welivesecurity
Security news, views and insight from the ESET experts

Dino – the latest spying malware from an allegedly French espionage group analyzed

BY JOAN CALVET POSTED 30 JUN 2015 - 11:12AM

In this blog we describe a sophisticated backdoor, called Dino by its creators. We believe this malicious software has been developed by the Animal Farm espionage group, who also created the infamous **Casper**, **Bunny** and **Babar** malware. Dino contains interesting technical features, and also a few hints that the developers are French speaking.

Animal Farm is the security industry's name for a group of attackers first described by Canada's Communications Security Establishment (CSE) in **a set of slides** leaked by Edward Snowden in March 2014. In those slides CSE assess with "moderate certainty" that this group is a French intelligence agency. Since then, several examples of malware created by Animal Farm have been found and publicly documented, in particular:

- Casper, a stealthy first-stage implant, **documented by ESET in last March**
- Bunny, a Lua-based backdoor, documented by **Marion Marschalek (Cyphort)**
- Babar, an espionage platform, also analyzed by **Marion Marschalek**

The connection between those pieces of malware and the group described in CSE slides has been convincingly established, for example by **Paul Rascagnères (G Data)**.

In this blog post we add a new piece to the puzzle with Dino, another malicious program belonging to Animal Farm's arsenal.

Introduction

The sample of Dino documented in this blog post was used in 2013 against targets in Iran. The original means of infection is unknown, though we believe Dino was installed by another program, as it contains an uninstallation command without the corresponding installation procedure. Given the set of commands it can receive, Dino's main goal seems to be the exfiltration of files from its targets.

The binary's original name, "Dino.exe", has been left visible by its authors, as was the case with Casper. Dino – which could be referring to the pet character from **The Flintstones** cartoon show – was already mentioned in a recent **Kaspersky blog** as a "full-featured espionage platform," but no technical analysis has been published yet.

Roughly, Dino can be described as an elaborate backdoor built in a modular fashion. Among its technical innovations, there is a custom file system to execute commands in a stealthy fashion, and a complex task-scheduling module working in a similar way to the "cron" Unix command. Interestingly, the binary contains a lot of verbose error messages, allowing us to see Dino's developers' choice of wording. Also, a few technical artefacts suggest that Dino was authored by native French speakers.

Dino Basics

Modules List

Dino has been developed in C++ and presents a well-defined modular architecture. The following array lists the modules contained in this Dino binary; the module names are those assigned by the developers.

Module	

Name	Module Purpose
PSM	Encrypted on-disk copy for Dino modules
CORE	Configuration storage
CRONTAB	Task scheduler
FMGR	File upload and download manager
CMDEXEC	Command execution manager
CMDEXECQ	Storage queue for commands to execute
ENVVAR	Storage for environment variables

Data Structure

Dino heavily relies on a custom data structure named “DataStore” by the Animal Farm developers. In particular, all Dino’s modules store their content inside this structure, making its understanding one of the keys to analyzing Dino.

A DataStore is a **map** from string keys to values of 8 possible types, such as integers or strings. The implementation of this data structure is based on a **hash table**. It means that to retrieve the value associated with a key, one has to calculate the hash of the key to locate a bucket from which the value can be retrieved.

Dino’s hash is a one-byte value calculated with a series of XOR operations on the key, and each bucket starts a linked list containing key/value pairs. The code responsible for retrieving the value associated with a key is shown in Figure 1.

```
// Get the hash of the key to look for
hash = 0x55;
for ( i = arg_key_to_look_for; *i; key_char = *i )
{
  ++i;
  hash ^= key_char;
}
// Look for the key in the linked list starting
// at the index "hash % number of buckets"
for ( current_field = (datastore->array_of_buckets
                       + 4 * (hash % datastore->number_of_buckets));
      current_field;
      current_field = (obj_datastore_field *)current_field->next_field )
{
  // True when the two strings are equals
  if ( !_stricmp((const char *)current_field->key, arg_key_to_look_for) )
    break;
}
```

Figure 1

Finally, DataStore objects can be serialized in a custom format, which begins with the magic word “DxSx”. This is used in particular by the PSM module to save the content of Dino modules in an encrypted file. More precisely, when a modification is made to a module’s content in memory, the PSM module saves it as a serialized DataStore. When Dino restarts, the module is deserialized from the file and loaded into memory. Funnily enough, the key serving to encrypt the file on disk is “PsmIsANiceM0du1eWith0SugarInside”.

