EvilBunny: Suspect #4

EVIL BUNNY: SUSPECT #4

SAMPLE DETAILS

Dropper

MD5	c40e3ee23cf95d992b7cd0b7c01b8599
SHA-1	1e8b4c374db03dcca026c5feba0a5c117f740233
File Size	943.5 KB (966144 bytes)
Compile Time	2011-10-25 19:28:00

Suspect #4

MD5	3bbb59afdf9bda4ffdc644d9d51c53e7
SHA-1	1798985f4cc2398a482f2232e72e5817562530de
File Size	773.5 KB (792064 bytes)
Compile Time	2011-10-25 19:28:39

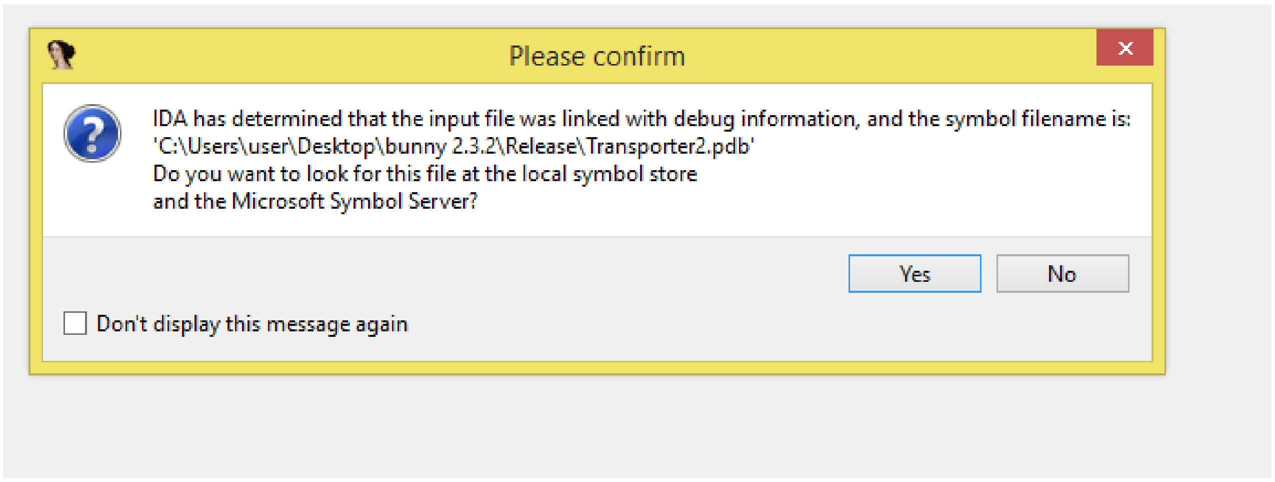
Marion Marschalek (marion@0x1338.at), 16th of October 2014
BCC0 7607 2FFA BCA8 9048 D648 D169 73AF F372 F2CA

The EvilBunny attacks have been presented at Hack.lu Conference 2014 in Luxembourg (http://2014.hack.lu/index.php)

CONTENTS

1. THE BUNNY DROPPER

Suspect #4 is the fourth out of a row of malware samples, which are believed to stem from the same malicious actor. Insights on the campaign were first presented at Hack.lu Conference 2014 in Luxembourg [8]. The name 'EvilBunny' is derived from a link to debug information embedded in the dropper binary of suspect #4. The path to the associated .pdb-file contains the name of the project and sub project, namely 'bunny 2.3.2' and 'Transporter2'.


Please confirm

IDA has determined that the input file was linked with debug information, and the symbol filename is:
'C:\Users\user\Desktop\bunny 2.3.2\Release\Transporter2.pdb'
Do you want to look for this file at the local symbol store
and the Microsoft Symbol Server?

Illustration 1: IDA Pro's request for the associated .pdb-file

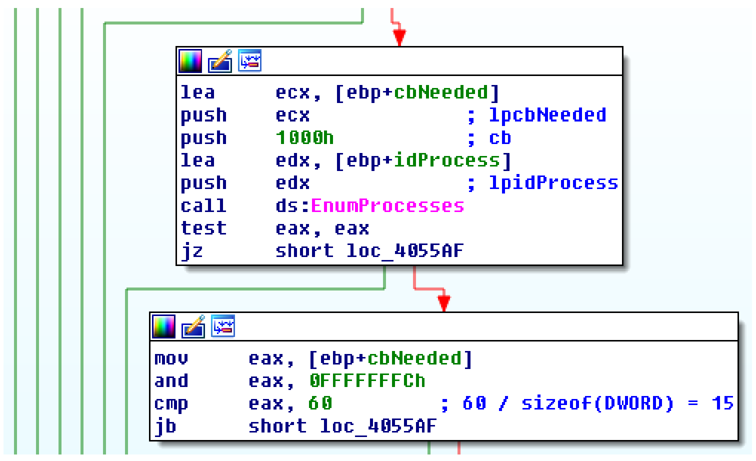
EvilBunny has been seen spreading through a malicious PDF document, exploiting CVE-2011-4369 on 20th of December 2011 [1]. The exploit would download the bunny dropper and install suspect #4 on the infected system. Both, dropper and payload were compiled on 25th of October 2011, while CVE-2011-4369 was released only on 16th of December. It is unclear as of when the attacks started; although, clearly, either the attackers adopted the vulnerability within 4 days or the attacks were performed using an AdobeReader 0-day.

The functionality of the dropper can be summarized in the following steps:

- Sandbox check and anti virus product enumeration
- Dropping payload 'netmgr.exe'
- Creating a registry key for persistence
- Creating a registry key for deletion of the dropper

Searching for a sandbox environment, the malware takes a number of simple steps to verify. It tests the module file name to see if it is less than 5 characters long or if it contains either of the four strings 'klavme', 'myapp', 'TESTAPP' or 'afyjevmv.exe'. Also, it verifies if less than 15 processes are running in the environment, using the API call EnumProcesses.

In case any of the conditions is met execution will abort. Up to the time of writing it remains unclear which environment the mentioned strings aim to identify, if any.



```
lea     ecx, [ebp+cbNeeded]
push    ecx                     ; lpcbNeeded
push    1000h                   ; cb
lea     edx, [ebp+idProcess]
push    edx                     ; lpidProcess
call    ds:EnumProcesses
test    eax, eax
jz      short loc_4055AF
```

```
mov     eax, [ebp+cbNeeded]
and     eax, 0FFFFFFFCh
cmp     eax, 60         ; 60 / sizeof(DWORD) = 15
jb      short loc_4055AF
```

Illustration 2: The process count as environment check

Accessing the systems WMI (Windows Management Interface) the malware queries the installed AntiVirus software by issuing 'SELECT * FROM AntiVirusProduct'. More specifically, the malware is interested in the data fields 'productUptoDate', 'VersionNumber', 'DisplayName' and 'productState'. The information about AntiVirus provided by WMI differs between Windows versions, so the requests address different namespaces, 'ROOT\SecurityCenter' for WindowsXP and 'ROOT\SecurityCenter2' for Windows Vista and newer versions.

