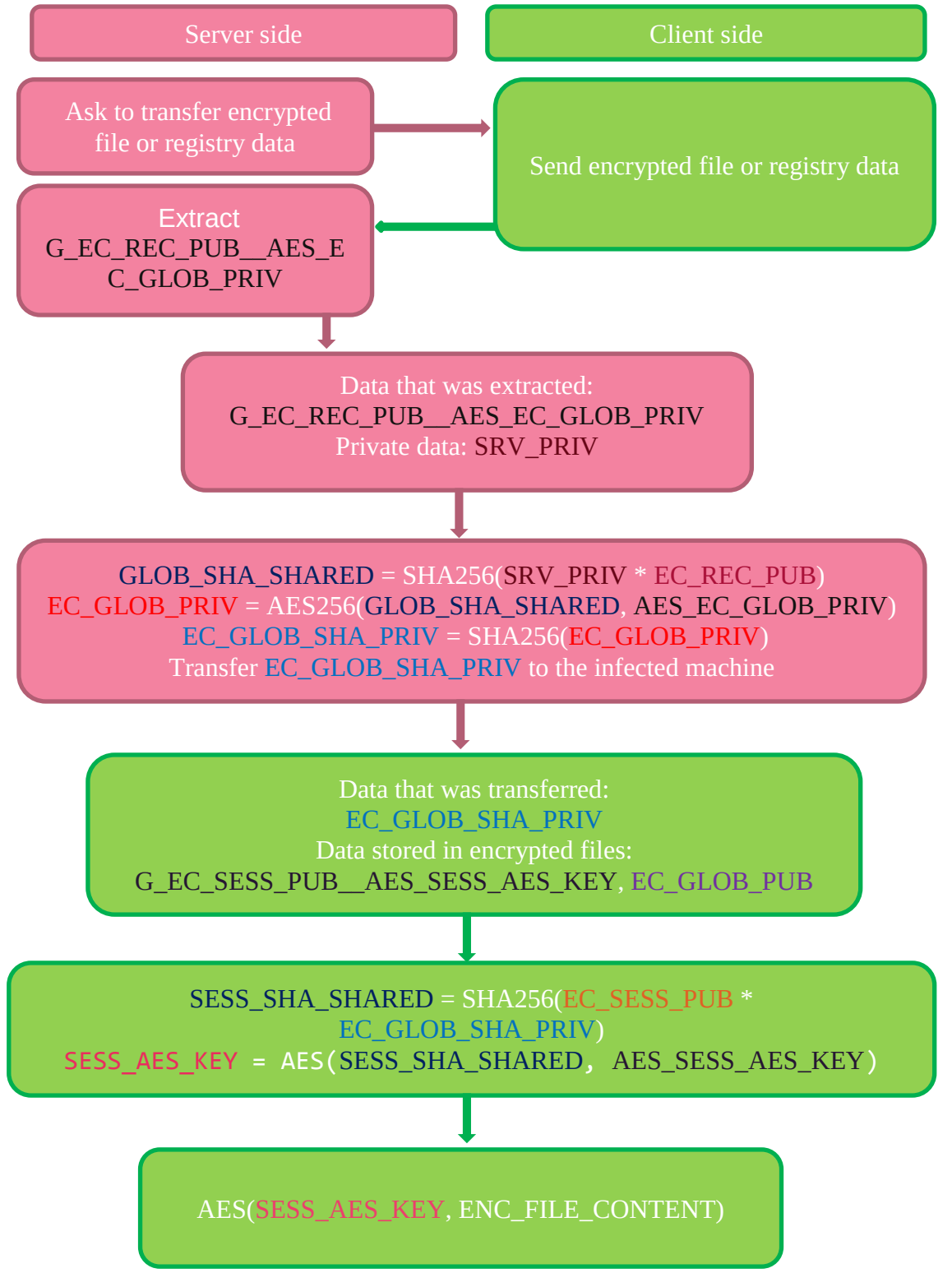
6. As a last step, the decrypter restores the files using an AES-CBC-128 algorithm with the `SESS_AES_KEY` key and the IV `\x12\x65\x1C\x01\x31\x8D\x69\x26\x17\x81\x97\xFF\x0E\xAD\xFA\xAA` that is stored in the encrypted file header.