Configuration

Dino’s configuration is initially stored in a serialized DataStore object contained in a zip archive at the end of the Dino binary. At runtime this object is deserialized and stored inside the CORE module. We can list the configuration’s content with Dino’s “conf –l CORE” command – described later – which displays on separate rows each key’s name, its associated value and the type of this value:

Started:5523F782 QWORD
InitialWaitDone:00000001 DWORD
InteractiveDelay:00000005 DWORD
MaxNothingSaidCount:00000078 DWORD
InstallDate: 5523F782 QWORD
fields:78537844…[*REDACTED*]…66B3900 BYTES
recID:11173-01-PRS WIDESTR
Version:1.2 WIDESTR
BD_Keys: 4D41474943424F58…[*REDACTED*]…9EB3506 BYTES
CC_Keys: 4D41474943424F58…[*REDACTED*]…0000000 BYTES

MaxDelay:00000E10 DWORD
ComServer0:hXXp://www.azhar.bf/…[*REDACTED*]…/postal.php STR
ComServer1:hXXp://www.rsvniima.org/…[*REDACTED*]…/din12/postal.php STR
ComServer2:hXXp://www.azhar.bf/…[*REDACTED*]…/postal.php STR
ComServer3:hXXp://www.rsvniima.org/…[*REDACTED*]…/din12/postal.php STR
ComServer4:hXXp://dneprorudnoe.info//…[*REDACTED*]…/postal.php STR
ComServer5:hXXp://dneprorudnoe.info//…[*REDACTED*]…/postal.php STR
ComServer6:hXXp://dneprorudnoe.info//…[*REDACTED*]…/postal.php STR
NextSendReceive:5CC33097FB72D001 BYTES
CC:000064F7-72E4-3F7D-C817-474D-A9BDBDF7 STR
DaysOfLife:00000000 DWORD
GUID:12FEB4A9EEDEE411B283000C29FD2872 BYTES
InitialDelay:00000000 DWORD
now:5523F78E QWORD
hash:A88E8181CA5CE35AE70C76145DFB820D BYTES
InitialCommands:78537844…[*REDACTED*]…000000 BYTES
xT0rvwz:DC188352A…[*REDACTED*]…00000 BYTES
tr4qa589:K/[RAFtIP?ciD?:D STR
jopcft4T:a.ini WIDESTR

While most of the keys have self-explanatory names, we would like to focus on the following keys:

- “recID”: Animal Farms binaries contain an ID whose decimal value appears to identify the target, “11173-01-PRS” in this case. For example Casper used an “ID” value set to “13001”, whereas some Babar samples used “12075-01” and “11162-01”. We do not know the meaning of the “PRS” suffix added in the case of Dino.

- “ComServer”: These keys contain the command and control (C&C) servers’ URLs. All the URLs were down when we started our analysis. Those C&Cs were compromised legitimate websites, which is standard operating procedure for Animal Farm.

- “Version”: Dino’s code version; here set to “1.2”, which is confirmed by the “din12” folder used in one of the C&C URLs. For the record, a “d13” folder has been seen on another Animal Farm C&C (see “3.7 Calling home” of **Marschalek’s Babar report**), indicating that Dino version 1.3 has also likely been deployed at some point.

- “BD_Keys” and “CC_Keys” contain cryptographic keys to encrypt the network communications with C&C servers. Their values start with the word “MAGICBOX”.
- The three last keys are displayed with obfuscated names (“xT0rvwz”, “tr4qa589” and “jopcft4T”) and store parameters for the custom file system we will describe later.

Commands

The following Table lists the commands accepted by this Dino binary with the names chosen by the developers. Each of those commands can take one or more arguments.

