MINERVA
It's malware bedtime

Evasive Techniques: An Introduction

Minerva Research Team

September 2016

Executive Summary

Evasion techniques are a set of malware capabilities that evolved as a result of the need to avoid execution within a hostile environment. Malware will test to detect traces of products belonging to the following four categories, which may pose a potential threat to it:

- Virtual machines and emulators – used for manual and automatic malware analysis.
- Sandbox – used to learn the behavior of suspicious binaries in a controled environment.
- Computer Forensics Tools – used by malware analysts to dissect malware samples.
- Security Products – specific anti-virus and anti-exploitation products.

According to research conducted by Qualys, **almost 89% of malware utilize at least one evasion or anti reverse-engineering technique.** The remaining 11% is significantly less sophisticated and can be easily detected by existing, more basic solutions.

Up to now the computer security industry tried to hide from evasive malware or to prevent it using classic methods. **Minerva's Environment Simulation Technology (EST) creates an environment in which malware refrains from execution, using the malware's own strength against it.**

MINERVA
It's malware bedtime

Table of Contents

What are Evasive Techniques?

For a criminal trying to pull off a a bank heist, busting through the front door with a shotgun pointed at the cashier probably would't be a recipie for success. The clever thief will disguise himself when he spots security guards and cameras, and will attemp to sneak all the way to the vault. Malware authors are much like "traditional" old-school criminals – just as a modern endpoint at an enterprise is a sort of a vault.

Untill fairly recently commiting a crime in cyber space was fairly easy. Defenses were sturdy but far less complex than todays solutions and once bypassed the attacker had full control of his target.

The subsequent adoption of the *defense in depth* concept resulted in the deployment of multiple advanced security solutions. This approach forced bad guys to adapt inorder to keep their buisnesses running as they now needed to battle more than a single "security guard".

Instead of fighting against the army of "security guards" malware authors decided to embrace ***evasive techniques*** as their response to the improved defenses. This approach enables cyber-criminals to pick their fights, bypass some products and avoid others by acting as legitimate software. Evasive techniques can be implemented in many ways but are usually added as a wrapping layer to a malicious binary and at the same time maintaining its original malicious functionality intact. Adding this layer of evasiveness is very common, as alluded to in reports by Qualys which suggest that almost 90% of in-the-wild malware are employing evasive measures.[1] [2]

Our paper will introduce the classes of malware "fears" and classify techniques employed by malware authors in order to fool computer security products and avoid detection. In the final chapter we will also go through the security industry response to the increasingly popular usage of these techniques.

[1] https://blog.qualys.com/securitylabs/2012/07/30/how-malware-employs-anti-debugging-anti-disassembly-and-anti-virtualization-technologies

[2] https://www.blackhat.com/docs/us-14/materials/us-14-Branco-Prevalent-Characteristics-In-Modern-Malware.pdf

The Sum of All (Malware) Fears

Classifying Malware's Phobias

In order to better understand the usage of evasive techniques we need to think like the attackers and understanding what it is that they are afraid of, and why. In order to do so, we grouped those "fears" into four distinctive classes, each containing one genre of products searched for by malware.

In this section of the paper we will clarify what is included in each of these classes and why cybercriminals are terminating their malware once they detect its presence.

Virtual Machines and Emulators

Defenders Use Cases

Analyzing malware often results in infecting the machines to check for behavioral analysis, and so using virtual (or emulated) environment has four key advantages:

- The defender can execute malware conveniently without putting his/her production environment in immediate danger.
- Rolling back infected machines to a clean state is quick, about x200 faster than similar physical setups.
- In some of the products it is possible to take a snapshot of the machine during the execution of a malware sample. This enables uncomplicated simpler analysis and the ability to re-observe its behavior in specific stages.
- It is possible to run many instances of virtual machines in parallel on a single, powerfull, hardware – enabling scalable automation of malware analysis.

Due to these factors we are seeing intensive usage of VMs and emulators for:

- Manual malware analysis by a human analyst.
- Automated analysis on a large scale, mostly as an infrastrucure to sandbox solutions.

What causes an attacker to refrain from attacking?

By detecting the presence of a virtual environment and halting any malicious activities – malware can evade both manual and automatic analysis, killing two birds with one stone. And thereby making tests for virtual environments very popular. Among the products we observed searched in the wild are:

- VMware (multiple products)
- Oracle's VirtualBox
- QEMU
- Xen
- Bochs

Sandboxes

Defenders Use Cases

This class of products enables the execution of malware in a controlled environment, tracing the analyzed sample activity and providing easy access to its metadata. Modern sandbox solutions often rate how malicious a given sample is, and trigger signatures to warn of specific dangerous properties of a sample.

