Paper

Bitdefender®

New Pacifier APT Components Point to Russian-Linked Turla Group

Executive Summary:

New components belonging to the Pacifier APT linked to cyberespionage campaigns against government institutions reveal backdoor similarities in the underlying framework and functionality, attributed to the Turla group. Allegedly associated with the Russian government, the Turla group is believed to have been involved with other advanced persistent threats targeting governments and militaries since at least 2008.

The group has been known to use a broad arsenal of tools to infiltrate high-profile European and US institutions solely to acquire data and carry out cyber espionage. Although mostly Windows-based operating systems were targeted by Turla, evidence indicates they also developed Linux-based malware.

This latest analysis is further evidence that the group is highly versed in evasion techniques and constantly employs new attack and stealth mechanisms to dodge traditional security tools. What's interesting about this new research on Pacifier is that we've found new backdoor components that communicate with C&C servers using three very innovative techniques.

The first backdoor is a binary that can communicate with a command and control server (C&C) by proxying the connection through an internet-connected computer that shares the same LAN as the victim. The second backdoor is a Visual Basic Script that uses the browser's local cache to store commands and send data to C&Cs. And the third backdoor is a highly obfuscated JavaScript that constantly connects to C&Cs and sends information about the victim's system.

While the three analyzed backdoors are different, they all show just how versatile the Turla group is in terms of coding, implementations and data exfiltration techniques. We also found some other programs and tools for collecting data, including some freely available on the internet and probably uploaded by the attacker post-infection. These free tools enable memory dumping capabilities for 32-bit and 64-bit processes, such as Microsoft Outlook, intercept plain and encrypted browser traffic using man-in-the-middle techniques, or perform local network discovery, potentially to map networks and find other victims or valuable caches of information.

Since more backdoors are likely to be discovered and analyzed by security researchers, it's safe to speculate that the group is comprised of skilled members with a deep understanding of security evasion techniques.

Key Takeaways:

- New backdoor modules potentially developed by the Turla group;
- Innovative Visual Basic Script backdoor that uses the browser's (Internet Explorer) local storage mechanism to communicate with the C&C server;
- Communication with C&C triggered by user activity (IE launch), not automated by the malware;
- Data exfiltration from non-internet connected victims and backdoor signed with valid digital certificate;
- Ability to inject Firefox, Chrome, Browser, Opera, and Safari browsers as well as Microsoft's Internet Explorer to disguise C&C communication;
- Custom deployment of open source tools for reconnaissance and data exfiltration;
- Network topography reconnaissance before deploying tools and malware.

What We Know About the Turla Group, So Far

In 2016, Bitdefender uncovered a new advanced persistent threat dubbed Pacifier, targeting government institutions starting in 2014. Using malicious .doc documents and .zip files distributed via spear phishing e-mails, attackers would lure victims with invitations to social functions or conferences into executing the attachments. Previous analysis of Pacifier components revealed that it's capable of dropping multi-stage backdoors and that the analyzed first stage dropper is also known as "Skipper" by other security vendors.

The Turla group is known for its variety of APT attack tactics ranging from spear phishing to watering hole campaigns aimed at selectively infecting victims. While the previous sample analyzed by Bitdefender researchers dropped a Trojan using an infected attachment, ESET researchers uncovered a watering hole campaign that instructed victims to install a JavaScript backdoor presented as a Firefox extension. While implemented differently, there were striking similarities to the way the Turla group implements functionalities, according to the report.

In our first report, the attachment was a double extension file (e.g. "*.doc.js") that dropped a clean file and the malware. This method allows them to bypass the "enable macro" feature built into Word documents, that's known to raise suspicion among users as it's often abused by malware.

The new modules analyzed in this paper were found on an infected computer, probably targeted during the original campaign. While many of the found components are identical to our original research, new modules and tools point to fresh clues into the sophistication of the attack.

The command and control domains seem to be legitimate domains that have been compromised by attackers and repurposed for their needs. Since the purpose of this paper is not to investigate how the websites were compromised, it's reasonable to assume that their owners were unaware that their domains were being used as C&Cs for malware.

Below, you'll find a detailed technical analysis of the three new backdoor modules, as well a short description of their capabilities and features. It's worth noting that we've addressed the custom-deployed tools under a different chapter, separate from the overall analysis of the three Trojans.

1. No Internet? There's a Backdoor for That.

Security researchers have long explored ways of compromising air-gapped systems by creating malware designed to exfiltrate data by tampering with fans, heat emissions, and even scanners. The Turla group has implemented a way of exfiltrating data from victims, even if they're not directly connected to the internet.

Of course, this is not the first time cybercriminals have specifically crafted malware for air-gapped systems. The Flame (or Flamer) malware discovered in 2012 is believed to have been operating since at least early 2010. It was one of the first modular pieces of malware designed to exfiltrate data from non-internet connected PCs, by covertly infecting USB media storage devices and piggybacking on them until they were plugged into an internet-connected PC. Also, only specific USB media storage devices were infected and used to smuggle data, hinting that the cybercriminal group knew exactly who to target and how to dodge security mechanisms.

The first analyzed backdoor is a binary that, once installed, can dial home even if the infected computer does not have an active internet connection. Using an internet connected computer that shares the same local area network as the victim, it proxies the C&C communication through it. The communication between the victim and the internet-connected computer is hard-coded within the backdoor, meaning the infected computer knows exactly which network PC is directly connected to the internet. This is effective especially if you're looking to not give away any signs of infection by "talking" to other network connected devices, seeking internet connectivity. This demonstrates that attackers have a pretty good understanding of the victim's network topography before deploying the backdoor.

The backdoor component was signed with a – probably stolen - digital certificate valid from October 15th 2015 to October 15th 2016, precisely during the time of the attack. We believe some modules are still missing while previously known components are present.

Technical Analysis

It is worth noting that this malware sample (64-bit **st.exe**) shares similarities with „Carbon", described by ESET here, which is attributed to the Turla group, the same group that used the „Uroburos" rootkit. Both „Carbon" and this version of the backdoor use the same type of underlying framework split between multiple components with nearly identical functionality, which communicate through a similar communication channel. Furthermore, both have similar formats for their log files, and the same aspects are configurable in both malwares although the configuration is stored in a different manner.

Each of the following modules creates a file in the "%TEMP%" folder with name "CVRG***XXXX***.tmp.cvr" where ***XXXX*** is a hardcoded hex value that differs for each module of the malware. The name of this file is specifically chosen to resemble files created by Microsoft Windows programs when they crash, that have the format ***"CVRXXXX.tmp.cvr"***. This file contains very detailed debug information, encrypted with DES, that documents every step the malware takes toward infecting the system. Therefore most of the names of functions and variables in the following analysis are the actual names the malware authors intended for use.

Functionality Summary

In terms of functionality, below is a simplified representation of how the Trojan actually works.

