IT'S A FILE INFECTOR... IT'S RANSOMWARE... IT'S VIRLOCK

Vlad Craciun, Andrei Nacu & Mihail Andronic Bitdefender, Romania

Email {vcraciun, anacu, mandronic}@ bitdefender.com

ABSTRACT

Win32.Virlock, with all its variations, is both a new kind of file infector and a piece of ransomware (screen-locker) at the same time. In this paper, we aim to cover the techniques used by this virus and discuss methods that can be used to detect and disinfect systems affected by it.

Virlock uses several techniques, including code obfuscation, staged unpacking, random API calls and large/redundant areas of decrypted code, to make it difficult to analyse. It also protects its code by decrypting only the sequences that are going to be executed. After a sequence of code is executed, Virlock encrypts it again. By staggering the decryption/ encryption process, it ensures that a memory dump at a certain point will not reveal its features but only the piece of code that is being executed at that time.

There is also a moment in its first execution when it shifts its shape by changing certain instructions and encryption keys so that new generations will look different. Each new infection is different from any other, mostly because of the timestamps that play an important role in computing the encryption keys. Having these protection methods will also make any clean-up attempt quite a challenge. The disinfection process for this virus involves searching inside malware code for specific instruction arrangements.

We will present some ideas that could help in detecting and disinfecting a Virlock-infected system.

INTRODUCTION

Malware has grown significantly in the last decade, both in prevalence and complexity. It has developed from innocent bad jokes and simple trojans to advanced polymorphic file infectors, rootkits and ransomware. While security companies have studied all the types of malware and built specific categories for them, it can be difficult, today, to categorize a malicious application as a trojan, a piece of spyware, or even a file infector, as they tend to be more complex and to embed several different kinds of behaviour at once.

Security vendors have been forced to develop different kinds of engines to reach faster conclusions in malware analysis, be it static or dynamic, but security products by definition are usually a step behind the malware creators, even if we try to minimize that time-interval. The security industry had tried to figure out better solutions and better engines to prevent malware execution in advance by using artificial intelligence, but no matter how hard we try, or how much time we invest in research, there is always something new which doesn't get caught. There are many cases in which we reach the conclusion that an engine is not doing the best to protect against a new piece of malware, or that making a small improvement will slow down the entire product. In some cases we reach the conclusion that a particular detection method is simply not adequate for a specific piece of malware.



Figure 1: Example of a common file infector (appended code to clean application).

1. RANSOMWARE AND FILE INFECTOR EVOLUTION

1.1 Old file infectors, behaviour and purpose

Known categories: appenders, prependers, EPO, polymorphic, interleaved.

Purpose:

The first file infectors were just bad jokes or proofs of concept. The earlier ones interleaved malicious code with original application code or prepended malware code to a clean application. By prepending the malicous code to a clean application, the authors increased the time needed for analysis, and also gained time for their malware to spread while users were searching for solutions. This is also a safe way to expose users' computers to hackers; file infectors act like agents, collecting confidential user data, or continuously delivering other kinds of malware to the infected system.

Behaviour:

Malicious code is executed first, infecting the system or ensuring it is running within another process or thread and eventually deploying any missing files, then it executes the original application. When a portion of the clean application is executed, the malware will also be executed at some point, this being triggered by a patched API import or by malicious code insertion. After the malicious code has finished running, the clean application's code continues to be executed from where it was left off.

1.2 Old screen-lockers: behaviour and purpose

Purpose:

An easy way to get money from users by blocking access to their working environment. (Childish play for grownups!!!)

Behaviour:

This kind of malware creates an additional desktop and switches to the new environment, just as if another user had logged on. Some of them may encrypt user files, but most of them don't. The ones that do encrypt user files, like some CryptoLockers, do not lock the user's screen, because the damage is already at a stage where the user might wonder where the backup is, or whether a decryption tool is worth paying for.

Let us mention some of the well known pieces of ransomware among both families:

• ACCDFISA, PornoBlocker, Rannoh, IcePol, CryptoWall, CoinVault.

In the following chapters we will uncover the main features and components of Virlock; however we are not going to focus on the infection process. This type of malware has the vaccine within itself, but only applies it for each infected file at runtime. We will focus mainly on its design and its abilities to sneak past some security solutions.

2. ANALYSING VIRLOCK, REFINING BEHAVIOUR, COMBINING PURPOSE

Virlock combines the technology of file infection with the



Figure 2: Ransomware blocking user screen and requesting payment.



Figure 3: RSA1024 CryptoLocker displaying message to user.

screen-locking features of regular screen-lockers. The authors embed both infection and disinfection tools, throwing away the management system to bind infected users to some private decryption keys. Their remaining concern is about users who are willing to pay their fee rated in bitcoins.

The screen-locking picture is very similar to that of those pieces of ransomware that pretend to be some higher authority with full rights to request certain amounts of money from home-users – for example as fines (see Figure 4). Most texts appearing on the locked screen are trying to scare the users, for example threatening them with prison for up to five years or more if they do not pay the money.

2.1 Analysing Virlock – refining behaviour

Virlock is changing the way in which the infection process takes place:

- It has an ingenious polymorphic engine (most file infectors don't come with such an engine), making the detection process more difficult with each infected system.
- It doesn't just insert a piece of code into the clean application as most file infectors do, but the entire clean

This comput	This computer contains pirated software and has been blocked by ICE-Homeland Security Investigations.											
					HARCENTY BATTONY STANS TOTAL							
federal pr	ison, a \$250,000 fin	e, forfeiture and r	estitution (17 L	.S.C s.506, 18 U	.S.C s.2319)							
As a first-tir If the fine is which will b How to pay J.You can p Click the tal 2.(Offline O A special res To regain ac 1NdR9tEKR After the pa Amount:	ne offender you are requ s not paid within three d: e forwarded to your loc: a fine? There are two wa ay the fine online throug so below to find the near ption) You can come to storation software will b- ccess now transfer BitCol BoQ10iyAPhpuks9Uct6X yment is finalized enter Transfer ID:	tired by law to pay a fi ays, a warrant will be i al authoritics. You will ys to pay a fine: gh BitCoin. BitCoin is a set vendor. Your com your local courthouse e sent to you by mail v mis to the following ad ftEdW Transfer ID below.	ne of 500 USD ssued for your arr be charged, fined wailable nationwi- puter will be unlor and pay the fine a vithin a week after dress (click to cop	est, , convicted for up to le. ked after the payme t the 'Cashiers' wind the payment is mad ();	> 5 years. ent is made. low. le.							
Amount:	I ransfer ID:					PAY FINF						
DIGENT/73 I PREFINE Note: All files on this computer have been encrypted with a strong symmetric algorithm and a 4096-bit key. Files will be inaccessible until the fine is paid. View encrypted files Attempt to remove this message will result in irreversible damage to your files, hardware and Windows installation. View encrypted files Payment Bit/Coin Information Bit/Coin Exchanges Bit/Coin ATMs Internet Browser												
<u>i ayıncın</u>	DROOM INCOMBLICIT	Diroom Exchanges	DROOM PATINS	Internet Drowser	THORONAU							
Project Global 3 is a	a coordinated effort by U.S., Car argeting computers with pirated	nadian, European, Australian, I content and their operators.	New Zealand and other	aw enforcement agencies								

Figure 4: Virlock screen lock.

application becomes a small piece of the malware itself (similar to Morto/Sality/ACCDFISA).

- It uses techniques to cheat users at first glance (seen in a few other pieces of malware), to bypass users' doubts that an infected file is really malicious.
- It has a lot of features (not new, but different) that make the reverse-engineering process more difficult, overload the analysts and annoy them.
- It has screen-locking (borrowed from screen-lockers) to increase the time taken to get to an infected sample – most home-users prefer to reinstall their operating system rather than trying to remove the malware.
- It uses multi-threading and rooting into the environment to get full control over the infected systems without the need for drivers, and to execute different paths inside the same application, but from different points of view (running processes/services/threads).

2.1.1 Not embedding malware code, but embedding a clean file

The infection process is somewhat different from the infection process of other known file infectors. However, there are small similarities between Virlock and both the Sality file infector and the ACCDFISA ransomware:

- Virlock and Sality: both replace the clean application with the malware which contains the original application packed or modified.
- Virlock and ACCDFISA: ACCDFISA uses the RAR archiver to make all the infections self-extractable – this is very similar to Virlock's behaviour but with the small difference that Virlock uses its own techniques to accomplish the same behaviour.

2.1.2 Anti-analysing techniques

At the moment we know about five different Virlock

versions. They're not too different but they do differ in such a way that some simple checks will not catch them all.

2.1.2.1 Code obfuscation

One of the main techniques used to harden the reverse engineering and analysis process is obfuscation. Obfuscation is present in all five versions and is similar between some and different between others. However, while obfuscation may contribute to detection, it is not a key-point in doing that.

Figure 5 shows some screenshots of obfuscated code from four different versions.

