Confidentiality and trace in malware samples

Master Thesis

David Álvarez Pérez

January 19, 2020

Advisors: Manuel Fernández Veiga

Máster Interuniversitario en Ciberseguridad, UVigo & UdC

Abstract

Malware sample confidentiality is an unsolved problem which affects the major security products including antiviruses, threat hunting and threat intelligence products[1][2] which, always, in greater or lesser volume, potentially collect confidential elements[3][4][5] (including files, system events, network traffic because of firewall capabilities, etc.). These are not necessary malware, but are also paradoxically called malware samples. This happens because new knowledge cannot come but from other place than unknown and undetected elements. We consider this a critical issue because the capabilities of surveillance of free and paid security products (including those that come built-in the operating system) are huge[6] and growing[7][8] in terms of local system and local area network access. The solution proposed in this work to this confidentiality risk and access logging problem involves the design of a new encrypted malware sample format: Universal Malware Sample Encryption (UMSE). UMSE is a rich format. It can represent: software threats, hardware threats[9], the mixing of both and takes into account very important forgotten aspects like: potential malicious elements context[10][11], life cycle[12] and variety of nature of the elements (not limited to files[13][14][15]) and some other things which finally improve the sample quality acquisition, storage and transport positively impacting all subsequent reverse engineering tasks and users confidentiality.

Contents

Chapter 1

Introduction

Understanding by malware sample the evidence set of any kind of malicious element, hardware and/or software, in a vital cycle state and belonging a context, the present work addresses the malware sample process: acquisition, efficient storage, confidentiality and access logging problem.

The result of this work is an encrypted malware sample format called Universal Malware Sample Encryption (hereafter, UMSE). The development, far from being limited to the sheer format specification, consisted in the following:

- UMSE format documentation and specification.
- A Microsoft Windows Portable Executable[16] dynamic linking library[17] implementing all necessary functions to work with UMSE, for instance: generate an UMSE malware sample, decrypt some sample parts regarding individual parts confidentiality, etc. This library was developed in C/C++ to be used by security products which are mostly developed in these languages (well, although it can be called from other languages).
- A very elementary antimalware agent simulator which acquires system elements demonstrating how easy is to integrate the UMSE dynamic linking library with existing security products.
- An intelligence tool. It consists in a web panel allowing to operate and manage UMSE malware samples stored in a generic database. All malware samples come from the mentioned antimalware agent simulator, which recollects potential malicious elements and sends them to this server. On the other hand, by virtue of UMSE format, each operation over the samples can be logged, as is this case here.
- A Shell, allowing the malware analyst to communicate with the intelligence tool to work with samples. Possible operations are some

of which, at least two, deserve mention: UMSE sample downloading and UMSE sample parts decryption in case that the analyst was sufficiently privileged in comparison to, not only the global, but also the confidentiality level for the specific individual sample parts.

- A tool to quickly and easily generate samples, useful when encryption is not a must.

Below is the status of current malware sample acquisition, storage and confidentiality in order to show the problem addressed in the rest of this work.

Chapter 2

Malware and malware samples

First of all, before addressing the question of what a "malware sample" is, let us analyze what a "malware" is.

The National Institute of Standards and Technologies (NIST) throws the following definition of malware: "Software or firmware intended to perform an unauthorized process that will have adverse impact on the confidentiality, integrity, or availability of an information system. A virus, worm, Trojan horse, or other code-based entity that infects a host. Spyware and some forms of adware are also examples of malicious code."[18]

And RFC4949[19] defines "malicious logic" as "Hardware, firmware, or software that is intentionally included or inserted in a system for a harmful purpose."

A lot of definitions of "malware" do exist[20] but in this work we propose to use this one to remark some key points of malware nature:

> Any element, hardware, software and/or firmware, determined malicious in some context and in a state of its life cycle.

Notice that proposed malware definition has a strong subjective component because it is not possible to say if something is malicious in a fully objective way[21]. Therefore, an element is not inherently malicious. Instead, an element is determined malicious in a state of its life cycle and taking into account the context.

To clarify this idea with an example: the TCP/IP swiss army knife `netcat` does not seem to be designed specifically with malicious purposes so, if context is ignored, it must be classified as goodware. But, installed by an attacker and with the purpose of giving access to a remote computer, this tool must clearly be classified as malware. It is in these cases considered

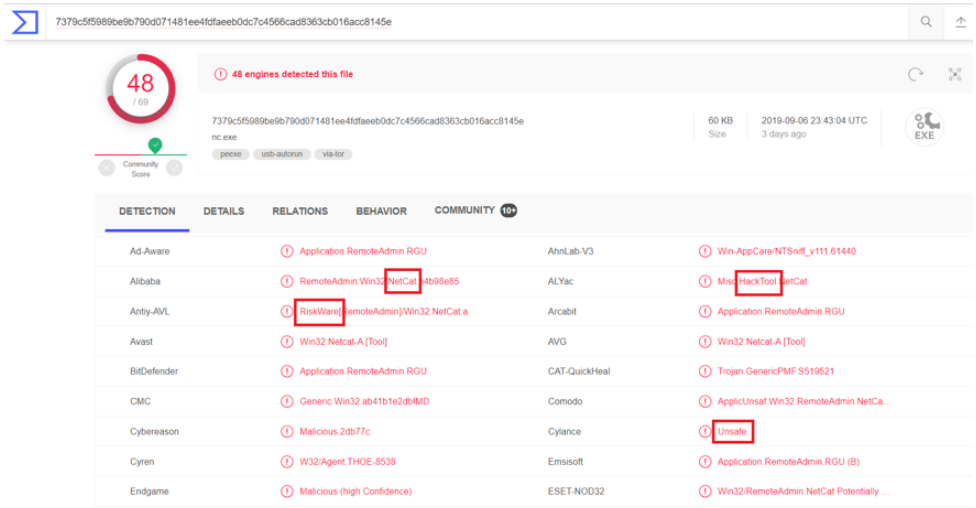


Figure 2.1: An illustration of a VirusTotal analysis of Netcat.

inconclusive (and in some others cases not relevant here) in which industry uses intermediate terms such as “riskware”, “hacktool”, “grayware” or “potentially unwanted program”.

