Paper

Bitdefender

Terdot: Zeus-based malware strikes back with a blast from the past

Author: Eduard Budaca – Antimalware Researcher

[2]

Foreword

Malware authors are surely known for their ability to fly under the radar. But every once in a while, details about their operations surface on the web. This is the case of a handful of malware operations that managed to gain unwanted attention by having their source code leaked. Mirai, KINS, Carberp and Zeus are among the malware families that went “open-source”, either voluntarily or because of operational negligence. And when this happens, high-quality code is rapidly adopted and integrated by less-skilled criminal groups looking for shortcuts to financial success.

This whitepaper is a technical analysis of the Terdot, a Banker Trojan that derives inspiration from the 2011 Zeus source code leak. Highly customized and sophisticated, Terdot can operate a MITM proxy, steal browsing information such as login credentials and stored credit card information, as well as inject HTML code in visited Web pages.

Particularly interesting about Terdot, though, is that, just like the Netrepser targeted attack, it leverages legitimate applications such as certificate injection tools for nefarious purposes, rather than specialized utilities developed in house. Another discovery worth mentioning is that, even if Terdot is technically a Banker Trojan, its capabilities go way beyond its primary purpose: it can also eavesdrop on and modify traffic on most social media and email platforms. Its automatic update capabilities allow it to download and execute any files when requested by its operator, meaning it can develop new capabilities.

The Trojan’s list of regular banking websites mostly includes Canadian institutions such as PCFinancial, Desjardins, BMO, Royal Bank, the Toronto Dominion bank, Banque Nationale, Scotiabank, CIBC and Tangerine Bank. Apart from its targets in the banking sector, the samples that form the object of this research also target information from e-mail service providers such as Microsoft’s live.com login page, Yahoo Mail (all top-level domains) and Gmail (all top-level domains). Targeted social networks inclide Facebook, Twitter, Google Plus and YouTube. Interestingly, the malware is specifically instructed not to gather any data from vk.com, Russia’s largest social media platform.

1. Initial compromise and spreading

This analysis is based on information we have been collecting since October 2016, when Terdot made a successful comeback. This malware family has been observed to arrive via infected e-mail messages rigged with a button with the PDF icon on it. When clicked, a piece of obfuscated Javascript code is executed. This code downloads and runs the malware file.

Other samples are apparently delivered in malware campaigns via the Sundown Exploit Kit, which appears to be the primary infection vector for the Terdot family.

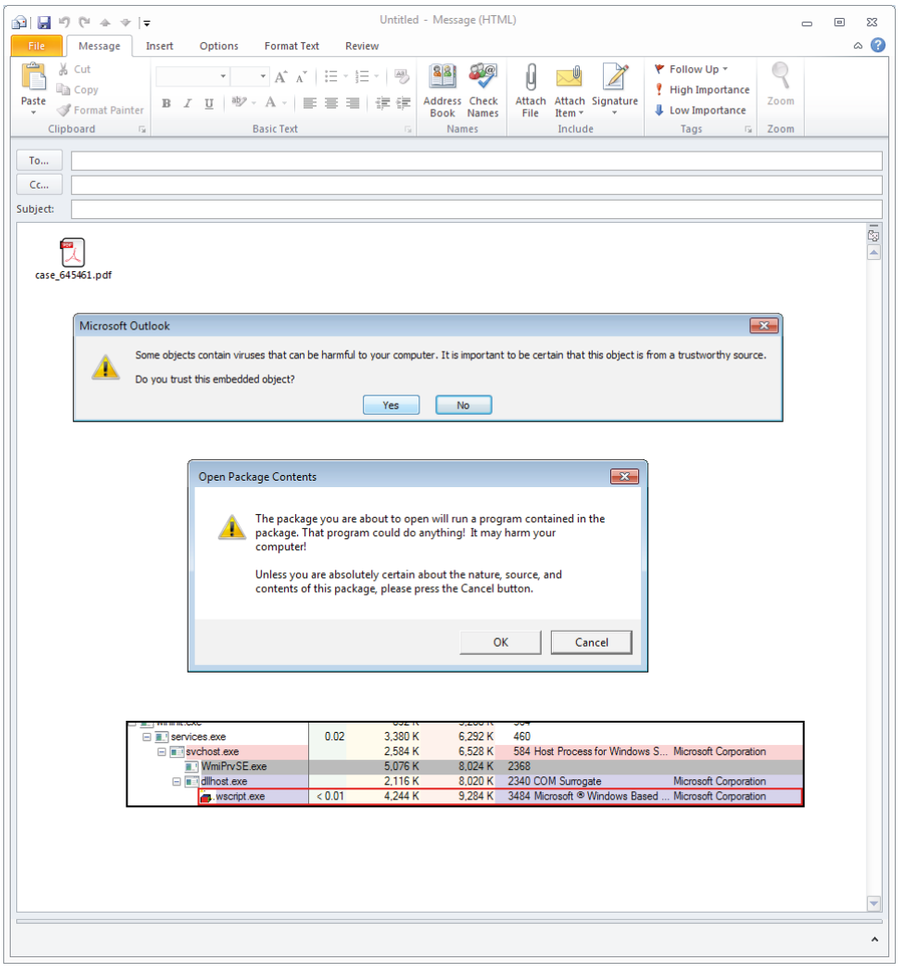

Fig. 1: Infection flow for a payload that arrives via infected e-mail. Warning dialogs and the new process spawned for the download and execution of the malware payload.

To protect the payload, the Terdot delivery mechanism uses a complex chain of droppers, injections and downloaders until the Terdot files and third-party utilities are downloaded on the disk. The infection chain is depicted in the image below:

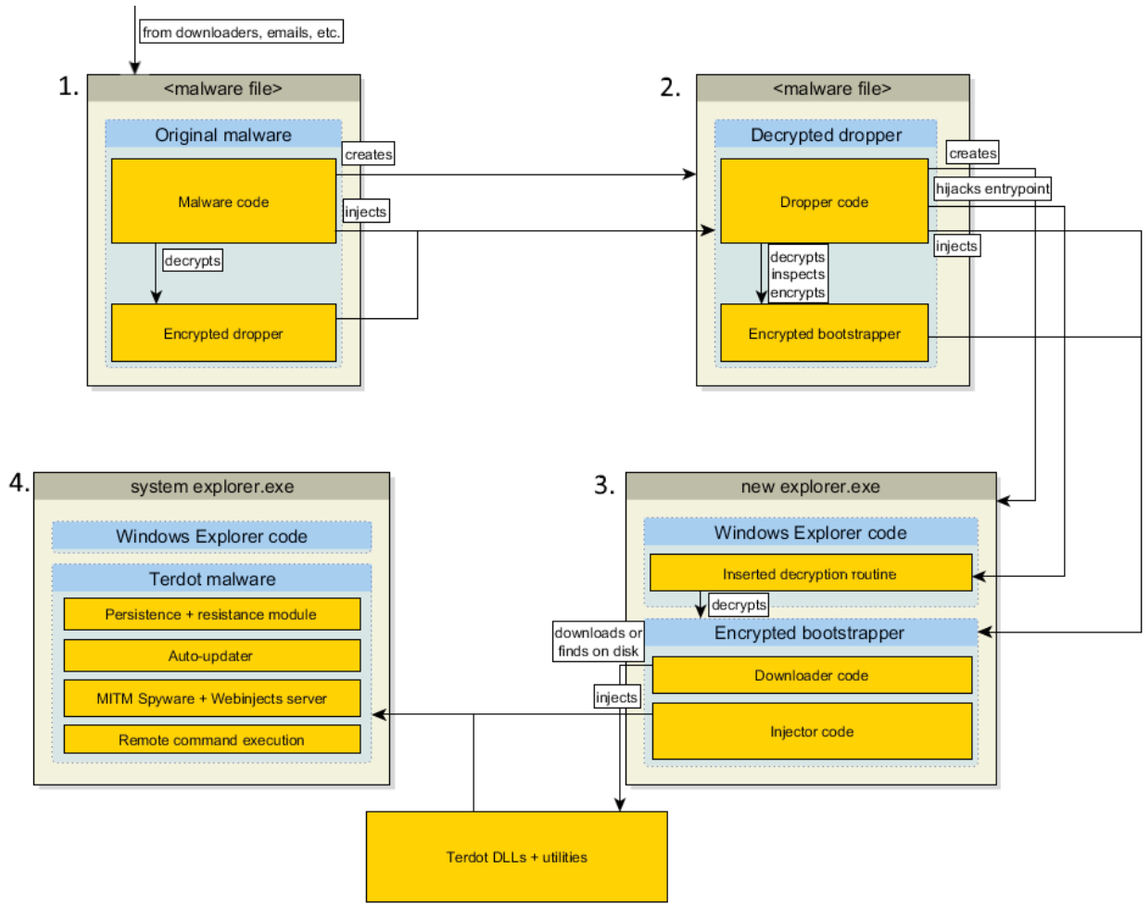

Fig. 2: Terdot initial injection chain

To read browser traffic, Terdot injects itself into browser processes, where it hooks very low-level network socket operations to direct all connections to its own local proxy server. This server inspects the traffic, forwards the request to the intended target, receives the response, then it sends it back to the victim's browser, possibly altering it in the process.