If we are going to trace the entropy of those pieces of code, or count the number of some target instructions which repeat excessively, we can create some checkpoint conditions that Virlock infections will not pass. Code can be obfuscated in lots of configurations, but some of them are built based on some basic principles. It is not too difficult to observe the criteria with which an obfuscation engine was built.

We could also de-obfuscate some instruction blocks by following the true aim of an obfuscated piece of code. However, de-obfuscation becomes irrelevant when one can look at the execution traces. They are still a plus when building documents to reveal the true meaning of some code.

Obfuscation also contributes to making the static analysis procedure more difficult.

2.1.2.2 Anti-debugger

There are lots of anti-debugger techniques, and usually, malware creators combine those features with techniques to detect virtual machines, emulators or supervisor tools like *PIN* from *Intel* (which allows one to instrument an executed application), or API loggers which inject tracing modules or pieces of code into a target process.

Virlock does not combine all of these, but it uses the strongest of them all, in order to bring the analyst to a point where he/ she could easily give up.

mov	edx, Øx4ecf	add esi.ebx	push Øx4a087fac	ney ecx
sub	eax, 0x6841a	add edx.ecx	jmp (8) loc_4F5DEB	xor ecx, edi
add	edx 0x69f18		call ebx	xchg ecx, ebx
sub	eav ØydedØ5	ala adi Ava	imm (9) loc 4F6250	xor edi, esi
add	edy Øyh26e8		non eby	jmp (2) loc_46DAB5
add	0202000 X X X X X X X X X X X X X X X X	mov esi, eui	imp (0) los AFFFF7	mov [esp+0x4c], edx
auu	- J., 0.,27004	and eax, ebx	$\frac{1}{2}$	xor ebx. ecx
ՏԱՌ		add esi, ecx		sub esi.ecx
ααα	eax, 0xy7b23	mov edi, edx	Jmp (B) 10C_4F5743	mou esi edi
sub	edx, Uxc9eeU	add ebx, edx	push 0x49ed2460	add add 00000000
sub	eax, Øx6fcd7	and edx, eax	jmp (C) Loc_4F6174	aud eur, expretaree
add	edx, Øx43ab1	sub eax. 0x83aad42f	pop ebx	sub ebx, eui
sub	eax, 0x9404f	xcho ecx. eax	jmp (D) loc_4F5AEB	xcng ecx, esi
sub	edx Øx9e42c	ycho edy eav	call (E) sub_4F59D2	Jub (3) TOC_40D013
sub	eax Øxe917e	yon eay eby	.imp (F) loc_4F5776	sub ed1, 0x37f07414
add	edv Øva913d	volv odv ooi	push 0x4a0ab01c	xor ecx, ebx
aub	any Avgaca6	xung eux, esi	imm (G) loc 4F5992	mov esi, ebx
sub	eax, exictae	xor eax, ecx	non ehv	sub edi, esi
Տահ		mov ebx, edi	imp (H) loc 4F6005	sub edi, ecx
Տար	eax, exercar	mov eax, edi	call aby	mov edi, 0x37f15f8c
aaa	eax, UX178Db	or ecx, eax	inn (I) loo (FER29	bswap ecx
add	eax, 0x5467d	xor edi, esi	Jub (1) 100_413120	add ecx.esi
sub	edx, Øxa84a6	xor edx, esi	pop enx	imm (4) loc 46F2C4
sub	eax, Øxff9d5eb8	sub edx, edi	JMD (J) LOC_4F5B17	
sub	edx, 0x6fbe8516	xor edi.esi	.0C_4F55FH=	inc edi
mov	[eax], edx	or edx. ebx	ret	on ediecy
mov	ebx_ Øxf69a	mou esi ehv	Jmp (R) Loc_4F5614	not odi
mov	eax ØxcalcØ	sub edv Øv&f9cfa2f	call ebx	an can add
add	ehx Øx4d248	mou aby aay	jmp (L) loc_4F5845	or ecx, eui
add	eax Øxa8ac7	mou esi eby	xor al, [ebx]	auu eur, eux
sub	ehy Øydb48f	an adi aay	jmp (M) loc_4F5D43	auu eux, ecx
sub	eav Øvibbii	or eur, ecx	nop	JMP (5/ 100_461443
Տար		or eax, eax	imm (N) loc 4F55F0	mov edx, fs:[edx]

Figure 5: Obfuscated code inside four different Virlock versions.

Multi-staged unpack

This is a known technique for making the reverse engineering procedures harder, for both static and dynamic analysis. If a piece of code is unpacked piece by piece, one at a time, while it is executed, then performing a static analysis could be very difficult. Following the modifications inside a debugger might also be tricky, as some debuggers simply refuse to disassemble the code at the point where they think that there is no code in the first place. If we add to that the fact that code might re-encrypt the previously executed code, then things get really interesting.

Staged unpack

Staged unpack is a feature which minimizes the 'area' of 'plain-text' code at any time. There is a piece of code, more like a template, which repeats itself along the execution of the malware, and at each step:

- It hashes the buffer to be unpacked
- It decrypts the next piece of code, only if the hashes match
- It executes the code inside the decrypted chunk (possible more function-templates)
- It rehashes the unpacked code and alters the hash, inside the code

• It re-encrypts the previously decrypted code.

The template follows the data structure of a linear linked list, where each node is itself a linear linked list of many possible function calls. We are seeing linked lists inside linked lists mainly because each function call inside such a code-chunk calls another unpack-execute-repack template.

Figure 7 shows the code template for the mentioned trick inside a particular infection, which starts by checking the integrity of the packed chunk-code at 40193F, decrypts the buffer at 4019C0, jumps to unpacked code at 401A7E, and finally rebuilds the HASH for the unpacked code which it overwrites at the beginning of the code template and re-encrypts the entire code starting at 401A7E.

If someone is trying to make some process-dumps to have a look at the code inside the malware while it's executing, they might be surprised to find that the malware is almost fully packed, just as it was in the first place. The surprise gets bigger, as one is thinking that the malware might have some running threads which did not get dumped at the time of the process dump and while trying to grab all the memory pieces, one will obtain nothing more than the first process dump.



Figure 6: Short example of execution flow, following the chunk encryption/decryption template.



Figure 7: Template-code for staged unpack (yellow square -> unpacked code).

004959F8 004959FD 004959FF 00495A02 00495A04 00495A04 00495A0F 00495A0F	BA 905531F8 3106 83C6 04 EB B4 81F2 A84C90F7 BB CDAD76FD 81F3 2A786AFD 81F3 BFC82DF9	ÉU1° 1≢ â⊧∳ δ- ü≥ċLÉ≈ π=åν² ü≤*x.j² ü≤*x.j²	movedx, Øxf8315590xor[esi], eaxaddesi, Øx4jmp(1)loc_495988xoredx, Øxf7904ca8movebx, Øxfd76adcdxorebx, Øxfd6a782ain all bit of the polycletic
00495A1B 00495A21 00495A24 00495A24 00495A26 00495A28	64 A1 30000000 8A40 02 3C 01 75 05 E8 2CBEF6FF	dí0 è0 8 <© u\$ ፬,∃÷	mov eax, fs:[0x30] mov al, [eax+0x2] cmp al, 0x1 jnz (S) loc_49512D call (6) sub_401859
00495A2D 00495A2D 00495A37	C705 EB584900 0F31	\$6XI ≉1	mou dword [Øx4958eb], Øxc10bcc rdtsc

Figure 8: Anti-debugger checking inside PEB.



Figure 9: Code executed when debugger is found.

Checking for the presence of a debugger

Every infected sample checks for the presence of a debugger at some point. There is a standard way to do that, which is by querying a flag inside PEB, called isDebuggerPresent at [fs:[30h]+2], bit 0 (see Figure 8).

In our example, if it's being debugged, the code jumps to 0x495A2D. If we are taking a closer look we can see in

Figure 9 that the code is being executed in those conditions. Eventually we find a piece of code looping on itself and calling Sleep.

Most of the time, we can trick the application by changing the condition flags; and thus the condition itself or the value being compared. However, the time spent getting one's hands on that piece of code is sometimes too much to continue with the dynamic analysis that way.

Rooting inside the execution environment

We mentioned earlier that the malware does not use all known methods to harden the analysis procedure, but it uses the strongest of all methods gathered together to at least discourage analysts or to create problems for automated tools.

The technique described in this section does not refer to a behaviour that rootkits are using, but rather to a behaviour which spreads the infection inside the infected system, making self-copies and additional processes or services, each of them with a couple of threads. If the malware gets to execute inside such a configuration, then the synchronization policies between processes and threads will enable it to do its main job, otherwise one will not get anything useful from it.

At the beginning of the execution, an infected sample will first create two copies (of the original infection core – morphed) inside hidden folders with random names but constant length (eight characters), one located in %AllUsersProfile% and one inside %UserProfile%:

 $[\% UserProfile\% [a-zA-Z] \{8\} [a-zA-Z] \{8\}.exe]$

[%AllUsersProfile%\[a-zA-Z]{8}\[a-zA-Z]{8}.exe]

The copy located in the %UserProfile% folder is executed first using CreateProcess and it is also set as a starting point inside the startup key:

[HKCU\Software\Microsoft\Windows\CurrentVersion\Run].