Names of AntiVirus products are represented as hard coded SHA-256 hashes, namely the following:

```
d4634c9d57c06983e1d2d6dc92e74e6103c132a97f8dc3e7158fa89420647ec3
4db3801a45802041baa44334303e0498c2640cd5dfd6892545487bf7c8c9219f
bfe74ca464620a62f11b8c47a3778bb132d84fecd90ce7c75817970f2eeeca51 Antivirus
443b6fb65fa57d57ee3113e48e9b4ed1db2921d5352e27fa85064cd60553c3ff BitDefender
e1625a7f2f6947ea8e9328e66562a8b255bc4d5721d427f943002bb2b9fc5645
588730213eb6ace35caadcb651217bfbde3f615d94a9cca41a31ee9fa09b186c
f1761a5e3856dceb3e14d4555af92d3d1ac47604841f69fc72328b53ab45ca56 Kaspersky
```

Only three of the seven hashes were identified. If any of the indicated products is installed and active on the machine execution will abort.

The dropper comes with its payload attached in the resource section. The resource RCData contains the entire netmgr.exe binary; alleged suspect #4. The binary is dropped as netmgr.exe, either located in %APPDATA%\Perf Manager\ or %WINDIR%\msapps\. The final directory location depends whether the dropper is running with administrative privileges or not.

Interestingly, the dropper also changes the creation time stamp of the newly created binary to time stamp of the system executable explorer.exe. This serves to avoid raising suspicion of a forensic analysts or tools.

As a means of persistence the dropper creates an entry under [HKLM|HKCU]\...\CurrentVersion\Run which points to the dropped netmgr.exe binary. This ensures that netmgr.exe will be executed every time the system boots. Furthermore, the dropper generates another entry in the Windows registry, under [HKLM|HKCU]\Software\Microsoft\IPSec. This entry is named 'isakmpAutoNegociate' and points to the dropper binary. This key will be requested by netmgr.exe later to locate the dropper binary and delete it.

The naming of the key for cleanup of the dropper is chosen to confuse analysts, although the name is clearly misspelled.

Another interesting side fact is that the dropper does not start the dropped malware, neither does it contain code which could even do so. This implies, the final infection stage does not start unless the infected machine is rebooted. Also, the dropper binary will not be deleted in the meanwhile.

2. SUSPECT #4

Suspect #4 was the fourth discovered sample in a row of malware related to the EvilBunny attacks; even before its dropper. Yet, it is by far the most interesting one. It is a multi-threaded bot with an integrated scripting engine. It incorporates a Lua engine and downloads and executes Lua scripts to reach a certain level of polymorphism. The Lua scripts can call back into the C++ code to alter the malware behavior at runtime.

The malware seeks to keep a low profile on the infected machine, while executing the botmaster's commands and Lua scripts. In total Suspect #4 exhibits three different methods for receiving C&C input and executing commands; directly via HTTP, through a downloaded database file or as a scheduled task. Also, the malware will generate numerous files to help its execution and frequently reply back to the C&C with status messages.

The initial purpose of the malware seems to be sharing execution load among infected host machines. However, due to the lack of the original Lua scripts and the extensive functionality of the embedded Lua engine the original intentions of the attackers remain unknown.

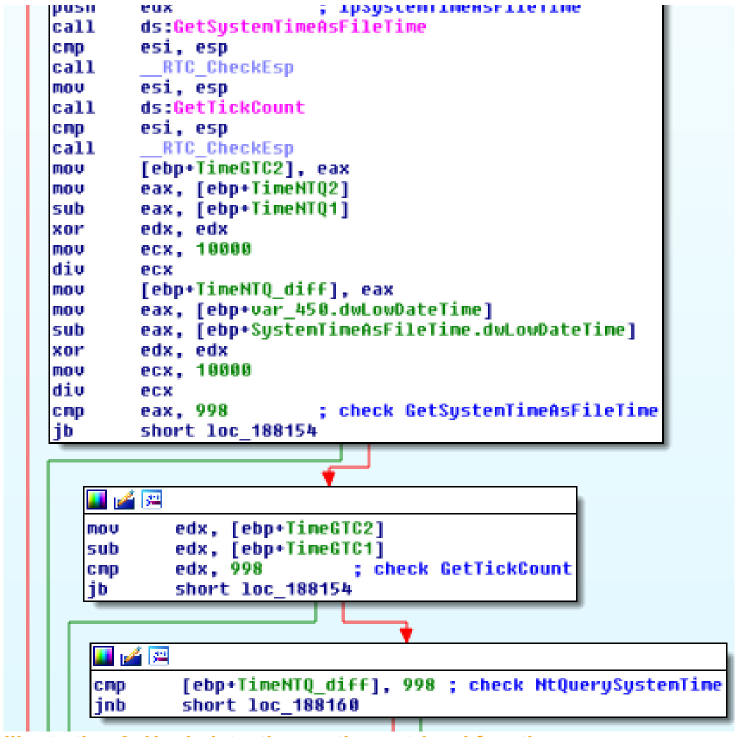
2.1 MALWARE ANTI-ANALYSIS

Suspect #4 is written in C++, and just like its dropper, it is not protected by any runtime packer or crypter. It is, however, compiled with a performance optimization option which causes a lot of code in-lining and duplicate code in the binary. Duplication also applies to string constants, as every string is present once per function, if the given function uses it. This might accelerate execution speed of the bot at runtime, but implicitly or even on purpose hinders static analysis of the binary code. Cross referencing of strings or helper functions is ineffective due to the duplications.

Only few anti-analysis measures can be found in the binary. One function exists, which resembles the sandbox check of its dropper with minor extensions. This sandbox detection consists of three parts:

- The module path name should be more than 5 characters long and contain either 'msapps\' or 'Perf Manager\', which is the case if the sample was legitimately dropped by its dropper.
- The module file name should not contain either of the following strings: 'klavme', 'myapp', 'TESTAPP' or 'afyjevmv.exe'.

- The function performs a hook detection on time retrieval APIs. These are NtQuerySystemTime, GetSystemTimeAsFileTime and GetTickCount. Every API is called twice to calculate a delta, while performing a sleep(1000) operation between iteration one and iteration two. The final condition is, if any of the three deltas is below 998 milliseconds execution will abort. This can only be the case if any of the three API's return values is modified by a system monitoring solution, like a sandbox.



