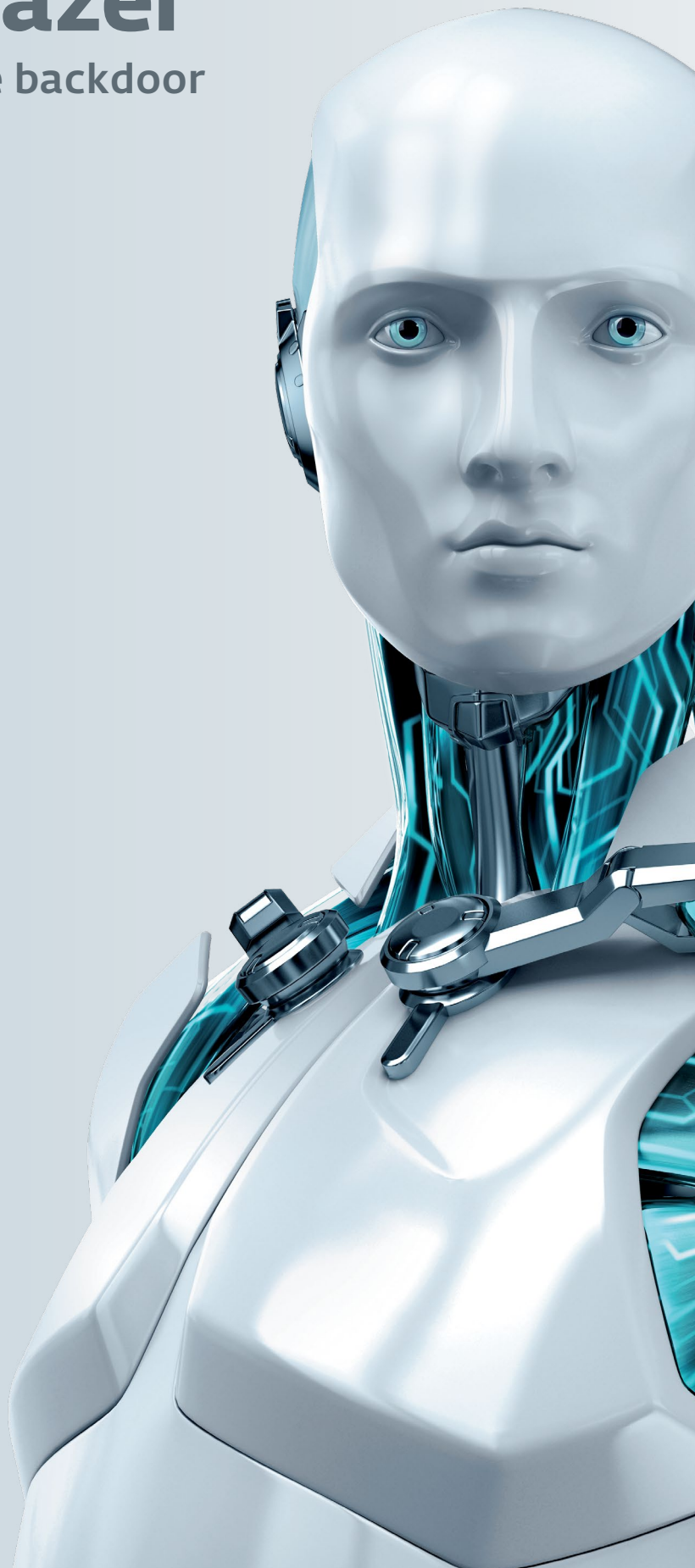
Gazing at Gazer

Turla's new second stage backdoor

August 2017



ENJOY SAFER TECHNOLOGY™

Gazing at Gazer

Turla's new second stage backdoor

August 2017

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Herein we release our analysis of a previously undocumented backdoor that has been targeted against embassies and consulates around the world leads us to attribute it, with high confidence, to the Turla group. Turla is a notorious group that has been targeting governments, government officials and diplomats for years. They are known to run watering hole and spearphishing campaigns to better pinpoint their targets. Although this backdoor has been actively deployed since at least 2016, it has not been documented anywhere. Based on strings found in the samples we analyzed, we have named this backdoor "Gazer".

Recently, the Turla APT group has seen extensive news coverage surrounding its campaigns, something we haven't seen for a long time. The Intercept reported that there exists a 2011 presentation by Canada's Communication Security Establishment (CSE) outlining the errors made by the Turla operators during their operations even though the tools they use are quite advanced. The codename for Turla APT group in this presentation is MAKERSMARK. Gazer is, similar to its siblings in the Turla family, using advanced methods to spy and persist on its targets. This whitepaper highlights the campaigns in which Gazer was used and also contains a technical analysis of its functionalities.

SUMMARY

Based on our research and telemetry on the different campaigns where Gazer was used,we believe that Southeastern Europe as well as countries in the former Soviet Union Republichas recently been the main target. The witnessed techniques, tactics and procedures (TTPs) are in-line with what we usuallysee in Turla's operation: a first stage backdoor, such as Skipper, likely delivered through spearphishingfollowed by the appearance on the compromised system of a second stage backdoor, Gazerin this case.

Although we could not find irrefutable evidence that this backdoor is truly another tool in Turla's arsenal, several clues lead us to believe that this is indeed the case. First, their targets are in line with Turla's traditional targets: Ministries of Foreign Affairs (MFAs) and embassies. Second, the modus operandi of spearphishing, followed by a first stage backdoor and a second stage, stealthier backdoor is what has been seen over and over again. Skipper, which has been linked to Turla in the past, was found alongside Gazer in most cases we investigated. Finally, there are many similarities between Gazer and other second stage backdoors used by the Turla group such as Carbon and Kazuar.

As usual, the Turla APT group makes an extra effort to avoid detection by wiping files securely, changing the strings and randomizing what could be simple markers through the different backdoor versions. In the most recent version we have found, Gazer authors modified most of the strings and inserted "video-game-related" sentences throughout the code. An example of such a string is depicted in Figure 1.

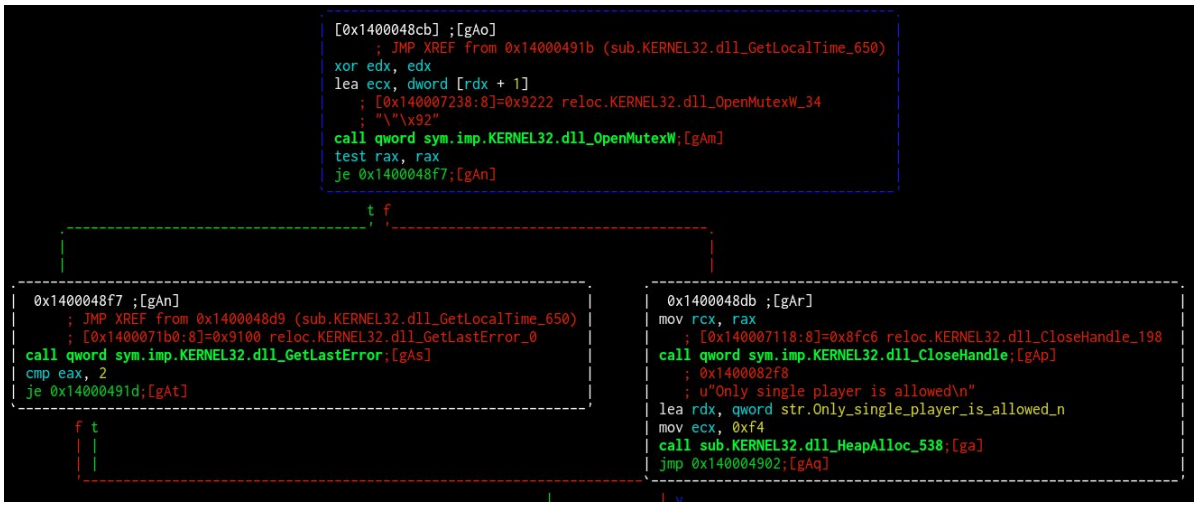


Figure 1. **Turla author's sense of humor**

SIMILARITIES WITH OTHER TURLA TOOLS

Gazer is written in C++ and shares several similarities with other malware from the Turla APT family. Indeed, Gazer, Carbon and Kazuar can receive encrypted tasks from a C&C server, which can be executed either by the infected machine or by another machine on the network. They all use an encrypted container to store the malware's components and configuration and they also log their actions in a file.