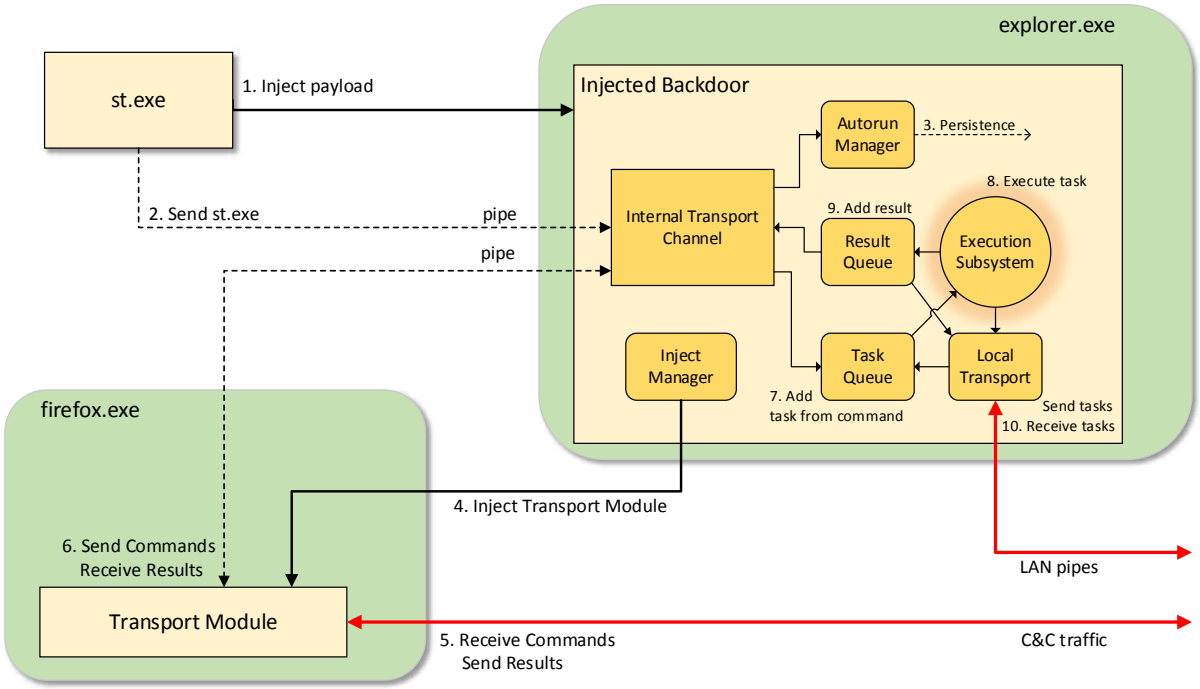

Figure 3- Functionality Summary

1. The main loader injects the payload (backdoor) into ***explorer.exe.***

2. The main loader sends its executable file to the Internal Transport Channel (from the injected backdoor) through a pipe.

3. The executable received over the pipe is used by the Autorun Manager for persistence.

4. The Inject Manager from the backdoor injects the Transport Module in a browser.

5. From the browser, the Transport Module receives commands and later sends results to the C&C.

6. The Transport Module sends the C&C commands to the Internal Transport Channel through the pipe. Results after execution are sent through the pipe to the Transport Module which forwards them to the C&C servers.

7. The Internal Transport Channel adds the command (task) in the Task Queue.

8. The Execution Subsystem executes tasks from Task Queue.

9. If needed, the Execution Subsystem adds results from executed tasks to the Result Queue from which the results will go to the Internal Transport Channel, then to the Transport Module and then to the C&C.

10. Some tasks are sent by the ***Execution Subsystem*** to the ***Local Transport Manager*** which then sends the tasks through pipes over LAN to backdoors on other infected computers. When a task is received on the LAN pipe, the ***Local Transport Manager*** adds the task to the ***Task Queue***.

Main Loader

The main function of this loader is to inject another executable into a process. The name of the process and the executable itself are taken from its resources, a resource for the process name (***"explorer.exe"*** in our case), and another one for the executable to be injected. Resources of the main loader are unencrypted, so dumping the injected executable becomes a trivial task.

Running the malware on a machine with Windows 10, or in the WOW64 (Windows 32-bit on Windows 64-bit) subsystem will abort the injection. The WOW64 check is a remaining artifact of a 32-bit version of the malware, since it is impossible for a 64-bit executable to run in WOW64.

A mutex (*{531511FA-190D-5D85-8A4A-279F2F592CC7}*) is created and checked to signal itself whether injection is already in progress. Furthermore, if a pipe with a certain name exists, injection is supposed to have already happened, as the pipe is created by the payload.

The desired process is searched for and the payload is injected using a custom injector. Afterwards, the main loader sends its own body over a pipe created by the payload with a function called ***GeneratePipeName*** that is implemented by all of this malware's modules. This pipe has the following printf format string: "*\\\\.\\pipe\\Winsock2\\CatalogChangeListener-%02x%02x-%01x*", where the 3 arguments depend on the current date and the SID (Security Identifier) of the current user.

The resource that contains the name of the process to inject into also contains the number of times to retry writing to said pipe, and the number of seconds to sleep between tries. In this case, the values are „explorer.exe", 7 retries, and 5 seconds to sleep between retries. These values can easily be changed between variants.

Injected Payload

This executable is written in C++ and follows a lot of object-oriented programming rules and patterns. It is highly modular and suggests that this is the work of several authors that probably worked in parallel, using a predefined set of specifications for each module.

It is also very configurable from its resources, as most settings are stored in its resources (e.g.: C&C servers and settings, persistence settings etc.). Therefore, the malware can be configured differently without recompiling the source code, just changing its resources. Unlike in the Main Loader, the resources are now compressed with bzip2 and encrypted with a PGP-like encryption, actual data is encrypted with a DES key, and the DES key is encrypted with RSA. The RSA key needed for decryption of the resources is itself stored as a resource in the malware.

The first step involves instantiating a ***PEStorage*** module, as this module's purpose is to provide an abstract interface for loading the resources of the executable. Any need for a resource of the executable will be satisfied by calling the specific function in the ***PEStorage*** module.

Immediately after, a ***Crypto*** module is instantiated, as this too is another module designed to abstract a functionality: encrypting and decrypting data. Note that everything encrypted or decrypted by the malware uses the type of PGP-like encryption described below, except for the debug logs.

Fig. 1 describes the encrypted message format used by the malware where:

1. Encrypted Session Key – First 128 bytes of the message. They contain a Triple DES key encrypted with RSA PKCS#1 v1.5 standard. The keys for RSA are on 1024 bits.

2. Message Signature – 128 bytes in size. Digital signature using RSA PKCS#1 v1.5 standard. RSA keys are on 1024 bits. The hashing algorithm used is MD5. The compressed plaintext (not the ciphertext nor the original plaintext) is signed.

3. Encrypted Message – The original plaintext compressed with bzip2 and then encrypted with 3DES with PKCS#7 padding. The 3DES key is 24 bytes long and is encrypted in the first 128 bytes of the message (see 1.).

4. MD5 – MD5 hash on the whole, final message. Used only by the *Transport Module* for integrity.

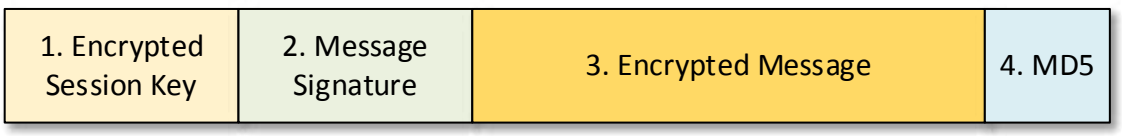

Figure 1 - Encrypted Message Format

The malware migrates its resources from the executable into a so-called External Storage. This External Storage can be of two types - RegStorage (storage in registry), and FSStorage (storage in files on disk) - depending on how the malware was configured. In this sample, the malware was configured to use RegStorage with root key: "\HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Explorer\ScreenSaver", each different resource of the malware is copied in a subkey of the above registry key. Analogous for FSStorage, different alternate data streams of the same file are used to copy the resources into. In the same storage, migrated information is encrypted with an RSA key randomly generated on the spot, and saved for further use. Both RegStorage and FSStorage provide a common abstract interface used to access or modify resources. This migration is made so that any further modification of the malware's configuration doesn't result in an alteration of the binary, but rather in an update of its External Storage. However, this only occurs if a valid External Storage is not found on the computer.