Command	Purpose
sysinfo	Retrieve reconnaissance information from the machine
killBD	Uninstall Dino using the custom file system (see ramFS description below for details)
!	Execute Windows batch command passed as a parameter
cd	Change the current work directory
pwd	Retrieve the current work directory path
dir	List files in a given directory with various additional information
set	Set or remove environment variables stored in the ENVVAR module
conf	Display or update module content
search	Search for files whose names match given patterns. The files found are packed in an archive, which is then scheduled for upload to the C&C using the FMGR module.
archive	Create an archive from given file paths
unarchive	Unpack an archive to a given location
download	Schedule a file transfer to the C&C using the FMGR module
cancel	Remove the next file transfer scheduled in the FMGR module
cancelall	Remove all scheduled file transfers in the FMGR module
cronadd	Schedule a command to be executed at a certain time by the CRONTAB module (see CRONTAB description below for details)
cronlist	List registered entries in the CRONTAB module
crondel	Remove an entry in the CRONTAB module
wakeup	Schedule a wake-up of the malware after a certain amount of time using the CRONTAB module
restart	N/A: the command is actually not implemented
showip	Display the public IP of the infected machine
cominfos	Display information about the currently used C&C server
comallinfos	Display information about all known C&C servers
wget	Download a file from the currently used C&C server onto the machine
delayttk	Delay the de-installation of the malware, if scheduled

One command of particular interest is “search”, which allows the operators to look for files very precisely. For example, it can provide all files with a “.doc” extension, the size of which is bigger than 10 kilobytes, and that were modified in the last 3 days. We believe this exfiltration of files to be Dino’s end goal.

At startup Dino executes successively the commands stored in the “InitialCommands” field in its configuration; in the sample we analyzed they are:

```
sysinfo
cominfos
!ipconfig /all
!ipconfig /displaydns
!tracert www.google.com
```

Those commands serve as a reconnaissance step for the operators. Their execution is managed by the CMDEXEC module, the commands being stored in a queue inside the CMDEXECQ module. The result is reported to the C&C server.

After having described Dino’s basics, we are now going to dig into two particularly interesting components; first, a custom file system used by the malware, and then the CRONTAB module in charge of task scheduling.

RamFS: A Temporary File System

Dino contains a custom file system named “ramFS” by its developers. It provides a complex data structure to store files in memory, each of them bearing a name corresponding to filenames used by usual file systems. RamFS also comes with a set of custom commands that can be stored in files and executed. It should be noticed that ramFS is also present in other Animal Farm binaries (see attribution paragraph below), but since we are unaware of previous analysis of ramFS, we are describing our findings here.

Architecture

RamFS content is initially stored encrypted in Dino’s configuration under the key “xT0rvwz”, whereas the corresponding RC4 key is stored under the key “tr4qa589”. Once the file system has been decrypted, it is stored in memory as a linked list of 512-byte memory chunks, each one of them being individually RC4-encrypted. When looking for a file in ramFS, each chunk is decrypted, processed and then re-encrypted. Hence there are very few noticeable traces of ramFS during its use.

Here are some high-level characteristics of this file system:

- File names and file content are encoded in Unicode
- File names length is limited to 260 characters
- Once decrypted, file content is manipulated as chunks of 540 bytes
- There is no metadata associated with the files

We could not find an existing file system matching the memory structures and the characteristics of ramFS, and therefore we believe this file system to be an original creation of the Animal Farm group.

Commands

Several commands can be executed in the context of ramFS, as listed in the following Table.

Command	Meaning
CD	Change the current work directory on the real file system
MD	N/A: the command is actually not implemented
INSTALL	Installation or de-installation of Dino, in Windows registry and/or as a service
EXTRACT	Extracts a file stored in ramFS onto the machine
DELETE	Deletes a file stored on the machine
EXEC	Executes a file stored in ramFS
INJECT	Injects a file stored in ramFS into a running process
SLEEP	Sleeps for a given amount of time
KILL	Terminates a running process
AUTODEL	N/A: the command is actually not implemented

Usage of ramFS in Dino

In the case of Dino, ramFS serves as protected storage for one specific file containing the instructions to remove the malware from the machine. The developers named this file the “cleaner” and it is executed when Dino receives the command “killBD” (the “BD” acronym is the developers’ designation of the malware).