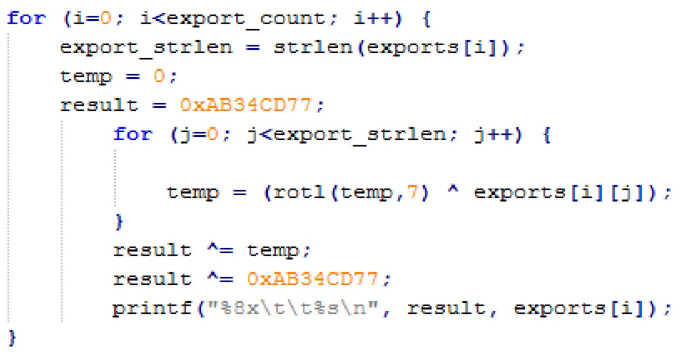
```
push    eax             ; lpSystemTimeAsFileTime
call    ds:GetSystemTimeAsFileTime
cmp     esi, esp
call    __RTC_CheckEsp
mov     esi, esp
call    ds:GetTickCount
cmp     esi, esp
call    __RTC_CheckEsp
mov     [ebp+TimeGTC2], eax
mov     eax, [ebp+TimeNTQ2]
sub     eax, [ebp+TimeNTQ1]
xor     edx, edx
mov     ecx, 10000
div     ecx
mov     [ebp+TimeNTQ_diff], eax
mov     eax, [ebp+var_450.dwLowDateTime]
sub     eax, [ebp+SystemTimeAsFileTime.dwLowDateTime]
xor     edx, edx
mov     ecx, 10000
div     ecx
cmp     eax, 998        ; check GetSystemTimeAsFileTime
jb      short loc_188154

mov     edx, [ebp+TimeGTC2]
sub     edx, [ebp+TimeGTC1]
cmp     edx, 998        ; check GetTickCount
jb      short loc_188154

cmp     [ebp+TimeNTQ_diff], 998 ; check NtQuerySystemTime
jnb     short loc_188160
```

Illustration 3: Hook detection on time retrieval functions

Another evasion trick implemented by the malware is the partial obfuscation of API names. For no obvious reason though, the obfuscation algorithm is only applied to a small subset of APIs used in the code. The algorithm uses the fixed value AB34CD77h and a combination of XOR and ROL instructions to calculate hashes from the export names of kernel32.dll, advapi32.dll and psapi.dll. It is important to note that this obfuscation trick and the according hash function remain consistent through out all related samples.



```
for (i=0; i<export_count; i++) {
    export_strlen = strlen(exports[i]);
    temp = 0;
    result = 0xAB34CD77;
        for (j=0; j<export_strlen; j++) {

            temp = (rotl(temp,7) ^ exports[i][j]);
        }
        result ^= temp;
        result ^= 0xAB34CD77;
        printf("%8x\t\t%s\n", result, exports[i]);
}
```

Illustration 4: The API name hashing function

2.1.1 ENUMERATION OF FIREWALL AND ANTIVIRUS PRODUCTS

Similar to its dropper, suspect #4 enumerates the system's installed AntiVirus product and checks firewall settings. For AntiVirus detection it utilizes WMI and issues 'SELECT * FROM AntiVirusProduct'. Identification of installed products is achieved by comparing the hashed product name to a set of SHA-256 hashes; although, a different and much bigger set than used by its dropper. With the help of hash tables from perturb.org and md5decoder.org it is possible to revert some of the hashes to their original text form:

a48be88bed64eff941be52590c07045b896bc3e87e7cf62985651bbc8484f945	MAfee
1ba035db418ad6acc8e0c173a49d124f3fcc89d0637496954a70e28ec6983ad7	
a7f9b61169b52926bb364e557a52c07b34c9fbdcd692f249cd27de5f4169e700	
2bc42b202817bdab7d49506d291e3d9624ae0069087a8949c8fcb583c73772b1	Norton
f7d9ea7f3980635237d6ea58048057c33a218f2670e0ff45af5f4f670e9aa6f4	Panda
522e5549af01c747329d923110c058b7bb7e112816de64bd7919d7b9194fba5b	Rising
4db3801a45802041baa44334303e0498c2640cd5dfd6892545487bf7c8c9219f	
d4634c9d57c06983e1d2d6dc92e74e6103c132a97f8dc3e7158fa89420647ec3	
9e217716c4e03eee7a7e44590344d37252b0ae75966a7f8c34531cd7bed1aca7	Trend
e1625a7f2f6947ea8e9328e66562a8b255bc4d5721d427f943002bb2b9fc5645	
588730213eb6ace35caadcb651217bfbde3f615d94a9cca41a31ee9fa09b186c	
ab6ed3db3c243254294cfe431a8aeada28e5741dfa3b9c8aeb54291fddc4f8c3	
b3fe0e3a3e3befa152c4237b0f3a96ffaa44a2d7e1aa6d379d3a1ab4659e1676	AntiVir
0d21bd52022ca7f7e97109d28d327da1e68cc0bedd9713b2dc2b49d3aa104392	
c0ffcaf63c2ca2974f44138b0956fed657073fde0adeb0b1c940b5c45e8a5cab	avast
249a90b07ed10bd0cd2bcc9819827267428261fb08e181f43e90807c63c65e80	AVG
b39be67ae54b99c5b05fa82a9313606c75bfc8b5c64f29c6037a32bf900926dd	
4b650e5c4785025dee7bd65e3c5c527356717d7a1c0bfef5b4ada8ca1e9cbe17	CA
c8e8248940830e9f1dc600c189640e91c40f95caae4f3187fb04427980cdc479	
97010f4c9ec0c01b8048dbad5f0c382a9269e22080ccd6f3f1d07e4909fac1a5	
aa0ad154f949a518cc2be8a588d5e3523488c20c23b8eb8fafb7d8c34fa87145	
333e0a1e27815d0ceee55c473fe3dc93d56c63e3bee2b3b4aee8eed6d70191a3	G
977781971f7998ff4dbe47f3e1d679f1941b3237d0ba0fdca90178a15aec1f52	Jiangmin
f1761a5e3856dceb3e14d4555af92d3d1ac47604841f69fc72328b53ab45ca56	Kaspersky

The malware retrieves the attributes 'productState', 'DisplayName' and on WindowsXP also 'productUptoDate'. It is noteworthy though, that for netmgr.exe the existence of any of the indicated AntiVirus products does not pose a reason to end execution. In fact, these hashes are the only measure for obfuscating strings in the binary.

Firewall detection works equally via querying WMI through "SELECT * FROM FirewallProduct", while this time the malware searches for the attributes 'enabled', 'VersionNumber' and 'DisplayName'.

AntiVirus and firewall information is stored in objects in a linked list at runtime. It will later either be exfiltrated to a C&C server directly or dumped to a local file on disk. Either way, the results of this check do not influence the flow of execution.

2.2 SYSTEM INFILTRATION

After start-up the malware aims to inject itself to an svchost.exe process. For stealthiness the malware can either inject itself as a remote thread to a running svchost.exe process or perform the same procedure on a newly created one.

As a singular means of persistence the malware relies on the auto-start registry key created by its dropper under [HKLM|HKCU]\...\CurrentVersion\Run, which will start netmgr.exe on every system boot.

Also, the payload retrieves the dropper's path from 'isakmpAutoNegociate' under [HKLM|HKCU]\Software\Microsoft\IPSec, then deletes the binary and the 'isakmpAutoNegociate' registry key.

2.3 CONFIGURATION AND COMMAND & CONTROL SETTINGS

At start-up netmgr.exe decrypts a configuration file stored in its resource section, revealing three URLs, among timeout settings and encryption keys:

- http://le-progres.net/images/php/test.php?rec=11206-01
- http://ghatreh.com/skins/php/test.php?rec=11206-01
- http://www.usthb-dz.org/includes/php/test.php?rec=11206-01

All three of these URLs served as C&C contacts when the attack was still ongoing, sending commands or Lua scripts to the infected host. Interesting is that two of the domains are actual fake domains resembling legitimate websites while one is a legitimate domain nowadays. Le-progres.net could be easily confused with leprogres.fr, the official website of the Lyon news paper Le Progres. Usthb-dz.org is remarkably similar to usthb.dz, the website of the University of Sciences and Technology Houari Boumediene in Algiers, Algeria.