Second and (in some cases) third copies are written in the %AllUserProfile% folder inside different subfolders. One of them is executed like the first copy in order to work together with it (one of the copies ensures that the other is not killed, and if that happens then it just recreates it), and the other is created as a service to supervise some tasks and gain privileged access to operating system components.

It is important at that point to note that the malware copies are not only different from the first one (using a polymorphic packer), but also have some key-flags changed. The changing of flags will enable, for example, one of the copies to execute a slightly different path inside the malware just like a switchcase block. For example, the malware self-disinfects the file inside it, only if a certain flag located at a hard-coded address says that this can be done.

A series of batch-files and VBS scripts are written on the disk temporarily to help the malware infect files by first making a backup and then overwriting the target file. Scripts are also used to change security policies inside the registry, in order to hide the malware or to disable default security features.

The following is a list of commands altering registry entries:

reg add HKCU\Software\Microsoft\Windows\ CurrentVersion\Explorer\Advanced /f /v HideFileExt /t REG_DWORD /d 1

reg add HKCU\Software\Microsoft\Windows\ CurrentVersion\Explorer\Advanced /f /v Hidden /t REG_ DWORD /d 2

reg add HKLM\SOFTWARE\Microsoft\Windows\ CurrentVersion\Policies\System /v EnableLUA /d 0 /t REG_DWORD /f

Straight after the installation, the malware tries to brute-force the user logon account password with at least a few thousand common password templates, and straight after that creates a new user with a random name and full administrator rights.



Figure 10: New account created by Virlock after successfully brute-forcing the administrator password.

The following are just a few examples of passwords that had been tried by the malware:

password, P@ssw0rd, 1234, Password1, 123456, admin, 12345, Passw0rd, p@ssw0rd, Pa\$\$w0rd, !QAZ2wsx, test, sunshine, P@ssw0rd, 1qaz@WSX, 123456789, 12345678, abc123, qwerty, letmein, changeme, master, Password!, passw0rd, 1q2w3e4r, Password01, password1, hunter, qazwsx, welcome, Welcome123, secret, orig_Administrator, princess, dragon, pussy, baseball, football, monkey, 696969, operator123, N0th1n9, !qaz@wsx, 1q2w3e4r5t6y7u8i, abcd12345, 7654321, Administrator, q1w2e3r4, q1w2e3r4t5.

A process created with the following command line will discard any possible API-tracer or debugger following the process execution. However, we can still trick such behaviours by altering the code at the entry-point and forcing a debugger to enter first, modifying the parameters for CreateProcess, or using some advanced environment emulators:

```
CreateProcessW("%TEMP%\AccMwMEs.bat", " "%TEMP%\
AccMwMEs.bat" "C:\samples\virlock.exe" ", ......)
```

[AccMwMEs.bat] echo WScript.Sleep(50)>%TEMP%/file.vbs cscript %TEMP%/file.vbs del /F /Q file.js del /F /Q %1 del /F /Q %0

When an infected sample gets to execute on a clean system, we say that the sample is the original one which is the primary cause of the infection. This sample is almost like any other fresh infected sample, which was not executed after the infection. There are some flags hard-coded into the malware so that it knows, at runtime, whether the sample being executed is a fresh infection that has not been executed before, or a drop made by malware targeted as a service or a malicious process running on the user's system. Figures 11 and 12 illustrate that behaviour.

2.1.2.3 Anti-emulation

Most malware creators integrate into their applications techniques to escape emulation and/or virtual machines. There are a number of known methods to accomplish that, we won't discuss all of them, but mainly those used by Virlock.

Among all the techniques which can cause emulators not to work, there are time constraints and unimplemented emulated API calls. Some emulators which are at the beginning, might have problems overcoming both of these, others might give up over time constraints (mainly because authors consider this a performance hit), and other advanced emulators could solve all of these in more efficient ways. However, most emulators are somewhere in the middle most of the time. We have to consider the possibility that from time to time

Hard-coded value	Meaning
0	Installed malware process, usually two synchronized processes
1	Original sample, installs malware components
2	Intermediate actions (while rooting into environment), brute-force user account password
3	Multithreading and synchronization (screen-locking, online payment)
4	Sample is running as service

Table 1: Associations between hard-coded values and their meaning.

0040156F	C705 B8184000	£q 10	mov	dword [0x4018b8],	0xc0f	
00401579	C705 BC184000	<u></u> &0↑0	mov	dword [0x4018bc],	0x401	908
00401583	FF35 D8184000	~5 ÷ †@	push	dword [0x4018d8]		
00401589	8F05 C0184000	A≪@	νου	dword [0x4018c0]		
0040158F	E8 C8000000	호민	call (1)	sub_40165C		
00401594	E8 6F030000	Σo♥	call (2)	sub_401908		
00401599	833D C8184000	â=Ľ↑@	стр	dword [0x4018c8],	Øx1	Sample is original one
004015A0	75 3C	u<	jnz (3)	loc_4015DE		
004015A2	A1 D0184000	í [⊥] †@	mov	eax, [0x4018d0]		
004015A7	0305 D4184000	₩₫₽₽₽	add	eax, [0x4018d4]		
004015AD	8D3D 00144000	ì= 900	lea	edi, [0x401400]		
004015B3	033D CC184000	¥= ¦†@	add	edi, [0x4018cc]		
004015B9	033D C4184000	¥=—10	add	edi, [0x4018c4]		
004015BF	83C7 08	âlk	add	edi, Øx8		
004015C2	8B1D E0184000	ï⇔α10	mov	ebx, [0x4018e0]		
004015C8	A3 B8184000	úq 10	mov	[0x4018b8], eax		
004015CD	893D BC184000	ë≟J†@	mov	[0x4018bc], edi		
004015D3	891D C0184000	ë⇔L†@	mov	[0x4018c0], ebx		
004015D9	E8 7E000000	Σ~	call (4)	sub_40165C		
004015DE			loc_4015DE:			
004015DE	E8 5A060000	ΩZŧ	call (5)	sub_401C3D		
004015E3	833D C8184000	â=Ľ↑@	cmp	dword [0x4018c8],	0×0	Sample is one of the 2 processes
004015EA	74 09	tO	jz (6)	loc_4015F5		
004015EC	833D C8184000	â=Ľ↑@	стр	dword [0x4018c8],	Øx4	Sample is running as Service
004015F3	75 ØB	นชี	jnz (7)	loc_401600		· ·
004015F5			loc_4015F5:			
004015F5	A1 F4184000	í (†te	mov	eax, [0x4018f4]		
004015FA	3105 DC184000	14_10	XOP	[0x4018dc], eax		

Figure 11: First context switching actions.



Figure 12: Last context switching actions.

malware creators reverse our engines and create malware which might target some of these security engines. If that is the case, then no matter how strongly an emulator is built, it might become useless if it's being targeted by malware.

Randomly chosen API calls

In an attempt to morph itself, Virlock rebuilds itself inside each infection, decorating the core of functionalities with things like random API calls from randomly chosen modules. The malware uses some tables, meaning that it does not choose from a huge set of possibilities but from a finite set. It chooses a random number of libraries which the future infection will import, and from those libraries, some random APIs inside each of them are chosen as imports.

If emulators are only emulating a certain set of APIs, then that might impede their ability to continue at the point of an unknown API call, or an API call not implemented accordingly (Figure 13).

Increasing the number of executed instructions

Most malware, be it packed or unpacked, does not require more than a few million instructions to be executed. At that point there are optimizations such as binary translation, which tries to improve performance over emulated loops like decryption blocks which get to be executed by the real processor and not by the emulator. Binary translation is sometimes combined with file-read operations – the best emulators will try to reduce the number of read operations and at the same time the maximum number of instructions allowed to be executed.

All versions of Virlock have a first stage decryption. Without it, any further code execution is basically impossible. There is

currently no version that executes fewer than 60M instructions for that purpose, and the number of instructions increases for bigger files and larger obfuscated loops, to hundreds of millions of instructions. Some infections also spread the obfuscated loops over a large area of the infected file, thus passing to emulators the pain of consecutive file reads, which also is a hit for performance.

There are many cases where the binary translation for loops is almost impossible if we are not first going to de-obfuscate the code being executed by the loop. Figure 14 shows such a case where just three calls to load more than 180 APIs from different modules is taking at least 500k instructions.

2.1.3 Cheating users

Very rarely seen in other pieces of malware of this kind (which embed the clean file into a totally different file), Virlock tries to cheat users into thinking that an infected file is actually what its icon claims it to be. There is a stage in the infection process where the malware searches inside the registry for the application associated with an extension type, in order to get to the file containing the icon of the associated application. This is a primary step for grabbing the icon and embedding it into the final infected file as an icon-resource. At a first glance, there is no difference between the original file and the infected one.

Straight after the infection, the malware will set a registry setting to hide extensions for known filenames. That way users will see their original files with their relevant icons and no EXE extension, so no one will ever doubt the actions of the file.