Malware context is absolutely fundamental during analysis[10][11] so, it must to be included in the collected samples. The concept of malware sample is broad, since its nature consists of a composition of miscellaneous elements[13][14][15]. We must, therefore, include different kind of elements (memory dumps, files, pictures and circuit specifications in case of hardware malware, network traffic captures, environment variables, etc.) and to enrich these elements with metadata to indicate the analyst what exactly is any kind of element appended as metadata as necessary.

In contradiction with any definition of “malware”, currently a “malware sample” is any kind of file that, potentially, could be malware. In other words, and to keep it simple, malware samples are not always malware. Let us clarify this issue next.

2.1 The state of the art in malware samples

It is known that intelligence tools simply store files, without any rigor of selection beyond self-tagging by the user who, while it is true, accepts the EULA (End User License Agreement), probably desiring strongly that potential sensitive data are properly encrypted.

In the same way, antimalware solutions also take malware samples from the system of its clients and, with high probability, all the undetected and unknown elements, because new detections cannot come from other place

than the knowledge acquired from the unknown and undetected samples. Therefore, those files are paradoxically also called malware samples. This is not a new thing. For instance, Kaspersky Antivirus has starred the news not long ago on this issue[22][23]. This highlights that pointing to Kaspersky Antivirus as espionage tool by EE.UU and EU is not other thing than to accidentally discover the general lack of current malware sample treatment in general, not by a specific company or product. In other words, a treatment which does not take into account, among other issues not at all negligible, data confidentiality and user privacy.

This incident meant great losses, leading Kaspersky to create a "Transparency Center"[24] and Eugene Kaspersky, CEO of Kaspersky Lab, to publish many releases like this: "Were even willing to meet with any of them and give them our source code to thoroughly review it, as weve got nothing to hide"[25]. Certainly, Kaspersky antivirus is not a spy tool by itself (as happens with any software piece, it depends on the context, it depends on who uses it), you can check the entire source code line by line and you will not notice specifically designed code for espionage, but you will notice the real issue: samples treatment! And no direct actions were taken in this sense by antivirus industry because, if it is not well done, it can reduce the protection rate that, unfortunately, is the only thing the customer demands.

We reproduce as en example two EULA blocks chosen at random because to quote all will be too much extensive and repetitive. These are quite similar for all antivirus companies and products.

12.1 The Software or Support may employ applications and tools to collect Personal Data, sensitive data or other information about Company and End Users (including End Users name, address, e-mail address and payment details), their computers, files stored on their computers, or their computers interactions with other computers (including information regarding network, licenses used, hardware type, model, hard disk size, CPU type, disk type, RAM size, 32 or 64 bit architecture, operating system types, versions, locale, BIOS version, BIOS model, total scanners deployed, database size, system telemetry, device ID, IP address, location, content, McAfee products installed, McAfee components, processes and services information, frequency and details of update of McAfee components, information about third party products installed, extracts of logs created by McAfee, usage patterns of McAfee products and specific features, etc.) (collectively, Data).[a]

[a] https://www.mcafee.com/enterprise/en-us/assets/legal/end-user-license-agreements-en-us.pdf

SECTION B. CONDITIONS REGARDING DATA PROCESSING

Provision of information (if applicable) In order to enhance the protection of information and improve the quality of the Software and services, You agree to automatically provide Kaspersky Lab with the following information of a statistical and administrative nature: information about installed programs, license data, information on detected threats and infections, checksums of processed objects, technical information about the Computer and devices connected to it, information about online activity of the device as well as You agree that such information can be provided to third-party service providers. More information is available at help.kaspersky.com. In order to identify new information security threats and their sources, enhance the operational protection of Users of the Software, and improve the quality of the product, You agree to automatically provide Kaspersky Lab with information specified in the Terms of Use of Kaspersky Security Network. Also, You can activate and deactivate the Kaspersky Security Network service at any time in the Software settings window. You further acknowledge and agree that any information gathered by Rightholder can be used to track and publish reports on security risk trends at the Rightholders sole and exclusive discretion. If you do not wish to provide information to the Kaspersky Security Network service, You should not activate the Kaspersky Security Network service. If service is already activated, you should immediately de-activate the Kaspersky Security Network service.

Kaspersky Lab protects the information received in accordance with applicable governing law and Kaspersky Lab's rules. Data is transmitted over a secure channel.[a]

[a] https://products.s.kaspersky-labs.com/homeuser/kav2020/20.0.14.1085abc/english-INT-0.2007.0/3231373433327c44454c7c4e554c4c/eula_en.txt

As you can read from its own words: sensitive data is collected. You can check not only literature but also the code to verify this.

2.2 Antivirus telemetry

2.2.1 Analysis

Our starting hypothesis is that any antimalware solution is an espionage tool in potential, and that not only files are sent to the server but also intelligence information that can be used to spy the users.

Let us do some reverse engineering on an antivirus product in order to know what kind of information is sent to the company. Let us do this with `Malwarebytes` because telemetry DLL is very easy to identify, this is actually the main reason to chose this antivirus product for investigating the issue.

The file responsible of `Malwarebytes` antivirus telemetry is the

```
TelemetryControllerImpl.dll
```

shown in Figure 2.2. If we take a look at the exported functions, we can see what kind of information is collected (Figure 2.3).

This list of exported function names seems to be self-explanatory, and it reveals without obfuscation what kind of information is sent. It can be summarized as follows.

- Malware information (ransomware is treated in its own specific way).
- Exploits information.
- Client data.
- License data.
- Error information.
- Quarantine information.
- Statistics.

We can also contrast the summary developed above with every of the third endpoint path names of the `Malwarebytes` telemetry exposed API, shown in Figure 2.4. The endpoint path is the following: `https://telemetry.malwarebytes.com/api` and, as an off-topic observation, the development endpoint is also publicly exposed: `https://telemetry.dev.malwarebytes.com/api`.