The list of C&C servers is encrypted and embedded in Gazer's PE resources. They are all compromised, legitimate websites (that mostly use the WordPress CMS) that act as a first layer proxy. This is also a common tactic for the Turla APT group.

Another interesting linkage is that one of the C&C servers embedded in a Gazer sample was known to be used in a JScript backdoor documented by Kaspersky as Kopiluak.

Last but not least, these three malware families (Gazer, Carbon and Kazuar) have a similar list of processes that may be employed as a target to inject the module used to communicate with the C&C server embedded in the binary. The resource containing this list can change from one sample to another, it is likely tailored to what is installed on the system (for example, on some samples, the process name "safari.exe" can appear on the list).

CUSTOM ENCRYPTION

Gazer's authors make extensive use of encryption. They don't use the Windows Crypto API and don't seem to use any public library. It looks as if they are using their own library for 3DES and RSA.

The RSA keys embedded in the resources contains the attacker's public key which is used to encrypt the data sent to the C&C server, and a private key to decrypt resources embedded in its binaries. These keys are unique in each sample.

These resources are structured in the same way as RSA from OpenSSL, but these values (p, q, etc.) are computed by the custom implementation of Gazer's authors.

For 3DES, the IV and a static key are hardcoded and are the same in all samples. This 3DES key is randomly generated and XORed with the static key. The random data used to XOR the static key is prepended to the logfile header. This key is then used in the regular 3DES algorithm.

GLOBAL ARCHITECTURE

In this section, we will describe in detail each component of Gazer.

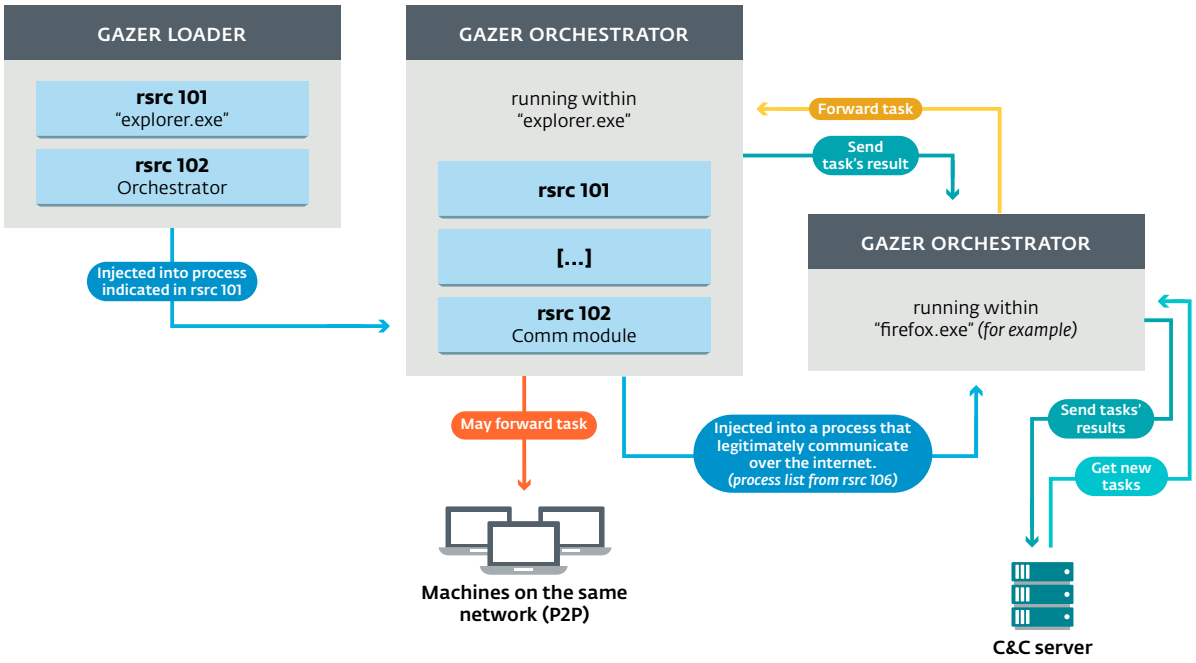


Figure 2. **Gazer architecture**

Loader

The loader is the first component of the malware to be executed on the system. Two resources are stored unencrypted in the binary:

- 101: the process name to inject the orchestrator into[1]
- 102: the orchestrator

The following mutex is created to ensure that only a single instance of the malware is running:

```
{531511FA-190D-5D85-8A4A-279F2F592CC7}
```

Named pipe generation

To establish a communication channel between Gazer components, a named pipe is initiated.

The named pipe is generated from this string:

```
\\\\.\\pipe\\Winsock2\\CatalogChangeListener-FFFF-F
```

1 Note that in all samples we have analyzed the process name is "explorer.exe"

The pattern "FFFF-F" is replaced with values computed from the security identifier (SID) of the current user and the current timestamp.

Let's take for example the current date as: "2017/04/24" and the SID: `"S-1-5-21-84813077-3085987743-2510664113-1000"`.

To generate the pattern at the end of the named pipe, some arithmetic is performed:

```
time = SystemTime.wDay * Systemtime.wMonth * SystemTime.wYear =
24 * 04 * 2017 = 0x2f460
x_sid = (1 * 21 * 84813077 * 3085987743 * 2510664113 * 1000) &
0xFFFFFFFF = 0xefa252d8

((time >> 20) + (time & 0xFFF) + ((time >> 12) & 0xFFF)) % 0xFF =
0x93
((x_sid >> 20) + (x_sid & 0xFFF) + ((x_sid >> 12) & 0xFFF)) % 0xFF =
0x13

((time * x_sid >> 24) + (uint8_t)(time * x_sid) + ((uint16_t)(time *
x_sid) >> 8) + (uint8_t)(time * x_sid >> 16)) % 0xf) = 0xa
```

In this case, the named pipe will be:

`\\\\.\\pipe\\Winsock2\\CatalogChangeListener-9313-a`

If the current user's SID cannot be retrieved, the named pipe `\\\\.\\pipe\\\Winsock2\\CatalogChangeListener-FFFE-D` will be used by default.

Code injection through thread hijacking

A not-so-common trick is used in order to inject the orchestrator into a remote process. Indeed, a running thread from the remote process is hijacked in order to run shellcode that will execute the communication module entry point.