Figure 2 shows the code responsible for executing this cleaner file. First, it retrieves the name of the file from Dino’s configuration (“a.ini”), then it retrieves the key to decrypt ramFS, and it finally mounts the file system in memory in order to execute the cleaner file stored inside. The verbosity of the error messages makes it particularly easy to understand the purpose of the code.

```
// Search for the cleaner file name in Dino configuration
cleaner_file_name = DataStore::SearchForKey(dino_config_datastore, k_cleaner_file_name);
if ( cleaner_file_name && cleaner_file_name->type == WIDESTR )
{
  // Search for the key to decrypt ramFS
  ramfs_crypto_key = DataStore::SearchForKey(dino_config_datastore, k_ramfs_crypto_key);
  if ( ramfs_crypto_key && ramfs_crypto_key->type == STR )
  {
    if ( MountRamFS(...)
    {
      ExecuteCleanerRamFS((int)&var_ramfs_obj);
      DataStore::StoreValue(v14, "results", L"cleaner executed, exiting", a1);
      DataStore::StoreValueFixedSize(arg_datastore, "destroyPSM", 1, 0, 1);
    }
    else
    {
      DataStore::StoreValue(v11, "results", L"Unable to mount cleaner RamFS, exiting", a1);
    }
  }
  else
  {
    DataStore::StoreValue(v10, "results", L"No cleaner Passphrase Found, exiting", a1);
  }
}
else
{
  DataStore::StoreValue(a1, "results", L"No cleaner Script Found, exiting", a1);
}
```

Figure 2

The cleaner file contains the string “INSTALL -A “wusvcd” -U” which, once executed, will uninstall the malware from the machine – “wusvcd” being the name used to register Dino on the machine.

Hence, ramFS serves as a protected container for files to be executed on the machine, offering a disposable execution environment to the operators and leaving very few traces on the system.

Tasks scheduling in a Unix fashion

The commands “cronadd”, “cronlist” and “crondel” serve respectively to add, list, and remove scheduled tasks registered in the CRONTAB module. Those tasks are Dino’s commands.

The syntax to define scheduled tasks is similar to the one used by the **cron Unix command**. In particular the time at which to run a command is given by a string following the format “minute hour day month year dayofweek”. Alternatively, this string can be replaced by “@boot” for a command to run at each startup – whereas some Unix cron implementations accept “@reboot”.

As an example, here is the output of the “cronlist” command after a “wakeup” command has been scheduled to run on 7th April 2015 at 15:44:

Id	Cron String						Local	Count	Command	Visibility
C1	44	15	07	04	2015	*	-d	-1	wakeup	regular

As we can see, each entry is identified by an “Id”, an incrementing hexadecimal number starting at 0xC0. The purpose of the “Local” field remains unclear (the other possible value being “-l”). The “Count” parameter counts the number of times a command has been executed, “-1” indicating the command will be executed only once. Finally, the “Visibility” field defines whether the command execution will be reported to the C&C (the other possible value being “Silent”).

Attribution

Dino Belongs To The Farm

The amount of shared code between Dino and known Animal Farm malware leaves very little doubt that Dino belongs to Animal Farm’s arsenal. Among these shared features, we can cite the following:

- At the very beginning of Dino execution, the current process name is checked against process names used by some sandboxes:

```
// Converts the file name to lowercase
_wcslwr_s(Filename, 0x104u);
// Checks the file name against sandbox names
if ( wcsstr(Filename, L"klavme.exe") )
  ExitProcess(0);
if ( wcsstr(Filename, L"myapp.exe") )
  ExitProcess(0);
if ( wcsstr(Filename, L"testapp.exe") )
  ExitProcess(0);
result = (DWORD)wcsstr(Filename, L"afyjevmv.exe");
if ( result )
  ExitProcess(0);
```

Figure 3

A very similar check (against “klavme”, “myapp”, “TESTAPP” and “afyjevmv.exe”) is present in Bunny samples, and in some first-stage implants deployed by Animal Farm.

- To hide its calls to certain API functions, Dino employs a classic Animal Farm ploy: a hash is calculated from the function’s name and used to look for the address of the API function. The actual hashing algorithm used

in Dino is the same that was used in **Casper**, namely a combination of rotate-left (ROL) of 7 bits and exclusive-or (XOR) operations.