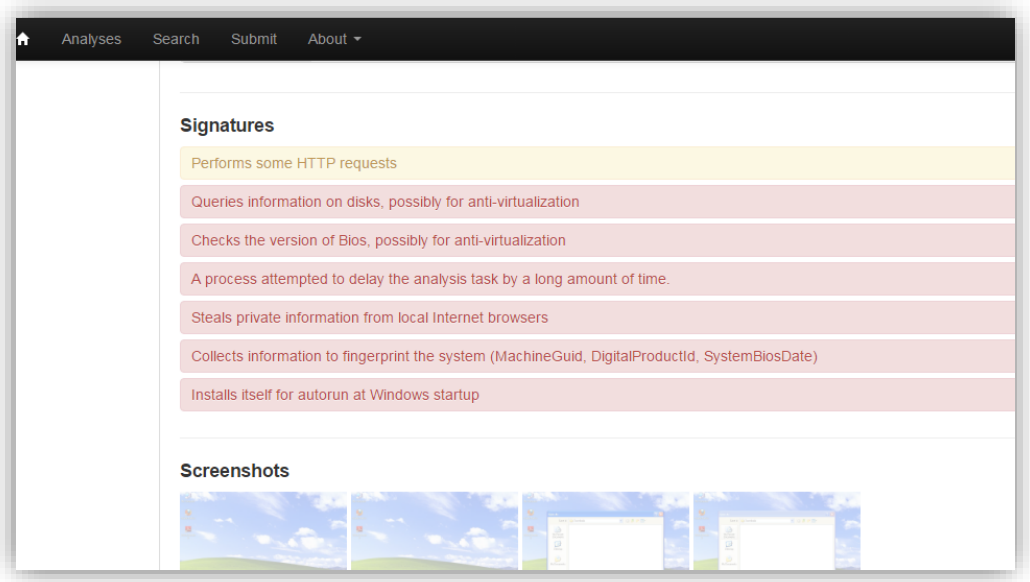


Figure 1: Signatures triggered by a sample analyzed by the Cuckoo sandbox

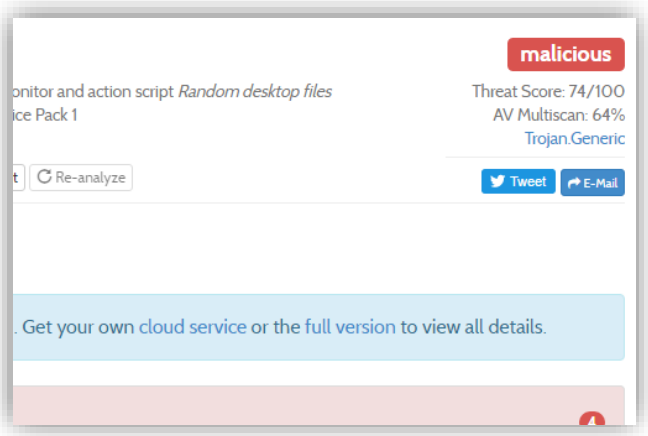


Figure 2: Hybrid-Analysis scoring a malicious sample

A sandbox can be used either as a filter to prevent execution of potentially hostile files in a "real" environment or as a tool in the hands of the malware analyst, summarizing the actions of a given sample.

What causes an attacker to refrain from attacking?

If a malware will displays “good behavior” during its sandboxed execution it won't trigger any signatures and subsequently achieve a low score. This may enable it to slip into the organization if the sandbox is acting as a "gatekeeper", or to fool the malware analyst to think it is legitimate code.

We observed malware searching many sandbox products including:

- Cuckoo
- FireEye
- Threat Stream
- VirusTotal
- Joe's Sandbox
- Hybrid Analysis
- Anubis
- Sunbelt

Some of these products are obsolete, yet, still searched by malware.

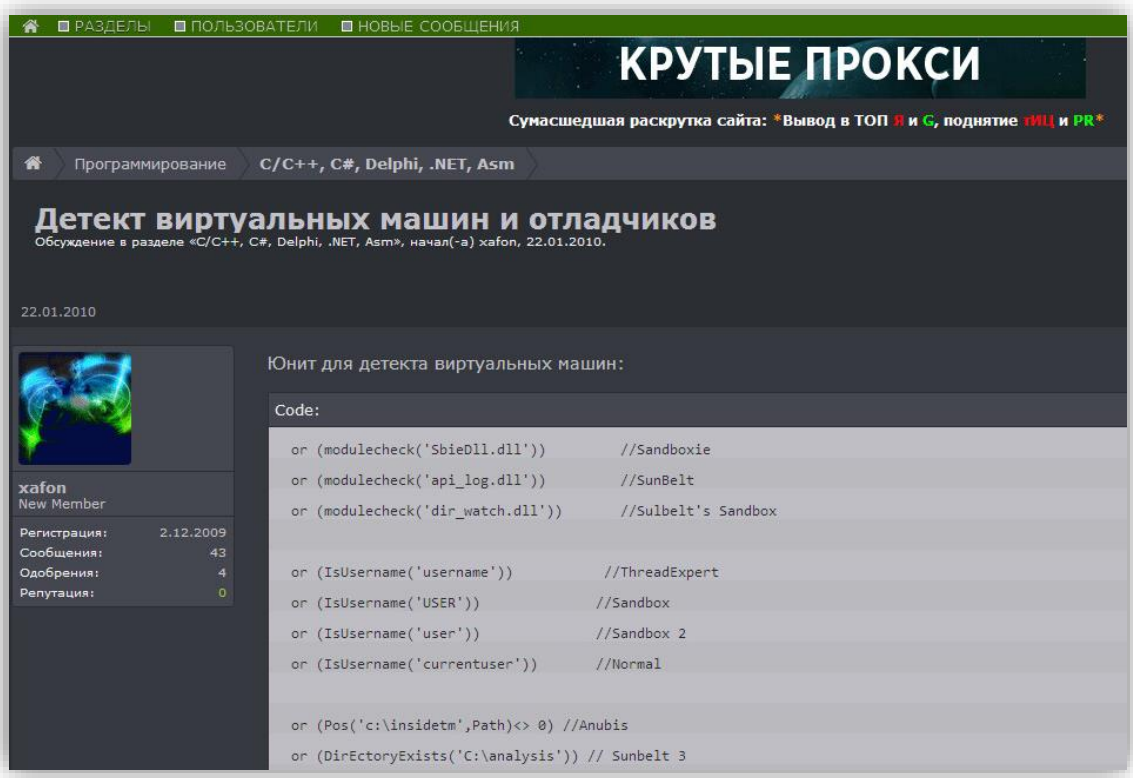


Figure 3:Thread from a russian underground forum discussing VM and sandbox evasion[3]