- The whole module and shellcode are copied into the remote process;
- the function `ZwQuerySystemInformation` is used to retrieve the total number of the running threads in the targeted process;
- the following operations are attempted on each of those threads:
  - the thread is suspended with the `OpenThread/SuspendThread` functions;
  - the thread context is retrieved using `GetThreadContext`;
  - the context's instruction pointer is saved and modified to point to the shellcode (through `SetThreadContext`);
  - the thread is resumed using `ResumeThread`.
- if one of the previous operations fails, the thread is resumed and the same actions are attempted on another thread.

```
launcher:
    push rax
    sub rsp, 38h
    movabs rax, 5D20092               ; @ end of payload
    mov qword ptr ss:[rsp+28], rax    ; lpThreadId
    mov qword ptr ss:[rsp+20], 0      ; dwCreationFlags
    xor r9d, r9d                      ; lpParameter
    movabs r8, 5D20046                ; lpStartAddress => @payload
    xor edx, edx                      ; dwStackSize = 0
    xor ecx, ecx                      ; lpThreadAttributes = NULL
    call qword ptr ds:[CreateThread]
    movabs rax, 90A7FACE90A7FACE      ; replaced by the saved
                                      instruction pointer from
                                      thread context ;)
    add rsp, 38h
    xchg qword ptr ss:[rsp], rax
    ret

payload:
    sub rsp, 28
    movabs r8, 5D20096
    mov edx, 1
    movabs rcx, 4000000000000000
    call qword ptr ds: [DllEntryPoint]
    xor ecx, ecx
    call ExitThread
    int 3
    xxxx;  @DllEntryPoint
    xxxx ; @CreateThread
    xxxx; @ExitThread
    xxxx
    xxxx
    xxxx
    xxxx ; TID
```

The shellcode is just a loader that will execute the module entry point in a new thread.

Persistence

The loader sends binary data through the named pipe to the orchestrator. This blob contains:

- a command ID (2): `CMC_TAKE_LOADER_BODY`
- the loader path file
- the loader PE

Once this message is received by the orchestrator, the loader is securely deleted by overwriting the file content and deleted through the DeleteFile function.

Afterwards, the persistency is set up. The persistency information is retrieved from the resource "105" and stored in the Gazer storage. Among these data, there is a `dword` value that is used to choose which persistency mode will be applied.

The resource 105 is structured in the following way:

- a `dword` value representing the persistence mode
- a `dword` value representing the size of the data
- the persistence information

There are 6 different persistence modes.

0: ShellAutorun
Persistence is achieved through the Windows registry by setting the value "Shell" with `"explorer.exe, %malware_pathfile%"` under the following key:

`HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon`

1: HiddenTaskAutorun
It is very similar to the "TaskScheduler Autorun (4)" method described below. The main difference is that the task is hidden from the user by using the `TASK_FLAG_HIDDEN` flag (set up via the `SetFlags` method from the `ITask` interface).

2: ScreenSaverAutorun
In this mode, Gazer achieves persistency by setting up in the Windows registry the executable file used for the screensaver.

Many values are created under the `HKCU\Control Panel\Desktop` registry key:

- `SCRNSAVE.exe` with the malware executable path
- `ScreenSaveActive` is set to "1": enable the screensaver
- `ScreenSaverIsSecure` is set to "0": specifies that the screensaver is not password-protected
- `ScreenSaveTimeout` is set to a value given in the resource. It specifies how long the system remains idle before the screensaver (in this case: the malware) starts.

3: StartupAutorun
If the resource 105 begins with the `dword` value "3", a LNK file will be created in the Start Menu. The resource will also provide a description for the shortcut file, the path for the target and the filename for the LNK.

The `IShellLink` interface is used to create the shell link.

4: TaskSchedulerAutorun
This method is used to achieve persistence by creating a scheduled task.

The task is created and set up through COM interfaces related to tasks (`ITaskService`, `ITaskSettings, …`).

Some information such as the task name and its description is retrieved from the resource. For example, in one of the sample's resources, the persistency mode is set to 04 (TaskSchedulerAutorun) with the persistency data:

```
%APPDATA%\Adobe\adobeup.exe Adobe Acrobat Reader Updater. This task was
generated by Adobe Systems, Inc to keep your Adobe Software up-to-data.
\Adobe\AcrobatReader.Adobe
```

In this example, a scheduled task will be created and set up thus:

- Task name: "`Adobe Acrobat Reader Updater`"
- Executable: "`%APPDATA%\Adobe\adobeup.exe`"
  - The orchestrator will copy the loader received through the named pipe to this location

- Task description: "`This task was generated by Adobe Systems, Inc to keep your Adobe Software up-to-data`"
- Task folder: "`\Adobe\AcrobatReader.Adobe`"

Last but not least, the task is configured to be started by the task scheduler at any time after its scheduled time has passed. The task will be triggered when the current user logs on.

5: LinkAutorun

This persistence method modifies existing LNK files to execute the malware through `cmd.exe`.

For each LNK file in the folder given in the resource, the icon and arguments are removed and the path is set to "`cmd.exe`" with the argument set to:

```
/q /c start "%s" && start "%s"
```

In most of the samples we analyzed, the configuration file specified that the TaskSchedulerAutorun persistence method should be used.

Logs

All three Gazer components log their actions into logfiles. They are encrypted with the same algorithm: 3DES.

In some versions of Gazer, it is easy to retrieve these logfiles because their filenames are hardcoded into the binaries:

- `%TEMP%\CVRG72B5.tmp.cvr`: the logs from the loader
- `%TEMP%\CVRG1A6B.tmp.cvr`: the logs from the orchestrator
- `%TEMP%\CVRG38D9.tmp.cvr`: the logs from the communication module

Each logfile is structured in the following way:

- [LOGSIZE][DECRYPTION_KEY][ENCRYPTED_LOG]
  - logsize: when this value (2 bytes) is subtracted from the magic value 0xf18b, it gives the encrypted log size
  - decryption_key: when this 12 bytes blob is XORed with another hardcoded key of 12 bytes, it gives the 3DES key that can be used to decrypt the log
  - encrypted_log: log encrypted with the 3DES algorithm in CBC mode

Once decrypted, each log entry is formatted in the following way:

|Hour:Min:Sec:Ms| [log ID] [log]

Here is an example with the decrypted orchestrator logfile:

```
|10:29:56:197|   [1556]
|10:29:56:197|   [1557]    ******************[...]****************
|10:29:56:197|   [1558]    DATE: 25.05.2017
|10:29:56:197|   [1559]    PID=900 TID=2324    Heaps=32
C:\Windows\Explorer.EXE
|10:29:56:197|   [1565]    DLL_PROCESS_ATTACH
|10:29:56:197|   [1574]    4164
|10:29:58:197|   [0137]    ==================[...]================
|10:29:58:197|   [0138]    Current thread = 2080
|10:29:58:197|   [0183]    Heap aff0000 [34]
|10:29:58:197|   [0189]    ### PE STORAGE ###
|10:29:58:197|   [0215]    ### PE CRYPTO ###
|10:29:58:197|   [0246]    ### EXTERNAL STORAGE ###
|10:29:58:197|   [1688]    Ok
|10:29:58:197|   [0279]    Path = \HKCU\Software\Microsoft\
Windows\CurrentVersion\Explorer\ScreenSaver
|10:29:58:197|   [0190]    \HKCU\Software\Microsoft\Windows\
CurrentVersion\Explorer\ScreenSaver
|10:29:58:197|   [0338]    ---FAILED
|10:29:58:197|   [0346]    Initializing standart reg storage...
|10:29:58:197|   [0190]    Software\Microsoft\Windows\
CurrentVersion\Explorer\ScreenSaver
|10:29:58:197|   [2605]    Storage is empty!
|10:29:58:197|   [0392]    ### EXTERNAL CRYPTO ###
|10:29:59:666|   [1688]    Ok
|10:29:59:713|   [1473]    Ok
|10:29:59:760|   [1688]    Ok
|10:29:59:775|   [1473]    Ok
|10:29:59:775|   [1688]    Ok
|10:29:59:775|   [1473]    Ok
|10:29:59:791|   [1688]    Ok
|10:29:59:791|   [1473]    Ok
|10:29:59:806|   [1688]    Ok
|10:29:59:806|   [1473]    Ok
|10:29:59:806|   [0270]    08-00-27-90-05-2A
|10:29:59:806|   [0286]    _GETSID_METHOD_1_
|10:29:59:806|   [0425]    28 7 8 122
|10:29:59:806|   [0463]    S-1-5-21-84813077-3085987743-
2510664113-1000
|10:29:59:806|   [0471]
|10:29:59:806|   [0787]    Ok
|10:29:59:806|   [1473]    Ok
|10:29:59:822|   [0514]    ### QUEUES ###
|10:29:59:822|   [0370]    T Empty
|10:29:59:822|   [0482]    R Empty
|10:29:59:822|   [1754]    Ok
|10:29:59:822|   [1688]    Ok
|10:29:59:822|   [1473]    Ok
|10:29:59:838|   [0505]    R #4294967295 PR_100 TR_00000000
SZ_172 SC_0(50)  --+-       EX_0
|10:29:59:838|   [0625]    ### TRANSPORT ###
|10:29:59:838|   [0286]    _GETSID_METHOD_1_
|10:29:59:838|   [0425]    28 7 25 122
|10:29:59:838|   [0463]    S-1-5-21-84813077-3085987743-
2510664113-1000
|10:29:59:838|   [0471]
|10:29:59:838|   [0165]    \\.\pipe\Winsock2\
CatalogChangeListener-2313-4
|10:29:59:838|   [0131]    PipeName = \\.\pipe\Winsock2\
CatalogChangeListener-2313-4
|10:29:59:838|   [0041]    true
[...]
```