We can also examine some function of those (`SendMwacReport`, for instance, has an interesting name) in order to understand how are samples treated

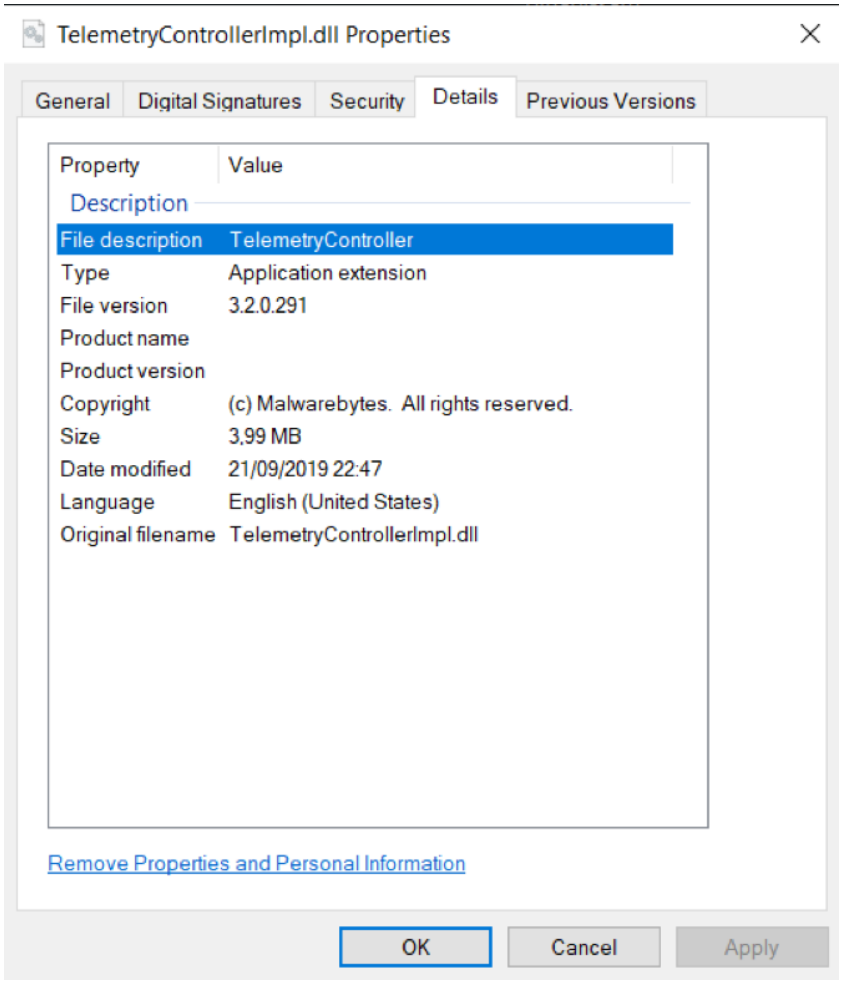


Figure 2.2: Reverse engineering in `Malwarbytes`.

in terms of confidentiality. This function sends a JSON file `mwcstream.json` using Microsoft Winsocks library.

The function responsible of the "send" action is

```
TelemetryControllerImpl::SendMwacReport
```

as shown in the following disassembly listing:

We can take another function from the list and we can see that all information is sent in the same way. It uses also a JSON file (in this case it is named `malwarestream.json`):

And finally the appropriate report sending function for this kind of JSON structure, `TelementryControllerImpl::ReportMalwareStream` in this case:

```
[0x18022653c]> iE
[Exports]
000 0x00026bb0 0x1800277b0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_AddHostToExploitRecord
001 0x00026710 0x180027310 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_AddMwacDetection
002 0x00025c20 0x180026820 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Create
003 0x00025ce0 0x1800268e0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Destroy
004 0x00028230 0x180028e30 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_DisableGameMode
005 0x00028190 0x180028d90 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_EnableGameMode
006 0x00025f00 0x180026b00 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_GetAllowDataCollection
007 0x00026b00 0x180027700 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_GetNewExploitRecord
008 0x00025d60 0x180026960 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Initialize
009 0x00026590 0x180027190 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_MalwareStream
010 0x00026690 0x180027290 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_NewMwacReport
011 0x000262b0 0x180026eb0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Proxy_SetAuthEnabled
012 0x00026090 0x180026c90 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Proxy_SetEnabled
013 0x00026110 0x180026d10 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Proxy_SetHostName
014 0x00026460 0x180027060 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Proxy_SetPassword
015 0x00026240 0x180026e40 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Proxy_SetPortNumber
016 0x00026330 0x180026f30 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Proxy_SetUserName
017 0x000265f0 0x1800271f0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendClientData
018 0x00026c60 0x180027860 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendExploitRecord
019 0x00026650 0x180027250 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendLicenseData
020 0x00027f50 0x180028b50 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendMsgToErrorStream
021 0x00028020 0x180028c20 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendMsgToLogStream
022 0x000268e0 0x1800274e0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendMwacReport
023 0x00026a10 0x180027610 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendOneMwacRecord
024 0x00028340 0x180028f40 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendOneMwacRecordV2
025 0x00027e90 0x180028a90 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendQuarantineAction
026 0x00027ef0 0x180028af0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendQuarantineRefresh
027 0x00026cd0 0x1800278d0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendRansomwareStreamData
028 0x000280f0 0x180028cf0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendUIEvent
029 0x00027230 0x180027e30 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SendUserInteractionData
030 0x00025fa0 0x180026ba0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetAllowDataCollection
031 0x00027ce0 0x1800288e0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetEndpointAgentInstalled
032 0x00026f60 0x180027b60 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetGetComponentPkgVersionCallback
033 0x00027140 0x180027d40 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetGetConfigDataCallback
034 0x00026d80 0x180027980 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetGetLicenseConfigDataCallback
035 0x00026e70 0x180027a70 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetGetRulesDBVersionCallback
036 0x00027880 0x180028480 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetIsAntiExploitEnabled
037 0x00027910 0x180028510 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetIsArwEnabled
038 0x00027ad0 0x1800286d0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetIsMwacEnabled
039 0x000277f0 0x1800283f0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetIsRtpEnabled
040 0x000279a0 0x1800285a0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetIsSelfProtectionEnabled
041 0x00027260 0x180027e60 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetLogCallback
042 0x00027350 0x180027f50 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetMaxLogLevel
043 0x00027dd0 0x1800289d0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetNebulaAttributes
044 0x00027d70 0x180028970 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetNebulaJWT
045 0x00027b60 0x180028760 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetScanConfig
046 0x00027050 0x180027c50 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTelemConfigChangedCallback
047 0x00027490 0x180028090 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTotalArwDetections
048 0x00027760 0x180028360 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTotalExploitsBlocked
049 0x00027400 0x180028000 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTotalMalwareDetections
050 0x000276d0 0x1800282d0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTotalMwacBlocks
051 0x000275b0 0x1800281b0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTotalPumDetections
052 0x00027520 0x180028120 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTotalPupDetections
053 0x00027640 0x180028240 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTotalRTPBlocks
054 0x00027370 0x180027f70 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetTotalScans
055 0x000282d0 0x180028ed0 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_SetUseAlternateURLsOption
056 0x00025e30 0x180026a30 GLOBAL   FUNC    0 TelemetryControllerImpl.dll_Shutdown
```

Figure 2.3: Exported functions.