The third domain though, ghatreh.com, is the legitimate address of an Iranian news website. According to domaintools.com, this domain has been created in November 2004 while the last change happened in September 2012. Also, domaintools.com reveals 14 unique IP addresses and 4 different registrars. It is assumed that the domain has been dropped by the attackers in 2012 and picked up by legitimate operators.

خانه - خبر - پیشخوان - آب و هوا - تماس با ما - استخدام - ویدئو های ورزشی - طنز و کاریکاتور - بازیهای آسیایی - نتایج زنده - ارسال خبر   ورود - عضویت

Sat, 15th Nov 2014 - 1393 یکشنبه، 25 آبان

فقط تصاویر جستجو

سیاسی | اقتصادی | اجتماعی | ورزشی | فرهنگی | پزشکی | علمی | فناوری | حوادث | بین الملل | استان ها | تصاویر

پست الکترونیکی:

همه خبرها

عضویت در خبرنامه

- 1 + 5
- یارانه ها
- مسکن مهر
- قیمت جهانی طلا
- قیمت روز طلا و ارز
- قیمت جهانی نفت
- اخبار نرخ ارز
- قیمت طلا
- قیمت سکه

موتور جستجوی قطره

بروزرسانی: 02:13 RSS

بزرگترین کارخانه طلای خاورمیانه افتتاح شد

ایمیدرو - برای اجرای این طرح، 170 میلیارد تومان سرمایه گذاری ارزی و ریالی شده و برای ...

تسلیت متفاوت علی ضیا /عکس

ملیت - علی ضیا با انتشار این تصویر در صفحه فیس بوکش یاد مرحوم مرتضی پاشایی را گرامی داشت.

صلاح دین، مکتب ما و صلاح مملکت ما این است که همه با هم یک کار را دنبال کنیم/دولت ...

بوسه خبرساز اوباما +عکس

اقتصاد ایران آنلاین - رابطه صمیمانه باراک اوباما با "آنگ سان سوچی" در کنفرانس خبری مشترک این ...

ایجاد شبکه علمی جزء ضرورت های امروز کشور است

شورای انقلاب فرهنگی - عضو کمیسیون آموزش و تحقیقات مجلس شورای اسلامی تأکید کرد: تبدیل علم به عمل ...

زنگنه: بازگشت به قیمت های گذشته نفت دشوار است

جماران - ، ولی باید قیمت را در حدی که وضعیت جدید

کد نمایش در سایت و وبلاگ

تبلیغات در قطره

اولین کنفرانس بین المللی

Illustration 5: The website of ghatreh.com today

The IP addresses of the domains in 2011, at a time when dropper, payload and C&Cs were still active, were 69.90.160.65, 70.38.107.13 and 70.38.12.10. All three IPs are located in Canada, two relate to Montréal, one to Oakville. The complete passive DNS history can be reviewed in appendix A.

The binary also contains the URL 'http://1.9.32.11/bunny/test.php?rec=nvista', but the purpose of it remains unclear. This IP address is never contacted by the malware.

Find the entire decrypted configuration below:

```
<COMMAND_KEY>wqNbo8DSbWZiq1QZ4NnmTcibYRJIFxmlSBQh1zdF8IlDKin2zkNOVtKvlx92Q0QOQnA
Nt1NCbDItwVQWY0HBQ2GKYPecpfuQ6JPZG0dQRarhCy7nTT3ukET9YzQEPN81</COMMAND_KEY>
<STORAGE_KEY>lbuLmdAouhUkSsMXckMUevjNf87S4ATWYwLYGLM44O1ortQEE2vr34QGzjPhrWeosFB
YQpT6dRwDuPPuP13zzqZhW3PeCQA7OUEghovMwEGYx1annboIwRxKbhUOIMsf</STORAGE_KEY>
<RESPONSE_KEY>8kMQOWondQc2ZgYgirg9NHg0QUiIIBdPe0YtIVfvxa9rvJ9wtFGx8ko4oFY34PrSkW
9cKzEFZYDnM54qeWcvSMIi9sBAkcD5ggFDmQOGdO42ef344I7s9wAKoAXKeq1s</RESPONSE_KEY>
<COMMAND_FILE_PATH_AND_PREFIX>%SELFDIR%\net.cap</COMMAND_FILE_PATH_AND_PREFIX>
<URL1>http://le-progres.net/images/php/test.php?rec=11206-01</URL1>
<URL2>http://ghatreh.com/skins/php/test.php?rec=11206-01</URL2>
<URL3>http://www.usthb-dz.org/includes/php/test.php?rec=11206-01</URL3>
<INJECTED>0</INJECTED>
<STARTDELAY>0</STARTDELAY>
<DIEDELAY>120</DIEDELAY>
```

The configuration provides three keys for encryption and decryption of data. The *command_key* is used to decrypt commands received from the C&C server, the *storage_key* is used to encrypt and decrypt data being stored to files on disk and the *response_key* will encrypt callbacks to the C&C server. All data exchanged with any of the C&C servers as well as data stored to a file on disk is encrypted with the according key.

Diedelay and *startdelay* are timeout settings, the exact use of the parameter *injected* is unclear up to the time of writing. *Command_file_path_and_prefix* defines the name of the file which will holds C&C commands and Lua scripts at runtime. The malware performs C&C command parsing and Lua script execution with dedicated threads, of which every thread creates its own net.cap file, extending the name by a fixed digit namely 0, 1, 3 or 4.

At runtime the malware adds additional configuration parameters, which are not assigned an obvious name like those retrieved from the resource section. The synonyms for all encrypted configuration values as they are stored in Windows registry are:

- ipsecFilterData
- ClassID
- ipsecID
- ipsecISAKMP
- ipsecData
- ipsecDSA
- ipsecSA
- ike
- ikeID
- isakmpID
- isakmpKey
- HMACSHA1
- isakmpData

After the configuration is retrieved from the resource section the malware will create a subkey for every parameter under [HKLM|HKCU]\Software\Microsoft\IPSec. This key has initially been created by the dropper. The attributes are encrypted before being written to registry, and will be decrypted on the fly every time the malware requests a specific parameter.

2.4 THE MULTI-THREADING MODEL

Suspect #4 comes with a solid multi-threading model, which seeks to assure fail-safe and high-performance execution. The malware runs a main thread, which manages four worker threads and performs C&C command parsing and Lua script execution. The worker threads are dedicated to receive commands and scripts through different ways. Next to that, the main thread also runs sub threads to maintain log files the malware creates at during execution and to keep track of the overall system load the malware creates.

The threads are coordinated via named events, global flag variables, in some cases also mutexes or semaphores are used. The main action of the malware is carried out in the main thread, which parses commands and executes Lua scripts, provided by the worker threads via command files.

The threading model is summarized in the following graphic.