2.1.4 Polymorphic engine

The thing that makes Virlock so special is that it has a polymorphic engine which mutates its shape in future



Figure 13: Consecutive blocks of random API calls, trying to escape emulators from the beginning.

0045AE71	B8 4C000000	ηL	mov eax, 0x4c ;'L '
0045AE76	A3 C5B64500	ú H E	mov [Øx45b6c5], eax
0045AE7B	EB ØF	δŵ	jmp (2) Entry Point
0045AE7D			loc_45AE7D:
0045AE7D	E8 65060000	δe●	🖕 call (3) sub_45B4E7 Search for ModuleHandle 👘
0045AE82	E8 85040000	₽à♦	🖌 call (4) sub_45B30C Search one API
0045AE87	E8 AE020000	∑ <68	call (5) sub_45B13A GetProcAddress for API
0045AE8C			- Entry Point
0045AE8C	833D D9B64500	â=- E	cmp dword [0x45b6d9], 0x0
0045AE93	77 E8	wΣ	ja (6) loc_45AE7D
0045AE95	C705 70A14500	 ‡píE	mov dword [0x45a170], 0xb6f89f
0045AE9F	0F31	×1	rdtsc
0045AEA1	3105 EFA14500	1 2 0íE	xor [Øx45a1ef], eax

Figure 14: Loading some APIs (calling is based on templates discussed in 2.1.2.2).



Figure 15: Infected files with extensions revealed.

infections. In this section we reveal the techniques used by the malware to accomplish this task.

Straight after the API-loading process, the malware allocates two buffers (one of them big enough to hold the core of the malware) to prepare the morphing process for the infections to come. The core of the malware is somewhere inside the infected application, but only visible after a few stages of successive decryption procedures. Figure 16 shows the schematics of the core, which resides packed, layered inside any infected file.

A polymorphic engine is located in our example at 0x45E636 and it is called several times during the installation of the malware into the newly infected system. Each new malware copy will also have modified the flags discussed previously, accordingly.

The process of shape-changing is accomplished in two steps, for each of the two dropped files which are going to do the



Figure 16: Virlock core with embedded clean application.



Figure 17: Code calling the polymorphic engine.

real infection. Figure 18 shows the preparation for the reshaping of a self-copy.

The first stage consists of preparing random file names, some random seeds, and the buffers involved in the morphing procedure (see Table 2).

We also see at this step the creation of two different MZPE file headers, originally packed inside the malware (see

00459A77	FF15 51B74500	§QπE	call (3) [kernel32.dll	:SetFileAttributesW]
00459A7D	6A 00	j i	push 0x0	
00459A7F	6A 00	3	push 0x0	
00459A81	6A 00	3	push OxO	
00459A83	6A 00	3	push ØxØ	
00459A85	6A 00	ă 👘	push ØxØ	
00459687	FF35 18984500	51¥E	push dword [0x4598]	18]
00459A8D	6A 00	.i	push ØxØ	
00459A8F	E8 A24B0000	Σ όΚ	call (4) sub_45E636	Fill Mornhing TABLES
00459A94	6A 10	.i►	push 0x10	Third phing TAbles
00459A96	8D05 11934500	ĭ≎4ôE	lea eax. [0x45931]	1]
00459A9C	50	P	push eax	
ØØ459A9D	FF15 11B74500	-δ-4nE	call (5) [kernel32.dll	Rt1ZeroMemory1
00459AA3	6A 44	.iD	push Øx44 ;'D	
00459665	8D05 21934500	ĭ¢!ôE	lea eax. [0x45932]	1]
00459AAB	50	P	push eax	
00459AAC	FF15 11B74500	§∢nE	call (6) [kernel32.dll	:Rt1ZeroMemory]
00459AB2	6A 07	.i•	push Øx7	
00459AB4	FF35 18984500	Š5†₩E	push dword [0x4598]	18]
00459ABA	FF15 51B74500	SQnE	call (7) [kernel32.dll	:SetFileAttributesW]
00459AC0	68 11934500	h∢ôË	push 0x459311	
00459AC5	68 21934500	h‡ôE	push Øx459321	
00459ACA	6A 00	j.	push ØxØ	
00459ACC	6A 00	ă –	push ØxØ	
00459ACE	6A 00	ă 👘	push ØxØ	
00459AD0	6A 00	ă 👘	push ØxØ	
00459AD2	6A 00	ă 👘	push ØxØ	
00459AD4	6A 00	ă 👘	push ØxØ	
00459AD6	6A 00	3	push OxO	
00459AD8	FF35 18984500	5†ÿE	push dword [0x4598]	18]
00459ADE	FF15 61B74500	SanE	call (8) [kernel32.dll	CreateProcessW1
00459AE4	B8 01000000	∃ 🖯	mov eax, 0x1	
00459AE9	C3	ŀ	- ret	

Figure 18: Preparing the reshape of a self-copy.

Buffer alias	Buffer size	Buffer ptr	Description
TAB1	0x200	0x970000	Randomization table 1
TAB2	0x2300000	0x1100000	Working buffer for reshaping procedure
TAB3	0x10000	0x9A0000	Intermediate table 1
TAB4	0x10000	0xAA0000	Intermediate table 2
TAB5	0x200	0x980000	Randomization table 2

Table 2: Buffers involved in the morphing procedure.