If the infected machine was not able to communicate with the C&C server, the `G_EC_REC_PUB__AES_EC_GLOB_PRIV` data was not sent. To decrypt data without sending the `SRV_PRIV` key to the infected machine, the cybercriminals may ask the user to send one of the encrypted files. As the encrypted file header contains the `G_EC_SESS_PUB__AES_SESS_AES_KEY`, `EC_GLOB_PUB`, and `G_EC_REC_PUB__AES_EC_GLOB_PRIV`, the cybercriminals can then perform the same actions as described previously.

Decryption process when communication with C&C succeeded



Decryption process when communication with C&C failed



SUMMARY

Based on the information we presented, it seems it is impossible to decrypt data without paying a ransom, as we have to solve the discrete logarithm problem. As keys are generated using a self-implemented randomization function (that still contains unpredictable data), cryptographic attacks on that function may be successful.

INDICATORS OF COMPROMISE

Static indicators

- Presence of files with the following SHA256 checksums in the filesystem:

```
3e730bb707b5c9d45e10dc500f0281a50e58badbbdfa6f5038e077c4ace125d4
366d1629a83acad94ced95c4b782ec00a2cc0096598b6824421aed1859d37c1a
7097913d473590c8fc507d8b8b6eaee8cd9db77888ebb14fc193eafeac039d7a
79743fb8f3dbef7b6066ed030ac488fe63038708cd227e7f52f4411540f2d5a4
8008a7f9920f8d61f2295ab82a9a3efac0c3a1c466213fe43afe7407bdab03d7
f2a7d3bd2430d3d8b56d04d2d56a67cb57452e5bacf85ffe37433c73cee6d40d
7b709122af3222d4e533ade64ab9bef3f79c6aa97370f876af5a6b90a834c7fe
```

Dynamic indicators

- Communication with the following C&C servers:

```
http://ricardomendezabogado.com/components/com_imageshow/wstr.php
http://opravnatramvaji.cz/modules/mod_search/wstr.php
http://gianservizi.it/wp-content/uploads/wstr.php
http://ptlchemicaltrading.com/images/gallery/wstr.php
http://3m3q.org/wstr.php
http://suratjualan.com/copywriting.my/image/wstr.php
http://imagescroll.com/cgi-bin/Templates/bstr.php
http://music.mbsaeger.com/music/Glee/bstr.php
http://stacon.eu/bstr.php
http://surrogacyandadoption.com/bstr.php
http://worldisonefamily.info/zz/libraries/bstr.php
http://biocarbon.com.ec/wp-content/uploads/bstr.php
```

- Presence of the following mutex in the system:

```
8765-123rvr4
```

- Importing a specific plaintext decryption key (see **Appendix B**):

```
kasdfgh283
```

- HTTP specific packets during the network communication (see **Appendix C**).
- Presence of the following registry keys:

Registry Key	Subkey	Value
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run	String of length 0x0C that consists only of lower letters	"C:\WINDOWS\system32\cmd.exe /c start "" "${PATH_TO_EXE}""
HKEY_USERS\S-1-5-18\Software\zzzsys	ID	REG:BINARY of size 8 bytes
HKEY_CURRENT_USER\Software\zzzsys	ID	REG:BINARY of size 8 bytes
HKEY_USERS\S-1-5-18\Software\zzzsys\hexlified_ID	data	REG:BINARY of size 0x100 bytes
HKEY_CURRENT_USER\Software\zzzsys\hexlified_ID	data	REG:BINARY of size 0x100 bytes

- Presence of the following files in the system:

```
CSIDL_MYDOCUMENTS\recover_file_${random_0x09_lower}.txt
${DESKTOP}\RECOVERY.TXT
${DESKTOP}\RECOVERY.HTM
${DESKTOP}\RECOVERY.png
*\_RECoVERY_+${random_0x05_lower}.txt
*\_RECoVERY_+${random_0x05_lower}.html
*\_RECoVERY_+${random_0x05_lower}.png
```