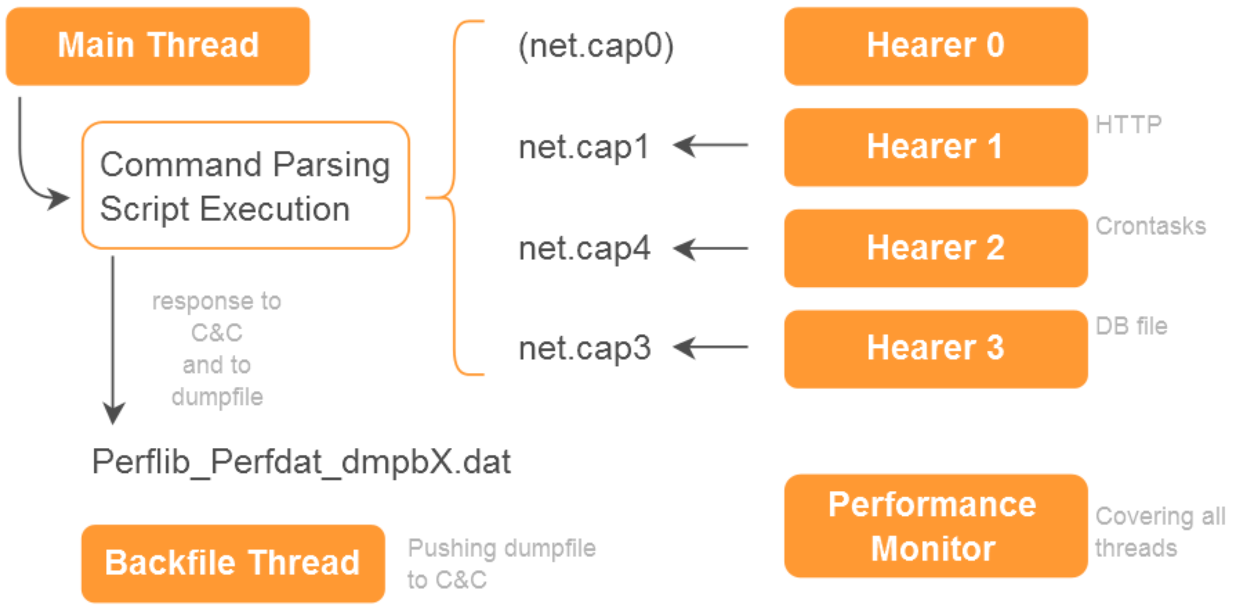


Illustration 6: Summary of the thread model

2.4.1 PERFORMANCE MONITORING

A dedicated thread is present for monitoring the CPU load the malware itself generates. Therefor the thread enumerates all threads from the current process and calculates their execution time.

The user- and kernel land execution time is measured for every thread using the kernel32! GetThreadTimes API; then aggregated, so the final value holds the overall CPU usage time of the malware process. The thread keeps track of the time passing between executions. If the malware's active execution time in percentage of this delta exceeds a configured threshold, the malware will suspend all threads of its process for at least 5s. The threshold is retrieved from configuration in Windows registry, and can be controlled by the botmaster via the C&C command setcpulimit (see section 2.5).

This clearly is an attempt to keeping a low profile on the infected machine. By maintaining its own execution load the bot's compromised svchost.exe process will not catch attention on the process list.

2.4.2 THE BACKFILE LAUNCHING THREAD

The backfile launching thread got its name from the named event 'backfilelaunching' which is set to indicate that the dedicated thread has started. Also in a 1s-frequency the malware runs the backfile thread to push dump files to one of its remote servers.

The files are named Perflib_Perfdat_dmpbX.dat, where X is an index number starting with 0. The data written to the file are status messages including a time stamp, which are returned by command and Lua script execution. The data is encrypted using the *response_key* from the configuration.

2.4.3 THE HEARER THREADS

The term 'hearer' can be found plenty of times in the binary, it is assumed that it is equivalent to 'listener' of some sort. All together suspect #4 manages 4 hearer threads including one command file per thread. The command file is named net.capX, where X is the index of the thread; 0, 1, 3 or 4.

The hearer thread's purpose is to receive instructions from the remote servers and provide them to the main thread. Such instructions are commands and/or one or more Lua scripts. Each hearer has a dedicated method to receive instructions which is either separately via HTTP from the server, aggregated through a downloaded data file or as tasks to be configured as scheduled tasks. The hearer threads dump the received instructions to their net.cap-files, from where the main thread's command parsing routine fetches and executes them. The data is encrypted with the *storage_key* before written to disk.

HEARER 0

Generic hearer thread. Started, but not involved in command fetching and parsing.

HEARER 1

Hearer 1 receives commands and Lua scripts directly via HTTP from one of the C&C servers. These are encrypted with the *storage_key* and stored to net.cap1 in the malware's working directory. Interestingly, the malware handles successful downloads with the HTTP status code 418. This code is not part of the original HTTP standard, but was introduced in RFC2324, 1998's traditional IETF April Fool's joke. The textual representation of status 418 is “I'm a teapot.”.

HEARER 2

Hearer 2 is responsible for managing the scheduled tasks, titled 'crontasks' by the malware. The tasks are stored in a file named perf.tmp, and when due for execution are copied to the according net.cap-file, net.cap2 in this case. From there the main thread can fetch and execute them in its command parsing loop. Hearer 2 also creates a swap file named perf.tmp2, which helps in selectively editing the initial task file and in managing the task list.

HEARER 3

Hearer 3 searches for a file named netdump.db in its local working directory. If found, the thread will read its content in chunks representing different commands and paste these to net.cap3. Again, the data is encrypted with the *storage_key*. No functionality could be found in the malware code to write to netdump.db, which leaves the conclusion that this file will be downloaded from a C&C server.

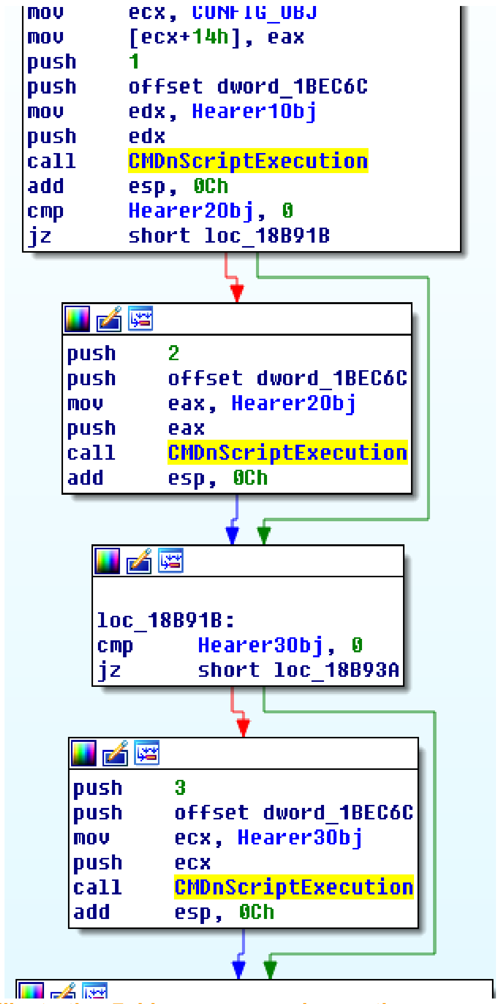
2.5 C&C COMMAND PARSING

After initializing and starting all four hearer threads the main thread enters a command parsing loop. For every hearer the main thread invokes the command and script execution function. Therefor data is read from the encrypted net.capX-file, where X indicates the index of the hearer; 0, 1, 3 or 4. Once decrypted, this data is parsed to find valid C&C commands and / or valid Lua scripts to execute. More details on command and script execution can be found in section 2.6.