pusn	DXT000																											· · · · · · · · · · · · · · · · · · ·
push	0×400	00052FF0	63	EB .	DA BI	4 3 E	มขา	2B F	2 BA	30	03 01	1 100	83 E	9 04	I EB	C3 8	3 FA	05 7	'D B4	BH	CE 1	73 U	L FE	EB	רע .	BB (:8	10-11; -<< ₩ 00 ab +0 fa •2>111;s - 10 - 10
push	0×0	00053010	85	24	FE B	9 7F	' 00 I	90 OI	0 EB	07	83 F9	7 01	7D B	2 EE	B 36	EB 5	1 C3	49 B	3A 5E	75	BE I	7D EI	B EE	BA	26 1	6D 1	18	à\$∎¦Δ δ•â•©>‰δ6δQ+I ^u∃²δ€ &m↑
call	[kernel32.dll:UirtualAlloc]	00053030	FD	C7	05 71	B 2 F	45 (10 6:	1 9E	31	00 E8	8 B3	FF F	F FF	FBB	61 2	2 F9	F9 E	B 48	46	BA I	E8 F8	3 10	F7	EB 3	D5 8	33	2∭\$Č∕EaB1.⊉[¨πa"••δHF∭29⊏≋δrâ
mou	[Ry452252] Any	00053050	120	04	7 1 A	n I C 1	ER 9	C B	R 90	64	ED ES	7 FR	C5 R	IN FG	113	AC E	0 33	06 8	13 CG	04	RR 4	IN 11	2 F1	172	FR ·	18.8		•• 3 EL6F ====================================
nuch	0.400	00053070	10	04	FR SI	FR	20	0 4	5 00	RR	F4 10	120	F9 R	18 FT	111	FF F	FRO	5F 1	3 97	172	FR I	14 8	2 19	04	FR	CO T	'n	
push	07100	000000000	10	nn i	29 01	126	TTN 1	70 7	3 88	FF	EF DI	0 02	ົວດໍກັ	E EC	91	122 7	9 20	70 0	TE DO	21	TE S	ic E	5 10	82	ดีวิเ	66 6	ลัด	4-1)-6200 -L6 1012 / 11/18-040
pusn	Cax	BRAESADA	1210	F	00 h	2 04	82	2 6	2 04	102	11 DI		10 A	10 00	5 61	00 A	a h2	45 b	10 DO	82	IAL 1	D D/	- ac	102	04	DA 6	20	
Call	LKerneijz.all:Ktizeronemory]	000033000	128		70 D	4 40	00 1	2 0	2 0.4	02	03 FI		D2 0	12 DO	5 02	01 1	0 02	23 0	50 02	83	OL 1	LF DI	UE	102	UL .	D-I E	97	
mov	edi, 10x4521571	00023000	U U	21	55 0.	L 4U	עטי							P 07		DT		03 0				1 2	0 04			<u>(4</u>		= CL=11115 program cannot be ru
lea	esi, LUx4530b0J	00023010	1 P F		рд рі	22	44 -	H 5.	3 20		ph p.	1 65	ZE U	זה חו	N MH	24 U	2 07	65 0	лл н з	87	21 t	SC C	U U4	21	66	CD I	14	In in DOS mode. The mean citerian
xor	ecx, ecx	00053110	21	60	CD D	1 A	23 1	DF D	4 2 B	60	CD D	1 21	EC C	D D4	1 20	ec c	D D4			68		C CI	D D4	D2	10	50 3	15	I I = 5xs E+I=E I=E I=EKich! I=EmPPE
mov	ebx. Øx12?	00023130	D2	02	4C U.	L N 3	D2 1	11 F	8 20	<u>H B</u>	54 Da	2 108	FND	10 20	LØF	ดา ด	вы	<u>67</u> 6	IC D2	01	NP 1	JZ 0.	3 04	D2	- YN	10 1	22	<u>пшг⊜ь⊔⊜,-⊼т⊔бж⊓екере</u> ъти⊜ д ие∌⊔
call	sub_4533DC 1st file HEader unpack	20053150	03	10	D2 Ø	3 20	I D2 (14 41	0 D2	02	10 D2	2 03	02 D	12 02	2 04	D2 Ø	7 04	D2 Ø	17 OD	30	D2 6	J3 Ø4	1 D2	06	02 :	D2 0	95	│
push	0x40 ;'0 '	0005-170	10	D2	02 11	ð D2	04 :	LØ D:	2 02	10	D2 Ø6	5 10	D2 Ø	IB 10	C 20	D2 Ø	2 3C	D2 5	64 20	D2	02 1	LC D2	2 1B				78	I⊳ղ©≻ղ⇔⊳ղ©≻ղ∰-ղő∟ ղ©ՀղI ղ©–ղ+.tex I
nush	0×1000	00053190	74	D2	03 4	8 04	D2 0	33 11	0 D2	03	Ø6 D2	2 03	04 D)2 ØE	E 20	D2 Ø	2 60			61		1 D:	2 02	EØ	D2	04 2	20	tnvH♦nv⊳nv≙nv♦nJ n8`.rdatan8xn♦
nush	0×600	000531B0	D2	03	Ø2 D:	2 03	: ØA])2 Ø	E 40	D2	Ø2 4					D2 Ø	3 ØD	D2 Ø	14 30	D2	03 6	12 D:	2 03	ØC	D2	0E 4	10	nwonwonfende.datany.fn+0nwonwenfe
nush	AxA	000531D0	D2	02	CØ D:	2 EC	: D2 1	EC 4	D 5A	УИ	D2 M1	1 103	D2 14	I3 N4	1 D2	ИЗ F	FFF	D2 Ø	12 B8	D2	07 <	10 D:	2 23	B8	D2	Ø3 Ø	JE	ng non incinenten weweren in incinententententententententententententent
call	[kewnel32_dll:lliwtual0]loc1	000531F0	1F	BA I	NF D	2 01	B4 1	19 C	D 21	B 8	Ø1 40	c lõb					0 70			22							IP .	TVIII - 1 a BL This program canno
BOUL	[AvdE24Eb] asy	00053210	24														4 65	9	in an	00	24 1	12 01	7 65	lan	03	87 2	21	t he pup in DOS node PESsuee Bict
nuch	0.600	00053230	60	CD	N4 2-	1 60	cn i	14 2	1 60	ch	D4 01	F 73	DF D	14 21	a lac	ch h	4 21	60 0	n n4	20	60.0	ים חי	1 52			68 2	1	1=bt1=bt1=bye lb+1=bt1=b 1=bliebt
push	0,000	000000200	lčč	čň	D 4 D 4	110	CO .	ic n	2 02	40	ดังดิว	5 1 1 2	61 D	10 21	A D D	E4 n	2 60	EG D	กัว ดีส	ñr.	6 1 6	ลัก ดี	Î	lac.	n 2	01 0	36	1-L-DE-0104-00-VT-R-0x020A0-04
pusn	Cax []	000000000	102	62	04 D		101	13 0	5 1 6	D 2	02 00	3 55	04 4	0 11	1 82	10 0	5 63	00 D	12 01	04	D2 0	0 0	4 03	107	80	20 1	20	
Call	LKernel32.dll:KtlZeronemory1	80053270 80053270	02	80	D-2 04	6 07	101	22 U. AE 41	0 10	02	10 0	9 04	10 0	10 12	104	70 D	6 10	D2 0	12 U2	201	D2 6	10 20	1 D2	EA	20	20 1	22	
mov	ea1, L0X45215D1	000003270	103	00	4 10 01	2שן כ	. DZ 1	12 1	0 02	02	40 0	1 09	10 1	0 02	100	D2 0	0 10	D2 0	10 10	40	D2 6	32 31	5 02	2.4	20	DZ 8	92	
lea	esi, L0x4531d71	000532B0	10	DZ :			05			03	48 0	1 02	03 I	DZ 02	2 03	00 1	2 03	ย6 ม	12 OE	20	D2 6			1.64	64			LULTCOXCUANAUALUATUATUA UA TUATUA
XOP	ecx, ecx	00023200	61	<u>D2</u>	02 E	a 112	04	10 D	2 03	02	DZ 0.	100	D2 0	IL 40	a 102	<u>02</u> 4	U ZE	64 b	1 74	61	D2 6	17 [0]	צע ע	04	30.	ע צע	93	
nov	ebx, 0x15b	TATATAL S CON	02	D2	03 0	E D2	0E	IN D	2 02	СN	D2 E0	5 D2	EC 6	A D2	2 01	<u>68</u> N	5 30	40 D	2 01	68	D2 4	11 31	J 40	D2	101	<u>68 I</u>	22	ຕັມຈຸນພາຕຸມຕູມຕູ ມແຫຼງມີຕູມສູດຕູມຕູມຕູດຄຸມຕູລາມ
call	sub_4533DC 2nd file Header unpack	BUUE 231 B	01	E8 :	170	4 D2	02 (SA D	2 01	E8	16 04	4 D2	02 E	8 17	7 04	D2 Ø	2 E8	1E Ø	14 D2	02	E8 1	13 04	1 D2	02	C3 :	D2 I	EB	<u>ՅՉ‡♦</u> Π@jnΘΣ_♦ Π @Σ <u>‡♦</u> Π@Σ≜♦ŋ@Σ!!♦π@ πδ
mov	dword [0x452f7b], 0x2dc58	00053330	D2	EB	6A 4I	4 68	UU .	ש ש.	0 00	68	ยย ยะ	1 00	00 6	N UK	d FF	15 E	D B6	45 U	10 A 3	-57	2F 4	15 UI	1 68	90	04	UU 1	90	πδjθh r n v j s¤∥≞ ա⊮∕≞ n v
rdtsc		00053350	50	FF	15 1:	L B7	45 (8 00	B 3D			00	8D 3	15 BØ	30	45 0	0 33	C9 B	3B 27	01	00 0	30 E8	3 6D	00	00	00 6	5A .	Psan£i=W∕Eì5⊗9E3mm′© ହn j
XOP	[0x45302d]. eax	00053370	- 40	68	00 11	3 00	00 (58 ØI	0 06	00	ØØ 6A	1 00 F	FF 1	.5 EL	D B6	45 Ø	5 A3	5B 2	F 45!	00	68 6	<u>10 0(</u>	500	00	50	FF 1	15	eh ⊧h ÷j§øEú[⁄Eh ÷P§
XOP	[0x453041], eax	00053390	11	B7	45 ØI	2 8 E	3D :		F 45	00	8D 35	5 D7	31 4	15 00	3 33	C9 B	B 5B	01 0	10 00	E8	30 6	<u>10</u> 01	3 00	C7	05 '	7B 2	2F	
YOP	[0x4530921 eax	000533B0	45	90	58 DI	C 02	90 0	IF 3:	1 31	Ø5	2D 36	45	00 3	1 05	5 41	30 4	5 00	31 Ø	15 92	30	45 F	10 E	3 3F	FC	FF 1	FF 6	13	Е Х — 8 ж11 Ф – ИЁ 1 ФАЙЁ 1 ФЕЙЕ Ф? ¹¹ ú
call	sub 45300F	00053300	ÂĎ	2F -	45 ØI	4 E8	18 1	C F	FFF	90	90 C	3 60	81 3	D EF	8 33	45 Ø	0 E1	6B F	19 00	ØĒ	84 4	ID Ø	เดิด	00	EB	04 I		i∕E ō→" éé l`ii=∩3E βk• xáM⊡ δ♦β
BOU	[0v452fad] eav	000533F0	6B	F9	ññ Bi	B 25	24	17 F	F 81	F2	FF D	5 DC	FC 8	1 F3	A A A	F6 2	F FB	BA 4	IS ED	F3	F9 1	8 B	วัดดี	ดด	ดดั	BA F	38	k· mySG ii2 r="ii(á÷"\[[Fø\-6m]0
call	sub 452FF3	00053410	Řő	ĈÁ	F9 8	I F3	4F 1	8 2	5 F8	B B	1F 19	5 21	FD 3	D 3F	E FF	ÂĔĎ	à ÂF	85 2	4 F4	FĂ	ÊÊ Î	รัต ดีเ	A BB	108	B3	ดีจี โ	28	11 ^μ ·ii(0aγ ⁰ αγδα ² =:6αbba5Σ·δ α±10 ⁰
Dan		00053430	67	05	FR 3	2 40	ด้ด	N G	R RO	ññ	RR A	alon	D6 R	E ES	2 66	ดดี ดี	ดัดดิ	RR C	18 30	86	F8	70 01	. 00	00	ññ	RE 2	9Ř	
nop			_	_		_		_		_	_			_				_			_		_	_	_			
nop																												

Figure 19: Preparing headers for the files to be constructed.