APPENDIX A: AES ENCRYPTION ROUTINE

```
from Crypto.Cipher import AES

def aes_encrypt(sc, iv, pt, fp=True):
  bs = AES.block_size
  cipher = AES.new(sc, AES.MODE_CBC, iv)
  if len(pt) % bs:
    pad_len = bs - (len(pt) % bs)
    pt += chr(pad_len) * pad_len
  elif fp:
    pt += chr(bs) * bs
  return cipher.encrypt(pt)
```

APPENDIX B: DISK ENCRYPTION PREVENTION BY IMPORTED CRYPTOGRAPHIC KEY

TeslaCrypt uses the following key to decrypt internal data:

```
kasdfgh283
```

This key is imported with the WinAPI function:

```
BOOL WINAPI CryptImportKey(
  _In_  HCRYPTPROV hProv,
  _In_  BYTE       *pbData,
  _In_  DWORD      dwDataLen,
  _In_  HCRYPTKEY  hPubKey,
  _In_  DWORD      dwFlags,
  _Out_ HCRYPTKEY  *phKey
);
```

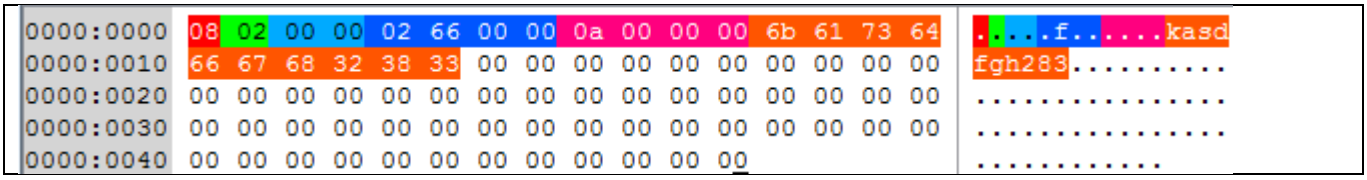
The ransomware uses the following parameters (important ones are colored in red):

```
CryptImportKey(
  AnyCryptoProv,
  PublicKey,
  0x4c,
  0,
  0,
  AnyAddr
);
```

`PublicKey` represents the following *`PublicKeyBlob`* structure:

```
typedef struct {
    BLOBHEADER blob_hdr;
    ULONG cbPublicKey;
    BYTE PublicKey[1]
} PublicKeyBlob;

typedef struct _PUBLICKEYSTRUC {
  BYTE   bType;
  BYTE   bVersion;
  WORD   reserved;
  ALG_ID aiKeyAlg;
} BLOBHEADER, PUBLICKEYSTRUC;
```

When importing the following configuration of `PublicKey`:



As the same parameters are passed to the *`CryptImportKey`*, the following steps can be performed to detect TeslaCrypt processes:

1. Intercept the *`CryptImportKey`* call and check if `dwDataLen`, `hPubKey`, `dwFlags` contain the values specified above. If so, go to the next step. Otherwise, continue function normal execution.
2. Check if the `pbData` structure contains the data presented above. If the data is the same, we can assume that the calling process is TeslaCrypt and break its execution.

APPENDIX C: DISK ENCRYPTION PREVENTION BY NETWORK COMMUNICATION SIGNATURES

When communicating with the C&C servers, the ransomware encrypts the sent data using an AES-CBC-128 algorithm. The following AES-related data is used:

```
IV = '\xFF\xFF\xAA\xAA\x00\x00\xBE\xEF\xDE\xAD\x00\x00\xBE\xFF\xFF\xFF'
key = SHA256('032453242372394857237945324985 7')
```

The encrypted data is hexlified and used as a value for the `data` key:

```
http_body = 'data=%s' % hexlify(data_encrypted)
```

To detect the presence of the TeslaCrypt ransomware, perform the following operations with the HTTP packet:

1. Check if User-Agent equals:

```
UserAgent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko
```

   If so, go to the next step.