Note that in older Gazer versions, the "log ID" was replaced by the name of the current function. We believe that this log ID is an ID for the function where the log occurs.

Working Directory

Using the Windows Registry

All the files related to Gazer (except the logs) are stored encrypted within the registry. The orchestrator's resource "109" contains the root storage path (it will be designated `%RootStoragePath%` in the rest of this paper). In every sample we examined, this resource pointed to the same storage path:

`HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\ScreenSaver`

If this resource is empty, the registry key above is used by default. Except for RSA keys, all the data in the storage is encrypted[2].

Several subdirectories (whose names are hardcoded in the binary) are created.

- `%RootStoragePath%{119D263D-68FC-1942-3CA3-46B23FA652A0}`
  - Object ID: a unique ID to identify the victim
- `%RootStoragePath%{1DC12691-2B24-2265-435D-735D3B118A70}`
  - Task Queue: linked list of tasks to be executed
- `%RootStoragePath%{28E74BDA-4327-31B0-17B9-56A66A818C1D}`
  - Plugins
- `%RootStoragePath%{31AC34A1-2DE2-36AC-1F6E-86F43772841F}`
  - Communication Module: the DLL that communicates with the C&C server
- `%RootStoragePath%{3CDC155D-398A-646E-1021-23047D9B4366}`
  - Autorun: the persistency method
- `%RootStoragePath%{4A3130BD-2608-730F-31A7-86D16CE66100}`
  - Local Transport Settings: the computers IPs that are on the same network
- `%RootStoragePath%{56594FEA-5774-746D-4496-6361266C40D0}`
  - Last Connection: last connection time with the C&C server (structure `SYSTEMTIME`)
- `%RootStoragePath%{629336E3-58D6-633B-5182-576588CF702A}`
  - RSA Private Key: generated on the fly; used to decrypt the data from Gazer storage.
- `%RootStoragePath%{6CEE6FE1-10A2-4C33-7E7F-855A51733C77}`
  - Result Queue: linked list of the tasks results
- `%RootStoragePath%{81A03BF8-60AA-4A56-253C-449121D61CAF}`
  - Inject Settings: the list of processes to use to try to inject the communications module
- `%RootStoragePath%{8E9810C5-3014-4678-27EE-3B7A7AC346AF}`
  - C&C servers

2 See the "Gazer Resources" section for details

Using Alternate Data Streams

If it is not possible to access the registry, these configuration items are stored using alternate data streams.

The function `GetVolumeInformation` is called to ensure that the volume "C:\" supports named streams in order to use ADS.

The same GUIDs as above are used to hide the different data in an ADS for the file (hardcoded in the binary):

`"%TEMP%\KB943729.log"`

For example, here is the full path to access the object ID:

`%TEMP%\KB943729.log:{1DC12691-2B24-2265-435D-735D3B118A70}`

Orchestrator

Gazer Resources

The Gazer-related files are stored in the orchestrator's resources.

File format

There are a total of 11 resources (101 to 111) each structured in the following way:

- [DATATYPE][SIZE][DATA][PADDING]
  - DATATYPE: A `dword` that specifies the type of data in the resource
    - 0x0: raw data
    - 0xFFFFFFFF: empty
    - 0x4: undefined
    - 0x1030001: strings array
    - 0x1: binary
  - SIZE: the size of the data (without padding)

Encryption

Except for resources 101 and 102 which are RSA keys, every resource is compressed with BZip and encrypted with 3DES.

[RSAEncryptedBlob][SignatureBlob][3DESBlob]

- RSAEncryptedBlob: The first 1024 bits of the data is a blob that contains a 3DES key. This blob is encrypted using RSA and can be decrypted using resource 101.
- SignatureBlob: The second part of the data is a blob of 1024 bits containing the signature of the last part of the data once decrypted.
- 3DESBlob: The last part is the effective data, which is encrypted with the 3DES key from the first blob.

Each resource is decrypted on the fly; the signature is compared with the decrypted data to check the integrity. Decrypted resources that pass this integrity check are encrypted with a new RSA key generated randomly by the orchestrator code. The private key and the encrypted resource are then stored in the registry under a specific GUID subkey.

Resources listing

- 101: RSA private key. It is used to decrypt the other resources.
- 102: an RSA public key.
- 103: empty
- 104: unknown
- 105: store the persistency information
- 106: the list of processes to use to try to inject the communications module
- 107: C&C communication DLL
- 108: C&C server list
- 109: Gazer working directory path
- 110: plugins list
- 111: local transport information

Task Execution

When a task is retrieved from the C&C, it is either executed by the infected machine or by another computer on the same network through a P2P mechanism (in the same way this was done in Carbon and Snake).

The task can be:

- file upload
- file download
- configuration update
- command execution

The result of the task is stored in a queue and forwarded to the module that communicates with the C&C server when access to the Internet is available.

Classes Hierarchy

The malware is written in C++ and the RTTI that contains information about the objects used in the code is not overwritten.

There are 5 abstract classes that have several implementations.

Table 1. **Abstract Class Autorun**

Class Name
LinkAutorun
StartupAutorun
ShellAutorun
ScreenSaverAutorun
TaskSchedulerAutorun
HiddenTaskAutorun

Table 2. **Abstract Class Queue**

Class Name
TaskQueue
ResultQueue

Table 3. **Abstract Class Storage**

Class Name
ExeStorage
FSStorage
RegStorage

Table 4. **Abstract Class TListenerInterface**

Class Name
LTMessageProcessing
CMessageProcessingSystem

Table 5. **Abstract Class TAbstractTransport**

Class Name
LTNamedPipe
TNPTransport

Communication Module

The communication module is used to retrieve tasks from the C&C server and to dispatch them to the orchestrator.

This library is injected into a process which can legitimately communicate over the Internet. The injection library is the same as the one found in the loader to inject the orchestrator into "`explorer.exe`".

Communication Initialization

If a proxy server exists, it is retrieved and used by Gazer to make the HTTP requests. There are two different methods used to retrieve this value, either by requesting the following registry key:

`HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings`

or through the function `InternetQueryOption` with the flag `INTERNET_OPTION_PROXY` if the proxy server cannot be retrieved through the registry.

The system user agent is then set up:

- the default value of the "`HKCU\Software\Microsoft\Windows\Current Version\Internet Settings\User Agent`" key is retrieved;

- the value keys under "`HKLM\Software\Microsoft\Windows\Current Version\Internet Settings\5.0\User Agent\Post Platform`" are enumerated and those that contain the sub-string "`IEAK`" are concatenated with the user agent string from the previous step;

- in the case that no user agent was found in the registry, the hardcoded UA `Mozilla/4.0 (compatible; MSIE 6.0)` is used

Before attempting any contact with the C&C server, the internet connection is checked by trying to reach the following servers one by one until one returns a HTTP status code 200:

- `update.microsoft.com`
- `microsoft.com`
- `windowsupdate.microsoft.com`
- `yahoo.com`
- `google.com`

C&C server communication

The malware communicates with its C&C server to retrieve tasks (through HTTP GET requests) and to send the tasks results (through HTTP POST requests).