This malware family is highly versatile and can steal authentication data in two ways: by inspecting the client's requests, or by injecting spyware Javascript code in the responses. Terdot keeps logs of relevant data in HTTP requests and uploads them periodically to the C&C servers.

Because most banks, including those targeted by Terdot, have universally adopted HTTPS, the malware includes capabilities to bypass the restrictions imposed by TLS by creating its own Certificate Authority and generating certificates for every visited domain in a classic man-in-the-middle attack. For Internet Explorer, the malware installs hooks to Win32 API certificate checking functions to trick the browser into trusting these forged certificates, and for Mozilla Firefox, Terdot adds the root certificate to the browser's trusted CA list, using legitimate tools provided by Mozilla.

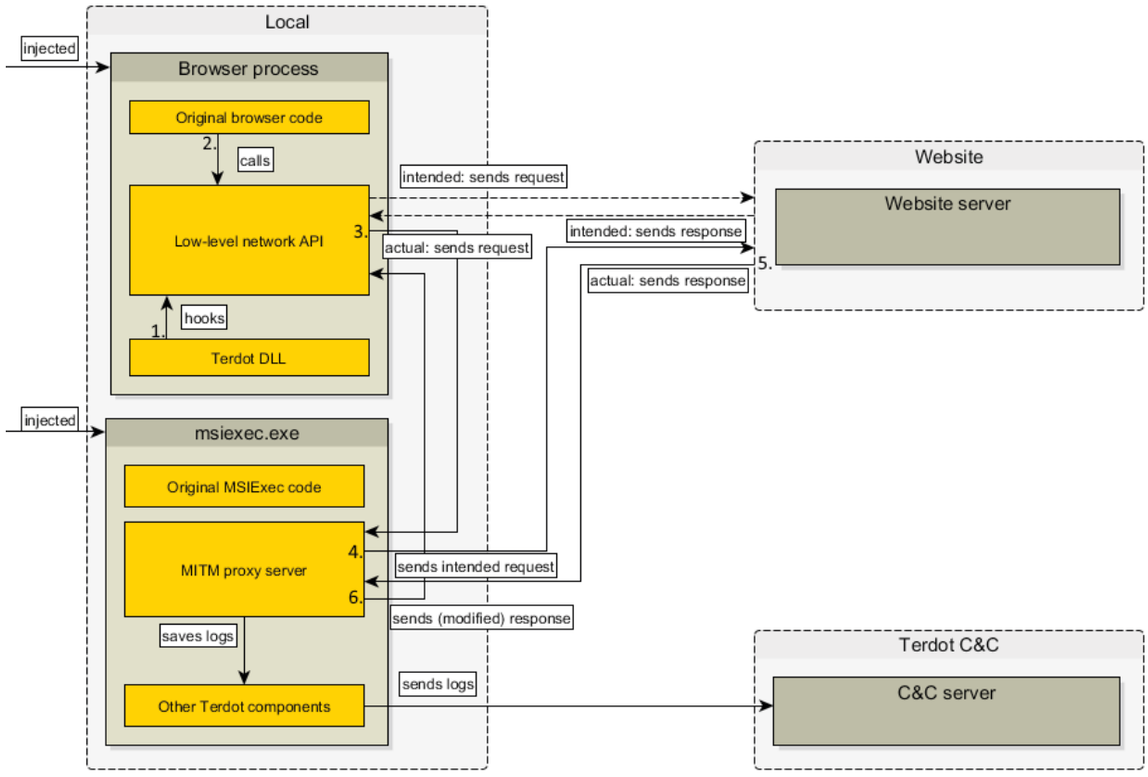

Fig. 3: Man-in-the-Middle attack implementation used by Terdot

Terdot's components are split across multiple processes, each with a specific role. Windows Explorer processes, along with other normal long-running Windows processes, are either injectors responsible for spreading the infection inside the machine, or watchdogs, processes that make cleanup more difficult. Terdot's MITM proxy that receives traffic from browser processes runs in a msiexec.exe process.

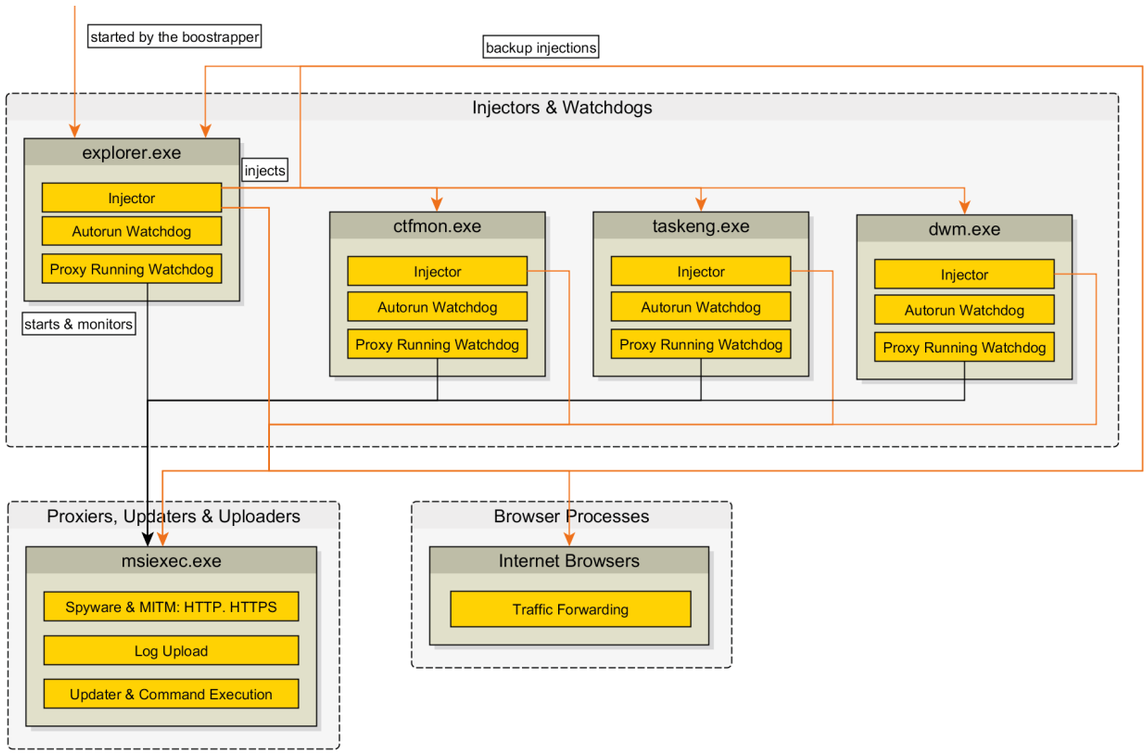

Fig. 4: Terdot components, along with the injection chain.

1.1. Technical dive into the sample

The sample we analyzed is an executable file written in Visual Basic. When executed, it creates a new process with the same name, but with different data than what is on file. The new data only gets written inside the virtual memory, with none of it written on the disk. For some samples, the injection process is skipped altogether and the new data is simply overwritten to the current process.

The new executable image (which acts as a dropper) contains two sections: code, as well as an encrypted DLL (bootstrapper) that gets injected during the next step. The code section features different obfuscation for all samples analyzed by Bitdefender's researchers.

However, the obfuscation algorithm seems to use similar manipulations of the ESP register, as well as XCHG-ing valus on the stack and registers. This renders static analysis of this section inefective. All analyzed dropper samples, however, have the encrypted DLL located at RVA 0x348 and gets decrypted as follows (the key is an unsigned integer and data is a byte array):

```
data[index] ^= (char)key
key = (key << 8) + index
```

The key is different from one executable file to another and cannot be statically retrieved because the section's offsets are different. However, given the fact that the encrypted data should always decrypt to an executable file, we can "predict" that the first two bytes will be the ASCII codes for M and Z, respectively. This is because we expect the MZ header, as defined by the structure of the PE file. The next two bytes of the key are easy to bruteforce, given that there are only 65,536 combinations. Optimizing for least necessary decryption to check the key, a brute-forcing algorithm written in Python takes less than 4 seconds on an i7-4790 CPU.

The dropper subsequently injects itself into a new explorer.exe procees and writes the encrypted DLL into the memory of the process outside of its image. This trick helps the malware authors conceal the malicious code even if a dump of the process image is taken, as the injected code is outside the original process image.

It also writes 38 bytes at the entrypoint, which contain the code that decrypts the second section and jumps to an address inside the decrypted buffer. This buffer processes relocations and imports and then runs the entrypoint of the bootstrapper DLL.

1.2. Data structures

The malware makes use of some data structure across its code. To better understand the inner workings of the malware, these structures are described below.

1.2.1. The **BaseConfig structure** format

This structure is used to keep information used mainly in the bootstrapping process. This data is identical across infection campaigns and never changes.

Field	Size
file mapping name	16 bytes
unknown	60 bytes
botnet ID	71 bytes
initial download URLs	266 bytes
configuration server URLs	263 bytes
upload timeouts	2*2 bytes
unknown	4 bytes
download timeouts	2*2 bytes
unknown	18 bytes
RC4 state buffer	258 bytes
unknown	17 bytes
flag: should send spyware logs	1 byte
unknown	9 bytes
flag: is first download compressed	1 byte
server RSA public key length	4 bytes
unknown	9 bytes
server RSA public key + padding	527 bytes
persistence structure subkey	8 bytes
.doc filename (unused)	17 bytes
persistence structure value	15 bytes

Notes:

- The URL fields are stored as semicolon-separated lists of strings. The initial download URLs represent the sources where the Terdot DLLs can be downloaded from; the configuration URLs are sources where a configuration data container structure can be downloaded from.