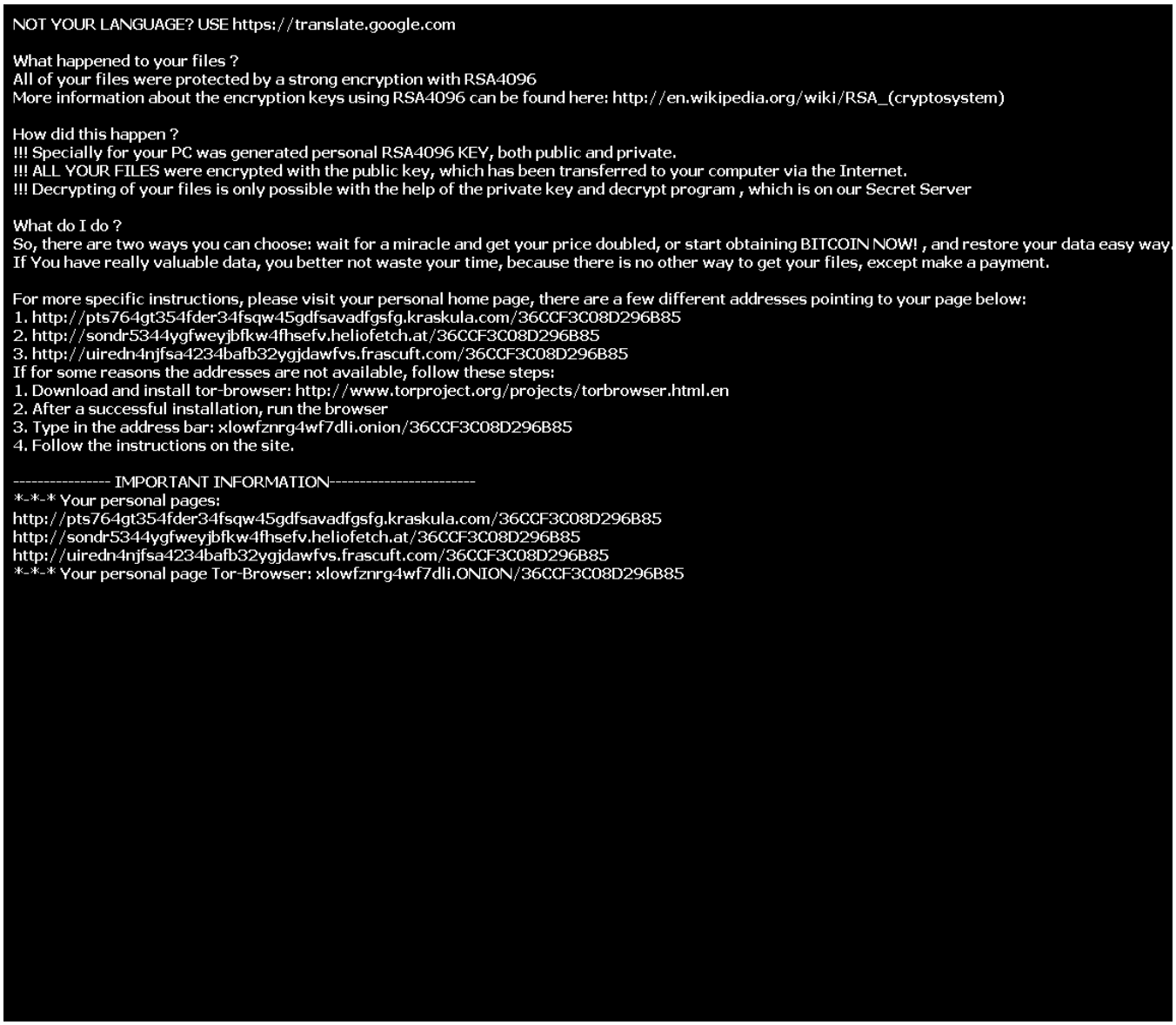
2. The code that is responsible for checking if the HTTP packet is likely sent by TeslaCrypt is presented below:

```
from binascii import unhexlify
from Crypto.Cipher import AES
from hashlib import sha256
from re import compile


def aes_decrypt(sc, iv, ct):
  cipher = AES.new(sc, AES.MODE_CBC, iv)
  return cipher.decrypt(ct)


def inet_http_body_decrypt(http_body):
  IV =
'\xFF\xFF\xAA\xAA\x00\x00\xBE\xEF\xDE\xAD\x00\x00\xBE\xFF\xFF\xFF'
  key = sha256('03245324237239485723794532498 57').digest()

  http_body_data = None
  http_body_delim = 'data='
  if not http_body.startswith(http_body_delim):
    return None

  http_body_data = http_body[len(http_body_delim):]
  try:
    http_body_data = unhexlify(http_body_data)
  except Exception:
    return None

  try:
    http_body_dec = aes_decrypt(key, IV, http_body_data)
  except Exception:
    return None

  pb = ord(http_body_dec[-1])
  http_body_dec = http_body_dec[:-pb]
```

```
  if ord(http_body_dec[-1]):
    return None

  return http_body_dec[:-1]


def inet_http_is_tesla(http):
  tc_ua = 'UserAgent: Mozilla/5.0 (Windows NT 6.3; WOW64;
Trident/7.0; Touch; rv:11.0) like Gecko'
  # Just for PoC
  tc_body =
compile('^Sub=.*&dh=.*&addr=.*&size=.*&version=.*&OS=.*&ID=.*&inst_
id=.*$')

  if http['User-Agent'] != tc_ua:
    return False

  http_body_dec = inet_http_body_decrypt(http['body'])
  if not http_body_dec:
    return False

  print '[+] Decrypted HTTP BODY: %s' % http_body_dec

  return tc_body.match(http_body_dec) is not None
```