An ***Inject Manager***, also configured in its resources, looks for processes with certain names, trying to inject a specific executable. Although the searched processes and the injected executable can have any functionality in theory, this ***Inject Manager*** was always configured to look for generic processes that communicate over the Internet (e.g.: iexplore.exe, firefox.exe, chrome.exe etc.) and inject payloads that contact and communicate with the malware's C&C servers. It uses the exact same injecting mechanism as the main loader.

Another module named ***Internal Transport Channel*** is started with its own thread. It acts as a server that listens for commands from other malware modules through a named pipe, generated with the same ***GeneratePipeName*** function that all executables implement. This pipe is the destination of the above-mentioned main loader body.

Accepted commands sent over the pipe are:

TAKE_TASK: add received task to the ***Task Queue.***

TAKE_LOADER_BODY: used by the main loader to send its own body, so it can be passed as an argument to the ***Autorun Manager*** module that handles persistence over the machine.

GIVE_RESULT: send the result of a specific task from the ***Result Queue***.

GIVE_SETTINGS: send the communication settings used when contacting the C&C.

TAKE_CONFIRM_RESULT: Used after a GIVE_RESULT, it's confirmation that the result can be deleted.

TAKE_CAN_NOT_WORK: used to signal that the transport module has no internet access.

TAKE_UNINSTALL: self-destruct feature.

TAKE_NOP: no operation, used by the transport module to check whether the communication settings changed.

NO_CONNECT_TO_GAYZER: used to signal that a C&C server cannot be contacted.

TAKE_LAST_CONNECTION: used by the transport module to send the last time it contacted a C&C server.

GIVE_CACHE, TAKE_CACHE: it has the same functionality as TAKE_NOP, but it was not implemented.

We believe GIVE_CACHE and TAKE_CACHE are present as remnants of the fact that the ***Local Transport Manager*** module described below was supposed to run in another process (similar to how the transport module does) but the idea was finally scrapped.

The ***Task Queue*** and the ***Result Queue*** are complementary queues used by the payload to manage commands received from the C&C servers. A task received from the C&C has the following C structure format:

```
{
DWORD id                    //unique ID used to identify each task
BYTE priority               //priority of the task
WORD maxTimeExecution       //maximum time in seconds the task should be allowed to run
DWORD transportID           //an ID used to identify the destination of the result (0x1030001 for this version of the transport module, 0x10100 for Local Transport Manager)
BYTE maxSendCount           //the maximum number of times the backdoor attempts to send the result of this task
DWORD size                  //the size of the task data
```

DWORD taskType //the type of the task (more details below)

BYTE taskData[size - 4] //data dependent on the type of the task (more details below)

}

Another module called ***Execution Subsystem*** has the responsibility of executing tasks from the ***Task Queue*** according to their priority, and adding the result of those tasks to the ***Result Queue***.

The following represent valid types of tasks:

EXECUTION (misspelled as ***EXECUSION***): receives a path, the contents of a file, and some command line arguments. Creates the file then executes it with the specified command line arguments. The output of the executed file is added to the ***Result Queue***.

CONFIGURE: receives an argument specifying which resource to modify from its ***External Storage***, and an argument specifying how.

UPLOAD: receives a path and the contents of a file. It creates that file on the infected machine.

DOWNLOAD: receives a path and adds its contents with the specified path from the infected machine to the ***Result Queue***.

REPLACEMENT: replaces itself in the ***Autorun Manager*** according to the given arguments. Can be used as an upgrade feature to overwrite the malware with a newer version.

DELETE: triggers the self-destruct feature.

LOCALTRANSPORT: sets a so-called packet for the ***Local Transport Manager***

The ***Autorun Manager*** has **6 different approaches to obtaining persistence**, of which the ones properly configured in the resources are used. The ***Autorun Manager*** thread also checks and reinstalls the configured persistence methods if one of them is removed by the user. The following approaches can be used:

ShellAutorun: adds the malware's path to the "HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell" key in the registry, so that the malware is executed at logon. It takes care to add "explorer.exe" as well, if the key did not exist prior to infection.

HiddenTaskAutorun: creates a hidden scheduled task that triggers at logon, using the ITaskScheduler COM interface. This method is only used on Windows XP.

ScreenSaverAutorun: replaces the current screensaver from the registry with the malware

StartupAutorun: copies the malware to the startup folder

TaskScheduler20Autorun: creates a scheduled task that triggers at logon using the ITaskService COM interface. This method is only used on Windows Vista, 7 or 8.

LinkAutorun: infects .lnk files by changing their target application to: "cmd.exe /q /c start "***OriginalPathOfLnk***" && start "***MalwarePath***""

This analyzed sample uses the ShellAutorun feature with path "%HOMEPATH%\ntuser.dat.LOG3", and the TaskScheduler20Autorun configured to disguise itself as an Adobe Updater task with path: "%homepath%\AppData\Local\Adobe\AdobeUpdater.exe", name: "User registry integrity check task", and description "This task was automatically generated by Microsoft Windows. To remove contact your system administrator".

The ***Local Transport Manager*** module is tasked with communicating throughout the local area network (LAN). Its main jobs are to create a named pipe over the local network and listen, and process packets received over the pipe. A packet roughly contains a list of tasks, a list of results, and a list of '***routes***'. A ***'route'*** describes the next machine to send the packet to, by their local network named pipe that was created by the backdoor. The same packet gets sent to the next entry in the route list after extracting, executing, and adding to the packet the results of the tasks that have the destination ID equal with the infected machine's ID. It acts as a server on LAN the same way the ***Internal Transport Channel*** does for local modules.

The diagram in Fig. 2 depicts the way a ***Local Transport Manager*** packet propagates through LAN.

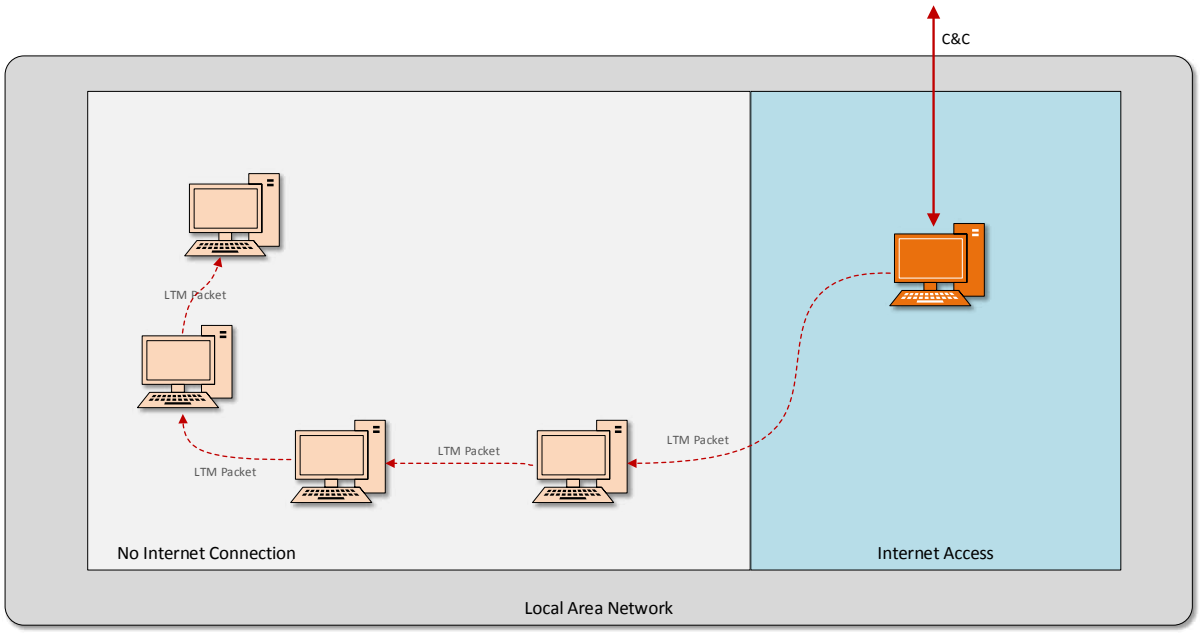

Figure 2 - LAN propagation

The following information is stored in the malware's ***External Storage***:

- RSA key pair generated at migration time, used to encrypt/decrypt data from the *External Storage* itself as well as tasks received from the C&C server
- An ID of the current infected machine, comprised of its MAC address followed by 16 bytes derived from the current user's SID
- Live backup of the *Task Queue*
- Live backup of the *Result Queue*
- The *Transport Module* executable that contacts the C&C server
- *Inject Settings* to be followed for the *Transport Module*'s injection
- *Transport Settings* – the C&C servers to contact, as well as settings of how the contact is established for the *Transport Module*
- *Autorun Settings* – settings that specify how malware persistence is obtained
- *Plugins* – unused
- *Last Connection* – the time the last connection with a C&C server was established
- *Local Transport Settings* – the name of the local network pipe to create and listen on (in this analyzed sample, these settings were null)
- *Cache* – serialized packets of the *Local Transport Manager*

Judging by the unused resource, referred to as ***Plugins*** - which was empty in all samples analyzed and was never actually used for anything other than being migrated to ***External Storage*** - we assume that either development of the malware was rushed at the final moment or this isn't its final version and future updates are being planned that use this resource. The unused GIVE_CACHE and TAKE_CACHE pipe commands described above support this notion as well.

Transport Module

This module is injected by the payload in a generic process that communicates frequently over the Internet (e.g.: iexplore.exe, firefox.exe, chrome.exe, opera.exe, outlook.exe etc.). It acts as a middle man between the main payload and the malware's C&C servers so the payload doesn't generate a lot of suspicious traffic from a process that should not be generating traffic, but instead blends its traffic with a legitimate process that generates a lot of traffic on its own. It uses the higher level ***wininet.dll*** library functions to send and receive data.

It uses the same ***GeneratePipeName*** function to generate the name of the pipe used to communicate with the payload. Using this pipe, it asks for the communication settings from the payload; those settings are also configurable in the resources of the main payload.

It checks whether some conditions comply with its configured settings. These conditions are rechecked at specific intervals and, if one check fails, a connection is not established. The following conditions can be checked if present:

Inactivity: if a number of hours specified in the settings have passed since the last connection, a connection is established regardless of other connection conditions.

Schedule: can be configured to contact the C&Cs servers at a certain hour each day, only on some days of the week or on an hourly, weekly or daily interval.

Blacklist: the connection isn't established if it finds a process with a name that matches one from the blacklist in its settings. It is probably used so it doesn't generate suspicious traffic when traffic monitoring processes are running.

Periodicity: checks whether a specified number of seconds have passed since the last connection was established

In the analyzed sample, only the ***Inactivity*** and ***Periodicity*** checks were configured, with 1 hour maximum inactivity time, and 1,200 seconds (20 minutes) between connections, but others can be configured on the fly if the ***CONFIGURE*** task is received from the C&C.

It contacts the C&C server to get a task, which it forwards through the pipe to the main module. It then waits 5 seconds, after which it tries to get a result through the same pipe to send back to the server.

The Transport Module is notified if a reconfiguration of its C&C settings happens in the main payload, so that it knows to change the way a connection is established.

All communications are encrypted and signed and incoming data is encrypted and decrypted using the RSA key pair that was generated and used upon migration to ***External Storage***. The public key is sent to the server when receiving a message that isn't properly encrypted and signed, along with the machine ID and a version hash. The task results sent over to the C&C servers are also encrypted using a hardcoded RSA public key from the malware's resources.

Other Variants

We found various versions of this backdoor with only minor differences between them, mainly pertaining to how their settings are configured in the executable. For example, "outlook.exe" is sometimes one of the processes searched and injected with the Transport Module, while in other instances "browser.exe" is scoped.

The C&C servers differ among all versions, but the Transport Module remains exactly the same, regardless whether the main loader is 32 or 64-bit.

Some versions opt to use ***FSStorage*** for storing data instead of the more common ***RegStorage***. However, the configuration path for ***FSStorage*** is "%TEMP%\storage"

2. Using Browser Cache to Evade Security

The second backdoor discovered during the investigation is a Visual Basic Script that has an innovative and covert communication mechanism with the C&C. The browser's caching mechanism is mostly used by websites to store local content to load resources faster if users visit the same website often. It's practically a browsing optimization design to boost performance, save bandwidth, and even store personalized user settings. Ordinarily, the browser cache is not the first place you would look for C&C communication or data exfiltration, but it seems innovation is not in short supply, even for cybercriminals.

One main advantage of using a Visual Basic Script is that it's far less suspicious than any other file type, and it's more likely to trick behavioral scanning engines as they don't necessarily execute these scripts. But probably the most important aspect of using Visual Basic Scripts is that they don't directly connect to the internet, but instead wait for the user to start the browser – in this case, Internet Explorer.

This analyzed Trojan sets a new default page for Internet Explorer so that each time the browser is launched, it automatically connects to the C&C. Since this behavior was only spotted on Internet Explorer, we can assume that the victim uses Internet Explorer as the main browser, so the group specifically coded the behavior for IE.

The website the Trojan connects to is perfectly legitimate, but it also contains a JavaScript (placed by the attacker) that uses a legitimate method of writing in the browser's local cache. Consequently, the JavaScript will write instructions from the C&C into the local browser cache of Internet Explorer. Since the script is not malicious per se, this is the novelty of the Trojan that reveals a radically new attack avenue.

On the victim's side, the backdoor simply checks if something new has been written in the browser's cache, looking for instructions. The Trojan doesn't constantly scan the cache for new instructions, it only does so at various time intervals to avoid triggering security warnings.

The results of each executed command are stored until IE (Internet Explorer) is started and can connect to the C&C page - set as the default homepage – to send the output of each executed command. The Trojan can also send emails with file attachments, although executing the command currently does nothing.

Using the browsers local storage mechanism is a new and very effective method for communicating with C&Cs as it's untraceable, it's a legitimate browser mechanism, and it raises no security warnings as almost all websites use the browser's local storage for locally storing resources. Another advantage of using a Visual Basic Script is that it "disguises" the C&C communication as legitimate browser behavior.

Worth noting is that, if the victim were not using Internet Explorer, the backdoor would not work. This is particularly interesting because it indicates that the cybercriminal group knew that Internet Explorer was the preferred browser, possibly by extensively analyzing the target in advance.

While the two backdoor modules are completely dissimilar in terms of coding, the only connection between them is that this Visual Basic Script has the ability to download the previously-described binary - that doesn't require the victim to have an active internet connection.

Technical Analysis

The technical analysis below includes the Visual Basic Script as well as the parameters it was designed to accept.

The "sc.wsf" is a Windows Script File containing 2,000 lines of VB script, built to act as a backdoor program. Interestingly, connection with the C&C is not done directly, but through the local storage mechanism of HTML5 and Internet Explorer browser. With local storage, web applications can store data locally within the browser. In JavaScript, you would use the local storage with something like:

```
localStorage.setItem("color", "Blue");

localStorage.getItem("color");
```

In Internet Explorer 8 through 11 the local storage is implemented as an .xml file that has nodes containing "name" and "value" attributes, allowing access to stored data in a key-value manner. Internet Explorer creates multiple local storage xml files that correspond to accessed pages. The backdoor searches all the local storage files and tries to find commands in them.