Notable is that the command parsing and execution happens in a serial fashion, while Lua script execution is handed over to dedicated threads. Also, no direct relation between C&C commands and Lua script execution could be identified.

The most notable actions of the bot can be summarized as follows:

- Downloads and executes Lua scripts to instrument its own code
- Can install managed tasks (named 'crontask') for its integrated engine
- Can maintain FTP connections
- Can send and receive files via HTTP
- Writes runtime information to local files



```
mov     ecx, CONFIG_OBJ
mov     [ecx+14h], eax
push    1
push    offset dword_1BEC6C
mov     edx, Hearer1Obj
push    edx
call    CMDnScriptExecution
add     esp, 0Ch
cmp     Hearer2Obj, 0
jz      short loc_18B91B
```

```
push    2
push    offset dword_1BEC6C
mov     eax, Hearer2Obj
push    eax
call    CMDnScriptExecution
add     esp, 0Ch
```

```
loc_18B91B:
cmp     Hearer3Obj, 0
jz      short loc_18B93A
```

```
push    3
push    offset dword_1BEC6C
mov     ecx, Hearer3Obj
push    ecx
call    CMDnScriptExecution
add     esp, 0Ch
```

Illustration 7: Linear command execution

- Encryption for local data and network communication

However, suspect #4 does accept a list of hard coded commands to be sent by the C&C server. These mainly serve the purpose of controlling operation of the bot on the infected machine, such as restarting threads or setting configuration parameters. The full list of C&C commands can be found below:

mainfrequency – Set a new timeout value for the command parsing loop.

getconfig – Get configuration from registry and push it to C&C.

ftpput – Push a file to an FTP server.

ftpget – Download and store a file from an FTP server.

sendfile – Send a local file to one of the C&Cs.

getfile – Download and store a file from one of the C&Cs.

uninstall – Set runtime flags to instruct main thread to terminate and delete related files.

restarthearer – Restart one of the hearer threads, clean its runtime files.

restart – Set runtime flag to instruct main thread to restart all threads.

cleanhearer – Clear all files related to an indicated hearer thread.

timeout – Configure a timeout value for HTTP connections.

waitfor – Set a timeout value for hearer 1.

updatedietime – Update the *dietime* value in registry.

crontaska – Add new task to perf.tmp for hearer 2 to execute.

crontaskr – Remove an entry in perf.tmp.

crontaskl – List tasks from perf.tmp, write to perf.tmp2 and send to C&C.

maxpostdata – Set a limit for data size to be sent to C&C via registry to limit data traffic.

seturl – Add an additional C&C server URL to configuration in registry.

stop – Set runtime flag to instruct main thread to shut down.

setcpulimit – Configure a limit in percent for CPU usage, stored in registry.

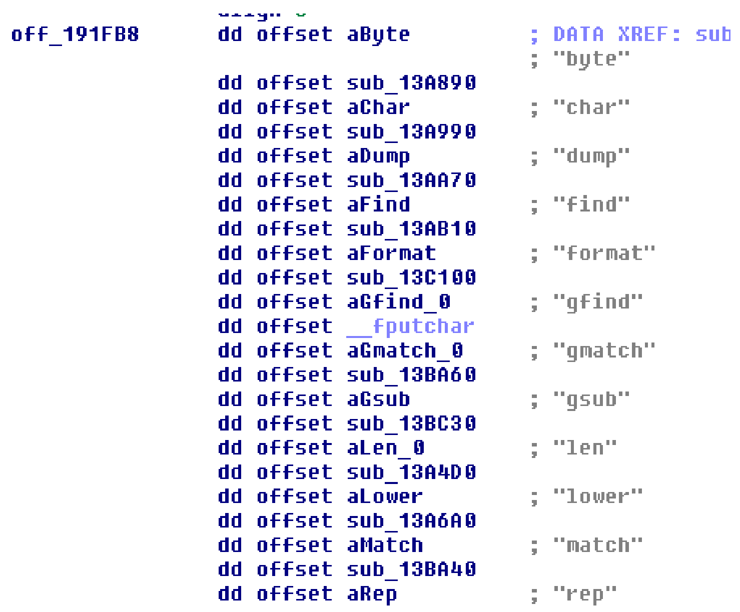
Every command execution function returns a status line, such as for example "ftp command succeeded for: %s!\n". The initial command, the status line and a current time stamp are encrypted and sent back to the C&C server. Additionally, the same data is written to the Perflib_Perfdat_dmpbX.dat file which is regularly pushed to the C&C by the backfile thread.

For hearer 1 and 3 the command parsing routine will additionally manage two continuous log files which are named Perflib_Perfdata41X.dat and Perflib_Perfdata_42X.dat, where X is the respective hearer thread's index. More specifically, command parsing will increment a global variable holding the total amount of executed commands and scripts. This 4-byte value is appended to Perflib_Perfdata41X.dat before and Perlfib_Perfdata42X.dat after command execution. These files survive restart or cleanup, so every time the malware boots it reads the current status from these logs and continues from the latest value. This mechanism helps the botmaster to keep track of the bots activity on an infected host and could also be helpful in trouble shooting.

2.6 LUA ENGINE INTEGRATION

Suspect #4 incorporates an interpreter for Lua 5.1, LuaSocket 2.0.2 and C/Invoke Lua bindings. Lua is a lightweight programming language designed as a scripting language which can be embedded into applications, providing a C API for doing so. C bindings are provided through C/Invoke and enable Lua scripts to perform callbacks to C/C++ code. This constellation can be found in many video game engines to provide polymorphic behavior in games. Engine and game play features are injected through Lua scripts, which instrument the game engine code.