Our next step is debug `Malwarebytes` in order to see how this JSON looks like. For instance, we can break in some point inside the function `SendOneMwacRecordV2` after analyzing (on demand) a file infector, and see the information. As you can see, apart of malware sample itself, other information is collected separately, unencrypted and stored in a non-standard format. We remark that the computer where these tests have been performed can be easily tracked by checking the unique identifier `machine_id`. Into the binary `.rdata` section a series of WMI queries do exist for machine identification purposes.

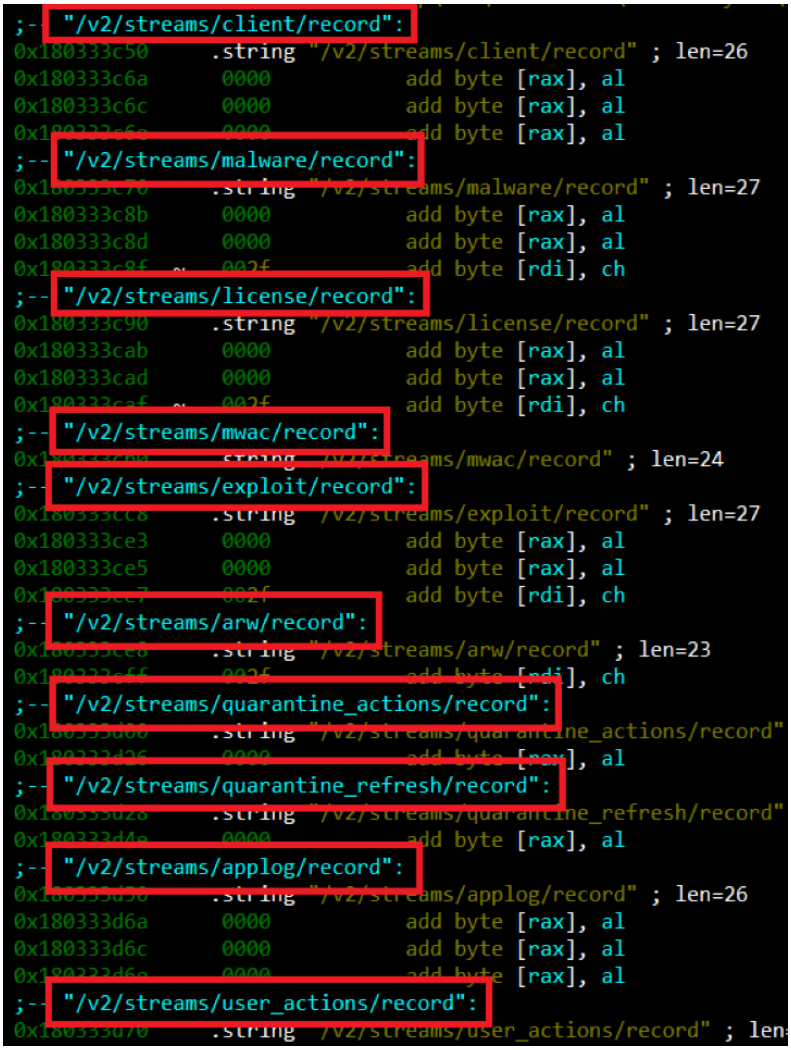
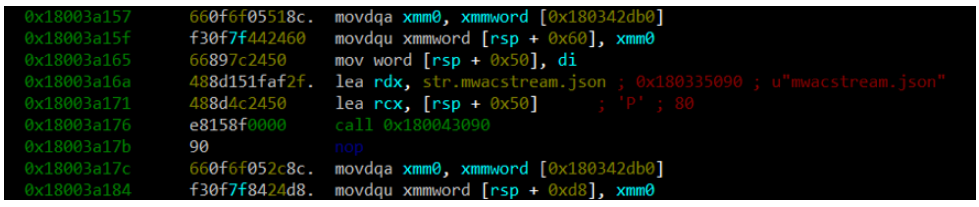
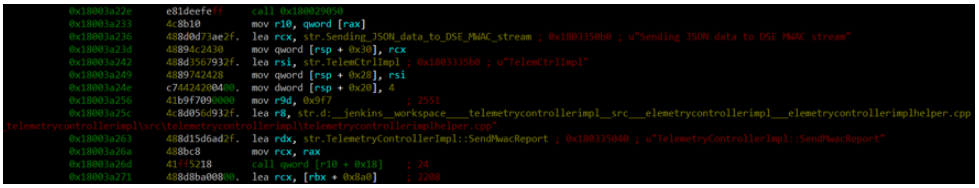
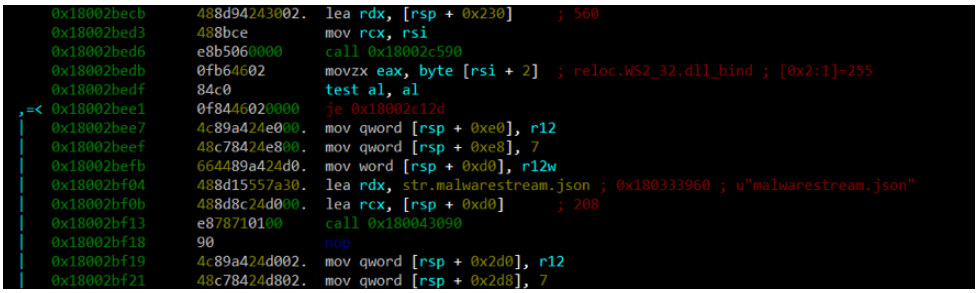


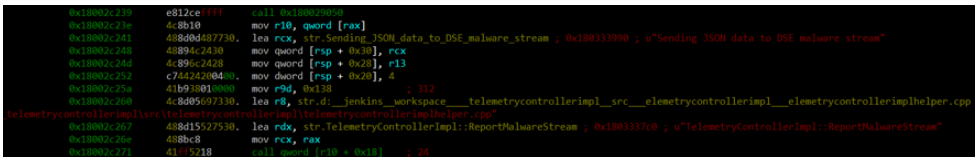
Figure 2.4: Malwarebytes telemetry API.




```
SELECT Index, MACAddress, Name FROM Win32_NetworkAdapter
       where AdapterTypeId=0
SELECT UUID FROM Win32_ComputerSystemProduct
SELECT processorID FROM win32_processor
SELECT SerialNumber FROM Win32_BIOS
SELECT Signature FROM Win32_DiskDrive WHERE Index=%u
SELECT serialNumber FROM Win32_PhysicalMemory
SELECT SerialNumber FROM Win32_DiskDrive WHERE Index=%u
```







Since Microsoft Winsock functions are used for network communications, we can also break in `Send` and `SendTo` functions and show de buffer content before telemetry data are sent (Figure 2.5). It is as easy as stated because the `Malwarebytes` self-defense is not enabled just after installing the product, a really hilarious security bug which can be used to attack the debugger without bypassing any kind of protection.