The communication with the C&C is done like this:

- The script sets the start page of Internet Explorer to a C&C address.
- When IE starts, the C&C page for the backdoor saves commands to the local storage.
- When the script runs again, it reads the commands from the local storage (parsing the .xml), executes them, and stores the results (if needed) back in the .xml file.
- When IE is started subsequently, the C&C page reads the results and the cycle repeats.

This way, the commands are stored and wait for the backdoor to execute them. Results are also stored until IE starts, so that it can connect to the internet to access the C&C page. The default C&C addresses are:

http://[redacted]/Framework/Extend/Library/ORG/Util/Image/Driver/ThinkImage.php
http://[redacted]form.com/status/libs/Zend/Feed/Entry.php
http://[redacted]/blog/wp-content/plugins/jetpack/modules/minileven/theme/pub/minileven/tweaks.php
http://[redacted].com/libraries/rokcommon/RokCommon/Doctrine/Platform/Joomla15.old.php

Each time the script runs, it will open Internet Explorer with one of those pages, because the start page for Internet Explorer is set to a C&C address. Another C&C address can be added later, received using the backdoor commands.

The script can take the following command line parameters:

-d - Sets the working directory. By default, the working directory is where the script resides.

-m - Sets the maximum size of the file containing the results. The default maximum size is 10MB

-t - Sets repeat interval in minutes for the scheduled task that runs the script. Default is 480 minutes.

-p - Sets the startup page for Internet Explorer

-n - Sets the new name of the script. The Default name is ***SystemSoundsServiceControl.***

-s - Sets an ID of the computer. When run without "-s" parameter, the script sets the ID to the MD5 hash of a string containing the current date, current time and computer name. After initializing the ID, the script will be set to run (from task scheduler or registry) with '-s' parameter with the generated ID, so that the ID remains the same.

When the script is executed for the first time, it will adopt the name of one random sub-folder from ***%PROGRAMFILES%*** with ***.wsf*** extension and it will move itself to a random sub-folder from ***%APPDATA%***. For persistence, the script uses a scheduled task named ***SystemSoundsServiceControl***, located in the ***Task Scheduler Library\WPD*** folder (that repeats every 480 minutes by default) and the autorun key: ***HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run***. The scheduled task is added using a PowerShell command. It logs most of its actions in a text file that has the name of the script, using the ***.log*** extension, and contains the date the script ran, details about its execution, and if functions succeeded or failed.

The backdoor commands are read from the .xml files that represent the local storage for Internet Explorer. The locations where the script searches for the .xml files are:

%userprofile%\AppData\LocalLow\Microsoft\Internet Explorer\DOMStore

%userprofile%\AppData\Local\Microsoft\Internet Explorer\DOMStore

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache\Extensible Cache\DOMStore\CachePath

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\LowCache\Extensible Cache\DOMStore\CachePath

The script will create a history file in the working directory, named ***Rezults.xml*** in which it will hold the commands that it received, as well as their results.

For every command received, the script will check in ***Rezults.xml*** if the command is present, which signals it was already executed. As the local storage file can store the commands indefinitely, this mechanism ensures that commands are executed only once. Interestingly, the ***.xml*** file is used as a configuration file, as the script stores in it other C&C addresses, encryption/decryption keys, install date. Figure 4 illustrates the way in which commands are received from the C&C server.

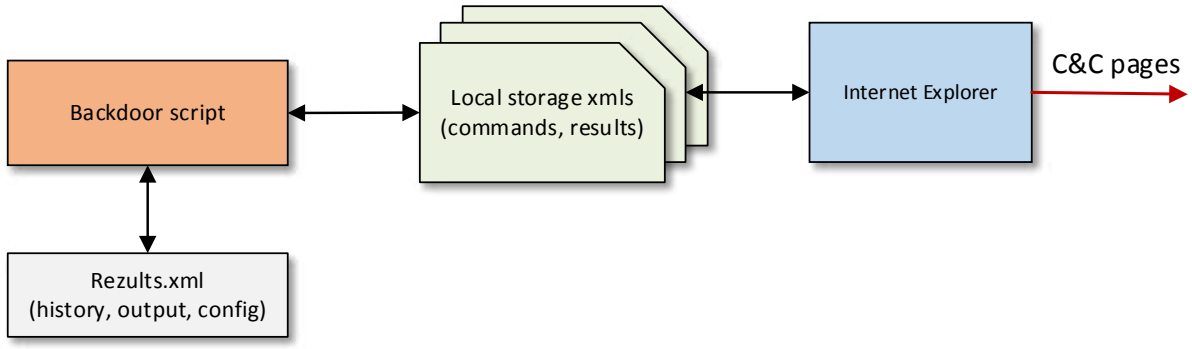

Figure 4 – Backdoor communication

Below, you'll find the variables from the local storage used between the backdoor and the C&C:

temp - Variable used to validate the .xml file. The backdoor tries to find commands only if this variable exists. Its value is ignored.

utm_ec - Contains an ID of the computer. The value of this variable is generated by the backdoor script (as described above), then stored in the local storage xml file. The C&C page will get this variable from the local storage and take differing actions based on its value.

utm_g - Contains the command for the backdoor. This variable is set by the C&C page. The value is encrypted (optional) with AES and encoded in Base64.

utm_k - Contains an AES key encrypted with RSA and encoded in Base64. The value is read by the script, decrypted with RSA, and the resulting AES key is used to decrypt the command string from utm_g. The encryption mechanism is optional. This mechanism could be used to ensure no one else can issue fake commands and that only the attacker-controlled C&C is capable of interacting with the target.

utm_p - The result from a command. This variable is added by the backdoor script. The C&C page will read the result. The result is encrypted with AES (optional) and encoded in Base64.

The text received in 'utm_g' variable (after decoding), consists of lines containing one or more backdoor commands. An example:

```
@start_instruction

@ident
6

@exec
cmd /u /c dir %APPDATA%\Microsoft & %APPDATA%\Microsoft\st.exe & DEL /Q %APPDATA%\
Microsoft\3.txt

@gfile
%APPDATA%\Microsoft\st.exe
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAA.....(truncated)

@end_instruction
```

The commands are actually strings that start with the '@' character. Results (saved in 'utm_p' variable) have the same structure as the command strings, but they follow each command. The backdoor commands are described below:

@start_instruction, @end_instruction - Not actual commands, just markers.

@ident - ID for a set of commands. Contains a number used by the backdoor to uniquely identify a set of commands. These commands and the ID are saved in ***Rezults.xml*** file by the backdoor and, every time it reads instructions from the local storage, it will search their IDs in ***.xml*** file. If it finds the ID, it means the commands were already executed, and they are ignored. If @ident is missing, the commands are not executed.

@about - Gets information about the system and stores it into the .xml file. The data gathered involves OS name, version, service pack, registered user, last boot time, wscript version.

@exec - Executes through cmd.exe. It runs ***"cmd /u /c ""(" & command & ")>>Rezult.txt""""***. The content of ***Rezult.txt*** will be stored in ***utm_p*** variable in the local storage.

@pfile - Receives one or more file paths, it reads the specified files, and stores them as the result in the local storage, Base64 encoded.

@efile - Receives a list of executable files and executes them.

@gfile - Receives one or more files encoded in Base64, each with a corresponding file path. After decoding, it will write the files on disk in the specified paths.

@rezult - Receives one or more command IDs and searches their results in its history file ***Rezults.xml.*** The results will be stored in the local storage.

@timeout_control - Receives a number of minutes which will be set as the repeat interval for the scheduled task of the script.