[3] hxxps://forum[.]antichat[.]ru/threads/172671

Computer Forensics Tools

Defenders Use Cases

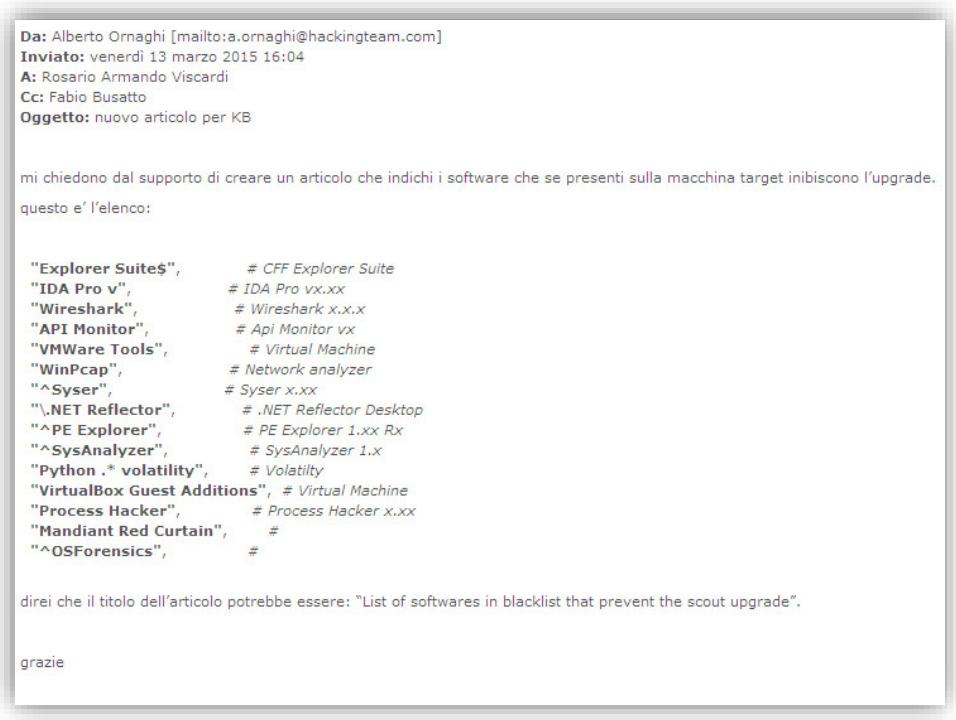
Debuggers, decompilers, network sniffers and dynamic analysis tools are part of the arsenal used by security researchers when they analyze malware. These tools give them the ability to study how a given malware sample behaves, how it is operated and how it can be detected.

Attackers Motivation to Avoid

Once malware is dissected by a flesh and blood researcher any evasive technique **can** be countered. Yet, handling this class of techniques may require advanced knowledge and consume significant amount of time, and if those are not in find – the malware won't be analyzed.

We see a good example of such behavior in Hacking team's leaked data. As a provider of cyber-warfare-class espionage tools, preventing analysis of their advanced assets was so critical to them that they first deployed a basic implant called *Scout*. Once it was deployed it assessed if it is being analyzed, and if so – prevented the upgrade to the advanced *Solider* and *Elite* implants.

In the image bellow you can see internal correspondence regarding blacklisted programs searched by Scout. All but two are used for computer forensics analysis:[4]



Da: Alberto Ornaghi [mailto:a.ornaghi@hackingteam.com]
Inviato: venerdì 13 marzo 2015 16:04
A: Rosario Armando Viscardi
Cc: Fabio Busatto
Oggetto: nuovo articolo per KB

mi chiedono dal supporto di creare un articolo che indichi i software che se presenti sulla macchina target inibiscono l'upgrade.

questo e' l'elenco:

```
"Explorer Suite$",          # CFF Explorer Suite
"IDA Pro v",               # IDA Pro vx.xx
"Wireshark",               # Wireshark x.x.x
"API Monitor",             # Api Monitor vx
"VMWare Tools",            # Virtual Machine
"WinPcap",                 # Network analyzer
"^Syser",                  # Syser x.xx
"\.NET Reflector",         # .NET Reflector Desktop
"^PE Explorer",            # PE Explorer 1.xx Rx
"^SysAnalyzer",            # SysAnalyzer 1.x
"Python .* volatility",    # Volatilty
"VirtualBox Guest Additions", # Virtual Machine
"Process Hacker",          # Process Hacker x.xx
"Mandiant Red Curtain",    #
"^OSForensics",            #
```

direi che il titolo dell'articolo potrebbe essere: "List of softwares in blacklist that prevent the scout upgrade".

grazie

Figure 4: Internal Hacking Team email regarding blacklisted processes

In the body of the email, one of Hacking Team's software architects asks another employee to write a KB article explaining why it is impossible to upgrade their implant on the victim's machine in the presence of the software mentioned above.

[4] hxxps://wikileaks[.]org/hackingteam/emails/emailid/502287

Endpoint Security Products

Defenders Use Cases

In an attempt to defend against sophisticated attacks – premium, old school anti-virus products enhanced with extensive behavioral analysis and anti-exploitation modules were deployed as a defensive measure. In some cases, multiple vendors were used to protect the same endpoint.