The file responsible of Malwarebytes antivirus cloud functionality is the `CloudControllerImpl.dll`.

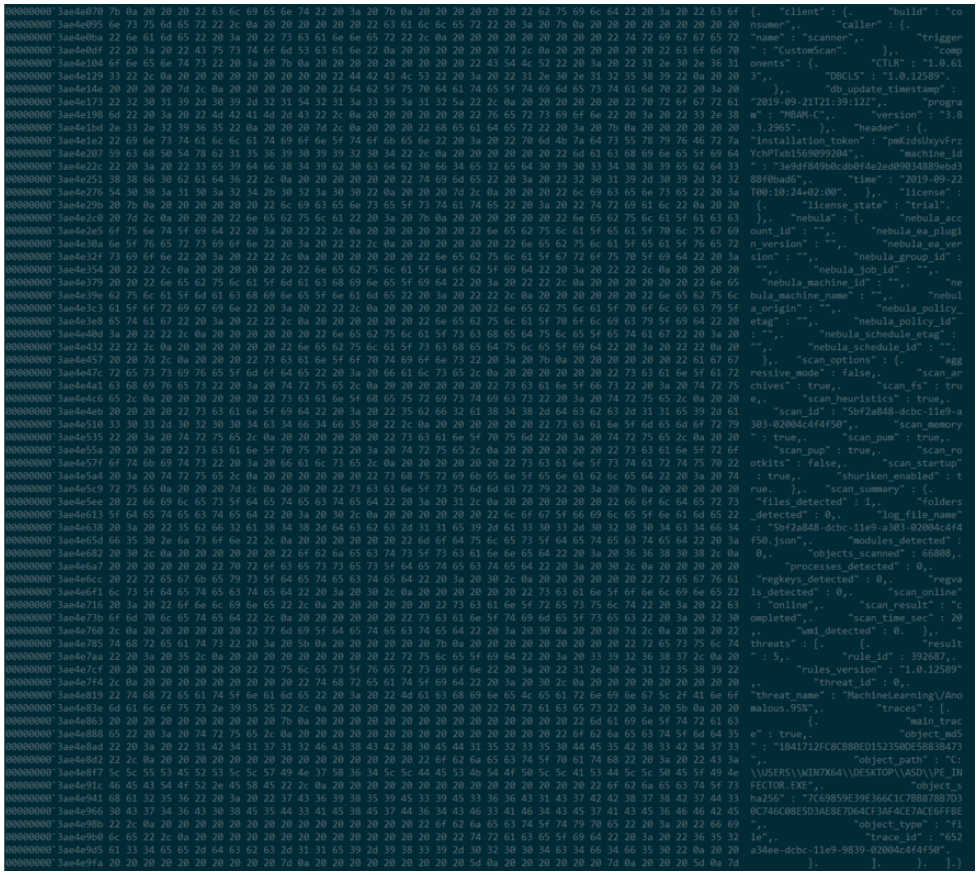
Let us follow the same analysis steps with this file. Now, export listing looks as follows:

There are five functions at the very end of the screenshot named starting by the prefix `Submit`. Those functions are responsible of submitting files and memory chunks to the `Malwarebytes` cloud storage, but those functions are only called when upgrading to the premium version of the product[28].

The reader should know what “exploit”, “ransomware”, and “rootkit” mean, but there are two function names which maybe seem a little stranger:

`SubmitDopplegangDetection` and `SubmitShurikenDetection` because they are `Malwarebytes` specific terms. The first one, `SubmitDopplegangDetection`, as you can see in the following picture, is used to send “scam” detections:

But `SubmitShurikenDetection` is much more interesting for us, because it is used to send heuristically detected samples, meaning, by definition, that



false positives will absolutely happens (goodware files are potentially sent to the cloud storage).

All of this information is submited to the following endpoints:

```
https://bactem-staging.mwbsys.com/files
https://bactem-staging.mwbsys.com/files
https://blitz.mb-cosmos.com/
https://static-blitz.mb-cosmos.com/
https://blitz.mb-cosmos.com/
https://static-blitz.mb-cosmos.com/
```

If the reader is interested about where those files are stored, we can also answer this question. These files are stored in the server referenced by the following IP address: `3.229.68.76` and it corresponds to Amazon Data Services NoVa:

This is an important point because the code is developed using Amazon S3 API[29]. The following is a little fragment of the `Shuriken` sending function:

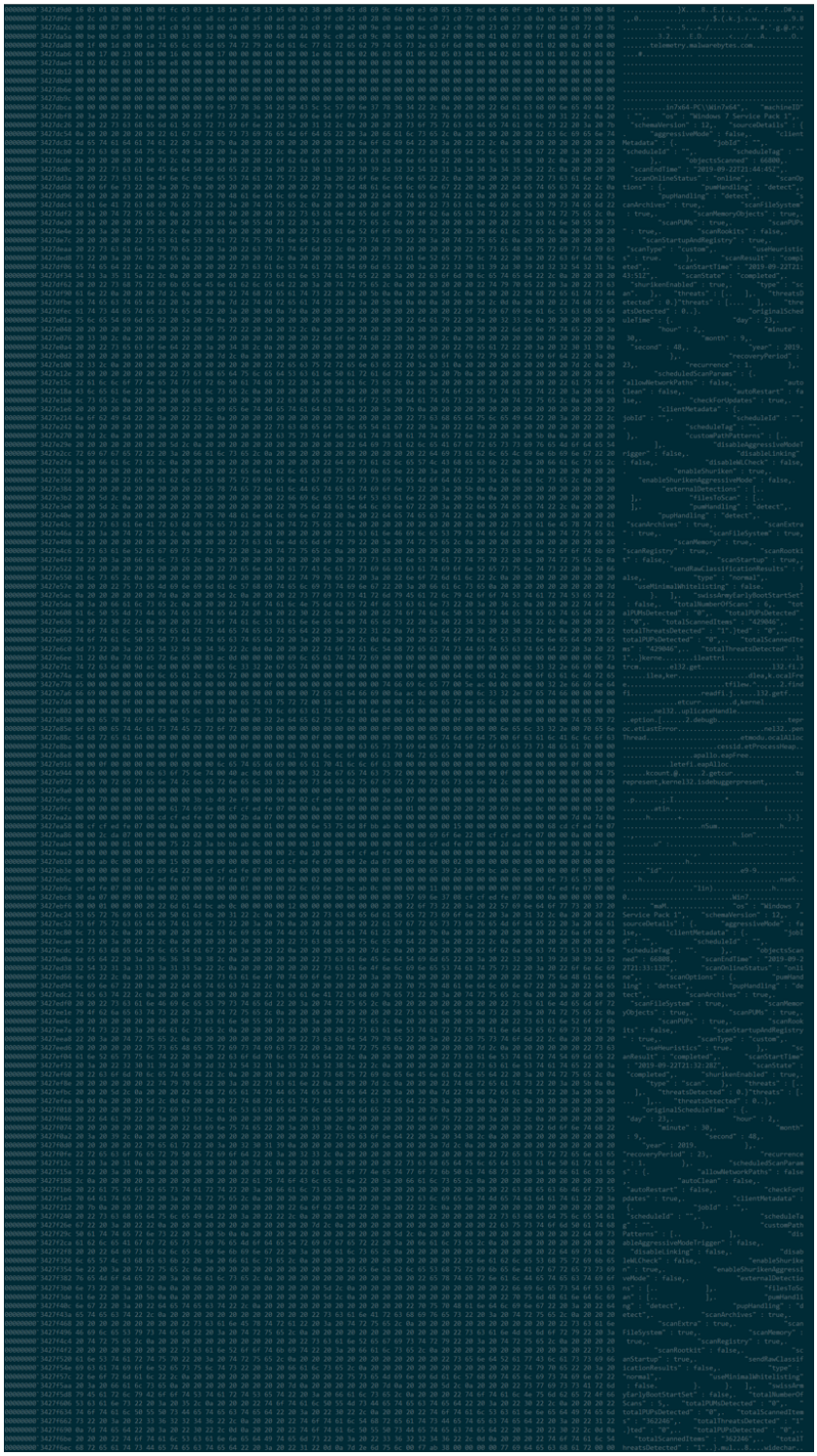
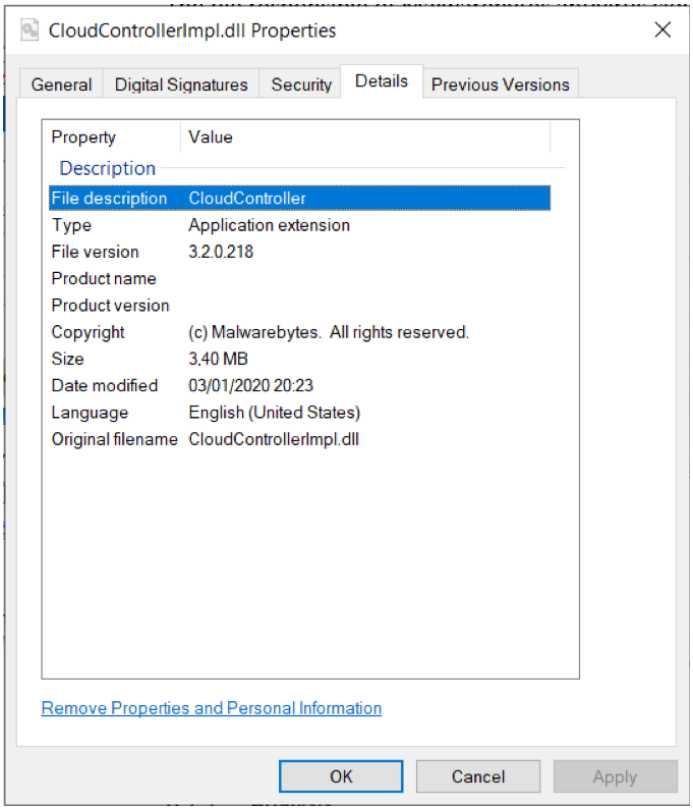
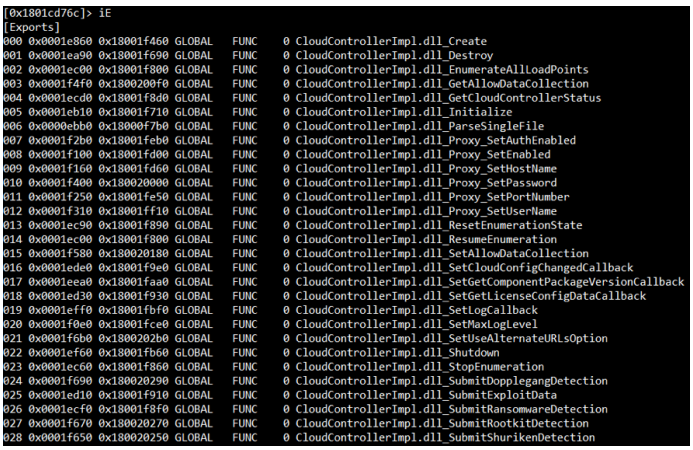