50	B2	1D	58	64	00	00	00	01	00	02	00	00	00	00	ØE	P‰+Xd ☺ 🛢 🎵
43				69				69				7 8		00	00	
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
ดด	ดด	ดด	ЙЙ	I ÃÃ	ดด	ЙЙ	ЙЙ	ÑĀ.	ดด	ดด	ดด	โดด	ดด	ดด	ЙЙ	
ดด	ดัด	ดัด	йй	lãã	ดัด	йй	ดัด	ЙЙ	ดัด	ดัด	ดด	FD	NR	8F	FS	2 1 60
FS	йй	йŇ	йй	йĭ.	йŇ	ดว	йй	йñ	йñ	йŇ	10	147	Ĩ.	74	ĀĔ	
70	70	20	20	25	20	54		CT.	74	70	² 0	26				
						60	00		60	60	00		60	00	66	
10		11	0.0		11	00	00	00	00	00	00	100	00	00	00	
00	99	90	90	90	90	90	99	90	90	90	90	166	00	90	99	S164 D
ิยย	บบ	พย	ัดด	00	พย	ัดด	ษย	Н5	17	26	31	71	<u>ال</u>	ียย	บบ	N+u1q©
61	บบ	บบ	บบ	เดด	บบ	61	10	47				172				⊎GetSyste
6D	44			61				55				6E				
67		00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00	00	00	00	C5	B1	66	2D	E6	00	00	00	01	00	00	00	+∭r−μ ☺
00	00	01	ØF	47				6F				6E	64			OxGetCommandLi
6E			ЙЙ	NO.	ЙЙ	ЙЙ	ЙЙ	NO.	ЙЙ	ЙЙ	ัดด	I A A	้ดด	ดด	ด้ด	
ดด	ดด	ดด	ดด	โดด	ดด	ดด	ดิด	ดด	ดด	ดด	ดด	โดด	ดด	ดด	ดด	
ññ	йй	йй	йй	йй	йй	йй	йй	йñ	йй	йй	йñ	lãã	йй	йй	йй	
00	51	64	RC	FF	йŇ	йЙ	йЙ	M 1	йŇ	M 1	ññ	lãã	йŇ	<u>й1</u>	10	0400 0 0 0+
47	ČĒ	14	43	120	20	10	60	27	61	90	64		60	1	-	CatClinhandSocu
26										00	00		00	00	00	
00				46				00		99	99	100	99	99	66	

Figure 20: Building a customized import table.

Figure 19). Their purpose is to fulfil the creation of the processes which will actually carry out the infection.

In the beginning of the second stage, the malware creates a custom import table, also based on time seeds (see Figure 20). The RDTSC instruction, which provides those time-seeds, is called very frequently, not only to randomize stuff, but also for choosing random locations in the target application, where relevant data regarding decryption keys, buffer pointers, etc., will be placed.

In Figure 21, we can see a sequence of instructions which progressively builds the decoration of the new infection.

All the steps required for a full file creations are called in a sequence of three consecutive calls, as shown in Figure 22 {reshape / append / recrypt}.

2.2 ANALYSING VIRLOCK – COMBINING PURPOSE

We have seen lots of malware categories that combine their powers with other malware categories. The results of those combinations have, most of the time, been some kind of surprise for security products. Not only do malware authors learn from security products how to improve their performance, but we also learn from malware authors that there is always something which we have not taken into account in the first place. This sounds like an evolving loop, where security products try to nullify malware actions, while on the other hand malware authors try to nullify security products' actions. Well, at least the loop is more like a threedimensional spiral, otherwise we would not exist at this moment in time.

The following is a brief history of combined malware actions including Virlock, which we find as a reference for this case:

- Viking / Jadtre rootkit and file infector
- CBDoorK rootkit and backdoor
- Sality file infector, botnet, worm
- Virlock ransomware, file infector.



Figure 21: Reshaping a new infection.



Figure 22: Main reshape steps for self-copies.

2.2.1 File infector and screen-locker

Until Virlock, no other malware combined these features. Malware authors who write ransomware are doing it for the money – they say as much in their readme files appearing on the infected computers. For example, a piece of ransomware using the Bitlocker feature from *Windows* tells the infected users that 'This is just how business works, pay and you'll get your data back.'

Early versions of ransomware only locked users' accounts, hoping that some of them would fall into their trap – and they succeeded, but there is always room for improvement. Some of the next versions tried to encrypt users' files with symmetric keys and locked the users' accounts, making it more difficult to revert the process. But as the security products improved their strategies and delivered rescue-CDs to users, malware authors improved their methods of cryptography, using asymmetric algorithms, and gave up the screen-locking. When infecting users with those kinds of ransomware, malware creators need a management system in order to bind private-keys with malware versions. Maybe they did not expect their methods to be so fruitful, but they seem to be overwhelmed by the number of infected users and public/private keys. It is not unusual for a user to try to pay, and get a decryptor which attempts to decrypt files from a different infection.

Virlock tries somehow to escape the load produced by the key-infection management system while improving the old techniques used in locking files and user accounts by embedding the clean file and packing it safe inside the malware with random and hard-coded keys. It also tries to crack users' account passwords, to lock their account in order to make it as difficult as possible for the users to recover their files. Using the presented technique for file infection, security products have to consider an entire arsenal of variables in order to begin a clean method, because it would be very easy to miss a certain hard-coded-key and to damage the file instead of recovering it.

3. GETTING TO THE CORE OF VIRLOCK

We've seen so far that Virlock uses a template-based reshape, so we can use that template as some kind of regular expression to find some inner pylons / code-blocks to start with. Studying the five different versions until now, there are certain similarities between them, which will lead us to classify a sample as infected.

In this chapter we will try to reveal the malware's weak points and see how those weaknesses may contribute to studying it better in all its present forms.

3.1 Revealing the core, inside different malware versions

First, there is an initial layer of decryption which will end up by continuing the execution somewhere at FirstSectionVA+0x400 or FirstSEctionVA+0x1000 with or without additional obfuscated code and possibly a short second decryption stage (Figure 23).

There are two major switch sections inside the malware which choose a path of execution depending on the hardcoded flag discussed in section 2.1.2.2. We will consider the

		DISASII
	Disasm	Entry Point
Disasm	emm dword [0x401442], 0xd0hc8a	amp durand [0y/01/12] 0yof0/01
cmp dword [0x401442], 0x1509fc	$\frac{1}{10}$ (1) $\frac{1}{100}$ 401507	
iz (1) loc 40158F		JZ (1) LOC_40154D
$\frac{1}{100}$ $\frac{100}{101}$	Jmp (2) Loc_401446	.imp (2) loc 401416
Jub vs roc-reriio	dec edi	test eav 0v8100d83f
int 3	petf	
cmpsb	toot duoud [any] 0x0b2o1Ebb	100-401410
iecxz (3) loc 401446	test uworu leaxi, oxonaciann	db Øxf2
	Loc_401446:	pop ds
100_101110.	idiv dword [edx-0x3199b91]	non edi
xor ebx, Øxf9f15aee	vov edv Øvfeh1h76f	$\frac{1}{2}$ (1) $\frac{1}{2}$ (0) $\frac{1}{4}$
mov edx, 0xfae24090	way adv avfdEabfd7	Jb (3) 10C 401410
mou edx. 0x9d9248		mov edx, Øxfd634b09
α_{11} (4) sub 40140	call (3) sub_401500	xor ebx. 0xfb490a68
	mov edx, Øxfacac4b8	vor ehv. Øvfdd11e29
xor eax, 0xfc0flala	xor edx. 0xfdd064a9	wan ody Øyffhd9aad
mov edx, Øxfd855def	you aby ayfd65cca9	XOP Cux, OXITDUTCAT
mov edx. Øxfd88c264		call (4) sub_401492
nou edv Øvfee329e7	xor eax, 0xfa51402a	mov edx, 0xf17790
	xor edx, 0xfc3f8a47	vov ebv. Øvf89df869
cmp eax, ex700a3037	cmn eax. 0x4h296727 :''g)K	mou ody Av£929bac6
jnz (5) loc_40197C	ing (4) loc 401978	MOV GUX, OXI 730JJACO
imp (6) loc 401514		cmp eax, 0x14c14bcc
sub ecv Øv4	Jub (2) TOC 401401	jnz (5) loc_401843
300 667, 671	mov ebx, Øxf8b8d3bf	imn (6) loc 401455
	Leven Facil	100 401455
		100_101103.
Diesen		

cmp	dword [0x401262],	Øxec1132
	100_101300	
յտք (2)	10C_401266	
xchg	ebp, eax	
CMC		
popf		
ĥĥĥ	<pre>lecx+0x4dd55af21.</pre>	al
loc 401266:		
db Øxfe		
mov	edx. 0xffd42f10	
xor	ebx 0xfb40704e	
call (3)	sub_40135A	
XOP	edx. Øxfcea7238	
mov	edx Øxfb87426e	
XOP	edx Axfe6e5846	
YOP	edy Øyf746d7e8	
VON	edv Øvfab5c993	
A01-	0.02xx2d2x4	
CMD	eax, oxzaazuzai	
jnz (4)	100_401681	
jmp (5)	loc_401369	
loc_4012A9:		
mov	esi, 0x4013a6	
	cmp jz (1) jmp (2) 200 xchg (2) xchg (2) 200 add (4) 00 (4) 200 loc 401266: db (4) 200 xor (4) xor (4) ymp xor (5) ymp (5) ymp jmp (5) ymp (5) ymp	Cmp dword [0x401262], jz 10c_401306 jmp 20 10c_401266 xchg ebp, eax cmc popf add [eex+0x4dd55af2], loc_401266: db 0xfe mou edx, 0xffd42f10 xor ebx, 0xfb40704e call (3) sub_401350 xor edx, 0xfb47426e xor edx, 0xfb87426e xor edx, 0xfb8746d78e xor edx, 0xf2665293 cmp (bc_401681) jmp (bc_401689) loc_441369 loc_441369 loc_