- The RSA public key is used to enforce the authenticity of the downloaded configuration data. It checks whether the C&C server, as opposed to any other source, has signed the downloaded configuration data.

- The persistence subkey and value fields correspond to a location under `HKCU\Software\Microsoft\`, where a persistence structure is stored. The bootstrapper uses the Registry to make sure that subsequent executions of the malware will use the same parameters.

Common values are specified in Appendix 1.

The **persistence structure** format

This structure is specific to every installation of Terdot; it holds machine-specific information and randomly-generated data.

Field	Size
structure size	4 bytes
main drive CLSID	16 bytes
computer ID	60 WCHARs = 120 bytes
unknown	8 bytes
autorun entry name	12 WCHARs = 24 bytes
unknown	104 bytes
RC4 state buffer	258 bytes
padding	2 bytes
mutex name seed	4 bytes
spyware log file key	4 bytes
HTTP proxy port	2 bytes
HTTPS proxy port	2 bytes
original copy filename	20 bytes
downloaded structure filename	20 bytes
spyware log filename	20 bytes
unknown	20 bytes
base registry key	10 bytes
registry value name of the configuration data container	10 bytes
registry value name of the main data container	10 bytes
unused registry value name	10 bytes

Notes:

- The structure size field is `0x29c`.
- The CLSID of the primary hard-disk drive is obtained by calling `CLSIDFromString` on the results of `GetVolumeNameForVolumeMountPointW` on the drive where Windows is installed.
- The computer ID is `%s_%08X%08X`, formatted with the computer name, or `unknown` if there is an error, the CRC32 of the `OSVERSIONINFOW` object and the CRC32 of 2 concatenated DWORDs: the `InstallDate` value inside `HKLM\Software\Microsoft\Windows NT\CurrentVersion` and the CRC32 of the `DigitalProductId` value inside the same registry key.
- The autorun entry name is the name of the value associated with the malware executable file inside the autorun registry key (`HKCU\Software\Microsoft\Windows\Currentversion\Run`).
- The mutex name seed is used as a randomizer for some mutexes used by the malware.
- The spyware log file key is used to decode the file where spyware logs are stored. This is generated during the first run and, as a result, varies from machine to machine.
- The proxy ports are the ports that the spyware proxy server is listening on. Again, this is generated upon the first run.
- The downloaded structure filename refers to a file where the downloaded payload containing the Terdot executables and tools are stored in.
- The base registry key is the name of a created registry key under `HKCU\Software\Microsoft`. It is used as a base key for all data stored in the registry.
- All filenames are relative to the `AppData` folder.
- All registry values are relative to the base registry key specified above.

1.2.2. The **downloaded structure** format

This structure is downloaded by the bootstrapper process and primarily contains the Terdot DLLs (these are not included in the original sample, but are required to be downloaded).

Field	Size
magic value	4 bytes
malware version	4 bytes
32-bit malware DLL length	4 bytes
32-bit malware DLL	variable
64-bit malware DLL length	4 bytes
64-bit malware DLL	variable
extra files: flags	4 bytes
extra files: length	4 bytes
extra files	variable

Notes:

- The magic value is `0x01000000`
- The malware version field is used to check if there is a newer version of the malware.
- The malware DLLs are variants of the popular ZeuS malware, and they are built on top of the source code that leaked in 2011. In the payload buffers we were able to retrieve, the 32-bit malware DLL has the export name "`client32.dll`" while the 64-bit DLL is named "`client64.dll`". The extra files are utilities used by Terdot in the man-in-the-middle process.

1.2.3. The **data container** structure format

Similar to ZeuS, which served as inspiration and foundation, Terdot uses a special container structure to store some configuration data. Uploads and downloads of configuration data to and from the C&C server also use this format. The configuration data container downloaded from the C&C server never changes and holds URLs and HTML code to be injected by the spyware proxy. This container is structured as such: a header, followed by an arbitrary number of blocks, each encoded separately and possibly compressed using the `nrv2b` algorithm. The format of the header is:

random data	container size in bytes	zero	number of blocks	MD5 hash of blocks
20 bytes	4 bytes	4 bytes	4 bytes	16 bytes

This header is immediately followed by a number of blocks, each in the following format:

type identifier	block flags	data length	uncompressed length	data
4 bytes	4 bytes	4 bytes	4 bytes	variable

Before being saved or sent, the data container undergoes some processing:

- If a block has the identifier combination identical to an earlier one, and it has bit 16 set in its flags field, then the newer block is discarded; if it has bit 19 set, the older block is discarded.
- A new block, between 128 and 1024 bytes long, filled with random data, is added at the end of the container. The MD5 hash of the blocks is calculated and saved (in binary format), in its designated field in the container header.
- The first 20 bytes in the container are filled with random data.
- The entire structure is encrypted using the following algorithm:

```
for (int i = 1; i < container_length; i++){
    container_data[i] ^= container_data[i - 1];
}
```

- The resulting data is encrypted using a copy of either the RC4 state buffer stored in the persistence structure (in the case of the data containers stored in the registry) or the RC4 state buffer stored in the BaseConfig structure (if the data container is sent to or received from the C&C server)

This is an example of a data container object:

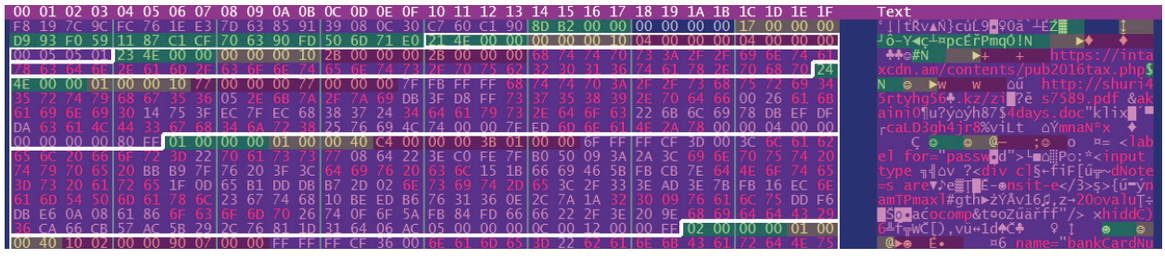

Fig. 5: HEX representation of a data container object

This is the start of the configuration data container downloaded from the C&C server, after it has been decrypted. However, note that some blocks are still compressed.

This object includes the following sections, color-coded for easy identification:

- Container header, in order:
  - magenta: random bytes
  - green: container size, in bytes
  - yellow: number of blocks
  - green: MD5 hash of the blocks
- Several blocks follow, separated by white outlines; for each, in order:
  - green: type identifier
  - yellow: flags
  - brown: raw length (first 4 bytes), uncompressed length (last 4 bytes)
  - magenta: raw data; note that, if the first bit of the flags is 1 (blocks 2, 3, 4), the data is compressed and the uncompressed length is usually larger

1.3. The Bootstrapper DLL

This DLL serves just one purpose, namely to find and run the Terdot DLL, either from an existing installation (in the case of a system reboot) or by downloading it. The export name of this file is `loader.dll`, and it has a single export, named `Begin`.

Our observations of the family of malware show that some samples first check if they have the necessary privilege to edit the Registry key over at `HKCU\Software\Microsoft\Test` by trying to create this key first. If the Registry access is denied (since `HKCU\Software\Microsoft` is owned by SYSTEM so it has additional protection), then the malware process doesn't momentarily have enough rights to achieve its goals, so it re-runs the original malware executable using the `ShellExecuteEx` API call with the command `runas wmic process call create [sample_path]`. This reruns the executable; however, the user will have to accept an UAC prompt such as the one illustrated below. Since the prompt only mentions Windows utilities, the potential victim is more likely to accept it.

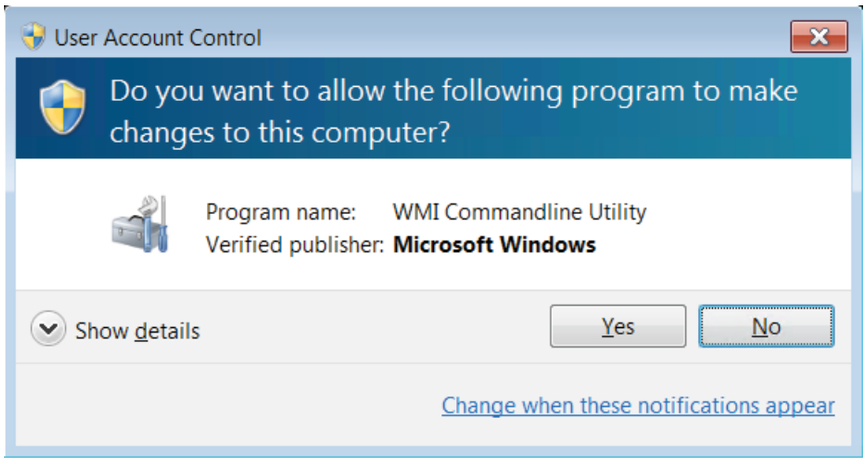

Fig. 6: Terdot requesting elevation via UAC

If this trick succeeds and the malware runs in an elevated context, it attempts to download the payload containing the Terdot DLL.