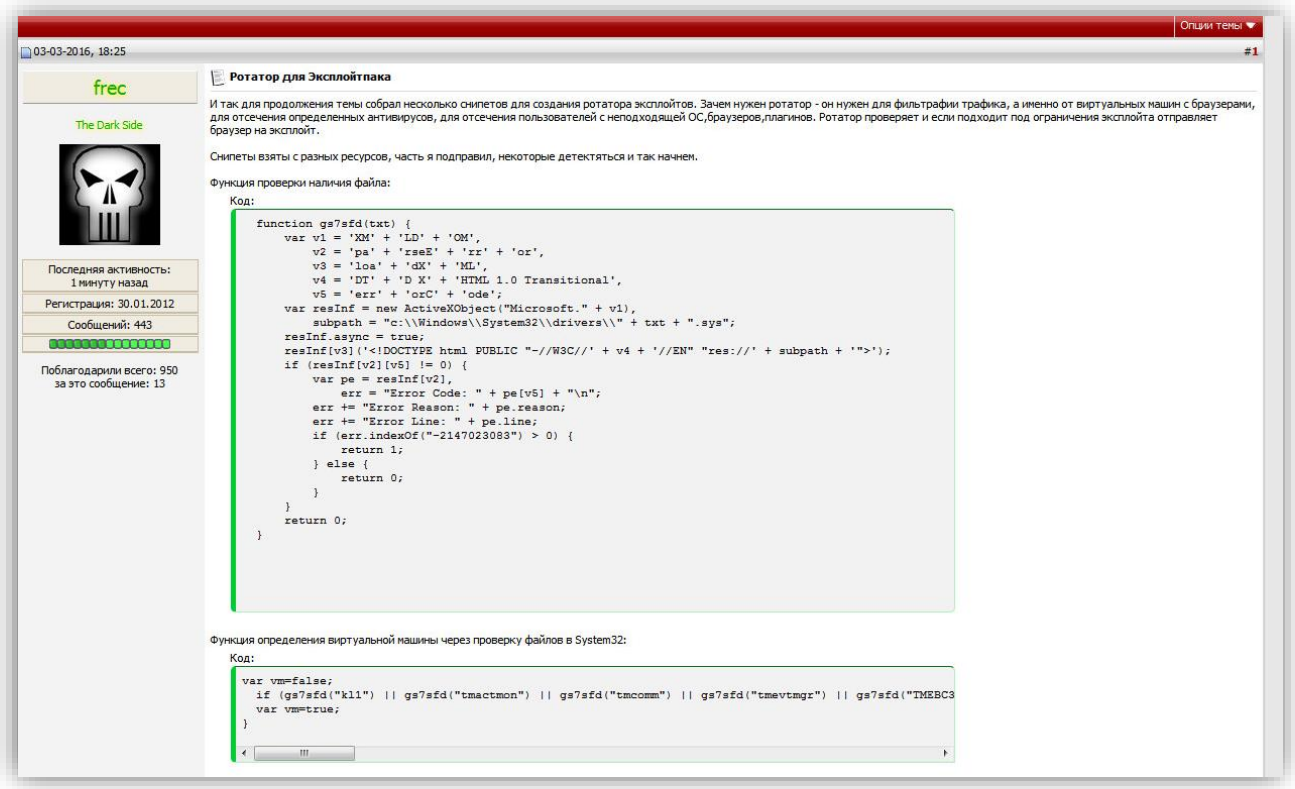
What causes an attacker to refrain from attacking?

Malware authors, often contend with cost/benefit issues when considering whether to attack a potential target or not:

- As software becomes more and more robust – exploits become pricier. Cyber-criminals wish to maximzie the usage of each exploit before it is detected and a patch is deployed. Anti-exploitation products are likely to catch expensive exploits, dettering possible attacks by their very presence.
- Malware authors have underground services providing them the ability to test their malware against up-tp-date copy of most AV and "next-generation" solutions. Avoiding 100% of the products is quite difficult but avoiding most of them is achievable. "Releasing" a malware which won't unpack and terminate in the presence of products detecting it will maximize the time it is out in the wild, generating revenue for the people behind it.

A classic example of a class of malware that avoids security products is exploit kits. The common scheme among exploit kits today is the use a "cheap" exploit, usually CVE-2013-7331 or CVE-2015-2413, to test for the presence of security products. Only if the environment is rendered "safe" it releases the pricy exploits enabling remote code execution (RCE).

Here we have a thread from a shady Russian forum called "FuckAV" (our sincere apologizes for their foul language). The thread contains a code exploiting CVE-2013-7331 and explanations of how this code may help filter out unattractive targets. In this snippet the exploit is used to avoid endpoints with files and folders implying the presence of Kaspersky and TrendMicro products.[5]



Опции темы

03-03-2016, 18:25 #1

frec

The Dark Side

Последняя активность: 1 минуту назад
Регистрация: 30.01.2012
Сообщений: 443

Поблагодарили всего: 950
за это сообщение: 13

Ротатор для Эксплойтпака

И так для продолжения темы собрал несколько снипетов для создания ротатора эксплойтов. Зачем нужен ротатор - он нужен для фильтрафии трафика, а именно от виртуальных машин с браузерами, для отсечения определенных антивирусов, для отсечения пользователей с неподходящей ОС,браузеров,плагинов. Ротатор проверяет и если подходит под ограничения эксплойта отправляет браузер на эксплойт.

Снипеты взяты с разных ресурсов, часть я подправил, некоторые детектяться и так начнем.

Функция проверки наличия файла:

Код:

```
function gs7sfd(txt) {
    var v1 = 'XM' + 'LD' + 'OM',
        v2 = 'pa' + 'rseE' + 'rr' + 'or',
        v3 = 'loa' + 'dX' + 'ML',
        v4 = 'DT' + 'D X' + 'HTML 1.0 Transitional',
        v5 = 'err' + 'orC' + 'ode';
    var resInf = new ActiveXObject("Microsoft." + v1),
        subpath = "c:\\Windows\\System32\\drivers\\" + txt + ".sys";
    resInf.async = true;
    resInf[v3]('<!DOCTYPE html PUBLIC "-//W3C//' + v4 + '//EN" "res://' + subpath + '">');
    if (resInf[v2][v5] != 0) {
        var pe = resInf[v2],
            err = "Error Code: " + pe[v5] + "\n";
        err += "Error Reason: " + pe.reason;
        err += "Error Line: " + pe.line;
        if (err.indexOf("-2147023083") > 0) {
            return 1;
        } else {
            return 0;
        }
    }
    return 0;
}
```

Функция определения виртуальной машины через проверку файлов в System32:

Код:

```
var vm=false;
  if (gs7sfd("kl1") || gs7sfd("tmactmon") || gs7sfd("tmcomm") || gs7sfd("tmevtmgr") || gs7sfd("TMEBC3
  var vm=true;
}
```

[5] hxxps://fuckav[.]ru/showthread[.]php?p=145933

Breaking the Fourth Wall

Swallowing the Red Pill

Malware authors motivation to avoid the four classes previously discussed is clear. But how do they achieve this, what are these evasive techniques? The most primitive of them is simply "sleeping-through" the sandbox, however– this technique is considered obsolete today as it is easily detected by sandboxes and analysts.