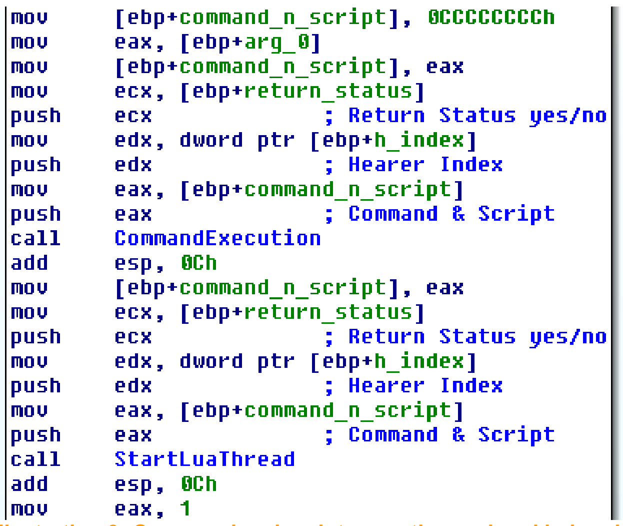
The Lua interpreter is very small, compiled roughly 180kB, thus can easily be integrated in an application. The C/Invoke bindings enable Lua to be completely independent from the C/C++ application, so injected scripts can be pure Lua code.



```
off_191FB8      dd offset aByte         ; DATA XREF: sub
                                        ; "byte"
                dd offset sub_13A890
                dd offset aChar         ; "char"
                dd offset sub_13A990
                dd offset aDump         ; "dump"
                dd offset sub_13AA70
                dd offset aFind         ; "find"
                dd offset sub_13AB10
                dd offset aFormat       ; "format"
                dd offset sub_13C100
                dd offset aGfind_0      ; "gfind"
                dd offset __fputchar
                dd offset aGmatch_0     ; "gmatch"
                dd offset sub_13BA60
                dd offset aGsub         ; "gsub"
                dd offset sub_13BC30
                dd offset aLen_0        ; "len"
                dd offset sub_13A4D0
                dd offset aLower        ; "lower"
                dd offset sub_13A6A0
                dd offset aMatch        ; "match"
                dd offset sub_13BA40
                dd offset aRep          ; "rep"
```

Illustration 8: A small portion of Lua callback functions

A script, if delivered to one of the hearer threads, will be provided through the corresponding net.cap-file of the hearer along with a C&C command. If a script is included in the data read from the net.cap-file, this is indicated by the start sentinel '<$'. Likewise, the end of a script is indicated by '$>'.



```
mov     [ebp+command_n_script], 0CCCCCCCCh
mov     eax, [ebp+arg_0]
mov     [ebp+command_n_script], eax
mov     ecx, [ebp+return_status]
push    ecx             ; Return Status yes/no
mov     edx, dword ptr [ebp+h_index]
push    edx             ; Hearer Index
mov     eax, [ebp+command_n_script]
push    eax             ; Command & Script
call    CommandExecution
add     esp, 0Ch
mov     [ebp+command_n_script], eax
mov     ecx, [ebp+return_status]
push    ecx             ; Return Status yes/no
mov     edx, dword ptr [ebp+h_index]
push    edx             ; Hearer Index
mov     eax, [ebp+command_n_script]
push    eax             ; Command & Script
call    StartLuaThread
add     esp, 0Ch
mov     eax, 1
```

Illustration 9: Command and script execution go hand in hand

Command execution goes hand in hand with Lua script execution, thus is dependent of the four hearer threads. The work flow for each hearer per execution loop in the main thread is as follows:

- Read and decrypt content from according net.cap-file
- See if an entry of the C&C command list can be found in the decrypted data

- If so, execute the command
- Hand the data over to the Lua interpreter thread
- See if a Lua script can be located, searching for start and end sentinel
- If so, start Lua thread and execute script
- If more than one script can be found start more Lua threads recursively

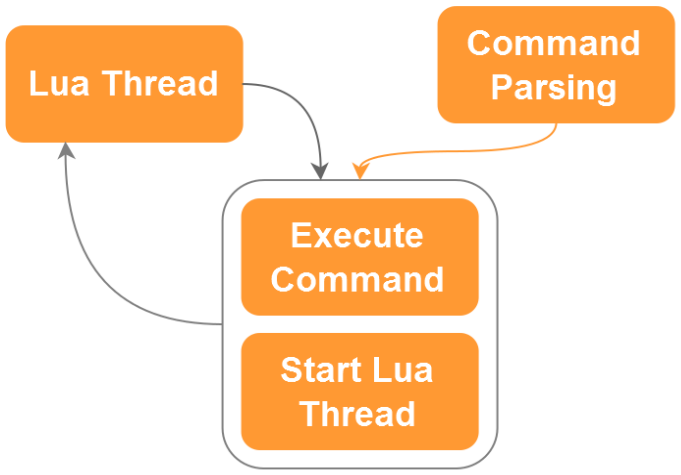


Illustration 10: Command and script execution in detail

After starting the Lua thread requests a value from Windows registry named 'EnableLua' under SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System. This key name stands for enable limited user account. When set to zero this key effectively disables UAC (User Account Control). The malware only requests the key value but does not change it. The value is later used by a callback function named 'popenWin' to decide on a way how a sub process should be started. Options are via WMIC using wmic.exe, Microsoft CScript engine using cscript.exe or directly via spawning a command line through cmd.exe.

However, spawning sub processes is not the only feature which comes with Lua. The Lua interpreter is a powerful code base which enables suspect #4 to change functionality on the fly, as different scripts are downloaded and executed. The scripts define the functionality as they perform callbacks to the C/C++ code in the malware binary. The potential of the Lua interpreter can be investigated on the project homepage [4].

Like command execution, Lua script execution also returns a status message which is extended by a time stamp and forwarded to one of the C&C servers, as well as written to the backfile log in Perflib_Perfdat_dmpbX.dat. Likewise, the Lua threads log their activity to Perflib_Perfdata_41X.dat/42X.dat.

Due to the absence of C&C servers which hold the Lua scripts for the bot's Lua interpreter the true nature of this feature remains unclear.

3. EVIL BUNNY IN THE SPOTLIGHT

Throughout analysis and due to the help of Paul Rascagniéres a number of binaries which relate to EvilBunny were identified [7]:

```
2a64d331964dbdec8141f16585f392ba
40E0F0681C79D70AC0329E68A94294CB
8132ee00f64856cf10930fd72505cebe
e8a333a726481a72b267ec6109939b0d
3bbb59afdf9bda4ffdc644d9d51c53e7
c40e3ee23cf95d992b7cd0b7c01b8599
330dc1a7f3930a2234e505ba11da0eea
83b7c532663f11bf994a1b518880557d
b8ac16701c3c15b103e61b5a317692bc
bbf4b1961ff0ce19db748616754da76e
```

The binaries share parts of source code, most notably is the identical API name obfuscation algorithm and respective key AB34CD77h.

The indicators of a relation among all samples or a subset of samples are the following:

- Same API name obfuscation algorithm and key
- Equal infection routines / source code
- Equal AntiVirus product enumeration technique
- Shared C&C domain names and equal C&C commands
- Equal proxy bypass technique / source code

Mentioned set of malware also shows parallels in coding style and structure. However, it has to be pointed out that suspect #4 without doubt sticks out in terms of complexity and functionality. It is assumed that suspect #4 was created by the same authors, but this cannot be proved.

Four of the set of identified samples are identical in functionality and capability. They possess respectable flooding capabilities and also provide remote access. However, they do not support data stealing or any spying functionality [8]. Two more C&C domains were identified for this group:

http://callientefever.info/img/new/n.php
http://fullapple.net/pictures/bkp/n.php

Regarding the nature of the attack or the actor behind EvilBunny only vague conclusions can be drawn. There has been public contemplation that the actor might be french and that the malware might be related to samples analyzed by the Canadian CSEC [9]. However, there hasn't been any hard evidence which links the attack to a group or a geographical location. Attribution is a tricky task and assumptions must not be drawn easily.