@kill - Deletes all local storage files for Internet Explorer and current user, and exits.

@page - Receives a list of C&C addresses and stores them to ***Rezults.xml.*** From there, it will load the addresses next time it runs. Each C&C address is used for a day. At every start, the script will open a C&C page in IE (opened with an invisible window), wait for a while for the page to access the local storage, then close IE. Also, the pages are set as start pages for IE.

@memory - Receives a value representing the maximum size of the ***Rezults.xml*** file, in kilobytes. It must be in the range 200 - 10240.

@update - Receives a new script file, encoded in Base64, used for self-update.

@rc - Put the script to autorun key in registry.

@common_key - Contains a pair of RSA public/private keys or only the private key. The private key is used to decrypt the AES key for the encrypted commands, while the public key is used to encrypt the AES key used to encrypt the results.

@self_key - Contains a pair of RSA public/private keys or only the private key. Used the same way as @common_key but it has priority over it.

Other versions

We come across another version of the script. It has minor differences from the version discussed, including new C&C addresses:

http://[redacted].com.br/includes/inc_drop.php
Http://[redacted].com/wp-includes/class-wp-ajax.php
http://[redacted].com/user_tpl/footerpage.php

The ***Rezults.xml*** file that contains the history and the results now has the same name as the script (but with the .xml extension) and is located in ***%Temp%*** folder. The history file is now encrypted with AES and a constant hardcoded key. The ***common_key, self_key*** RSA public/private keys are gone, as are the commands ***@common_key, @self_key***.

The commands are now always encrypted with AES and the default key is hardcoded in the script. The AES key can be changed later with a new backdoor command ***@key.*** A new command for the backdoor ***@timeout*** and a new command line parameter for the script (***-tIE***) sets a timeout variable that is not used, suggesting the script is not finished.

The timeout could be used as a timer for which Internet Explorer waits on a C&C page. The ID of the computer is now fixed to a hardcoded value and cannot be changed, so the ***–s*** command line parameter is gone. A new interesting command is ***@mail*** that also contains a SMTP server and port, an email and password, and a body. The command does nothing, but the functionality to send an email with a file attachment is present. A new localstorage variable ***utm_in*** is added by the script and represents the ID of a backdoor command result (result stored in ***utm_p*** variable).

3. Victim Profiling with a JavaScript Backdoor

The same investigation revealed the existence of a third backdoor written in JavaScript, also believed to be developed by the Turla group. The code is highly obfuscated, but the backdoor is simpler than the one written in Visual Basic Script. Running basic Windows commands to gather info and sending them to the C&C using HTTP is yet another non-standard technique that allows attackers to fly under the radar.

Technical Analysis

The "WindowsCache.js" backdoor copies itself in the "*C:\Users\<USER>\AppData\Local\Microsoft\Windows (<USER> is the current user)"*. If this folder does not exists it copies itself in C:\Users\<USER>\AppData\Local\Temp and it also puts itself to an autorun key in the Registry "*HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run*".

The backdoor has two C&C addresses:

hxxp://[redacted].co.il/wp-content/themes/san-kloud/admin.php
hxxp://[redacted].com/wp-includes/pomo/db.php

It connects to the C&Cs directly, through HTTP protocol, and sends a computer ID in the user-agent variable. First, the backdoor gathers some system information by running the following commands through ***cmd.exe***:

systeminfo
net view
net view /domain
tasklist /v
gpresult /z
netstat -nao
ipconfig /all
arp -a
net share
net use
net user
net user administrator
net user /domain
net user administrator /domain
set
dir C:\Users.**
dir C:\Users\<USER>\AppData\Roaming\Microsoft\Windows\Recent.**
dir C:\Users\<USER> \Desktop.**

The output of these commands is gathered in a temporary file (***~dat.tmp)*** located in the working folder of the script. The content of the file is encrypted with RC4 and encoded in base64, then sent to a C&C address with am HTTP POST request. The user-agent variable of the request is set to: ***"Mozilla/5.0 (Windows NT 6.1; Win64; x64); <32 digit ID>"***. The ID is generated from the computer name and user name (ex: *48988645562559440e32409812126788*). The C&C responds with a command for the backdoor. The commands are:

"good" – no action is taken.

"work" – the backdoor will send another POST request with the content ***"work"***, with the user-agent from above, and the C&C responds with an executable, encrypted with RC4. The executable will be placed in the work folder with the same name as the script but with ***.scr*** extension, will be executed and deleted.

"exit" – the backdoor terminates.

"fail" – the backdoor removes itself from disk and registry, then terminates.

After each command, the backdoor sleeps 5 seconds then repeats the process (sending the system info and receiving a command) with the other C&Cs defined. After all C&Cs are contacted (two in our case) the backdoor sleeps about an hour and repeats.

This is another tool used by Turla actors, which shows the diversity involved in this attack. Running basic Windows commands to gather info, using highly obfuscated JavaScript, all these kinds of non-standard techniques, allows them to fly under the radar.

Similar research and analysis on this Trojan was also published by Kaspersky researchers in early February 2017, pointing to the Turla group.

Custom Deployed Tools and Files

Some of the programs we found, used as tools, can be found on the internet. They were probably uploaded by the attacker using one of the backdoors analyzed above. Interestingly, the attacker dumped the memory of "outlook.exe" in a likely attempt to grab important emails and information. This is a particularly effective means of accessing data from active process, as the resulting .dmp file raises no suspicions of security solutions nor tech-savvy users. These files are constantly created by Windows if the operating system crashes, and IT admins often use them to diagnose problems. In the unlikely event a user would stumble across a file with that extension, he would not give it a second thought.

The same "outlook.exe" process was also injected with a tool designed to intercept plain and encrypted traffic, in an attempt to "see" everything sent and received via Outlook. This method allows an attacker real-time access to sent and received emails, without having to dump the processes' memory or performing other "stunts".

Technical Analysis

1. **dp_x32.exe, dp_x64.exe**

Dumps the memory of a process. 32 bit and 64 bit version. Creates files with ***.dmp*** and ***.sidms*** extensions.

2. **CVR_0000.tmp.cvr.sidms, outlookexe.dmp**

Memory dumps of outlook.exe.

3. **~nt00001.exe, DLL.dll, ~st0121e.bat**

Netripper tool (can be found here: https://github.com/NytroRST/NetRipper) injects DLL.dll in a browser and intercepts plain and encrypted network traffic. ***~st0121e.bat*** executes "***~nt00001.exe DLL.dll OUTLOOK.exe***" which injects DLL.dll in outlook.exe.

4. **Get-SubNetItems.ps1, ~sc00001.bat**

PowerShell script for finding computers over the network (can be found here: https://gallery.technet.microsoft.com/scriptcenter/SubNet-Scan-dad0311f). The bat file starts the script to find computers in the local area network.

Known Pacifier Components

These components are very much like the binary components discussed in our previous paper about Pacifier, with a few exceptions. Some files are identical to the old ones and will not be discussed. The rest will be briefly described next.