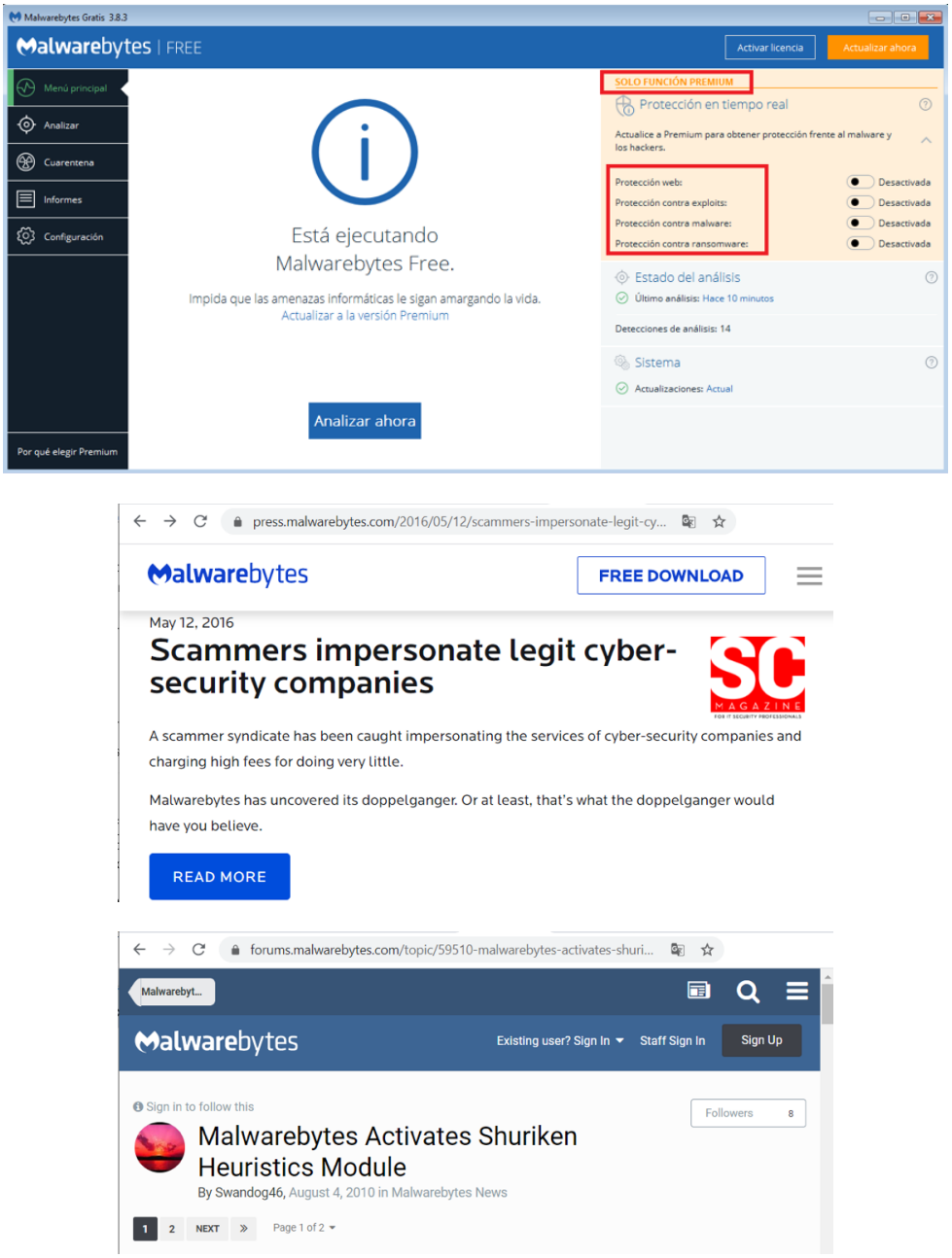
Figure 2.5: Debugging `Malwarebytes`.





As you can see, `x-amz-meta-payloadtype` is the "payloadtype" custom metadata parameter prefixed as specified by Amazon S3 API, and the rest means that a `Shuriken` sample submission is happening.

Thus, we have identified the mechanism used to send samples and telemetry data to the `Malwarebytes` cloud server. Another interesting thing is to know how file and memory samples are chosen by this product in order



to keep them in the server. Following the natural analysis flow, some interesting functions are located into `MBAMService.exe` which finally relies on `CloudControllerImpl.dll` where all the hard job takes place.

There are a lot of callbacks into this binary image. Two of them make reference to cloud submission and telemetry submission, specifically:

If reader is really interested about how `Malwarebytes` heuristically chooses

IP Lookup Results for:

3.229.68.76

IP	3.229.68.76	ASN No.	14618

IP Location via DB-IP (PRODUCT: API, REAL-TIME)

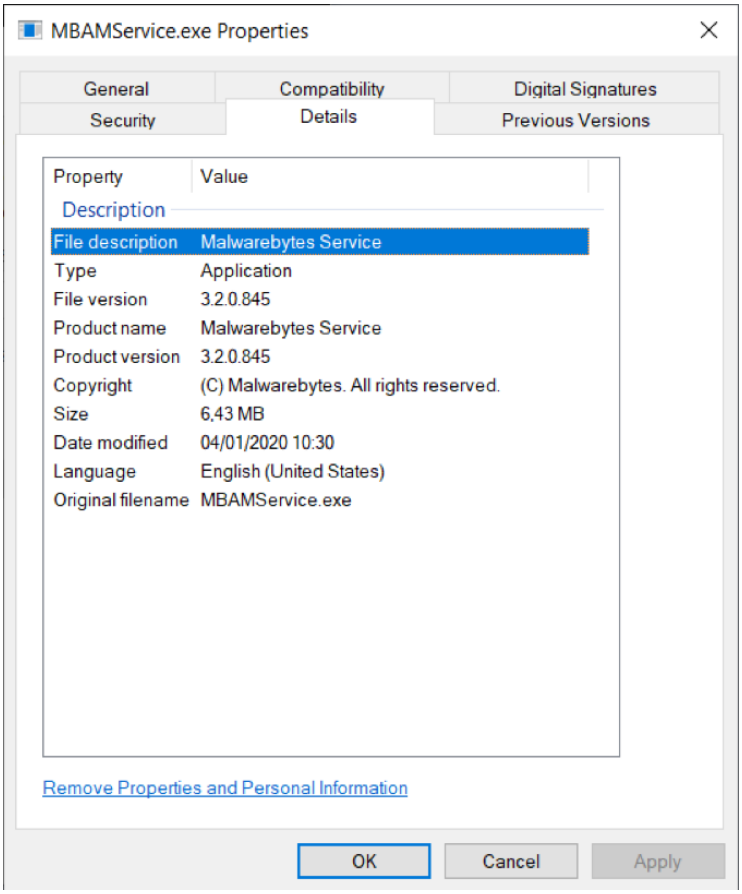
City	Ashburn	State	Virginia	Country	United States
ISP	Amazon Technologies Inc.	Latitude	39.0438	Longitude	-77.4874
Organization	Amazon Data Services NoVa	Is proxy	NO	Is crawler	NO
Threat Level	low				