It turns out that by querying the OS for the presence of specific artifacts it is possible to determine if the execution environment is hostile or not.

Here we will focus on the artifacts scanned for by malware and will provide examples of each of the categories presented. We will also show how in-the-wild malware searches for them. Please note that we cannot cover all of the possibilities for detection of a given artifact as there are too many to list here – some that are yet to be discovered by malware authors themselves.

Static Artifacts

This is a class of artifacts which are persistent in nature, and thus provide a higher level of certainty while testing for the presence of defensive tools.

Registry Keys and Values

The Windows registry is a hierarchical database storing Windows and other app settings. Almost all installed software creates registry keys and values.

One convention exploited by cyber criminals is that most products add a registry key under *HKLM\SOFTWARE* with its name. This enables them to test for the presence of specific products by going through the keys listed.

Ransomware such as *TeslaCrypt* and *Locky* are good examples of malware searching for AV products by this key. This for example is an execution of a *TeslaCrypt 3* sample[6] on a machine where the registry key *HKLM\SOFTWARE\ESET* is present:

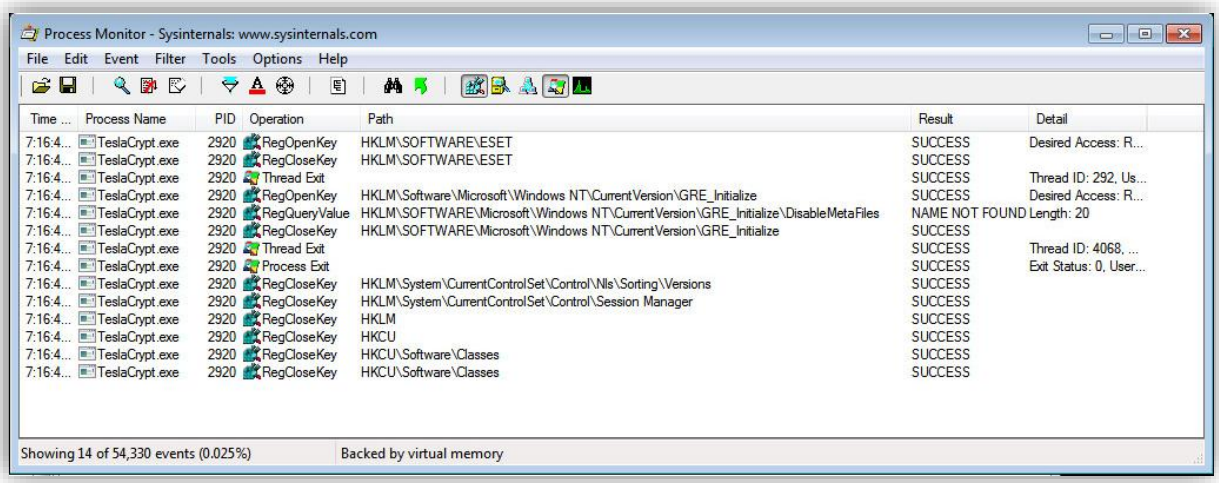


Figure 5:TeslaCrypt 3 terminates after assuming ESET is installed

The sample terminates immediately after it assumes that ESET is installed. The result can be compared with this following trace, demonstrating execution of the same malware on the same machine but without the registry key:

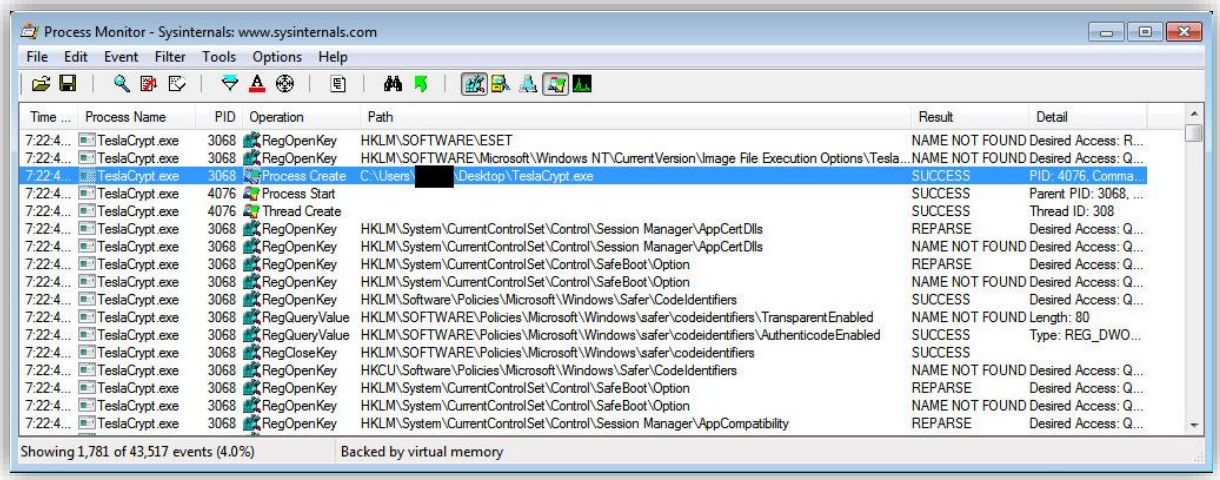


Figure 6: The same sample feels "safe", deploying its next stage

Not only does it continue its execution, but also starts to unpack a secondary stage to a new instance of itself.

[6] SHA-256: ca7cb56b9a254748e983929953df32f219905f96486d91390e8d5d641dc9916d

Files and Folders

Almost any installed program writes itself to the disk. Previously we saw that exploit kits halt the infection process in the presence of specific files – but many other types of malware halt its execution as well.