D	isasm	
	cmp	dword [0x401262], 0x4ffd3c
	jz	loc_4013B1
	jmp	loc_401266
	aam	Øxca
	iret	
	add	[edx-0x39a27e0], bh
	mov	ebx, 0xfe0236b0
	mov	edx, 0xff510487
	call	sub_40136C
	XOP	edx, Øxfb9bd78e
	mov	ebx, 0xfb879616
	XOP	edx, 0xfe036542
	mov	edx. Øxfbf5f934
	mov	ebx, 0x393578 ;'x59 '
	cmp	eax, Øxcea62818
	jnz	loc_401698
	յան	loc_401355
ŀ	- ret	

Figure 23: First chunk of relevant code in all five versions.

NOV EAX, OED9	MOV EAX, 0F93	MOV EAX, OD7E	MOV EAX, 7AE	MOV EAX, OD45	
LEA EDI,DWORD PTR DS:[401A66]	LEA EDI, DWORD PTR DS: [401A63]	LEA EDI, DWORD PTR DS: [401753]	LEA EDI, DWORD FTR DS: [401902]	LEA EDI, DWORD PTR DS: [40173C]	
MOV EBX, DWORD PTR DS: [401A36]	MOV EBX, DWORD PTR DS: [401A33]	MOV EBX, DWORD PTR DS: [401723]	MOV EBX, DWORD PTR DS: [4018D2]	MOV EBX, DWORD PTR DS: [40170C]	
MOV DWORD PTR DS: [401A16], OED9	MOV DWORD FTR DS: [401A13], 0F93	MOV DWORD PTR D5: [401703], 0D7E	MOV DWORD PTR DS: [4018B2], 7AE	MOV DWORD FTR DS: [4016EC], 0D45	
10V DWORD PTR DS: [401A1A]. v2.00401A66	MOV DWORD PTR DS: [401A17], v1.00401A63	MOV DWORD PTR DS: [4017071.00b904b9.00401753	MOV DWORD PTR DS: [4018861.c259785e.00401902	MOV DWORD PTR DS: [4016F01.0447773h.0040173C	
PUSH DWORD PTR DS: [401A36]	PUSH DWORD PTR DS: [401A33]	PUSH DWORD PTR DS: [401723]	PUSH DWORD PTR DS: [4018D2]	PUSH DWORD PTR DS: [40170C]	
POP DWORD PTR DS: [401A1R]	POP DWORD PTR DS: [401A1B]	POP DWORD PTP DS+[40170B]	POP DWORD PTP DS+[4018B1]	POP DWORD PTR DS: [4016F4]	
CALL v2.00401735	CALL v1.00401741	CALL 00b904b9.00401446	CALL c259785e.00401651	CALL 0447773h. 00401484	
CALL v2.00401A66	CALL v1.00401A63	CALL 00b904b9 00401753	CALL c259785c 00401902	CALL 0447773b 0040173C	
TWP DMORD PTR DS: [4014261.1	CMP DHORD PTR DS: [4014231.1	CMP DHORD PTP DS: [4017131 1	CMP DEODD PTP DS+[4018C21 1	CMP DWORD PTP DS-CAULERCL 1	
INZ SHOPT w2 00401677	JN7 SHOPT w1 0040167D	JN7 SHOPT 000-90409 00401431	JN7 SHOPT c259785a 004015D3	JWZ SHORT 0447773b 00401426	
NOV FAY DEODD PTP DS- [40142F1	MON FAY DHORD FTP DS+[401A2B]	MON RAY DECED BTD DS: (401718)	WOU RAY DUODD BTD DS: (401904)	MON FAY DHORD PTP DS: [401704]	
ADD FAY DUODD PTD DS: [401432]	ADD FAY DEODD PTP DS+[401A2F]	ADD FAX DWORD FIR DS. [401715]	ADD FAX DUORD FIR DS. [4010CR]	ADD FAX DUORD FIR DS.[401704]	
FA EDT DHODD BTD DS: [401400]	LEA EDT DHODD DTD DS: [401400]	ADD EAX, DWORD FIR DS: [401/11]	ADD EAX, DWORD FIR DS. [401002]	ADD EAX,DUORD FIR DS:[401700]	
ADD FDT DHORD PTD DS. [401400]	ADD EDI DHODD DID DS. [401400]	LEA EDI,DWORD FIR DS: [401000]	ADD NDT DUORD FIR DS:[401400]	LEA EDI,DUORD FIR DS:[401000]	
ADD EDI,DUORD FIR DS:[401A2A]	ADD EDI,DWORD FIR DS:[401A17]	ADD EDI,DWORD FIR DS:[401/17]	ADD EDI,DWORD FIR DS:[401006]	ADD EDI,DWORD FIR DS:[401/00]	
ADD EDI,DUORD FIR DS:[401A22]	ADD EDI,DWORD FIR DS:[401A1F]	ADD EDI,DWORD FIR DS:[401/0F]	ADD EDI,DWORD FIR DS:[4018BE]	ADD EDI,DWORD FIR DS:[4016F8]	
NUL EDI,6	ADD EDI,0	ADD ED1,8	ADD ED1,8	ADD ED1,8	
NUV EBX,DUURD FIR DS:[401A3E]	MUV EBX,DWORD FIR DS:[401A3B]	MUV EBX, DWURD FTR DS:[401728]	MUV EBX, DWURD FIR DS:[4018DA]	MOV EBX, DWORD FIR DS: [401714]	
NUV DWORD FIR DS:[401A16],EAX	MUV DWURD FIR DS:[401A13],EAX	MOV DWORD PTR DS:[401703],EAX	MOV DWORD FTR DS:[4018B2],EAX	MOV DWORD PTR DS:[4016EC],EAX	
MOV DWORD FTR DS:[401A1A],ED1	MOV DWORD FTR DS:[401A17],ED1	MOV DWORD PTR DS:[401707],EDI	MOV DWORD PTR DS:[4018B6],EDI	MOV DWORD PTR DS:[4016F0],EDI	
HOV DWORD FTR DS:[401A1E],EBX	MOV DWORD PTR DS:[401A1B],EBX	MOV DWORD PTR DS:[40170B],EBX	MOV DWORD PTR DS:[4018BA],EBX	MOV DWORD PTR DS:[4016F4],EBX	
CALL v2.00401735	CALL v1.00401741	CALL 00b904b9.004014A6	CALL c259785e.00401651	CALL 0447773b.004014A4	
CALL v2.00401E65	CALL v1.00401EBF	CALL 00b904b9.00401A97	CALL c259785e.00401C65	CALL 0447773b.00401A84	
CMP DWORD PTR DS:[401A26],0	CMP DWORD PTR DS:[401A23],0	CMP DWORD PTR DS:[401713],0	CMP DWORD PTR DS:[4018C2],0	CMP DWORD PTR DS:[4016FC],0	
JE SHORT v2.00401694	JE SHORT v1.0040169A		JE SHORT c259785e.004015EA	JE SHORT 0447773b.0040143D	
CMP DWORD PTR DS:[401A26],4	CMP DWORD PTR DS:[401A23],4		CMP DWORD PTR DS:[4018C2],4	CMP DWORD PTR DS:[4016FC],4	
JNZ SHORT v2.004016A5	JNZ SHORT v1.004016AD	JNZ SHORT 00b904b9.0040144A	JNZ SHORT c259785e.004015F5	JNZ SHORT 0447773b.00401448	
NOV EAX, DWORD PTR DS: [401A52]	MOV EAX, DWORD PTR DS: [401A4F]	MOV EAX, DWORD PTR DS: [40173F]	MOV EAX, DWORD FTR DS: [4018EE]	MOV EAX, DWORD PTR DS: [401728]	
KOR DWORD PTR DS:[401A3A],EAX	XOR DWORD PTR DS:[401A37],EAX	XOR DWORD FTR DS:[401727],EAX	XOR DWORD FTR DS:[4018D6],EAX	XOR DWORD PTR DS:[401710],EAX	
NOV EAX, 0F21A6	MOV EAX, OE8A2A	MOV EAX, 2975C	MOV EAX, 0A2590	MOV EAX, 1BAC28	
LEA EDI, DWORD FTR DS: [40294F]	LEA EDI, DWORD PTR DS: [402A06]	LEA EDI, DWORD PTR DS: [4024E1]	LEA EDI, DWORD PTR DS: [4020C0]	LEA EDI, DWORD PTR DS: [402491]	
MOV EBX, DWORD PTR DS: [401A3A]	MOV EBX, DWORD PTR DS: [401A37]	MOV EBX, DWORD PTR DS: [401727]	MOV EBX, DWORD PTR DS: [4018D6]	MOV EBX, DWORD PTR DS: [401710]	
MOV DWORD PTR DS: [401A16], EAX	MOV DWORD PTR DS: [401A13], EAX	MOV DWORD PTR DS: [401703] EAX	MOV DWORD PTR DS: [4018B2], EAX	MOV DWORD PTR DS: [4016EC].EAX	
NOV DWORD PTR DS: [401A1A], EDI	MOV DWORD PTR DS: [401A17],EDI	MOV DWORD PTR DS: [401707].EDI	MOV DWORD FTR DS: [4018B6].EDI	MOV DWORD PTR DS: [4016F0],EDI	
NOV DWORD PTR DS: [401A1E], EBX	MOV DWORD PTR DS: [401A1B], EEX	MOV DWORD PTR DS: [4017081.EBX	MOV DWORD PTR DS: [4018BA1.EBX	MOV DHORD PTR DS: [4016F41.EBX	
CALL v2.00401735	CALL v1.00401741	CALL 00b904b9.00401446	CALL c259785e.00401651	CALL 0447773b. 00401484	
MOV DWORD PTR DS: [4014421.0F7CB4F	MOV DHORD PTR DS: [4014421.0E3A6CC	MON DRORD PTP DS+14012621 OFFCAD4	MOV DWORD PTP DS+[4014121 832646	MON DWORD PTP DS+[4012621 0447773b 009DF595	
RDTSC	RDTSC	PDTSC	DDTSC	DDTSC	
KOR DMORD PTP DS+F4014C21 FAX	YOR DHORD FTR DS+[4015151 FAY	YOR DHORD PTP DS: [4013601 FAY	YOD DRODD DTD DS+F4014171 FAV	VOD DHODD RTP DS+[4013741 FAY	
YOR DHORD PTP DS- [4014D61 FAY	YOR DHORD PTP DS: [401524] FAY	YOR DEGED FIR DS. [401300], EAX	YOR DUORD FTR DS.[4014961 EAV	YOR DWORD FIR DS.[401374],EAX	
YOR DWORD PTP DS+[4015011 FAX	YOR DHORD PTR DS+[4015301 FAY	YOR DWORD PTP DS: [4013721 FAY	YOR DROED FTR DS: [4014FF1 FAY	YOR DWORD PTP DS+[4012F31 FAY	
CALL 112 00401500	CALL 11 004014CD	CALL 00000/040 0040136C	Chil c250795c 00/01526	Chil 0447772b 0040125b	
NOV DWORD PTP DS+ [40147F1 FAV	MOV DUODD PTP DS+F4014711 FAY	MON DHODD DID DE DE LOUISOCI ENV	WOW DHODD DTD DG: 54014421 EAV	MON DHODD BID DS. [401204] FAX	
CALL 122 00401440	CALL 11 004014F1	CALL OUNDOW ALK DOVEMONTO POLICIES	CALL 22502950 00401470	Chil 0447773b 00401323	
KOD	NOP	NOD	NOD	NOD	
KOD	NOD	NOP	NOP	NOP	
ND v2 0040204E	TMD w1 00402105	NUP CONCOMPO DO 100 071	NUP	NOP	
DELL 00-00-02741	DETH	011 00090409.004024£1	DETE	017 04477730.00402491	
ALC: UNK					