If such a HTTP packet was caught, we can assume that it was sent by a TeslaCrypt process. Find this process and break its execution.

APPENDIX D: RANSOM MESSAGE IMAGE



APPENDIX E: RANSOM MESSAGE WEB PAGE

NOT YOUR LANGUAGE? USE Google Translate

What happened to your files?

All of your files were protected by a strong encryption with RSA4096

More information about the encryption RSA4096 can be found http://en.wikipedia.org/wiki/RSA_(cryptosystem)

What does this mean?

This means that the structure and data within your files have been irrevocably changed, you will not be able work with them, read them or see them, it is the same thing as losing them forever, but with our help, you can restore them

How did this happen?

Especially for you, on our SERVER was generated the secret key

All your files were encrypted with the public key, which has been transferred to your computer via the Internet.

Decrypting of YOUR FILES is only possible with the help of the private key and decrypt program which is on our Secret Server!!!

What do I do?

Alas, if you do not take the necessary measures for the specified time then the conditions for obtaining the private key will be changed

If you really need your data, then we suggest you do not waste valuable time searching for other solutions becausen they do not exist.

For more specific instructions, please visit your personal home page, there are a few different addresses pointing to your page below:

1 - http://pts764gt354fder34fsqw45gdfsavadfgsfg.kraskula.com/C62DFE7EB4BD25
2 - http://sondr5344ygfweyjbfkw4fhsefv.heliofetch.at/C62DFE7EB4BD25
3 - http://uiredn4njfsa4234bafb32ygjdawfvs.frascuft.com/C62DFE7EB4BD25

If for some reasons the addresses are not available, follow these steps:

1 - Download and install tor-browser: http://www.torproject.org/projects/torbrowser.html.en
2 - After a successful installation, run the browser and wait for initialization.
3 - Type in the tor-browser address bar: xlowfznrg4wf7dli.onion/C62DFE7EB4BD25
4 - Follow the instructions on the site.

!!! IMPORTANT INFORMATION:

Your Personal PAGES:
http://pts764gt354fder34fsqw45gdfsavadfgsfg.kraskula.com/C62DFE7EB4BD25
http://sondr5344ygfweyjbfkw4fhsefv.heliofetch.at/C62DFE7EB4BD25
http://uiredn4njfsa4234bafb32ygjdawfvs.frascuft.com/C62DFE7EB4BD25
Your Personal TOR-Browser page : xlowfznrg4wf7dli.onion/C62DFE7EB4BD25
Your personal ID (if you open the site directly): C62DFE7EB4BD25