This for example is the main function of the IRONGATE SCADA targeted malware, note that before it performs malicious activity, it calls the function *detect_VMware*:[7]

```
def main():
    print "Installing scada update..."
    if detect_vmware():
        time.sleep(10)
        print "Installation complete."
        return None

    extracted_file = False
    extract_embedded_file(SHARES_BASE64, SHARES_EXECUTABLE_FILENAME)
    shares = get_shares()
    delete_file_no_errors(SHARES_EXECUTABLE_FILENAME)
    for share_path in shares:
        hot_folder_full_path = find_hot_folder(share_path)
        if hot_folder_full_path:
            if not extracted_file:
                extract_embedded_file(SCADA_BASE64, FILE_TO_COPY)
                extracted_file = True
##              print "FOUND: " + share_path
            move_file_over_shares(hot_folder_full_path)
##            print "File Moved"
    print "Installation complete."
```

Figure 7: IRONGATE main function

In this function it checks whether known VMware drivers are present in the system folder – and terminating itself if they are found:

```
def detect_vmware():
    detected = False

    # try:
    #     with OpenKey(HKEY_LOCAL_MACHINE,
    #                  r"HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0") as key:
    #         print 1
    # except Exception, e:
    #     print(e)

    try:
        with OpenKey(HKEY_LOCAL_MACHINE, r"SOFTWARE\\VMware, Inc.\\VMware Tools") as key:
            detected = True
    except Exception, e:
        pass
##        print(e)

    windows_path = os.environ['WINDIR']
    if os.path.isfile(os.path.join(windows_path, r"system32\drivers\vmmouse.sys")) or \
            os.path.isfile(os.path.join(windows_path, r"system32\drivers\vmhgfs.sys")):
        detected = True

    return detected
```

Figure 8:IRONGATE's VMware detection function

[7] https://www.fireeye.com/blog/threat-research/2016/06/irongate_ics_malware.html

Dynamic Artifacts

This class of artifact contains less reliable artifacts which may exist only temporarily. Generally, searching for these indicators is not as effective as searching for the static ones, but it enables detection of a wider range of products that either hide their static artifacts successfully, or simply don't have traceable files or registry artifacts.

Processes

"A process is an executing program", as Microsoft defines it.[8] When analyzing a malware, either manually or automatically, the analyst uses many programs – not hidden by default. Moreover, security products such as anti-virus usually have multiple processes focused on protecting the machine on the one hand, and displaying a nice user interface on the other.

Here, we see an example for an air-gap-leaping APT called USB Thief[9] searching Kaspersky's AV process that is in charge of the graphical user interface:

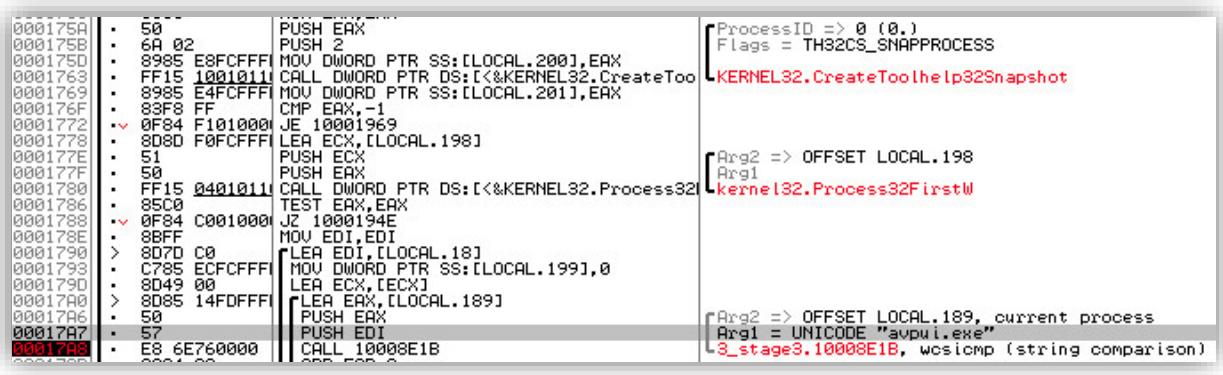


Figure 9:USB Thief testing if avpui.exe process exists

In this case it is implemented by calling the *CreateToolhelp32Snapshot* Windows API to obtain a list of running processes, and then comparing their names against a blacklist of the AV solutions' processes detecting the malware. If any of the processes on the list is found – the malware terminates immediately.

[8] https://msdn.microsoft.com/en-us/library/windows/desktop/ms684841(v=vs.85).aspx
[9] http://www.welivesecurity.com/2016/03/23/new-self-protecting-usb-trojan-able-to-avoid-detection/

Window Handles

Windows processes tend to have a graphical window-like interface. Malware can try and detect this object as demonstrated here by Mark Vincent Yason at Black Hat USA way back in 2007:[10]

```
    push    NULL
    push    .szWindowClassOllyDbg
    call    [FindWindowA]
    test    eax,eax
    jnz     .debugger_found

    push    NULL
    push    .szWindowClassWinDbg
    call    [FindWindowA]
    test    eax,eax
    jnz     .debugger_found

.szWindowClassOllyDbg   db "OLLYDBG",0
.szWindowClassWinDbg    db "WinDbgFrameClass",0
```

Figure 10: detecting OllyDbg and WinDbg by their windows

In this case the *FindWindowA* API will return true if *OllyDbg* or *WinDbg* debuggers are running.