First, the malware checks if a file mapping with a specific name exists. This name varies from sample to sample and common values are listed in Appendix 1. This file mapping is written by later stages of the malware, when an update has just been downloaded and executed and it needs to be configured. If it doesn't exist, the execution continues.

If this mapping exists, its contents is copied to memory then erased from the mapping. In this payload, the first DWORD represents the malware version. If it is a newer version, it is accepted, otherwise, the malware aborts execution. The file mapping contains a persistence structure and its global RC4 state buffer is used for decrypting the data; this file contains a previously downloaded data structure with the Terdot DLL and other utility executables and DLLs.

If this is an older version (determined by comparing the version field with the current version), the malware checks for a newer version of the payload from the C&C server. No other fields of this file mapping are used at this stage, but will be used later in the update process.

If the mapping does not exist, the malware tries to find another payload source at a certain location in the Windows Registry. Its presence is determined by checking if a specific Registry subkey and value name pair exists. It is located under `HKCU\Software\Microsoft\` and varies from sample to sample as shown in Appendix 1.

If it exists, it is decrypted with a copy of a global RC4 state buffer. The result is a persistence structure; this ensures that later runs (such as after a system reboot) will use the same parameters. Like in the case where the file mapping actually exists, this persistence structure contains a filename in the `Appdata` directory, whose contents are decrypted with an RC4 state buffer found inside the structure. When decrypted, this file also contains the Terdot DLL and other files. This data will be subsequently used to bootstrap the malware.

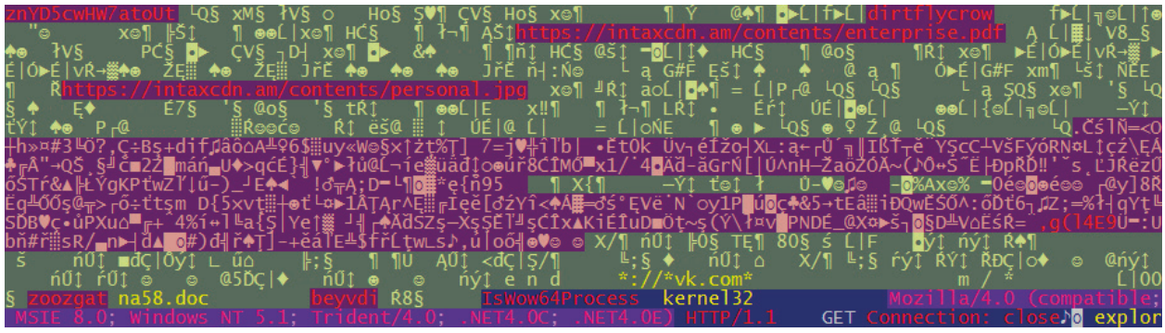

Fig. 7: *Part of the bootstrapper's program data; In yellow: the configuration structure (BaseConfig); in red, in order: the update mapping name, the botnet's name, the Terdot DLL download URL, the configuration download URL, the RC4 state buffer, the public RSA key of the C&C server (length, then data), and the persistence structure registry subkey and value; in magenta, the default user-agent.*

1.4. The download mechanism

If neither the file mapping nor the registry location exist, the malware tries to download the payload from the C&C server, which it reaches through one or more hardcoded, plaintext URLs. These URLs may vary from one sample to another, and the values they might take are listed in **Appendix 1**.

The user-agent is either injected in the program executable image by the earlier process, (the system user-agent retrieved with `urlmon.dll:ObtainUserAgentString`), or, if such an agent cannot be retrieved, a default, hardcoded, plaintext one (*`Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET4.0C; .NET4.0E)`*, corresponding to a standard IE8 user agent) is used.

It first tries the download using the system-configured proxy, if it fails, it tries without proxies. Also, if the protocol is HTTPS, invalid certificates are ignored. Otherwise, the download is a simple GET request. The response is decrypted using a copy of the same global RC4 state buffer, then the CRC32 of all but the last 4 bytes is checked to equal the checksum stored in the last 4 bytes. The buffer without the checksum is decompressed using the *`nrv2b`* compression algorithm (algorithm here). The decompressed data is a payload buffer like the one read from the files referenced in the persistence structure retrieved above, containing the Terdot DLLs and other files.

1.4.1. Terdot injection and exploitation mechanism

Once a payload buffer has been decoded, the appropriate malware DLL is injected into the system *`explorer.exe`* process. If the host is running a 32-bit version of the operating system, the malware opens the *`explorer.exe`* process, loads and writes the 32-bit version of the downloaded malware DLL, then runs the DLL export named `_Run@4`.

On a 64-bit operating system, the malware (running as a 32-bit process) needs to use an exploit to access the 64-bit version of the WIN32 API. It uses the "***Heaven's Gate***" exploit, which works as follows:

- temporarily changing the `CS` (code segment) register to `0x33`, which means that the machine code corresponds to x86-64 instructions;
- running x86-64 code;
- changing the `CS` register back to `0x23`, which means that the machine code is interpreted as x86 instructions.

By leveraging this exploit, the malware can call the 64-bit versions of *`NtAllocateVirtualMemory`*, *`NtWriteVirtualMemory`*, *`LdrLoadDll`* ***and*** *`CreateRemoteThread`* to load the 64-bit version of the downloaded malware DLL in *`explorer.exe`*'s address space and run the DLL export named `Run` in a new thread inside the system *`explorer.exe`* process. This exploit is necessary because, under Windows32-on-Windows64 (WOW64), 32-bit processes have limited access to 64-bit processes such as the system's *`explorer.exe`*.

If, for some reason, the injection into the system *`explorer.exe`* process fails, the malware runs the 32-bit malware DLL inside the address space of the current process (the overwritten *`explorer.exe`*, not the original, system *`explorer.exe`*). First the malware runs the DLL entry point, then the `_Run@4` export in another thread.

In any situation, when the malware runs the DLL export, the new thread receives as a parameter a buffer containing a BaseConfig structure, the original malware path, the start address of the DLL image, the downloaded structure and the data inside the file mapping. If the DLL is

running inside another process, this buffer is first written inside that process' memory, and its pointer is passed to the program.

If the injection is successful, the boostrapper proceeds to sleep in an infinite loop to allow the threads to continue. If the injection is not successful, the malware retries after 10 seconds.

```
mzStart = (char *)loadMzToMemoryAlignment((char *)downloaded->client32Start);
if ( mzStart )
{
  hExplorer = openExplorer();
  if ( hExplorer )
  {
    fullDownloadBuffer = (char *)VirtualAllocEx(hExplorer, 0, downloaded->srcLen, 0x3000u, 4u);
    if ( fullDownloadBuffer )
    {
      srcLen = downloaded->srcLen;
      NumberOfBytesWritten = 0;
      if ( WriteProcessMemory(
             hExplorer,
             fullDownloadBuffer,
             (LPCVOID)downloaded->source,
             srcLen,
             &NumberOfBytesWritten) )
      {
        if ( NumberOfBytesWritten == downloaded->srcLen )
        {
          mzImgSize = getSizeOfImage((int)mzStart);
          injectMzBuffer = VirtualAllocEx(hExplorer, 0, mzImgSize, 0x3000u, 0x40u);
          if ( injectMzBuffer )
          {
            NumberOfBytesWritten = 0;
            if ( WriteProcessMemory(hExplorer, injectMzBuffer, mzStart, mzImgSize, &NumberOfBytesWritten) )
            {
              if ( NumberOfBytesWritten == mzImgSize )
              {
                explorerPayloadBuf = VirtualAllocEx(hExplorer, 0, 0xCECu, 0x3000u, 4u);
                if ( explorerPayloadBuf )
                {
                  getRunConfiguration(&payloadBuf);
                  payloadBuf.mzStart = (int)injectMzBuffer;
                  payloadBuf.downloadedBuffer = fullDownloadBuffer;
                  payloadBuf.downloadedLen = downloaded->srcLen;
                  payloadBuf.flags = flags | 0x10;
                  if ( updateMapping )
                    memcpy(payloadBuf.updateMappingData, (const void *)updateMapping, 0x4B0u);
                  NumberOfBytesWritten = 0;
                  if ( WriteProcessMemory(hExplorer, explorerPayloadBuf, &payloadBuf, 0xCECu, &NumberOfBytesWritten)
                    || NumberOfBytesWritten != mzImgSize )
                  {
                    runExport = findExport(mzStart, "_Run@4");
                    if ( runExport )
                    {
                      threadCreated = 0;
                      if ( CreateRemoteThread(
                             hExplorer,
                             0,
                             0,
                             (LPTHREAD_START_ROUTINE)((char *)injectMzBuffer + runExport - mzStart),
                             explorerPayloadBuf,
                             0,
                             0) )
                        threadCreated = 1;
                    }
```

Fig. 8: Analyzed pseudocode of injection in system 32-bit explorer.exe process

2. Downloaded payload (the Terdot DLL)

The Terdot DLL can be executed in three different ways:

1. It can be statically imported; in this case, the `DllMain` is run, which creates a new thread. This thread calls the `init` function and then continues with host-specific behavior. We haven't seen this method used in the wild.