- The Dino's custom file system – the so-called ramFS – is present in several droppers used by Animal Farm. In those binaries the file system serves to set the persistence of the payload. For example, here is the command executed by some NBOT droppers in the context of ramFS:

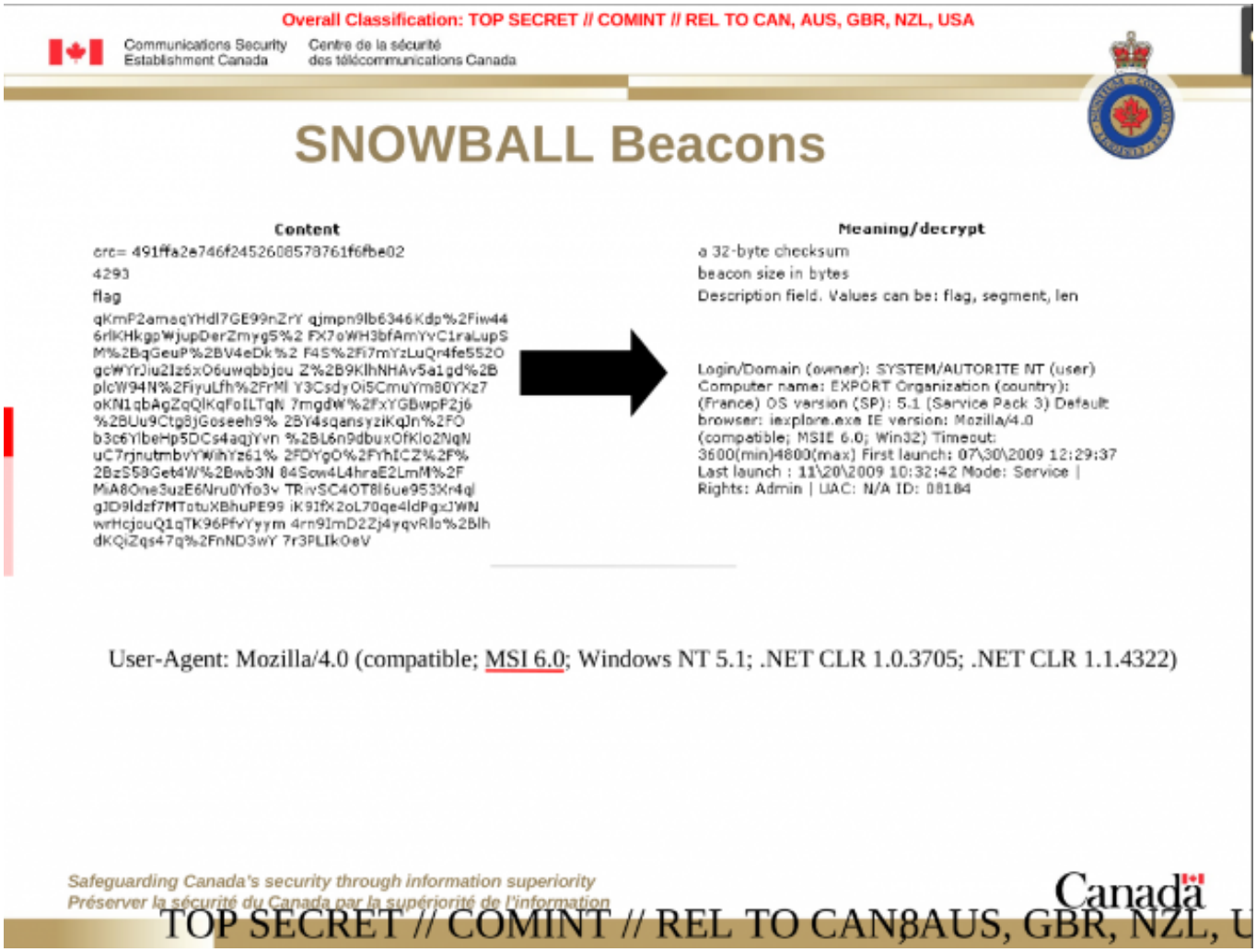
```
INSTALL -A "Net3D" -B "Net3d.exe" -D "3D Network Service" -C "3D Network Service" -F
```

- As a final indication that Dino belongs to Animal Farm menagerie, it is noticeable that the output of Dino's sysinfo command looks like an updated version of the "beacon" from the SNOWBALL implant described in the leaked CSE slides – part of operation SNOWGLOBE, which led to the discovery of Babar:

Dino's sysinfo example output

```
Login/Domain (owner): Administrator/JOHN (john)
Computer name: JOHN
Organization (country):  (United States)
RecId: 11173-01-PRS
MaxDelay: 3600
Version: 1.2
OS version (SP): 5.1 (Service Pack 3)
WOW64: No
Default browser: firefox.exe
IE version: Mozilla/4.0 (compatible; MSIE 7.0; Win32)
First launch: 04/01/2015 – 18:31:14
Time to kill: N/A
Last launch : 04/01/2015 – 19:21:44
Mode: N/A  |  Rights: Admin  |  UAC: No
ID: 4635BEF0-D89D-11E4-B283-000C-29FD2872
InstallAv: 0
Inj: Yes
```

SNOWBALL implant beacon



All these indicators together make us very confident that Dino was developed by the Animal Farm group.

French speaking Developers

Dino adds at least two more indicators to those already documented suggesting that Animal Farm developers are French speaking:

- Dino's binary contains a resource whose language code value is 1036. The original purpose of this language code is to allow developers to provide resources (menus, icons, version information…) for different locations in the world in the corresponding language. Interestingly, when a developer does not manually specify the language code, the compiler sets it to the language of the developer's machine. So, which language corresponds to the value 1036, or 0x40c in hexadecimal? **French (France)**.

Of course a non-French speaking developer could have deliberately set this value to mislead attribution efforts. But in more recent Animal Farm binaries (for example Casper), this language code has been set to the classical English (USA) language code. Therefore, it seems that Animal Farm developers forgot to set this value in their first creations, realized their mistake at some point, and decided to set a standard value. Someone using the language code as a false flag would have likely kept the strategy going.

For the record, this Dino sample is not the only Animal Farm binary with 1036 as language code.

- Dino's binary is statically linked with the **GnuMP library**, which is used to manipulate big numbers in cryptography algorithms. The GnuMP code in Dino contains file paths coming from the developer's machine:

```
..\..\src\arithmetique\mpn\mul.c
..\..\src\arithmetique\printf\doprnt.c
..\..\src\arithmetique\mpn\tdiv_qr.c
..\..\src\arithmetique\mpn\mul_fft.c
..\..\src\arithmetique\mpn\get_str.c
```

As the attentive reader has probably guessed, "arithmetique" is the French translation of "arithmetic".

Conclusion

Dino's binary shows an intense development effort, from custom data structures to a homemade file system. As with other Animal Farm binaries, it bears the mark of professional and experienced developers.

But Dino also shows a poor knowledge, or interest, from these developers in anti-analysis techniques – contrary to what was seen in Casper – as demonstrated, for example, by the verbosity of some Dino's log messages:

```
"update ttk with negative of null value is forbidden, consider using
killbd instead"

"update not done, value wasn't already in module and type mispelled or
missed"

"archive %s successfully created, but unable to schedule download.Try
to manually download/erase it."

"Date is invalid ! Date Format is ddmmyyyy"

"decyphering failed on bd"
```

All those messages provide substantial help in understanding Dino's internal workings. One will also appreciate the numerous misspellings contained in the messages.

Regarding Dino's victims, we know very little except that they were located in Iran in 2013. This is in accordance with the victimology described by Canada's CSE in its presentation:

That leads us to the final point of this blog: several signs suggest that Dino's creators are French speaking developers. These signs add to the pretty long list of indicators already supporting this hypothesis, in particular the ones mentioned by **Canada's CSE**.

Indicators of Compromise

Indicator	Value
Sample SHA1	*BF551FBDCF5A982705C01094436883A6AD3B75BD*
C&C URL	hXXp://www.azhar.bf/modules/mod_search/found/cache/postal.php
C&C URL	hXXp://www.rsvniima.org/templates/rsv/icons/din12/postal.php
C&C URL	hXXp://dneprorudnoe.info/sxd/lang/i18n/charcodes/postal.php
Path	C:\Program Files\Common Files\wusvcd\wusvcd.exe
Default storage file names	C:\Program Files\Common Files\wusvcd\wusvcd00000000-0000-0000-0000-00000000.{dax,dat,lck}
Downloaded file name extension	.tmp_dwn
Registry key	Software\Microsoft\Windows\Windows\CurrentVersion\Run\wusvcd

Author **Joan Calvet**, ESET