Current User Name

Some sandbox solutions and malware analysts show little creativity when selecting a username for their analysis machine. Malware takes advantage of this lack of creativity and can easily test it against a blacklist prior to unpacking itself. In the example bellow we see a malware call the *GetUserNameA*:

```
001D077C  FF 93 A9 09 00 00   call dword ptr ds:[ebx+9A9]      [ebx+9A9]:GetUserNameA
001D0782  09 C0               or eax,eax
001D0784  74 53               je 1D07D9
001D0786  FF 75 F8            push dword ptr ss:[ebp-8]
001D0789  8D 85 F8 FB FF FF   lea eax,dword ptr ss:[ebp-408]
001D078F  50                  push eax
001D0790  FF 93 A5 09 00 00   call dword ptr ds:[ebx+9A5]      [ebx+9A5]:CharUpperBuffA
001D0796  80 AD F8 FB FF FF 53 sub byte ptr ss:[ebp-408],53
001D079D  75 3A               jne 1D07D9
001D079F  80 AD F9 FB FF FF 41 sub byte ptr ss:[ebp-407],41
001D07A6  75 31               jne 1D07D9
001D07A8  80 AD FA FB FF FF 4E sub byte ptr ss:[ebp-406],4E
001D07AF  75 28               jne 1D07D9
001D07B1  80 AD FB FB FF FF 44 sub byte ptr ss:[ebp-405],44
001D07B8  75 1F               jne 1D07D9
001D07BA  80 AD FC FB FF FF 42 sub byte ptr ss:[ebp-404],42
001D07C1  75 16               jne 1D07D9
001D07C3  80 AD FD FB FF FF 4F sub byte ptr ss:[ebp-403],4F
001D07CA  75 0D               jne 1D07D9
001D07CC  80 AD FE FB FF FF 58 sub byte ptr ss:[ebp-402],58
001D07D3  75 04               jne 1D07D9
001D07D5  C6 45 FC 01         mov byte ptr ss:[ebp-4],1
001D07D9  0F B6 45 FC         movzx eax,byte ptr ss:[ebp-4]
```

Figure 11: detecting sandbox by logged the current user's username

At first glance it seems like a legitimate piece of code, but when inspecting the series of conditional jumps closely, we see that it compares the retrieved value with the hardcoded string "SANDBOX". If this is indeed the username – the malware terminates immediately.

[10] https://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf

Mouse Movement and other "Turing Tests"

If a malware is executed on a machine that has no man behind it – malware authors used to believe that the mouse cursor will stay at the same position.

It is easy to implement such a test, as demonstrated in the following code snippet:[11]

```
int gensandbox_mouse_act() {
    POINT position1, position2;
    GetCursorPos(&position1);
    Sleep(2000); /* Sleep time */
    GetCursorPos(&position2);
    if ((position1.x == position2.x) && (position1.y == position2.y)) {
        /* No mouse activity during the sleep */
        return TRUE;
    }
    else {
        /* Mouse activity during the sleep */
        return FALSE;
    }
}
```

Figure 12: testing cursor position difference

This test can be easily countered by randomly moving the mouse around the screen, as is implemented in many sandboxes. This for example is the Cuckoo function in charge of that tactic:[12]

```
RESOLUTION = {
    "x": USER32.GetSystemMetrics(0),
    "y": USER32.GetSystemMetrics(1)
}

def move_mouse():
    x = random.randint(0, RESOLUTION["x"])
    y = random.randint(0, RESOLUTION["y"])
    USER32.SetCursorPos(x, y)
```

Figure 13: moving the cursor randomly, countering the above evasion technique

It is a cat-and-mouse (pun intended) game, as the bad guys create more complex "Turing tests" to detect whether they are interacting with a human or not (e.g. – popping up a window and waiting for a user to click its button).

[11] https://github.com/a0rtega/pafish/blob/d13b9cb1d07f132b2071ee5d72e786e91b6a20e3/pafish/gensandbox.c
[12] https://github.com/cuckoosandbox/cuckoo/blob/master/analyzer/windows/modules/auxiliary/human.py

Hardware and x86 Tricks

The evasion techniques in the above classes were very straightforward, searching directly for traces of the products we discussed in the first part of the paper. In this section we will focus on more complex techniques which require an understanding of low-level computer mechanisms and direct usage of x86 instructions.

Joanna Rutkowska's Original Red Pill

This technique was introduced by a Polish security researcher named Joanna Rutkowska, possibly one of the most famous methods to detect virtualized environment.[13] She detected a correlation between the location of the interrupt descriptor table (IDT) in memory, and virtualization products.

```
int swallow_redpill () {
        unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
        *((unsigned*)&rpill[3]) = (unsigned)m;
        ((void(*)())&rpill)();
        return (m[5]>0xd0) ? 1 : 0; // return 1 if in VM
}
```

Figure 14: implementation of the Red Pill technique, full explanation is available in the reference

This is a result of the fact that each machine running a VM has only a single register holding the address of the IDT, but at least two OS (host and guest) sharing memory between them. The virtual machine manager (VMM) has to prevent collisions between them, so it allocates the guest IDT to constant addresses, as Rutkowska observed.

[13] https://web.archive.org/web/20070110201418/http://www.invisiblethings.org/papers/redpill.html

x86 Instructions and Low-Level WinAPI Abuse

Since the introduction of the original Red Pill, both researchers and malware authors have found new ways to detect whether the malware they are running is in a VM by using x86 instructions and often undocumented, low-level Windows API functions.

Examples are:

- **Unique CPU Vendor Detection**– when the *CPUID* instruction is executed and the EAX register contains the value 0, the vendor of the CPU will be written as a 12-charechter ASCII string to the EBX, ECX and EDX registers. If not configured explicitly to display another value. VMs and emulators will return distinct unique values such as "VMwareVMware" or XenVMMXenVMM".
- **Timing Attacks** – It is possible to use low level APIs and instructions to observe slight changes in the time it takes to execute a given binary on a physical and on a virtual machine. One method is calling the RDTSC instruction directly multiple times – while on a physical machine the returned values from consecutive calls will have very minor diffrences. VMs however, will typically return values with much higher differnces. However, this test is known to be somewhat inaccurate.[14]
- **Processor Count** – using low-level Windows API enables access to internal OS structures such as the process environment block (PEB). This structure contains sensetive information about the machine's hardware including the number of processors it contains. In some sandbox solutions there is only a single processor, which is very much unlike the majority of modern PCs – providing malware yet another way in which to determine whether or not it's in a sandbox.

[14] http://blog.badtrace.com/post/rdtsc-x86-instruction-to-detect-vms/

Anti-Evasion Techniques

Different Needs and Approaches

Owing to the fact that evasion techniques have become more prevalent, the information security industry needed to react and try to counter-punch. Each category of products has its own motivation to handle a malware implementing evasion techniques, and this part of the paper will cover why and how this is accomplished.