2. It can be run by calling the `_Run@4` (on the 32-bit DLL) or `Run` (on the 64-bit DLL) exported functions; this happens only in first-stage runs. Depending on input arguments, these functions can process relocations and imports and call `DllMain`. These steps are necessary if the DLL is injected inside the system `explorer.exe` process, but, if the malware runs inside the bootstrapper process, these steps are performed by the bootstrapper itself. In either case, processing continues with the initial configuration process after this call.

3. It can be run by calling the `__injectEntryForThreadEntry@4` (on the 32-bit DLL) `_injectEntryForThreadEntry` (on the 64-bit DLL) exported function; this happens only in second-stage injections. This function processes relocations and imports, saves the `BaseConfig` and persistence structures received as arguments, and runs `DllMain`. The behavior of `DllMain` in this case is identical to the static import case.

2.1. Initial configuration

For initial configuration during a first-stage run, the Terdot DLL saves the `BaseConfig` structure received as a parameter from the bootstrapper. It is stored as a global variable XORed with `0x15`. The configuration process subsequently follows 3 separate paths, depending on how the bootstrapper received the Terdot DLLs:

1. If the Terdot DLL was downloaded from the C&C server, first the `init` method is called, which initializes the global variables. Then the malware starts generating the persistence structure. To accomplish this goal, it creates:

- a random subkey inside `HKCU\Software\Microsoft` and 3 random registry values inside it: two of them used to store configuration information, and one unused
- the computer ID, main drive CLSID, a new RC4 state buffer from a random key, TCP port numbers for the HTTP and HTTPS spyware proxy, mutex name seeds, spyware log file key and autorun entry name
- 3 random filenames, each inside a random (different) subdirectory of `AppData`:
    - One of these files is used to store the original malware sample.
    - The second file is used to store the downloaded structure containing the Terdot DLLs and extra files (padded with random bytes and encrypted with the newly generated RC4 state buffer).
    - The third file is left empty and is used to store the spyware logs before being sent to the C&C server.
- an X.509 certificate, used in the spyware MITM process.

2. If the Terdot DLL was loaded from the file containing the decrypted payload structure object, and this filename was found inside the persistence structure in the registry, only `init` is called.

3. If the source of the Terdot DLL payload is the file mapping, the current Terdot DLL is the updated version of another Terdot instance running on the same machine, and was created and executed by it.

In this case, the file mapping also contains two event names; these are used to control the operation of the Terdot malware. The first one is the global shutdown event, which is checked throughout the malware code; signaling this event will cause the older version instances to shut down. The other event is set by the malware before exiting; the newer version waits for this event to be signaled to continue the configuration process.

2.2. The init function

The `init` function initializes globals and the OpenSSL library. It gathers identifying information about the execution environment, including:

- `user SID`
- `session ID`
- `main drive CLSID`
- `Windows version`

- current process authority
- process module name

If the malware is not running at (at least) a medium mandatory integrity level (determined through `GetTokenInformation`), the malware shuts down. This mandatory integrity level is given to processes started by regular users; a low integrity level, sometimes given to sandbox untrusted applications, would mean that the malware does not have enough rights to perform its tasks. Also, using a mutex, if another first-stage instance of the same malware campaign is running at the same time, the second instance waits 60 seconds, then exits.

Terdot creates generic security descriptors that disable auditing for their corresponding objects. With these security descriptors, it creates events used in thread and process synchronization and malware shutdown. The events' names are generated as to be highly specific to a machine and user.

Each class of host processes has a very specific role in the execution of the malware:

- the `explorer.exe` process is responsible for injecting into other hosts and ensuring that the autorun entry is not removed. Also, when injected in `taskeng.exe`, `ctfmon.exe` or `dwm.exe`, this DLL will fulfill the same role, namely to make the cleanup process more difficult.
- an `msiexec.exe` process is responsible for running the MITM proxy, sending the spyware logs to the C&C server, and downloading updates and other commands.
- browser processes are responsible for redirecting traffic to the MITM proxy.
- In addition, first-stage injections running under WOW64 (that is, 32-bit processes on a 64-bit system), fulfill both the `explorer.exe` and the `msiexec.exe` roles.

These roles, however, will not be assigned until later in the execution stage.

2.3. Host-specific behavior: `explorer.exe`, first-stage run

If the DLL is running inside an `explorer.exe` process or one of `taskeng.exe`, `ctfmon.exe` or `dwm.exe` (injected by the first-stage run), it creates 3 threads:

- **the injector thread**
  - tries to inject the same Terdot DLL in every valid host process (`explorer.exe`, `taskeng.exe`, `ctfmon.exe`, `dwm.exe`, `msiexec.exe` as well as browser processes)
  - for every candidate, it checks if it hasn't already injected the DLL in that process, using a list of already injected PIDs, as well as global mutexes created on each successful injection
  - if no checks fail, the DLL export named `__injectEntryForThreadEntry@4` is called with BaseConfig and the persistence structure as parameters
- **the autorun thread**
  - checks if the original sample file is deleted and the autorun entry is removed; in that case the autorun entry (in the registry) is restored with a different name. This is **probably a bug in the malware**, since, at that point, the file set to autorun doesn't exist anymore.
- **the msiexec runner thread**
  - responsible for starting an `msiexec.exe` host process for the Terdot DLL to be injected into.

Every thread runs in a loop until the global shutdown event has been signaled, at which point the thread closes. Also, every thread first waits for a specific, different mutex before starting; this ensures that no two hosts with the same role will perform overlapping actions, and that, if one host is terminated (by antimalware software, for example), another one can take its place.

2.4. Host-specific behavior: msiexec.exe, second-stage run

This step in the malware lifecycle is reached only if an infected `explorer.exe` process or a second-stage host injects the Terdot DLL inside an `msiexec.exe` host. In this case, four threads are started:

- Two of these threads run the spyware proxy, one for HTTP and one for HTTPS.
- A thread uploads the spyware logs to the C&C servers.
- A thread downloads configuration and malware updates and executes other commands sent by commanding servers.

Like in the case of `explorer.exe`-specific behavior, every thread first waits for a specific, different mutex before starting.

2.5. Host-specific behavior: browser processes, second-stage run

As with `msiexec.exe` injections, this step is reached only after an injection by an `explorer.exe` process or one with the same role. The created thread hooks functions inside the browser processes, to facilitate the MITM attack.:

- it hooks `Crypt32:CertVerifyCertificateChainPolicy` to always return true on SSL connections and `Crypt32:CertGetCertificateChain` to signal a valid certificate chain so that the user will not be warned about the MITM attack

- It hooks `ntdll:ZwDeviceIoControlFile` to redirect the traffic to its MITM proxy. This method is usually called to send a control code (`IOCTL`) to a device driver. However, behind the scenes, Windows implements socket operations as `IOCTL`s sent to the `afd.sys` driver. By hooking this method, the malware can intercept any socket functions for this browser process

  This hook only takes effect if all of the following conditions are met:

  - the `IOCTL` is either `0x12007` (corresponding to `Connect`) or `0x120C7` (corresponding to `SuperConnect`)
  - the port is either `80` or `443`
  - the IP is not inside the `192.0.0.0/8` or `127.0.0.0/8` subnets
  - the current process name is `iexplore.exe`, `microsoftedgecp.exe`, `chrome.exe`, `opera.exe`, `firefox.exe` or `WebKit2WebProcess.exe`

  If any of the conditions are not met, the hook simply calls the original `ZwDeviceIoControlFile` function. If all the conditions are met, however, the destination IP address is changed to the `127.0.0.1` and the port is changed to one of the ports inside the persistence structure. Then, the original function is called with the new arguments. If it succeeded, this socket is now connected to the spyware proxy server that runs inside the `msiexec.exe` process

2.6. Downloading configurations, updates and commands

This feature runs inside the `msiexec.exe` process, inside its own thread.

First, this thread tries to download the configuration data container (Fig. 9). It first checks if it doesn't already exist in the registry. If it doesn't exist, it must be downloaded. The URLs are retrieved from the BaseConfig structure (see Appendix 1 for values). A simple GET request is done; the malware retries 6 times with each URL, 3 times respecting and 3 times ignoring the proxy settings, sleeping 5 seconds between attempts. If the global shutdown event is signaled, the download process stops. If the download with all URLs is unsuccessful, the download process restarts.

On a successful connection to a C&C server, the malware receives an encrypted data container. The malware first verifies its authenticity; the last 256 bytes are a RSA signature of the SHA-256 hash of the rest of the encrypted container. The public key from the BaseConfig structure is used to check the signature. If the signature matches, the data container is decrypted. The data is re-encrypted, this time, using the RC4 state buffer from the persistence structure, then stored in the registry. Also, the CRC32 of the downloaded data is saved into the main data container.

The next action is to download updates and commands. The URL from which these are downloaded can be found in the configuration data container. The server expects a POST request containing an encrypted data container with:

- a unique computer identifier
- the malware version
- CRC32s of downloaded data
- system information (Windows version, processor architecture)
- system language
- network adapter IPs

Once generated, the data container is encrypted with the BaseConfig RC4 state buffer and sent to the URL mentioned above. The malware tries to POST the data three times or until the global shutdown event is signaled. If the POST succeeds, the C&C server responds with a data container encrypted with the same RC4 state buffer.