Before sending a request to the C&C, the command `CMC_GIVE_SETTINGS` is sent to the orchestrator through its communication channel (a named pipe, more on this in the next section). The message (`MSG`) contained in the packet in this case is a single byte set by the orchestrator for the command result status.

The orchestrator replies on the same channel with the settings retrieved from the working directory with the object id, the list of the C&C servers and the last connection date.

A GET request is performed to retrieve a task from the C&C.

The parameters of the GET request are chosen from amongst a hardcoded list of keywords that does not look suspicious. Their values are generated randomly in the charset [a-z0-9] with a random size from a range given for each parameter:

- `id` [6-12] *(As with all other parameters, if this parameter is used in the request, it will have a random value (of letters and digits) with a random size between 6 and 12 characters.)*
- `hash` [10-15]
- `session` [10-15]
- `photo` [6-10]
- `video` [6-10]
- `album` [6-10]
- `client` [5-10]
- `key` [5-10]
- `account` [6-12]
- `member` [6-12]
- `partners` [5-10]
- `adm` [6-12]
- `author` [6-12]

- `contact` [6-12]
- `content` [6-12]
- `user` [6-12]

Here are few examples of such requests:

```
xxx.php?album=2ildzq&key=hdr2a&partners=d2lic33f&session=nurvxd2x0z8bztz&video
=sg508tujm&photo=4d4idgkxxx.php?photo=he29zms5fc&user=hvbc2a&author=xvfj5r0q
9c&client=7mvvc&partners=t4mgmuy&adm=lo3r6v4xxx.php?member=ectwzo820&contact
=2qwi15&album=f1qzoxuef4&session=x0z8bztz8hrs65f&id=t3x0ftu9xxx.php?partners
=ha9hz9sn12&hash=5740kptk3acmu&album=uef4nm5d&session=dpeb67ip65f&member=arj6
x3ljjxxx.php?video=nfqsz570&client=28c7lu2&partners=818eguh70&contact=ibj3xch
&content=1udm9t799ixr&session=5fjjt61qred9uo
```

A timeout of 10 minutes is set for each request (send/receive/connect) through `InternetSetOption`.

Once the request is sent, the response is handled only if the returned HTTP status code is 404. The content of the response is encrypted and can be decrypted with the private RSA key generated by the orchestrator. The response body contains a blob of data and an MD5 hash of the data. The blob is hashed and compared to the MD5 to ensure the integrity of the server's response.

If the response size is 20 bytes (a blob of 4 bytes + the hash), there are no tasks to retrieve.

A command `CMC_TAKE_TASK` is sent to the orchestrator with the encrypted task received from the C&C server and its size. The orchestrator will be in charge of executing the task and will send the results to the communication module. Once the blob of the tasks results (encrypted by the orchestrator) is received, it is sent to the C&C server through a POST request in the same way that it was done for the GET request (using parameters with random values).

Messages between components

A global named pipe is used for the communication between the different components. The data sent through this named pipe is formatted in the following way:

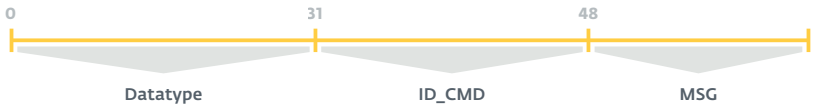


Figure 3. **Message format**

- DATATYPE: the same constants are used for the resources (check the File Format entry in the"Resources section")
- ID_CMD: the command name (check below for a complete list)
- MSG: the data to be sent

Here is a listing of the different commands:

- `CMC_TAKE_TASK (ID_CMD: 1)`
  - When a task is retrieved by the C&C server module, it is sent to the orchestrator, which stores the task in the task queue.

- `CMC_TAKE_LOADER_BODY (ID_CMD: 2)`

    - Wipe Gazer's original loader file, clean persistency and set up a copy of the loader and its persistency according to one of the resources (check persistency part for details).

- `CMC_GIVE_RESULT (ID_CMD: 4)`
    - When this message is received, the orchestrator will retrieve the task's result from the result queue, compress and encrypt it using the server's public RSA key (the one from the resource 102) and send the blob to the communication module which will send the whole result to the server through a POST request.
- `CMC_GIVE_SETTINGS (ID_CMD: 5)`
    - The communication module sends this message to the orchestrator to request the information needed to contact the server (list of the servers to contact, the last connection time and the victim ID).
- `CMC_TAKE_CONFIRM_RESULT (ID_CMD: 6)`
    - When the communication module sends a task's result to the server, a message is sent to the orchestrator that will remove the task's result from the queue.
- `CMC_TAKE_CAN_NOT_WORK (ID_CMD: 7)`
    - When an operation has failed (for example, if the communication module cannot correctly parse the data received from the orchestrator), this message is sent to the orchestrator with the last error code. The error code will be added to the logfile.
- `CMC_TAKE_UNINSTALL (ID_CMD: 8)`
    - Used to wipe a file from the disk.
- `CMC_TAKE_NOP (ID_CMD: 9)`
    - No operation
- `CMC_NO_CONNECT_TO_GAZER (ID_CMD: 0xA)`
    - This command is sent to the orchestrator when the communication module cannot contact any of the servers. In this case, if a pending task's results are in the queue, they are stored encrypted in Gazer's storage.
- `CMC_TAKE_LAST_CONNECTION (ID_CMD: 0xB)`
    - This command is sent from the communication module to the orchestrator each time a connection is established to the C&C server. It contains a structure `SystemTime` (filled with the current system time). Once the message is received by the orchestrator, the last connection date is stored compressed and encrypted in the Gazer storage (either the registry or ADS).
- `CMC_GIVE_CACHE / CMC_TAKE_CACHE (ID_CMD: 0xC / 0xD)`
    - Not implemented

GAZER VERSIONS

Four different versions have been identified.

In the first version, the function used to write logs has as its parameter the real function name where the log occurs. There were also different methods used to inject code (the one documented in this whitepaper and one based on window injection).

In a second version, the function names used as parameters are replaced by an ID and only one method is used for code injection. Also, the string "NO OLD METHODS" appears in this part of the code.

Some samples from the first versions were signed with a valid certificate issued by Comodo for "Solid Loop Ltd". The compilation date appears to be 2002 but is likely to be faked because the certificate was issued in 2015.

The latest versions are signed with a different certificate: "Ultimate Computer Support Ltd".

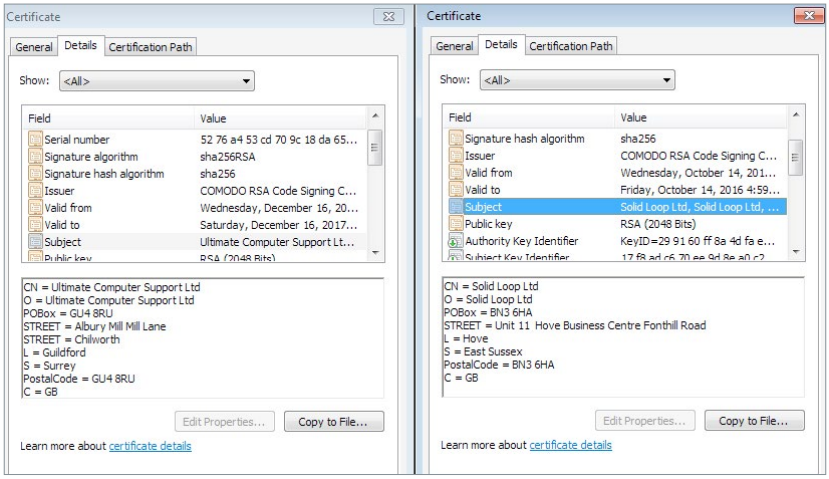


Figure 4. **Certificates used to sign the malware variants**

Some efforts have been made to obfuscate strings that can be used as IoCs. The mutex name and the named pipe do not appear in cleartext anymore; they are now encoded with a `XOR` key. On the previous versions, the logfile names were hardcoded in the binary. The function `GetTempFileNameA` is now used to generate a random filename. The C&C server returns a 404 or 502 status code page, whereas it was only a 404 in the previous versions.