Concealment

Sandboxes aspire to reflect the execution of a sample in a real machine, and if it successfully detects a sandbox environment, the sandbox will have ostensibly failed its mission. Modern sandbox products mask their presence to avoid this "observer effect". Tools such as pafish enable everyone to achieve similar results and are mostly free of cost.

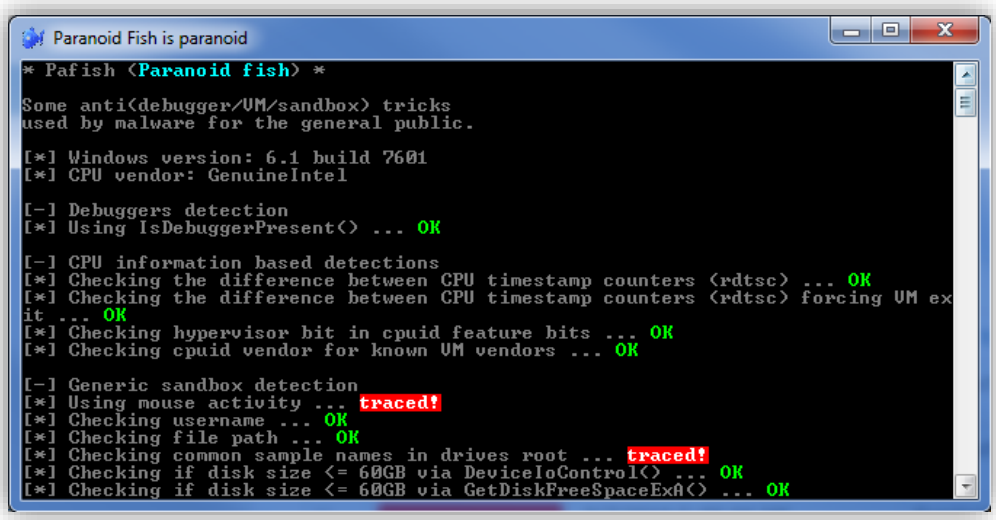


Figure 15: pafish execution, explaining the user how malware may detect its sandbox

Detection

Endpoint Security Products Behavioral Analysis

Anti-virus and NG solutions may classify some evasive techniques as suspicious, enabling them to detect malware by usage of evasive techniques, just like any other type of suspicious behavior. However, the problem of malware authors adapting to new solutions is still present – as they can improve their techniques offline to avoid detection.

Sandbox Products

A good sandbox will not only hide indicators that expose it, but will also send out an alert when a sample tries to perform evasive techniques. Here for example is a Cuckoo signature triggered as a result of a sample searching a registry key typical to a VirtualBox-based VM:[15]

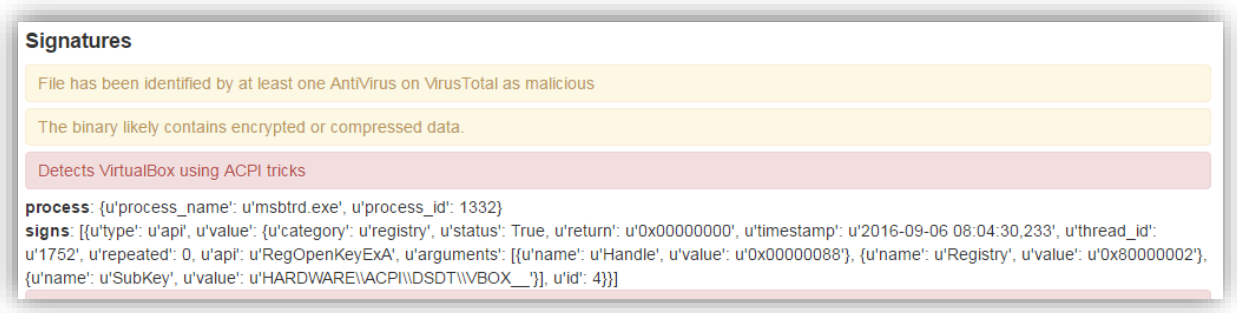


Figure 16: Cuckoo detecting evasive techniques

Prevention

Malware almost by defenition is paranoid, it is trying to avoid detection by employing many advanced evasion techniques which constitute its core strength, so we asked ourselves- why not to use the malware's strength against it? What would happen if we make the malware think that it is in a VM or a sandbox being analyzed by a reseracher at all times– while in fact its already in the targeted endpoint?

The concept of adding artifacts sought by malware is somewhat similar to the classic "vaccination" idea existing since mid-1990's, but at the moment only Minerva Labs' products achieve a similar effect by simulating an environment hostile to malware on each endpoint on an enterprise scale.

[15] https://malwr.com/analysis/ODUyMWRjYzc4Zjg2NDViYWEyM2FhODRhYmRjNTAzNjI/

Conclusions

We started this paper with a comparison between malware authors and traditional criminals. We found that in fact many malware authors are similar to legitimate software companies. They aspire to be a profitable venture either by running a shady operation of their own or like in the *malware as a service* (or *MaaS*) "business model"– and offer a superior product to their clients.

Evasive techniques are just one aspect of malware, but they are unique. At the moment we are witness to a direct arms race between "good and evil", with each new malware adding more and more sophisticated tests to be performed prior to the deployment of a payload. Internal competition between "malware vendors" just increases the numbers of techniques added to malware, as "clientele" often prefer the product offering them the ones containing the highest count of evasion techniques.

This explosion in the number of evasive techniques looks frightening at first sight but it creates new opportunities for defenders as well. Minerva's unique approach of simulating an environment hostile to evasive malware, takes advantage of malware paranoia and forces it to pick its poison. Does it terminate while trying to evade detection or perhaps give up on evasive techniques and get caught. The fact that only a single artifact searched by a malware is required in order to halt its execution makes Minerva's approach very potent and inspires optimism about the future of the war on malware.