The downloaded data container can include new configuration data or updates to the original sample of the malware (Fig. 2) (the executable file data is written to a random file in the temporary directory with the `.exe` extension). The file mapping used by the bootstrapper in the update process is written, containing the following fields:

malware version	global shutdown event name	update complete event name	persistence registry sub-key name	persistence structure
4 bytes	256 bytes	256 bytes	16 bytes	668 bytes

Once the mapping has been written, the malware starts the updated executable.

Regardless of success, the malware will save the update timestamp to the main data container (to prevent another update within 60 seconds).

If the malware can't download updates and commands, it tries alternative sources to download the configuration data container from. There are two other sources for this download: a list of URLs in the existing configuration container (sent by the C&C server as successors to it) and URLs generated using a DGA.

The original thread will sleep for either 12 minutes if the download of updates and commands succeeded, or 2 minutes if it failed. Also, if the global shutdown event is signaled, the thread closes. Otherwise, after the timeout, the malware will restart this procedure.

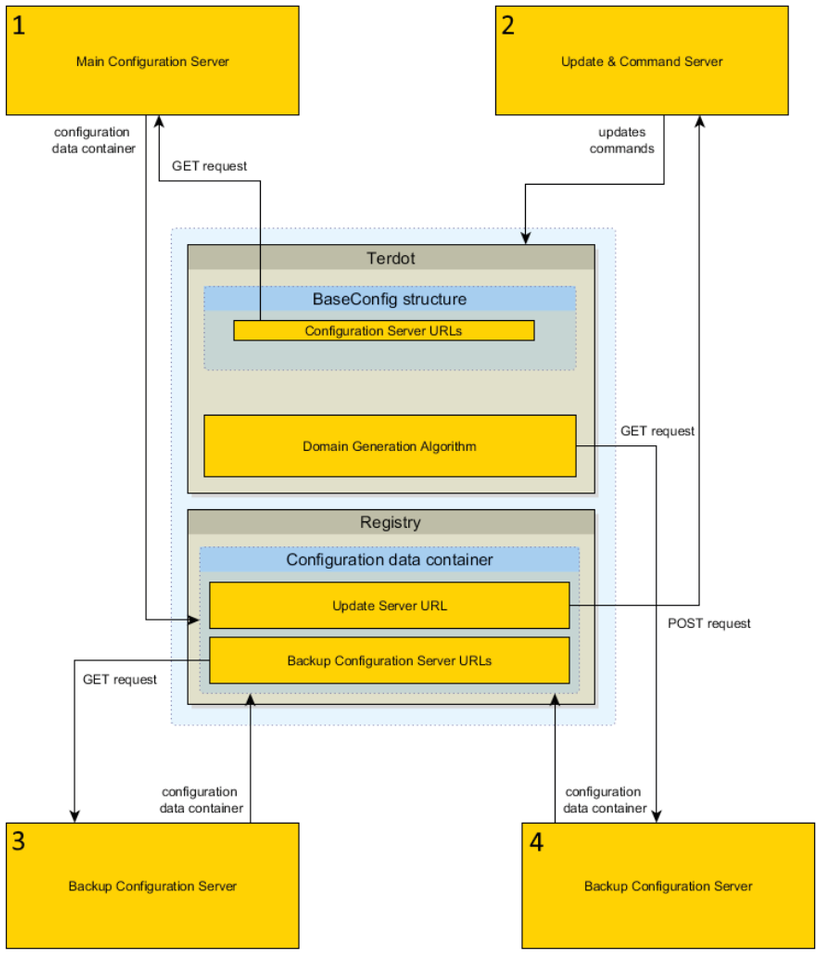

Fig. 9: Visual representation of the update download algorithm

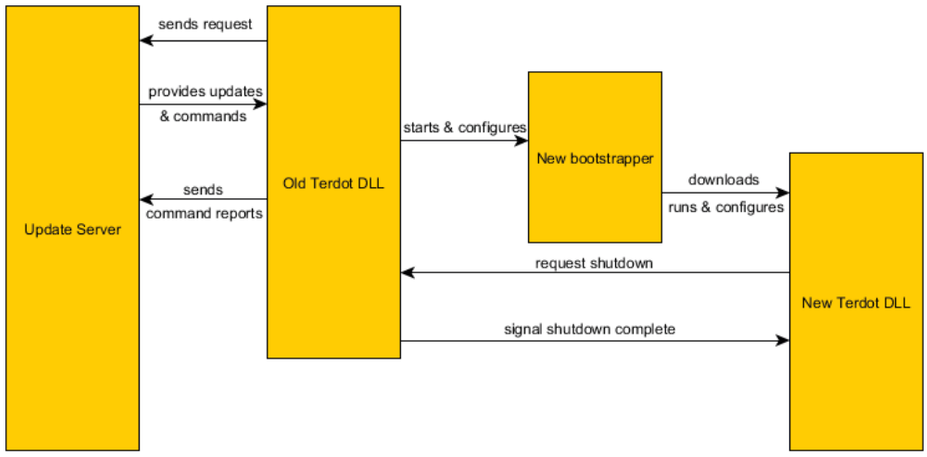

Fig. 10: Visual representation of the update process

2.7. Command execution

This feature runs in its own thread, injected into an `msiexec.exe` process. It receives a downloaded data container as a parameter, where specific commands are sent by the C&C server. The commands are grouped in scripts. Each script is preceded by a 16-byte unique identifier. In the main data container, the processed identifiers are kept as a concatenated list; if the identifier already exists, the script is skipped, otherwise the identifier is saved. If this is a new script, the rest of the buffer is converted from UTF-8 to UTF-16 and then executed. Each script consists of multiple newline-separated commands.

Valid commands are:

- `bot_uninstall` (no arguments): commands the malware to uninstall itself after all scripts have been executed.
- `user_execute file [arguments]`: the file is run using the specified arguments, if they exist. If the filename contains environment strings (like `%APPDATA%` or `%WINDIR%`), these are first expanded to folder paths.
- `user_execute url`: a simple GET request is executed using the URL passed as argument; up to 3 tries are done over HTTP and 3 over HTTPS; the downloaded file is run with the same procedure as above.
- `bot_httpinject_disable {URLs}`: these URLs are added to a list that signals the MITM spyware proxy to disable injection for them.
- `bot_httpinject_enable {URLs}`: these URLs are removed from the injection disable list mentioned above.
- `user_url_block {URLs}`: these URLs are added to a list that signals the MITM spyware proxy to block all access to.
- `user_url_unblock {URLs}`: these URLs are removed from the blocking list mentioned above.

The URL lists (both the injection disable and blocking lists) are stored in the main data container as concatenated lists of null-terminated strings, followed by an extra null byte.

After all commands in a script have been executed, a report is prepared to be sent to the C&C server. This report is a data container identical to the one POSTed to the C&C server to download the updates and commands. Apart from these fields, this report data container includes the following fields:

- script identifier
- is error (either 0 or 1)
- error message

If no error occurred, the returned error message is `OK`; otherwise, the returned errors are any of the following, but not limited to these:

- `Script already executed.`
- `Failed to execute command at line %u.`
- `Unknown command at line %u.`

After all scripts have been executed, if the malware was ordered to uninstall, it executes the following steps:

[19]

- it signals the global shutdown event to close all Terdot threads in all hosts.
- it signals an event to inform the `msiexec.exe` runner thread (injected in an `explorer.exe` process) not to terminate this process; this is necessary because the uninstall procedure hasn't finished.
- after sleeping for 5 seconds, it removes all created files and directories in the `AppData` directory.
- it removes all traces in the registry, including the autorun entry.
- finally, it terminates the host `msiexec.exe` process.

If the malware wasn't commanded to uninstall, the thread in which this procedure is running finishes execution.

2.8. The Domain Generation algorithm explained

Pseudorandomly generated domains are another way to find the locations for downloading the configuration data containers; these are only used if the server responsible for updates, commands and receiving the spyware logs is down, and all alternative sources from configuration data are also down.

This algorithm checks 128 domains generated from the current day, month and year. No lookback or lookahead is used, so a domain registered by the malware creators can only be used for one day.

A seed is generated from the current year, month and day by converting the concatenation of these variables (as strings, in base 10, without leading zeroes) to an integer (for example, the date `2016-12-01` would create the seed `2016121`). Then, 128 16-character long domain names are generated by iterating in the following way:

```
value = domainIndex + characterIndex + ((value >> 24) & 0xFF | (value << 8)) + 0x2AB3FEA3
domains[domainIndex][characterIndex] = abs(value % 25) + 'a'
```

The top-level domain is always `.com`. For each domain that is up, the malware tries to download the configuration from the URL "`{domain}/{mapping_name}.bin`" where `{mapping_name}` is the value of the corresponding field in the BaseConfig structure. For each URL, the malware first tries the download over HTTPS, then over HTTP. The download algorithm is a simple GET request, retried up to 6 times, alternatively respecting and ignoring the system proxy settings.

Regardless of success, the DGA continues until either the global shutdown event has been signaled, or the event indicating that updates and commands have been successfully downloaded, is set.

3. Web traffic interception

The malware has the capability to intercept all browser traffic, including HTTPS traffic, by forging SSL certificates. It saves specific information (including credentials and banking information), it injects custom HTML and Javascript into the victims' webpages, and it sends saved data to C&C servers.

3.1. The spyware MITM proxy server

When running inside an `msiexec.exe` process, Terdot starts two threads, each running a transparent proxy server. The former is responsible for HTTP connections, and the latter handles HTTPS connections. Each thread listens on the `localhost` address, on the corresponding port saved in the persistence structure.