In the latest versions compiled in 2017, the log messages are different (although they have the same meaning). For example: `"PE STORAGE"` is replaced by `"EXE SHELTER"`, `"PE CRYPTO"` by `"EXE CIPHER"` etc...

Last but not least, the compilation timestamp seems not to be faked anymore.

In conclusion, Gazer is a very sophisticated piece of malware that has been used against different targets in several countries around the world. Through the different versions we found and analyzed, we can see that this malicious backdoor is still being actively developed and used by its creators.

Indicators of Compromise can also be found on github. For any inquiries, or to make sample submissions related to the subject, contact us at: threatintel@eset.com.

IOCS

Filenames

- %TEMP%\KB943729.log
- %TEMP%\CVRG72B5.tmp.cvr
- %TEMP%\CVRG1A6B.tmp.cvr
- %TEMP%\CVRG38D9.tmp.cvr
- %TEMP%\~DF1E06.tmp
- %HOMEPATH%\ntuser.dat.LOG3
- %HOMEPATH%\AppData\Local\Adobe\AdobeUpdater.exe

Registry keys

- HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\ScreenSaver
- HKCU\Software\Microsoft\Windows NT\CurrentVersion\Explorer\ScreenSaver

C&C URLs

- daybreakhealthcare.co.uk/wp-includes/themees.php
- simplecreative.design/wp-content/plugins/calculated-fields-form/single.php
- 169.255.137.203/rss_0.php
- outletpiumini.springwaterfeatures.com/wp-includes/pomo/settings.php
- zerogov.com/wp-content/plugins.deactivate/paypal-donations/src/PaypalDonations/SimpleSubsribe.php
- ales.ball-mill.es/ckfinder/core/connector/php/php4/CommandHandler/CommandHandler.php
- dyskurs.com.ua/wp-admin/includes/map-menu.php
- warrixmalaysia.com.my/wp-content/plugins/jetpack/modules/contact-form/grunion-table-form.php
- 217.171.86.137/config.php
- 217.171.86.137/rss_0.php
- shinestars-lifestyle.com/old_shinstar/includes/old/front_footer.old.php
- www.aviasiya.com/murad.by/life/wp-content/plugins/wp-accounting/inc/pages/page-search.php
- baby.greenweb.co.il/wp-content/themes/san-kloud/admin.php
- soligro.com/wp-includes/pomo/db.php
- giadinhvabe.net/wp-content/themes/viettemp/out/css/class.php
- tekfordummies.com/wp-content/plugins/social-auto-poster/includes/libraries/delicious/Delicious.php
- kennynguyen.esy.es/wp-content/plugins/wp-statistics/vendor/maxmind-db/reader/tests/MaxMind/Db/test/Reader/BuildTest.php
- sonneteck.com/wp-content/plugins/yith-woocommerce-wishlist/plugin-fw/licence/templates/panel/activation/activation.php
- chagiocaxuanson.esy.es/wp-content/plugins/nextgen-gallery/products/photocrati_nextgen/modules/ngglegacy/admin/templates/manage_gallery/gallery_preview_page_field.old.php
- hotnews.16mb.com/wp-content/themes/twentysixteen/template-parts/content-header.php
- zszinhyosz.pe.hu/wp-content/themes/twentyfourteen/page-templates/full-hight.php
- weandcats.com/wp-content/plugins/broken-link-checker/modules/checkers/http-module.php

Mutexes

`{531511FA-190D-5D85-8A4A-279F2F592CC7}`

Hashes

Table 6. **Gazer sample hashes**

SHA1 hash	Component	Compilation Time	Certificate	Eset Detection Name
27FA78DE705EBAA4B11C4B5FE7277F91906B3F92	Gazer wiper x32	07/04/2016 15:04:24	not signed	Win32/Turla.CL
35F205367E2E5F8A121925BBAE6FF07626B526A7	Gazer loader x32	05/02/2002 17:36:10	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win32/Turla.CC
B151CD7C4F9E53A8DCBDEB7CE61CCDD146EB68AB	Gazer loader x32	05/02/2002 17:36:10	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win32/Turla.CC
E40BB5BEEC5678537E8FE537F872B2AD6B77E08A	Gazer loader x32	05/02/2002 17:36:10	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win32/Turla.CC
522E5F02C06AD215C9D0C23C5A6A523D34AE4E91	Gazer loader x64	05/02/2002 17:36:26	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win64/Turla.AA
C380038A57FFB8C064851B898F630312FABCBBA7	Gazer loader x64	05/02/2002 17:36:26	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win64/Turla.AA
267F144D771B4E2832798485108DECD505CB824A	Gazer loader x64	05/02/2002 17:36:26	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win64/Turla.AA
52F6D09CCCDBC38D66C184521E7CCF6B28C4B4D9	Gazer loader x32	04/10/2002 18:31:37	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win32/Turla.CC
475C59744ACCB09724DAE610763B7284646AB63F	Gazer loader x32	04/10/2002 18:31:37	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win32/Turla.CC
22542A3245D52B7BCDB3EAEF5B8B2693F451F497	Gazer loader x32	04/10/2002 18:31:37	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win32/Turla.CC
2B9FAA8B0FCADAC710C7B2B93D492FF1028B5291	Gazer loader x64	04/10/2002 18:34:18	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win64/Turla.AA
E05AB6978C17724B7C874F44F8A6CBFB1C56418D	Gazer loader x64	04/10/2002 18:34:18	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win64/Turla.AA
6DEC3438D212B67356200BBAC5EC7FA41C716D86	Gazer loader x64	04/10/2002 18:34:18	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win64/Turla.AA
B548863DF838069455A76D2A63327434C02D0D9D	Gazer loader x64	09/01/2016 19:30:10	not signed	Win64/Turla.AA
C3E6511377DFE85A34E19B33575870DDA8884C3C	Gazer loader x64	06/02/2016 19:29:15	admin@ ultimatecomsup.biz valid from 16/12/2015 to 16/12/2017	Win64/Turla.AA
9FF4F59CA26388C37D0B1F0E0B22322D926E294A	Gazer loader x64	16/02/2016 16:00:44	admin@ ultimatecomsup.biz valid from 16/12/2015 to 16/12/2017	Win64/Turla.AA