4. ACKNOWLEDGEMENTS

For providing technical insights, support and if I remember right even water, coffee or occationally rum, I want to express my deepest gratitude to Morgan Marquis-Boire, Sebastien Larinier, Paul Rascagniéres, Nicolas Brulez, Michael Shalyt and Inbar Raz.

Further I want to thank Fred Arbogast, Alexandre Dulaunoy, Raphael Vinot and the remaining organizing team of Hack.lu for their advice and effort, even in situations of widespread insomnia.

All of you – such wow, very awesome; rock on!

5. RESOURCES

[1] An Analysis Of CVE-2011-4369
http://blog.9bplus.com/analyzing-cve-2011-4369-part-one/

[2] ThreatExpert Report on c40e3ee23cf95d992b7cd0b7c01b8599
http://www.threatexpert.com/report.aspx?md5=c40e3ee23cf95d992b7cd0b7c01b8599

[3] How to disable UAC
https://social.technet.microsoft.com/Forums/windowsserver/en-US/0aeac9d8-3591-4294-b13e-825705b27730/how-to-disable-uac

[4] The Lua Project
http://www.lua.org/

[5] The C/Invoke Project
http://www.nongnu.org/cinvoke/

[6] Lua scriptable game engines (Wikipedia)
http://en.wikipedia.org/wiki/Category:Lua-scriptable_game_engines

[7] All file hashes related to the EvilBunny malware
http://pastebin.com/7iedeFF5

[8] Hack.lu 2014 Keynote on TS/NOFORN
http://www.slideshare.net/pinkflawd/tsnoforn

[9] Quand les Canadiens partent en chasse de « Babar »
http://www.lemonde.fr/international/article/2014/03/21/quand-les-canadiens-partent-en-chasse-de-babar_4387233_3210.html

6. APPENDICES

APPENDIX A – PASSIVE DNS RECORDS

Marked in Grey is the data of sinkholes for the mentioned domains, currently operated by Kaspersky.

Domain usthb-dz.org

Resolve	Location	Network	First	Last
95.211.172.143	NL	95.211.0.0/16	14.12.2013 00:00	30.12.2013 00:00
184.168.221.41	US	184.168.220.0/22	20.09.2012 00:00	14.12.2013 00:00
8.5.1.34	US	8.5.1.0/24	10.04.2011 00:00	20.09.2012 00:00
70.38.12.10	CA	70.38.0.0/17	10.03.2010 00:00	10.04.2011 00:00
216.108.239.153	US	216.108.239.0/24	01.06.2009 00:00	10.03.2010 00:00
213.186.33.19	FR	213.186.32.0/19	20.04.2009 00:00	01.06.2009 00:00
91.121.142.185	FR	91.121.0.0/16	12.01.2009 00:00	20.04.2009 00:00
91.121.137.201	FR	91.121.0.0/16	19.09.2007 00:00	12.01.2009 00:00
64.15.136.137	CA	64.15.128.0/19	19.09.2007 00:00	19.09.2007 00:00

Domain le-progres.net

Resolve	Location	Network	First	Last
95.211.172.143	NL	95.211.0.0/16	20.06.2014 00:00	09.11.2014 09:24
208.73.210.155	US	208.73.208.0/21	10.04.2011 00:00	27.07.2014 00:21
69.90.160.65	CA	69.90.160.0/22	02.11.2009 00:00	17.06.2014 07:16
74.54.82.222	US	74.52.0.0/14	13.04.2009 00:00	02.11.2009 00:00
74.54.82.228	US	74.52.0.0/14	16.03.2009 00:00	13.04.2009 00:00
208.87.149.250	US	208.87.148.0/22	08.12.2008 00:00	16.03.2009 00:00
216.36.248.134	US	216.36.192.0/18	02.11.2008 00:00	08.12.2008 00:00
216.36.248.128	US	216.36.192.0/18	25.09.2008 00:00	02.11.2008 00:00
216.36.248.134	US	216.36.192.0/18	06.12.2007 00:00	25.09.2008 00:00
209.62.20.175	US	209.62.0.0/17	04.11.2007 00:00	06.12.2007 00:00
69.46.226.168	N/A	69.46.224.0/20	01.11.2007 00:00	04.11.2007 00:00

Domain ghatreh.com

Resolve	Location	Network	First	Last
184.107.60.97	CA	184.107.0.0/16	07.09.2012 00:00	09.11.2014 09:45
70.38.107.13	CA	70.38.0.0/17	04.09.2012 00:00	01.04.2013 00:00
204.93.167.100	US	204.93.167.0/24	07.04.2012 00:00	04.09.2012 00:00
70.38.107.13	CA	70.38.0.0/17	14.03.2012 00:00	07.04.2012 00:00
70.38.107.12	CA	70.38.0.0/17	10.04.2011 00:00	14.03.2012 00:00
70.38.107.13	CA	70.38.0.0/17	23.03.2008 00:00	10.04.2011 00:00
67.19.84.46	US	67.18.0.0/15	16.10.2006 00:00	23.03.2008 00:00
67.18.209.222	US	67.18.0.0/15	07.10.2006 00:00	16.10.2006 00:00
204.13.160.25	US	204.13.160.0/22	30.09.2006 00:00	07.10.2006 00:00
64.20.43.107	US	64.20.32.0/19	23.09.2006 00:00	30.09.2006 00:00
204.13.160.25	US	204.13.160.0/22	01.07.2006 00:00	23.09.2006 00:00
69.25.212.153	US	69.25.208.0/20	29.04.2006 00:00	01.07.2006 00:00
66.45.225.11	US	66.45.224.0/19	14.02.2005 00:00	29.04.2006 00:00
67.19.22.234	US	67.18.0.0/15	12.02.2005 00:00	14.02.2005 00:00
204.157.11.208	US	204.157.0.0/16	22.11.2004 00:00	12.02.2005 00:00

Domain callientefever.info

Resolve	Location	Network	First	Last
95.211.172.143	NL	95.211.0.0/16	25.10.2010 00:00	07.10.2014 06:26
68.178.232.99	US	68.178.232.0/22	24.10.2009 00:00	25.10.2010 00:00

Domain fullapple.net

Resolve	Location	Network	First	Last
95.211.172.143	NL	95.211.0.0/16	09.12.2010 00:00	09.04.2014 10:48
68.178.232.99	US	68.178.232.0/22	23.01.2010 00:00	09.12.2010 00:00
209.51.136.27	US	209.51.128.0/19	13.12.2009 00:00	23.01.2010 00:00
72.9.244.162	US	72.9.240.0/20	02.12.2009 00:00	13.12.2009 00:00

APPENDIX B – LIST OF RUNTIME FILES SUSPECT #4

net.capX (where X is the index of the hearer thread; 0, 1, 3 or 4)

perf.tmp

perf.tmp2

netdump.db

Perflib_Perfdat_dmpbX.dat (where X is an incremental index)

PerfLib_Perfdata_41X.dat (where X is the index of the hearer; 1 or 3)

PerfLib_Perfdata_42X.dat (where X is the index of the hearer; 1 or 3)