If the server is proxying HTTPS traffic, it first must test the saved X.509 root certificate (used in the process of forging the SSL certificates) and add it to the system's list of trusted certificate authorities.

For the MITM process to be effective, the SSL trust chain must appear correct to the user's browser. For Internet Explorer, this is achieved by hooking Crypt32.dll functions to always return true, but Mozilla Firefox uses its own certificate store. Terdot scans for Firefox's certificate store; if the certificate to be added is not yet in the store, it terminates every `firefox.exe` process, then uses an application received alongside the Terdot DLL to add it.

The `certutil.exe` file found in this extra data is run with the following arguments:

```
certutil.exe -A -n "{random name}" -t "C,C,C" -i "{X.509 certificate filename}" -d "{cert8.db directory}"
```

There are strong indications that **this executable is part of Mozilla's NSS Tools package**, and most antivirus solutions report it (and accompanying DLLs) as clean. This command adds the root certificate to Mozilla Firefox's store, which ensures no security warnings will ever be presented to the user.

The spyware proxy server creates an IPV4 socket and binds it to the `localhost` address on the specified port, listens to it, and sets it to nonblocking (to prevent the application or the server from hanging waiting for data). Once a client has connected to it, the proxy accepts the socket then creates a new worker thread (up to 512 threads at a time for each running proxy) responsible for handling the connection. The proxy server keeps processing, forwarding and spying on data until either the connections are closed, or the global shutdown event has been raised.

The malware reads the HTTP request sent by the hijacked client. It does this in a loop, reading up to 65535 bytes from the socket, then feeding these bytes into a finite automaton responsible for parsing the HTTP headers and separating the content. These bytes are also saved into a full request / response buffer, reallocating it if necessary.

If the client sends a HTTP UPGRADE request, or a `Content-Type` containing `multipart/form-data`, the proxy server is unable to process other protocols or multipart encoded forms, and defaults to a simple passthrough from client to server and back. The malware will read from one end and write to the other until one of the connections is closed or the global shutdown event is signaled.

Otherwise, information is extracted from the HTTP request and (if it is relevant enough), it is saved to a file to be later sent to the C&C server. Afterwards, if the logging procedure hasn't signaled that the URL must be blocked (and not reach the victim's browser), the proxy server continues with gathering injections.

The proxy thread continues until the global shutdown event has been raised or until the connection is broken.

3.2. Logging the intercepted information

To gather information to send to the C&C servers, the malware extracts data from the requests. Information such as hostname, referrer, content type, user agent, cookies, content as well as the username and password from the `Authorization` header field, if it exists, are part of the spyware packets. However, **the username and password parsing procedure contains a bug** (detected and optimized by the compiler), since the parsing function is always called with a NULL pointer as the argument instead of the header value string.

To find the data to log, Terdot uses a configuration block found in the configuration data container. This configuration block is structured as a null-terminated list of null-terminated strings. Each string is a wildcard pattern prefixed by a character that signifies the action that the malware should take on URLs matched by that pattern (e.g. `#` matches any character, `*` matches zero or more characters). The first character of every string can be one of the following:

- **!** signals that the URL must be ignored by logging (excluded from the spyware logs).
- **–** or **^** signal that the server must block these URLs (they should not be forwarded to the victim), but keep them in logs.
- **@** signals that the URL may contain usernames or passwords, and these should be extracted and logged. However, this feature doesn't seem to be properly implemented.
- other characters signal that the log packet needs to be kept regardless of other circumstances

Below are some details about the actions taken depending on the request type and parameters:

- If the blocked URL list in the main data container includes a pattern matching the current URL, this connection is also marked to be blocked.
- If the content type header begins with *`application/x-www-form-urlencoded`*, this connection is marked to contain urlencoded forms.
- If a username and a password were saved at an earlier step, these are saved to a plaintext buffer to be logged later.
- If no wildcard pattern that matched the URL marked the packet to be kept, the request doesn't contain login credentials and the request is either a HTTP GET or has no body, it is marked to be discarded; otherwise its data is saved.
- If this request contains urlencoded HTTP form data, "+" characters in the request body are changed to spaces; if the request has any other `Content-Type` values, no changes are made, and if the `Content-Type` header field is missing, but there is content, the saved request body is replaced with `*UNKNOWN*`; if there is no HTTP request content, `*EMPTY*` is saved.

Terdot keeps the information gathered from spyware activities in a special file inside the `AppData` directory. The name of this file is kept in [21]

the persistence structure. Inside this file, this data is saved as a series of blocks, stored in chronological order, each in the following format:

data length	already uploaded	data
4 bytes	1 byte	variable

The data length is stored XORed with a value saved in the persistence structure. The data is stored as a data container encrypted with the persistence RC4 key, and is uploaded in a separate thread running inside an `msiexec.exe` host.

For each block in the file that doesn't have the 5th byte set (meaning that this block was not uploaded), the length is extracted and the data container is decrypted, then re-encrypted using the BaseConfig RC4 key and sent to the URL stored in the configuration data container. This is a simple POST request, using the same algorithm as the one used to download updates. In the end, the block is marked as read using the dedicated byte in the header.

Using this procedure, the blocks in the spyware log file are uploaded as data containers, one by one. After all of them have been successfully uploaded, the file is deleted, and the uploading process restarts (possibly uploading from a file with new data), after a timeout equal to 2 minutes on success or 16 minutes on failure.

Like most features of the malware, this thread is wrapped in its own mutex and terminates if the global shutdown event has been signaled.

2.3. The HTML injection component

Beyond sniffing and logging information, Terdot is capable of injecting custom HTML/Javascript into the servers' response. The URLs that Terdot is configured to inject HTML code in are saved in the configuration data container. This data is saved as a list of entries, each in the following format:

flags	entry length in bytes	URL pattern start offset	unknown	request ignore pattern offset	request match pattern offset	injection limiter pattern offset	response content pattern offset
2 bytes	4 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes

For an injection to be performed:

- the URL pattern must match the request URL,
- the request ignore pattern (if it exists) must not match
- the request match pattern (if it exists) must match the request content,
- the flags field must have either bits 1 or 3 set.

Also, the URL must not be in the list of URLs that the injection is disabled for.

Since the responses don't hold the URL, in the HTTP request phase all injections are checked and saved if all prerequisites are met. In the HTTP response phase, the injection can still be disabled based on the `Content-Type` header value.

The malware continues with the next rule in the injection buffer. After all the blocks have been processed, the new content is saved to be forwarded to the victim's browser.

4. Conclusion

Terdot is a complex malware, building upon the legacy of ZeuS. Its modular structure, complex injections and careful use of threads make it resilient, while its spyware and remote execution abilities make it extremely intrusive.

As described in this paper, Terdot goes above and beyond the capabilities of a Banker Trojan. Its focus on harvesting credentials for other services such as social networks and e-mail services could turn it into an extremely powerful cyber-espionage tool that is extremely difficult to spot and clean.

As a consequence, strong antimalware tools are necessary to protect PCs from these dangers.

Appendix

Appendix 1. BaseConfig structure values (tallied)

Other values exist; these are found in bootstrappers dropped by 114 different samples (in total 49 different bootstrappers). For some fields such as URLs, some samples hold more than one value.

Initial download URL:

URL	Count
https://piroco.com/b.bin	16
https://stream.vanindsdb.com/b2	15
https://stream.zoeltwsaw.com/b2	15
https://stream.salmftw.com/b2	15
https://stream.glawdfhow.com/b2	15
https://stream.gizdosales.com/b2	15
http://ubyrttgf.click/main/file/rembt.bin	13
http://cxqytmtflyj.click/main/file/rembt.bin	13
http://vfdyth.click/main/file/rembt.bin	12
http://ubyrttgrdf.bid/main/file/rembt.bin	12
https://obesca.com/b.bin	9
https://stream.gizdosales.com/f/z.php?f=b1	9
http://newstodayinasia88.asia/00/b.bin	8
http://resdomactivationa.asia/00/b.bin	6
http://ubyrbfghghvh.click/main/file/rembt.bin	5
http://ubyrbtrgcc.click/main/file/rembt.bin	4
https://pirococo2.com/b2.bin	4
https://pirococo3.com/b2.bin	4
https://piroco.com/b2.bin	4
https://login.webmail.com.am/styles/webmail_logo.png	3
http://resdomactivationa8.asia/00/b.bin	3
http://yidckntbrmhuuhmq.com/00/b.bin	3
http://cxqyvqjtmtflyj.com/main/file/rembt.bin	3
http://210.211.108.163/~chobt/images/temp/01tdu.bin	2
http://10cyberprojects2016.asia/b.bin	2
http://soelxtmj.click/main/file/rembt.bin	2
http://gegbghtyg.eu/main/file/rembt.bin	2
http://10cyberprojects20161.asia/00/b.bin	2
https://richardbbenoit.com/ttt/b42363b.bin	2
http://masterhost1333.asia/b.bin	2
http://masterhost8981.asia/b.bin	2
http://comodotrl.com/cgi/b59005b.bin	2
https://intaxcdn.am/2016/enterprise.pdf	2
http://remembermetoday4.asia/00/b.bin	2
https://clork.ru/xen/settings1.bin	1
http://startupproject33677.asia/00/b.bin	1
http://118.193.16.24/cwhht.com/cache/db/00b.jpg	1
http://210.211.108.163/~chobt/images/temp/mbb.jpg	1
http://axe.maintop.top/bot.bin	1
https://stream.afnfoundation.cf/f/z.php?f=b1	1
http://ausecurposcom.com/cxx/b18.bin	1
https://banneradspr.xyz/ipd.bin	1