SHA1 hash	Component	Compilation Time	Certificate	Eset Detection Name
029AA51549D0B9222DB49A53D2604D79AD1C1E59	Gazer loader x64	18/02/2016 15:29:58	admin@ ultimatecomsup.biz valid from 16/12/2015 to 16/12/2017	Win64/Turla.AA
CECC70F2B2D50269191336219A8F893D45F5E979	Gazer loader x64	01/01/2017 08:39:30	admin@ ultimatecomsup.biz valid from 16/12/2015 to 16/12/2017	Win64/Turla.AG
7FAC4FC130637AFAB31C56CE0A01E555D5DEA40D	Gazer loader x64	11/06/2017 23:43:51	admin@ ultimatecomsup.biz valid from 16/12/2015 to 16/12/2017	Win64/Turla.AD
5838A51426CA6095B1C92B87E1BE22276C21A044	Gazer loader x32	19/06/2017 01:28:51	admin@ ultimatecomsup.biz valid from 16/12/2015 to 16/12/2017	Win32/Turla.CF
3944253F6B7019EED496FAD756F4651BE0E282B4	Gazer loader x64	19/06/2017 01:30:00	admin@ ultimatecomsup.biz valid from 16/12/2015 to 16/12/2017	Win64/Turla.AD
228DA957A9ED661E17E00EFBA8E923FD17FAE054	Gazer orchestrator x32	05/02/2002 17:31:28	not signed	Win32/Turla.CF
295D142A7BDCED124FDCC8EDFE49B9F3ACCEAB8A	Gazer orchestrator x32	05/02/2002 17:31:28	not signed	Win32/Turla.CF
0F97F599FAB7F8057424340C246D3A836C141782	Gazer orchestrator x32	05/02/2002 17:31:28	not signed	Win32/Turla.CF
DBB185E493A0FDC959763533D86D73F986409F1B	Gazer orchestrator x32	05/02/2002 17:31:28	not signed	Win32/Turla.CC
4701828DEE543B994ED2578B9E0D3991F22BD827	Gazer orchestrator x64	05/02/2002 17:34:25	not signed	Win64/Turla.AA
6FD611667BA19691958B5B72673B9B802EDD7FF8	Gazer orchestrator x64	05/02/2002 17:34:25	not signed	Win64/Turla.AA
FCABEB735C51E2B8EB6FB07BDA8B95401D069BD8	Gazer orchestrator x64	05/02/2002 17:34:25	not signed	Win64/Turla.AA
75831DF9CBCFD7BF812511148D2A0F117324A75F	Gazer orchestrator x32	04/10/2002 18:31:28	not signed	Win32/Turla.CC
BAE3AE65C32838FB52A0F5AD2CDE8659D2BFF9F3	Gazer orchestrator x32	04/10/2002 18:31:28	not signed	Win32/Turla.CC
37FF6841419ADC51EEB8756660B2FB46F3EB24ED	Gazer orchestrator x64	04/10/2002 18:33:02	not signed	Win64/Turla.AA
9E6DE3577B463451B7AFCE24AB646EF62AD6C2BD	Gazer orchestrator x64	04/10/2002 18:33:02	not signed	Win64/Turla.AA
795C6EE27B147FF0A05C0477F70477E315916E0E	Gazer orchestrator x64	04/10/2002 18:33:02	not signed	Win64/Turla.AA
8184AD9D6BBD03E99A397F8E925FA66CFBE5CF1B	Gazer orchestrator x64	09/01/2016 19:28:29	not signed	Win64/Turla.AA
7CED96B08D7593E28FEE616ECCBC6338896517CF	Gazer orchestrator x64	06/02/2016 19:29:04	not signed	Win64/Turla.AA
63C534630C2CE0070AD203F9704F1526E83AE586	Gazer orchestrator x64	06/02/2016 19:29:04	not signed	Win64/Turla.AA
23F1E3BE3175D49E7B262CD88CFD517694DCBA18	Gazer orchestrator x64	18/02/2016 15:29:32	not signed	Win64/Turla.AA

SHA1 hash	Component	Compilation Time	Certificate	Eset Detection Name
7A6F1486269ABDC1D658DB618DC3C6F2AC85A4A7	Gazer orchestrator x64	01/01/2017 08:39:19	not signed	Win64/Turla.AG
11B35320FB1CF21D2E57770D8D8B237EB4330EAA	Gazer orchestrator x64	11/06/2017 23:42:28	not signed	Win64/Turla.AD
E8A2BAD87027F2BF3ECAE477F805DE13FCCC0181	Gazer orchestrator x32	19/06/2017 01:28:21	not signed	Win32/Turla.CF
950F0B0C7701835C5FBDB6C5698A04B8AFE068E6	Gazer orchestrator x64	19/06/2017 01:29:46	not signed	Win64/Turla.AD
A5EEC8C6AADF784994BF68D9D937BB7AF3684D5C	Gazer comm x64	05/02/2002 17:57:07	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win64/Turla.AH
411EF895FE8DD4E040E8BF4048F4327F917E5724	Gazer comm x32	05/02/2002 17:58:22	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win32/Turla.CC
C1288DF9022BCD2C0A217B1536DFA83928768D06	Gazer comm x32	06/02/2016 19:23:52	not signed	Win32/Turla.CC
4B6EF62D5D59F2FE7F245DD3042DC7B83E3CC923	Gazer comm x32	11/06/2017 23:44:24	not signed	Win32/Turla.CF
7F54F9F2A6909062988AE87C1337F3CF38D68D35	Gazer wiper x32	05/02/2002 17:39:07	admin@solidloop.org valid from 14/10/2015 to 14/10/2016	Win32/Turla.CL

APPENDICES

Function names

There are a few samples of Gazer that use the current function name as first parameter for the log function. Here is a list of some function names used in Gazer:

- AutorunManager Class
  - AutorunManager::~AutorunManger
  - AutorunManager::Init
  - AutorunManger::ReInit
  - AutorunManager::BuildAutorunSettings
  - AutorunManager::FreeAutorunsSettings
  - AutorunManager::FullCheck
  - AutorunManager::StartAutorunEx
  - AutorunManager::FullStart
- HiddenTaskAutorun Class
  - HiddenTaskAutorun::IsPathsEqual
- LinkAutorun Class
  - LinkAutorunClass::InfectLnkFile
  - LinkAutorunClass::ClearLnkFile
  - LinkAutorunClass::CheckLnkFile
- RemoteImport32 Class
  - RemoteImport32::RemoteImport32
  - RemoteImport32::GetRemoteProcAddress
  - RemoteImport32::GetRemoteModuleHandle
- ScreenSaverAutorun Class
  - ScreenSaverAutorun::ChangeScreenSaver
  - ScreenSaverAutorun::WndProc1
  - ScreenSaverAutorun::GetMessageThreadProc
  - ScreenSaverAutorun::CreateHiddenWindow
  - ScreenSaverAutorun::CloseHiddenWindow
- ShellAutorun Class
  - ShellAutorun::AutorunInstallEx
  - ShellAutorun::AutorunUninstallEx
  - ShellAutorun::AutorunCheckEx
  - ShellAutorun::IsPathsEqual
- StartupAutorun Class
  - StartupAutorun::AutorunInstallEx
  - StartupAutorun::AutorunUninstallEx
  - StartupAutorun::AutorunCheckEx
  - StartupAutorun::IsPathsEqual
- TaskScheduler20Autorun Class
  - TaskScheduler20Autorun::Init
  - TaskScheduler20Autorun::AutorunCheckEx
  - TaskScheduler20Autorun::AutorunInstallEx
  - TaskScheduler20Autorun::AutorunUninstallEx
  - TaskScheduler20Autorun::IsPathsEqual

- DllInjector Class
  - DllInjector::LoadDllToProcess
  - DllInjector::GetProcHandle
  - DllInjector::CheckDllAndSetPlatform
  - DllInjector::CopyDllFromBuffer
  - DllInjector::MapLibrary
  - DllInjector::Map86Library_tox64
  - DllInjector::CallEntryPoint
  - DllInjector::FindDllImageBase
  - DllInjector::WindowInject
- InjectManager Class
  - InjectManager::~InjectManager
  - InjectManager::BuildInjectSettingsList
  - InjectManager::FreeInjectSettingsList
  - InjectManager::Stop
  - InjectManager::DetachAll
  - InjectManager::FindAndInjectInVictim
  - InjectManager::FindProcessSimple2
  - InjectManager::LoadNtdll
  - InjectManager::UnLoadNtdll
  - InjectManager::LoadWinsta
  - InjectManager::UnLoadWinsta
  - InjectManager::SetStatusTransportDll
  - InjectManager::GetTransportState
  - InjectManager::DestroyManuallyCreatedVictim
  - InjectManager::VictimManualCreateIE
- TNPTransport Class
  - TNPTransport::Init
  - TNPTransport::ReInit
  - TNPTransport::~TNPTransport
  - TNPTransport::Receive
  - TNPTransport::RunServer
  - TNPTransport::ServerProc
- ExeStorage Class
  - ExeStorage::Migrate
  - ExeStorage::SecureHeapFree
- FSStorage Class
  - FSStorage::~FSStorage
  - FSStorage::Init
  - FSStorage::GetBlock
  - FSStorage::GetListBlock
  - FSStorage::Migrate
  - FSStorage::SecureHeapFree
  - FSStorage::Update
  - FSStorage::Empty
- RegStorage Class
  - RegStorage::~RegStorage
  - RegStorage::Init
  - RegStorage::FreeList