Figure 24: A comparison between all five versions inside context-switch sections.

two sections as the core of the malware, as they are present inside all versions, no matter how obfuscated the code is, and the path to those functionalities is unique if an emulator behaves just like a real operating system.

Not all versions are as compact, as shown in Figure 24. There are some cases where junk-code might appear between relevant instructions in our target code, but ignoring them is not as difficult as one may think.

3.2 Searching for a match

Most detection algorithms will just try to find a relevant piece of code inside a piece of malware. Looking at the code shown in Figure 25, we might be tempted to say that we found something relevant for our malware (a branching point where it chooses to execute as installed or as a fresh infection). However, in other malware versions we found other such pieces of code, doing the same thing but with modified instructions. Considering this, the detection cannot choose that sequence of instructions to follow, but we need some rules depending mostly on the constant addresses given in the piece of code and the instruction types, which are not so different across different malware versions. This kind of matching seems to be as powerful as a regular expression-matching algorithm, but additional changes have to be considered.

3.3 Cleaning infected files

To recover the clean file from the malware, we need to follow the code until a point at which we can check whether the infection contains a clean file (switch-flag == 1) or not (switchflag != 1). If we do have a clean file, we need to grab the hardcoded values inside the malware (different with each infected file) and to force the emulation of decryption functions.



Figure 25: Piece of malware code to decrypt clean file.



Figure 26: Finding clean file using hard-coded variables.





Country name	Inf. systems	Inf. files	Country name	Inf. systems	Inf. files
China	22	192	Canada	3	19124
United States	11	663	Australia	2	6
Germany	10	106	United States	2	281
Russian Federation	9	82	Indonesia	1	4
Canada	8	10101	Russian Federation	1	1
Australia	7	3708	Vietnam	1	115
France	5	172	Namibia	1	23
Switzerland	5	909	Switzerland	1	29
Romania	5	219			
United Kingdom	4	544			
Iran, Islamic Republic of	3	77	Country name	Inf. systems	Inf. files
Sweden	3	42	France	2	205
India	2	15	Canada	1	1
Bosnia and Herzegovina	2	47	Indonesia	1	13
Ukraine	2	9	United States	1	6
Netherlands	2	1178	Russian Federation	1	2
Poland	2	4	Vietnam	1	33
Indonesia	2	74	Namibia	1	2
Vietnam	2	1388	Switzerland	1	5

Figure 28: Left: Win32.Virlock.Gen.1, Top-right: Win32.Virlock.Gen.3, Bottom-right: Win32.Virlock.Gen.4.

A simple clean procedure is to use the emulation to execute the decryption function. After that, we can grab from memory, using the specified variables, the actual clean file. The starting point of a particular clean file inside the malware is shown in Figure 26.

4. STATISTICS

Figure 27 shows a graphic for the timeline of Win32.Virlock.Gen.1, which is the most widespread version at the moment.

In Figure 28, we see how many systems have been infected since March 2015 for the three most common detections. Almost 39,700 unique files were detected by *Bitdefender* on 148 systems in less than five months. The highest number of infections were detected in Canada – almost 30,000, representing 75% of all infections. We expect a small increase in the next few months as the authors of the malware seem to still be working on it, and a total decrease by the middle of next year, by which time many security products will have solutions for it.

5. CONCLUSIONS

It seems that malware creators are constantly learning from their mistakes and they always find new ways to bypass security products, be it with a small improvement such that their sample will not be detected for a few days, combining technologies that could force certain security products to redesign their engines (due to performance-hits) in order to come up with a feature to successfully detect and clean the malicious application, or forcing security companies to search for better solutions or to give-up by not being able to keep up with damages done by specific malware infections.

Virlock is among the few malware applications which combines different technologies to harden the reverse engineering process and at the same time to make the creators of security products question their technologies. The redesign process of certain engines is not always an easy step, and most of the time this is not a solution. For example, to add some features to emulators, in order to execute unimplemented APIs, to track a certain sequence of generic assembly instructions, or to increase the complexity of search algorithms near to the complexity of strstr(), might result in performance hits which will impact the overall functionalities of the security product. Some designers being inspired in the first place might laugh at the idea that an improvement could be made as a next step inside an already evolved tool, but that is not always the case.

With the advance of malware technologies in the last few years, we find it even harder to revert malware, or to revert the infection process and to restore the system to a clean state. Ransomware using asymmetric encryption algorithms is constantly destroying user-data requiring money to get data back. More than ever, we need methods to automate dynamic analysis and at the same time to extract relevant features from different infections along with improving the prevention techniques. Model-checking and symbolic simulation may be a solution from that point of view, and maybe combining that with time-line analysis and control of a running operating system environment, we might prevent, learn and successfully revert much more complex infections.

There is also a small chance that by using classifiers to extract common vector-features from traces obtained from emulation of such malware, and then dynamically observing the modifications which take place during the infection, one could generate the detection process (which resumes to a search problem in the space of files to be scanned), along with the disinfection process, in just one click.

ACKNOWLEDGEMENTS

This work was co-funded by the European Social Fund through Sectoral Operational Programme Human Resources Development 2007 – 2013, project number POSDRU/187/1.5/ S/155397, project title 'Towards a New Generation of Elite Researchers through Doctoral Scolarships.'