APPENDIX F: RANSOM MESSAGE TEXT

```
NOT YOUR LANGUAGE? USE https://translate.google.com

What happened to your files ?
All of your files were protected by a strong encryption with RSA4096
More information about the encryption keys using RSA4096 can be found here:
http://en.wikipedia.org/wiki/RSA_(cryptosystem)

How did this happen ?
!!! Specially for your PC was generated personal RSA4096 KEY, both public and
private.
!!! ALL YOUR FILES were encrypted with the public key, which has been transferred
to your computer via the Internet.
!!! Decrypting of your files is only possible with the help of the private key and
decrypt program , which is on our Secret Server

What do I do ?
So, there are two ways you can choose: wait for a miracle and get your price
doubled, or start obtaining BITCOIN NOW! , and restore your data easy way.
If You have really valuable data, you better not waste your time, because there is
no other way to get your files, except make a payment.

For more specific instructions, please visit your personal home page, there are a
few different addresses pointing to your page below:
1. http://pts764gt354fder34fsqw45gdfsavadfgsfg.kraskula.com/E406831E97DB790
2. http://sondr5344ygfweyjbfkw4fhsefv.heliofetch.at/E406831E97DB790
3. http://uiredn4njfsa4234bafb32ygjdawfvs.frascuft.com/E406831E97DB790
If for some reasons the addresses are not available, follow these steps:
1. Download and install tor-browser:
http://www.torproject.org/projects/torbrowser.html.en
2. After a successful installation, run the browser
3. Type in the address bar: xlowfznrg4wf7dli.onion/E406831E97DB790
4. Follow the instructions on the site.

---------------- IMPORTANT INFORMATION-----------------------
*-*-* Your personal pages:
http://pts764gt354fder34fsqw45gdfsavadfgsfg.kraskula.com/E406831E97DB790
http://sondr5344ygfweyjbfkw4fhsefv.heliofetch.at/E406831E97DB790
http://uiredn4njfsa4234bafb32ygjdawfvs.frascuft.com/E406831E97DB790
*-*-* Your personal page Tor-Browser: xlowfznrg4wf7dli.ONION/E406831E97DB790
```

APPENDIX G: FILE EXTENSIONS TO ENCRYPT

.r3d	.indd	.w3x	.xlsx	.bsa
.ptx	.cdr	.rim	.xlsm	.ltx
.pef	.erf	.psk	.xlsb	.forge
.srw	.bar	.tor	.xlk	.asset
.x3f	.hkx	.vpk	.ppt	.litemod
.der	.raf	.iwd	.pptx	.iwi
.cer	.rofl	.kf	.pptm	.das
.crt	.dba	.mlx	.mdb	.upk
.pem	.db0	.fpk	.accdb	.d3dbsp
.odt	.kdb	.dazip	.pst	.csv
.ods	.mpqge	.vtf	.dwg	.wmv
.odp	.vfs0	.vcf	.xf	.avi
.odm	.mcmeta	.esm	.dxg	.wma
.odc	.m2	.blob	.wpd	.m4a
.odb	.lrf	.dmp	.rtf	.rar
.doc	.vpp_pc	.layout	.wb2	.7z
.docx	.ff	.menu	.pfx	.mp4
.kdc	.cfr	.ncf	.p12	.sql
.mef	.snx	.sid	.p7b	.bak
.mrwref	.lvl	.sis	.p7c	.tiff
.nrw	.arch00	.ztmp	.txt	
.orf	.ntl	.vdf	.jpeg	
.raw	.fsh	.mov	.png	
.rwl	.itdb	.fos	.rb	
.rw2	.itl	.sb	.css	
.mdf	.mddata	.itm	.js	
.dbf	.sidd	.wmo	.flv	
.psd	.sidn	.itm	.m3u	
.pdd	.bkf	.map	.py	
.pdf	.qic	.wmo	.desc	
.eps	.bkp	.sb	.xxx	
.jpg	.bc7	.svg	.litesql	
.jpe	.bc6	.cas	wallet	
.dng	.pkpass	.gho	.big	
.3fr	.tax	.syncdb	.pak	
.arw	.gdb	.mdbackup	.rgss3a	
.srf	.qdf	.hkdb	.epk	
.sr2	.t12	.hplg	.bik	
.bay	.t13	.hvpl	.slm	
.crw	.ibank	.icxs	.lbf	
.cr2	.sum	.docm	.sav	
.dcr	.sie	.wps	.re4	
.ai	.zip	.xls	.apk	