[24]

URL	Count
https://wartelio.top/dpr.bin	1
https://prisectos.top/dpr.bin	1
https://intaxcdn.am/contents/enterprise.pdf	1
http://210.211.108.163/~chobt/images/temp/01t.bin	1
https://ugaugacongo.ru/xen/settings2.bin	1
http://startupproject20166.asia/b.bin	1
https://dayspirit.at/xen/settings.bin	1
http://multifacto.com/cutt/b550104j.bin	1
http://zompokrtut.ru/bpst.bin	1
https://deatroleo.xyz/dpr.bin	1
https://aspecto.top/dpr.bin	1
https://valinados.top/dpr.bin	1

Configuration server URL:

URL	Count
https://piroco.com/c.bin	16
https://stream.vanindsdb.com/c2	15
https://stream.zoeltwsaw.com/c2	15
https://stream.glawdfhow.com/c2	15
https://stream.salmftw.com/c2	15
https://stream.gizdosales.com/c2	15
http://cxqytmtflyj.click/main/file/applecfg.bin	13
http://ubyrttgrdf.bid/main/file/applecfg.bin	12
http://vfdyth.click/main/file/applecfg.bin	12
http://ubyrttgf.click/main/file/applecfg.bin	12
https://obesca.com/c.bin	9
https://stream.gizdosales.com/f/z.php?f=c1	9
http://newstodayinasia88.asia/00/c.bin	8
http://resdomactivationa.asia/00/c.bin	6
http://ubyrbfghghvh.click/main/file/applecfg.bin	5
https://pirococo2.com/c2.bin	4
http://ubyrbtrgcc.click/main/file/applecfg.bin	4
https://pirococo3.com/c2.bin	4
https://piroco.com/c2.bin	4
https://login.webmail.com.am/styles/buttons.png	3
http://yidckntbrmhuuhmq.com/00/c.bin	3
http://cxqyvqjtmtflyj.com/main/file/applecfg.bin	3
http://resdomactivationa8.asia/00/c.bin	3
http://10cyberprojects20161.asia/00/c.bin	2
http://remembermetoday4.asia/00/c.bin	2
http://comodotrl.com/cgi/c64374i.bin	2
http://masterhost8981.asia/c.bin	2
http://soelxtmj.click/main/file/applecfg.bin	2
http://masterhost1333.asia/c.bin	2
http://gegbghtyg.eu/main/file/applecfg.bin	2
http://210.211.108.163/~chobt/images/temp/01ttdunews.bin	2
https://intaxcdn.am/2016/personal.jpg	2
https://richardbbenoit.com/ttt/c3769837c.bin	2

URL	Count
http://10cyberprojects2016.asia/c.bin	2
https://banneradspr.xyz/stat.bin	1
http://zompokrtut.ru/ctim.bin	1
https://aspecto.top/dsr.bin	1
http://startupproject20166.asia/c.bin	1
https://dayspirit.at/xen/config.bin	1
https://intaxcdn.am/contents/personal.jpg	1
http://startupproject33677.asia/00/c.bin	1
http://ubyrttgf.click/main/file/rembt.bin	1
http://multifacto.com/cutt/c112683e.bin	1
http://axe.maintop.top/config.bin	1
https://prisectos.top/dsr.bin	1
https://valinados.top/dsr.bin	1
https://stream.afnfoundation.cf/f/z.php?f=c1	1
http://118.193.16.24/cwhht.com/cache/db/00c.jpg	1
http://210.211.108.163/~chobt/images/temp/01tt.bin	1
http://ausecurposcom.com/cxx/c23.bin	1
http://210.211.108.163/~chobt/images/temp/cb.jpg	1
https://deatroleo.xyz/dsr.bin	1
https://clork.ru/xen/config1.bin	1
https://wartelio.top/dsr.bin	1
https://ugaugacong.ru/xen/config2.bin	1

Persistence registry key & value

(formatted as key>value; these are located under `HKCU\Software\Microsoft\`.)

Key>Value	Count
ocisyxvu>mev	16
qaunno>pawyhe	14
qoxe>vio	9
tieni>oqko	6
eqduyb>doe	4
uhtil>vuvihu	4
oxiru>doo	4
idloet>atq	3
emids>haz	3
coiqd>goup	3
apdoob>avna	2
kuidy>fufyo	2
iflyb>ykem	2
axtohyo>kyn	2
wiev>yzy	2
neecry>ovo	2
bakoagxe>heu	2
vizyev>zygeuw	2
kyfaifym>haf	2
iflyil>qup	2
enez>radyq	2
loci>weehre	1

[26]

Key>Value	Count
seaqavyh>hoed	1
zoozgat>beyvdi	1
uhapin>byoq	1
ansy>enu	1
recoy>uwloty	1
kaezegid>liap	1
ciadop>meukma	1
irxy>gaxep	1
tukur>xixapa	1
ewerboc>uwo	1
laevwoa>hok	1
igabwym>ulsali	1
ygwapo>ihy	1
gosup>pyy	1
xaylzyf>tufu	1
debe>ivot	1
seuxxiih>cyydq	1
oszayl>azyxe	1
uqab>tapik	1
ceuxp>yntaeg	1
efeqywpo>xoge	1
myduate>ykpo	1
aqneireh>ury	1
fozyxao>azyp	1
siefud>hyca	1

Update file mapping:

Name	Count
shared_TEST1	35
shared_my_de	20
shared_jh3ghjT4Fj42Rv	17
shared_AX01	15
shared_update_64	4
shared_SH1	4
shared_tabooboy	3
shared_tempt	3
shared_my_botnet	2
shared_IRS2016	2
shared_fucker	2
shared_torture	2
shared_mone	1
shared_bbb01	1
shared_catcher1	1
shared_rude	1
shared_znYD5cwHW7atoUt	1

Appendix 2: Artifacts that tell an infection

- Unexpected registry keys under `HKCU\Software\Microsoft`:
  - one registry key with one value (see list in Appendix 1)
  - one registry key (random capitalized name, length 4-6 characters) with 1-3 values (random capitalized names, length 4-9 characters)
- 3 unexpected directories in the Application Data directory (random capitalized names, length 4-6 characters), each with a file (random lowercase name, length 4-5 characters)
- An autorun entry (saved in the registry, random name, length 5-12 characters) to one .exe file in one of the Application Data directories above
- Running processes:
  - extra `explorer.exe` process immediately after boot (shutting down after a few seconds)
  - extra `msiexec.exe` process (long-lived)
- Requests to certain domains and IPs (see lists in Appendix 1)
- While browsing in Mozilla Firefox:
  - unexpected crash of the browser once, at the time of infection
  - all HTTPS websites using certificates signed by the same root Certificate Authority (random words in the C, O and CN fields, capitalized, length 6-14 characters)
  - an unexpected certificate in Mozilla Firefox's trusted Certificate Authorities list – see under Options-Advanced-Certificates-View Certificates-Authorities
  - slightly slower than expected Internet connection (due to the spyware proxying)

Appendix 3: Victimology

Our internal telemetry systems reveal a vast spreading area for the Terdot samples. Most computers infected with Terdot are currently in Australia, the United Kingdom and the United States. A detailed breakdown per country is shown in the table below:

Country	Count
Australia	428
United Kingdom	130
United States	74
Germany	63
India	39
Bangladesh	36
Philippines	35
China	22
Malaysia	22
Romania	17
United Arab Emirates	16
Nepal	14
Indonesia	14
Brazil	11
Tunisia	10
Pakistan	8
Ghana	8
Mexico	8
Nigeria	7
Papua New Guinea	6
Austria	6
Russia	5
Netherlands	5
Canada	5
Singapore	5
France	4
Hungary	4
Laos	4
Iran	4
Spain	4
Switzerland	3
Colombia	3
Italy	3
Vietnam	3
Morocco	3
Cambodia	3
Ukraine	3
Sweden	3
Bhutan	2
Belarus	2
Costa Rica	2
Kenya	2

Country	Count
Uganda	2
Belgium	1
Bosnia and Herzegovina	1
Barbados	1
Japan	1
Hong Kong	1
Honduras	1
Rwanda	1
Portugal	1
Serbia	1
Taiwan	1
Sri Lanka	1
New Zealand	1
Thailand	1
Venezuela	1
Cameroon	1
Ecuador	1
Sao Tome and Principe	1
Qatar	1
Algeria	1

Bitdefender is a global security technology company that delivers solutions in more than 100 countries through a network of value-added alliances, distributors and reseller partners. Since 2001, Bitdefender has consistently produced award-winning business and consumer security technology, and is a leading security provider in virtualization and cloud technologies. Through R&D, alliances and partnership teams, Bitdefender has elevated the highest standards of security excellence in both its number-one-ranked technology and its strategic alliances with the world's leading virtualization and cloud technology providers. More information is available at http://www.bitdefender.com/

BD-Business-Nov.01.2017-Tk#: crea1826
Bitdefender-Whitepaper-TERDOT-crea2079-A4-en_EN