Technical Analysis

mskl32.exe

A dropper that contains only the 32bit files: ***msi.dll, msp.dll, wsm.exe, mskl.dll***.

wsm.exe

Starts the Trojan and injects the ***%appdata%\Microsoft\VisualStudio\9.0\msstyles.dll*** and ***%appdata%\Microsoft\VisualStudio\9.0\mskl.dll*** files into a process the. We do not have the ***msstyles.dll*** file for analysis.

msi.dll

Injects a dll into another process.

msp.dll

Gets the PID of a specific process. Export function ***p1*** returns the PID of ***dwm.exe, wscntfy.exe or sihost.exe***. Export function ***p2*** returns the PID of ***taskhostex.exe, sihost.exe, DWM.exe***. The PID is returned only if the searched process is under the same domain\user as the current process, in our case the process of ***wsm.exe***.

mskl.dll

A new component that takes screenshots: one every hour and one every time a window gets the focus. The pictures are saved to ***%temp%\1.tmp.dat, %temp%\2.tmp.dat, etc.*** Logs every key pressed and the name of the active window in ***%temp%\KBDTV10FY.dat*** and the file will be encrypted with a substitution cypher.

msstrt.exe

New module that copies itself to ***%CommonProgramFiles%\Microsoft Shared\NGENUP.exe***. It creates a task with the commandline: ***schtasks /Create /RU SYSTEM /SC ONSTART /TN "Microsoft\Windows\.NET Framework\NGEN Update" /RL HIGHEST /F /TR "%CommonProgramFiles%\Microsoft Shared\NGENUP.exe".*** To avoid suspicion, the task will be created in the folder ***Microsoft\Windows\.NET Framework*** with the name ***NGEN Update***. The task will be triggered at system startup. If ***msstrt.exe*** is executed as ***%CommonProgramFiles%\Microsoft Shared\NGENUP.exe,*** it executes the file ***%ALLUSERSPROFILE%\StartDotNetUpdate.cmd.*** We do not have the file ***StartDotNetUpdate.cmd*** for analysis.

msrar.exe

Rar archiver, console application.

mst60.dll

64 bit library contains functions for communicating with the C&C. Two C&C addresses are used: ***37.48.90.239/rss.php, 67.228.88.107/rss.php.***

IOCs

Known Pacifier components

%appdata%\Microsoft\VisualStudio\9.0

%temp%\1.tmp.dat, %temp%\2.tmp.dat, etc.

%temp%\KBDTV10FY.dat

%CommonProgramFiles%\Microsoft Shared\NGENUP.exe

NGEN Update (scheduled task name)

%allusersprofile%\StartDotNetUpdate.cmd

sc.wsf

utm_ec, utm_g, utm_k, utm_p (local storage variables names)

Rezults.xml, Rezult.txt (in backdoor working dir)

SystemSoundsServiceControl (scheduled task name)

wscript.exe.activation_config (in backdoor working dir, Active Configuration File specifying .NET 4 use)

COMPLUS_ApplicationMigrationRuntimeActivationConfigPath (environment variable, used for Activation Configuration File, may be used by other applications)

st.exe

File paths

%TEMP%\CVRG72B5.tmp.cvr – main loader debug information file

%TEMP%\CVRG1A6B.tmp.cvr – injected payload debug information file

%TEMP%\CVRG38D9.tmp.cvr – transport module debug information file

%APPDATA%\Microsoft\st.exe – main loader initial path

%HOMEPATH%\ntuser.dat.LOG3 – ***ShellAutorun*** path in some versions

%HOMEPATH%\AppData\Local\Adobe\AdobeUpdater.exe – ***TaskScheduler20Autorun*** path in some versions

%AppData%\Sun\Java\jucheck.exe – ***TaskScheduler20Autorun*** path in some versions

%TEMP%\storage – ***FSStorage*** path in a configured version

%TEMP%\KB943729.log – ***FSStorage*** default path if it isn't properly configured, or using the above path fails

Pipes

\\\\.\\pipe\\Winsock2\\CatalogChangeListener-%02x%02x-%01x – based on current date and user, changed with date change

Mutex

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

Registry Paths

\HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Explorer\ScreenSaver – RegStorage key configured to be used in some versions

Software\Microsoft\Windows\CurrentVersion\Explorer\ScreenSaver – RegStorage default key if it isn't properly configured, or using the above key fails

The following subkeys of one of the above keys are used for storing malware configuration data (note that the same names are used for

alternate data streams if the configuration is file-based):

- {629336E3-58D6-633B-5182-576588CF702A}
- {119D263D-68FC-1942-3CA3-46B23FA652A0}
- {1DC12691-2B24-2265-435D-735D3B118A70}
- {6CEE6FE1-10A2-4C33-7E7F-855A51733C77}
- {31AC34A1-2DE2-36AC-1F6E-86F43772841F}
- {81A03BF8-60AA-4A56-253C-449121D61CAF}
- {8E9810C5-3014-4678-27EE-3B7A7AC346AF}
- {3CDC155D-398A-646E-1021-23047D9B4366}
- {28E74BDA-4327-31B0-17B9-56A66A818C1D}
- {56594FEA-5774-746D-4496-6361266C40D0}
- {4A3130BD-2608-730F-31A7-86D16CE66100}
- {81A03BF8-60AA-4A56-253C-449121D61CAF}

Network activity

- zerogov.com/wp-content/plugins.deactivate/paypal-donations/src/PayPalDonations/SimpleSubsribe.php
- shinestars-lifestyle.com/old_shinstar/includes/old/front_footer.old.php
- 217.171.86.137/rss_0.php
- dyskurs.com.ua/wp-admin/includes/map-menu.php
- warrixmalaysia.com.my/wp-content/plugins/jetpack/modules/contact-form/grunion-table-form.php

It should also be noted that the domains mentioned above seem to be legitimate websites compromised by the attackers and used for C&C.