  - RegStorage::GetListBlock
  - RegStorage::DeleteListBlock
  - RegStorage::Migrate
  - RegStorage::SecureHeapFree
  - RegStorage::Update
  - RegStorage::Empty
- ResultQueue Class
  - ResultQueue::~ResultQueue
  - ResultQueue::DumpQueueToStorage
  - ResultQueue::RestoreFromStorage
  - ResultQueue::ClearQueue
  - ResultQueue::RemoveResult
  - ResultQueue::GetNextResultToSendWithModule
  - ResultQueue::SetPredeterminedResult
  - ResultQueue::print
- TaskQueue Class
  - TaskQueue::~TaskQueue
  - TaskQueue::DumpQueueToStorage
  - TaskQueue::RestoreFromStorage
  - TaskQueue::ClearQueue
  - TaskQueue::RemoveCompletedTasks
  - TaskQueue::print
- CExecutionSubsystem Class
  - CExecutionSubsystem::~CExecutionSubsystem
  - CExecutionSubsystem::Stop
  - CExecutionSubsystem::TaskExecusion
  - CExecutionSubsystem::TaskConfigure
  - CExecutionSubsystem::TaskUpload
  - CExecutionSubsystem::TaskDownload
  - CExecutionSubsystem::TaskReplacement
  - CExecutionSubsystem::TaskDelete
  - CExecutionSubsystem::TaskPacketLocalTransport
  - CExecutionSubsystem::FinishTask
  - CExecutionSubsystem::PushTaskResult
  - CExecutionSubsystem::UpdateStorage
- CMessageProcessingSystem Class
  - CMessageProcessingSystem::~CMessageProcessing
  - CMessageProcessingSystem::ListenerCallBack
  - CMessageProcessingSystem::WaitShutdownModule
  - CMessageProcessingSystem::SetCompulsorySMC
  - CMessageProcessingSystem::UnSetCompulsorySMC
  - CMessageProcessingSystem::IsCompulsorySMC
  - CMessageProcessingSystem::GetCompulsorySMC
  - CMessageProcessingSystem::Receive_TAKE_NOP
  - CMessageProcessingSystem::Receive_GIVE_SETTINGS
  - CMessageProcessingSystem::Receive_TAKE_CAN_NOT_WORK
  - CMessageProcessingSystem::Receive_GIVE_CACHE
  - CMessageProcessingSystem::Receive_TAKE_CACHE
  - CMessageProcessingSystem::Receive_TAKE_TASK
  - CMessageProcessingSystem::Receive_GIVE_RESULT

- CMessageProcessingSystem::Receive_TAKE_CONFIRM_RESULT
- CMessageProcessingSystem::Receive_TAKE_LOADER_BODY
- CMessageProcessingSystem::Receive_TAKE_UNINSTALL
- CMessageProcessingSystem::Receive_NO_CONNECT_TO_Gazer
- CMessageProcessingSystem::Receive_TAKE_LAST_CONNECTION
- CMessageProcessingSystem::Send_TAKE_FIN
- CMessageProcessingSystem::Send_TAKE_SHUTDOWN
- CMessageProcessingSystem::Send_TAKE_SETTINGS
- CMessageProcessingSystem::Send_TAKE_RESULT

- Crypto Class
  - Crypto::GetPublicKey
  - Crypto::EncryptRSA
  - Crypto::Sign
  - Crypto::EncryptAndSignBufferRSAEx
  - Crypto::DecryptRSA
  - Crypto::Verify
  - Crypto::DecryptAndVerifyBufferRSAEx
  - Crypto::EncryptAndSignBufferRSA1
  - Crypto::EncryptAndSignBufferRSAC
  - Crypto::DecryptAndVerifyBufferRSAO
  - Crypto::DecryptAndVerifyBufferRSA1
  - Crypto::DecryptAndVerifyBufferRSAL
  - Crypto::VerifyLoaderFile
  - Crypto::VerifyLoader
  - Crypto::CompressBuffer
  - Crypto::DecompressBuffer

- LTManager Class
  - LTManager::~LTManager
  - LTManager::Init
  - LTManager::GetResultFromQueue
  - LTManager::SetResultToCache
  - LTManager::GetTaskFromCache
  - LTManager::SetTaskToQueue
  - LTManager::IsSendPacketFurtherOnRoute
  - LTManager::SendPacketNextRouteUnit
  - LTManager::SetCache
  - LTManager::SetPacket
  - LTManager::DumpCacheToStorage
  - LTManager::DeSerializeCache
  - LTManager::DeSerializePacket
  - LTManager::DeSerializeRoute
  - LTManager::DeSerializeTask
  - LTManager::DeSerializeResult
  - LTManager::SerializeCache
  - LTManager::SerializePacket
  - LTManager::SerialiazeRoute
  - LTManager::SerializeTask
  - LTManager::SerializeResult
  - LTManager::ClearCache
  - LTManager::ClearPacket
  - LTManager::ClearRoute

- LTManager::ClearTask
- LTManager::ClearResult
- LTManager::PrintCache
- LTManager::CreateEvents
- LTManager::SetEvents
- LTManager::ResetEvents
- LTManager::WaitEvents
- LTManager::DeleteEvents

- LTMessageProcessing Class
  - LTMessageProcessing::ListenerCallBack
  - LTMessageProcessing::Send_TAKE_OK
  - LTMessageProcessing::Send_TAKE_ERROR_CRYPT
  - LTMessageProcessing::Send_TAKE_ERROR_UNKNOWN

- LTNamedPipe Class
  - LTNamedPipe::ReInit
  - LTNamedPipe::BuildLocalTransportSettings
  - LTNamedPipe::~LTNamedPipe
  - LTNamedPipe::Receive
  - LTNamedPipe::RunServer
  - LTNamedPipe::Stop
  - LTNamedPipe::CreateNewNPInstance
  - LTNamedPipe::ServerProc
  - LTNamedPipe::ClientCommunication

Yara rules

```
import "pe"
import "math"
import "hash"

rule Gazer_certificate_subject {
  condition:
    for any i in (0..pe.number_of_signatures - 1):
       (pe.signatures[i].subject contains "Solid Loop" or
pe.signatures[i].subject contains "Ultimate Computer Support")
}

rule Gazer_certificate
{
  strings:
    $certif1 = {52 76 a4 53 cd 70 9c 18 da 65 15 7e 5f 1f de 02}
    $certif2 = {12 90 f2 41 d9 b2 80 af 77 fc da 12 c6 b4 96 9c}
  condition:
    (uint16(0) == 0x5a4d) and 1 of them and filesize < 2MB
}

rule Gazer_logfile_name
{
    strings:
     $s1 = "CVRG72B5.tmp.cvr"
     $s2 = "CVRG1A6B.tmp.cvr"
     $s3 = "CVRG38D9.tmp.cvr"
    condition:
    (uint16(0) == 0x5a4d) and 1 of them
}
```