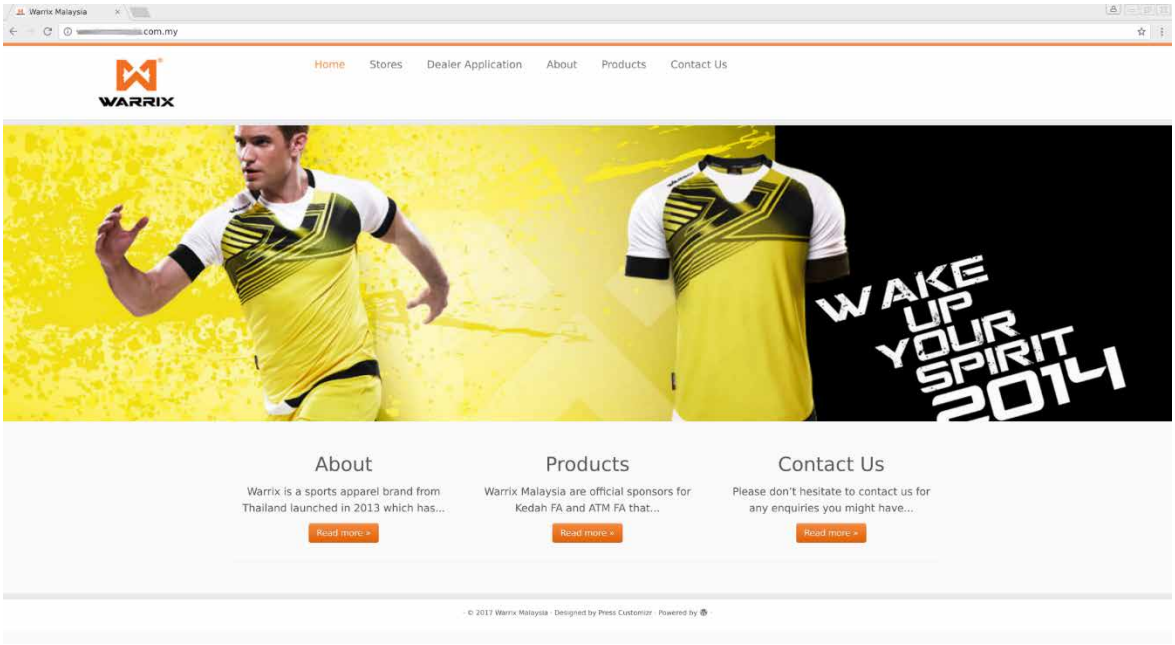

Figure 5 - A legitimate website compromised and used as a C&C server

Some lines from the debug information files *(CVRXXXX.tmp.cvr)*:

```
|13:11:23:205|  [0165|GeneratePipeName      ] \\.\pipe\Winsock2\CatalogChangeListener-5190-d
|13:11:23:205|  [0275|WinMain               ] PipeName = \\.\pipe\Winsock2\CatalogChangeListener-5190-d
|13:11:23:205|  [0277|WinMain               ] Checking for existence...
|13:11:23:205|  [0308|WinMain               ] --- Pipe is not installed yet
|13:11:23:205|  [0286|GetCurrentUserSID      ] _GETSID_METHOD_1_
|13:11:23:205|  [0425|GetUserSidByName      ] 28 7 1569648 122
|13:11:23:205|  [0463|GetUserSidByName      ] S-1-5-21-983755730-3884504757-593626067-1000
|13:11:23:205|  [0471|GetUserSidByName      ]
|13:11:23:205|  [0318|WinMain               ] Loading...
|13:11:23:205|  [0026|KernelInjector::KernelInjector     ] Address of marker: 0x000000000017F280 and cProcName:
0x000000000017F3A0
|13:11:23:205|  [0031|KernelInjector::KernelInjector     ] Value of marker = 0x0000000000000120
|13:11:23:205|  [0045|KernelInjector::KernelInjector     ] Address of function “WriteProcessMemory” is 0x00000000779DBFF0
|13:11:23:205|  [0088|KernelInjector::SetMethod          ] m_bAntiDEPMethod = 1
|13:11:23:205|  [0376|FindProcessesSimple   ] PROCESS NAME:  explorer.exe
|13:11:23:205|  [0345|WinMain               ] try to load dll to process (pid=1384))
|13:11:23:205|  [0088|KernelInjector::SetMethod          ] m_bAntiDEPMethod = 1
|13:11:23:205|  [0094|KernelInjector::LoadDllToProcess            ] MethodToUse = 1
|13:11:23:205|  [0171|KernelInjector::GetProcHandle    ] pid = 1384
|13:11:23:205|  [0314|KernelInjector::CopyDllFromBuffer           ] Trying to allocate space at address 0x0000000190010000
|13:11:23:205|  [0332|KernelInjector::CopyDllFromBuffer           ] IMAGEBASE = 0x0000000190010000  ENTRYPOINT =
```

0x0000000190048E20

|13:11:23:205| [0342|KernelInjector::CopyDllFromBuffer] ANTIDEP INJECT

|13:11:23:205| [0345|KernelInjector::CopyDllFromBuffer] Writing memory to target process...

|13:11:23:205| [0353|KernelInjector::CopyDllFromBuffer] Calling to entry point...

|13:11:23:205| [0598|KernelInjector::CallEntryPoint] CODE = 0x0000000002390000, ENTRY = 0x0000000190048E20, CURR = 0x00000000778ABF5A, TID = 1372

|13:11:23:205| [0786|KernelInjector::CallEntryPoint] _FINISH_ = 1

|13:11:23:205| [0372|KernelInjector::CopyDllFromBuffer] CTRLPROC = 0

|13:11:23:205| [0375|KernelInjector::CopyDllFromBuffer] + INJECTED +

|13:11:23:205| [0351|WinMain] +++ Load in 1384

|13:11:35:485| [0445|WinMain] Writing Loader to Pipe...

|13:11:35:485| [0169|WriteSelfToNamedPipe] +++ PIPE IS OPEN

|13:11:35:485| [0453|WinMain] SCT_FIN

|13:11:35:485| [0480|WinMain] ReleaseMutex = {531511FA-190D-5D85-8A4A-279F2F592CC7}

|13:11:35:485| [0483|WinMain] exit

.....

|13:11:25:838| [0749|MainThread] ### INJECT MANAGER ###

|13:11:25:838| [0286|GetCurrentUserSID] _GETSID_METHOD_1_

|13:11:25:838| [0425|GetUserSidByName] 28 7 0 122

|13:11:25:838| [0463|GetUserSidByName] S-1-5-21-983755730-3884504757-593626067-1000

|13:11:25:838| [0471|GetUserSidByName]

|13:11:25:838| [0882|InjectManager::LoadWinsta] fd9e0000

|13:11:25:838| [0852|InjectManager::LoadNtdll] 77ab0000

|13:11:25:838| [0080|DllInjector::SetMethod] m_bAntiDEPMethod = 1

|13:11:25:856| [1754|Crypto::DecryptAndVerifyBufferRSA1] Ok

|13:11:25:874| [1754|Crypto::DecryptAndVerifyBufferRSA1] Ok

|13:11:25:874| [0188|InjectManager::BuildInjectSettingsList] 1 iexplore.exe

|13:11:25:874| [0188|InjectManager::BuildInjectSettingsList] 2 firefox.exe

|13:11:25:874| [0188|InjectManager::BuildInjectSettingsList] 3 chrome.exe

|13:11:25:874| [0188|InjectManager::BuildInjectSettingsList] 4 browser.exe

|13:11:25:874| [0188|InjectManager::BuildInjectSettingsList] 5 opera.exe

|13:11:25:874| [0188|InjectManager::BuildInjectSettingsList] 6 safari.exe

|13:11:25:874| [1084|InjectManager::SetStatusTransportDll] ! StopFlag

File Hashes (SHA1)

```
702912637b9bb4356262b135cecc4b10741f7bb0
8606d1ab3c8fc932feffde1e8fd6a8687f641f19
c327f7a5602c7c4a25ff9edf6c714eca9f44a6b3
f2ca8fd540e6a1a8b20ca4577a798807f5e6bd91
14feb17c1c5fa76d8fb89113b49c2ee2258e2e5a
411ef895fe8dd4e040e8bf4048f4327f917e5724
46c563d56295efd198e59e0a2036900566ec2a49
afffcbfd6ff2d88943e44e4c0a4532654ee64c4e
527fb1247bcc0ce599e8e3ea55e3fb124c699db9
1978579d781e9a786f9b297c910ddacfc287e92a
13c5ac80ead15ce8667c2858910a9dc17b7bd618
81e21fb7ec096c2ea738a08522aa07941d121b0b
5ec7f506dbf23a34aa5f9b8f4fae782508c4a59b
da9b1374496145e4831b0494ea5da68d1bc4c0e2
1bff1d85439fe275e16ff2728b6b775b89fdc961
4c91c6c8499c07dc1509aedd75be299a0ec2531f
9a5d7df7a130b633a550983b6d565da6dbf5b78f
9b9e21d2720270f9f0967fc1ad06287b13758edd
41a6f8352974666ed67f56d889ccf6ee9d53f285
4568855d0ee5a9acc6ceb72025fcac3bc65af585
8d1095d6e235aaf7846327bba8b6f9be22c5994e
084b3243210faa875358a2497d3e5f407a2fc36c
a33f2802ba40a1a32bdb78ee28e1caaf234f01cc
832aa44f6bb76db42ffacbba6a9d0973da9a404b
ab8cd4cce1be285e16b57d92d890195aac457630
e19cdfb239c1d99bcf5c920c2544f8c696046767
c380038a57ffb8c064851b898f630312fabcbba7
```

Authors

Cristian Istrate - Antimalware Team Lead|
Andrei Ardelean - Malware Researcher
Claudiu Cobliș - Malware Researcher
Marius Tivadar - Antimalware Team Lead

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/

FOR MORE INFORMATION VISIT: enterprise.bitdefender.com

BD-Business-Aug.31.2017-TI#: crea1836
Bitdefender Whitepaper Pacifier crea1836 A4 en_EN