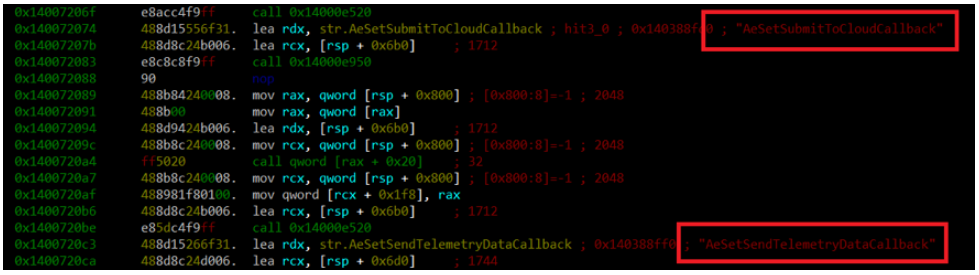
```
[0x18002da9f]> pd
          ;-- hit0_0:
          0x18002da9f      488d158ad229.  lea rdx, str.x_amz_meta_payloadtype ; 0x1802cad30 ; "x-amz-meta-payloadtype"
          0x18002daa6      488d8d300100.  lea rcx, [rbp + 0x130]     ; 304
          0x18002daad      e84e9dfeff     call 0x180017800
          0x18002dab2      41b808000000   mov r8d, 8
          0x18002dab8      488d15c9d229.  lea rdx, str.shuriken      ; 0x1802cad88 ; "shuriken"
          0x18002dabf      488d8dd00000.  lea rcx, [rbp + 0xd0]      ; 208
          0x18002dac6      e8359dfeff     call 0x180017800
          0x18002dacb      488d95d00000.  lea rdx, [rbp + 0xd0]      ; 208
          0x18002dad2      4883bde80000.  cmp qword [rbp + 0xe8], 0x10 ; reloc.WS2_32.dll_recv ; [0x10:8]=-1
          0x18002dada      480f4395d000.  cmovae rdx, qword [rbp + 0xd0]
          0x18002dae2      4c8b85e00000.  mov r8, qword [rbp + 0xe0] ; [0xe0:8]=-1 ; 224
          0x18002dae9      488d8d700100.  lea rcx, [rbp + 0x170]     ; 368
          0x18002daf0      e80b9dfeff     call 0x180017800
          0x18002daf5      41b806000000   mov r8d, 6
          0x18002dafb      488d154ed229.  lea rdx, str.sample        ; 0x1802cad50 ; "sample"
          0x18002db02      488d8d500100.  lea rcx, [rbp + 0x150]     ; 336
```

```
0x18002dd3f      488d8db00000.  lea rcx, [rbp + 0xb0]      ; 176
0x18002dd46      4883bdc80000.  cmp qword [rbp + 0xc8], 0x10 ; reloc.WS2_32.dll_recv ; [0x10:8]=-1
0x18002dd4e      480f438db000.  cmovae rcx, qword [rbp + 0xb0]
0x18002dd56      4c8b10         mov r10, qword [rax]
0x18002dd59      48894c2438     mov qword [rsp + 0x38], rcx
0x18002dd5e      488d0d3bca29.  lea rcx, str.Uploading_file:__hs ; 0x1802ca7a0 ; u"Uploading file: %hs"
0x18002dd65      48894c2430     mov qword [rsp + 0x30], rcx
0x18002dd6a      4c896c2428     mov qword [rsp + 0x28], r13
0x18002dd6f      c744242004..   mov dword [rsp + 0x20], 4
0x18002dd77      41b912080000   mov r9d, 0x812             ; 2066
0x18002dd7d      4c8d050cb129.  lea r8, str.d:__jenkins__workspace____cloudcontrollerimpl__src__cloudcontro
c\cloudcontrollerimpl\cloudcontrollerimplhelper.cpp"
0x18002dd84      488d1585cd29.  lea rdx, str.CloudControllerImplHelper::UploadShurikenData ; 0x1802cab10 ;
0x18002dd8b      488bc8         mov rcx, ra
0x18002dd8e      41ff5220       call qword [r10 + 0x20]     ; 32
0x18002dd92      498b8ef80400.  mov rcx, qword [r14 + 0x4f8] ; [0x4f8:8]=-1 ; 1272
0x18002dd99      488b01         mov rax, qword [rcx]
```

files to be sent to the cloud storage, remember that when using heuristics, by definition, no categorical conclusions are possible so false positives will happen. Therefore, confidential files in addition to potential malware embedding confidential data could be sent to the server. It is recommended to the restless reader to disassemble `CloudControllerImpl.dll` by his/her own to perform a further analysis.

At this point, it is necessary to install `Malwarebytes` Premium in order to investigate the file submission features. We obtained a trial[30] license for a limited period. In this Premium Trial version, real-time features are available. We noticed that this `Malwarebytes` version has more sample submission routines, and identified the following sample submission exports in the





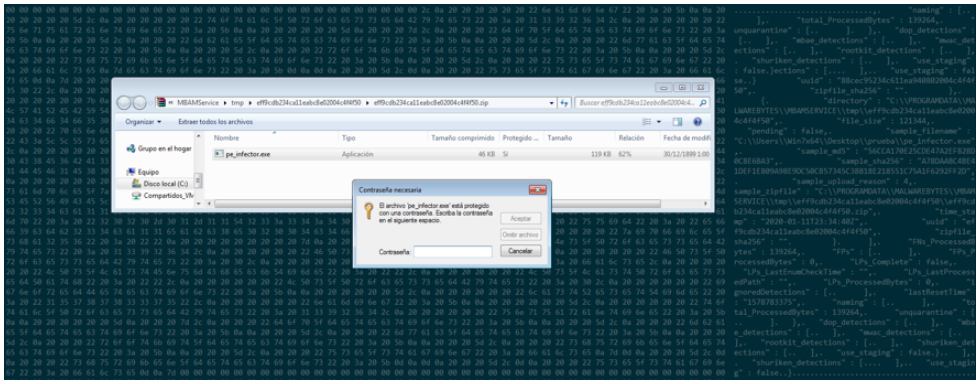
`CloudControllerImpl.dll` library file:

1. `SubmitDDSSample`
2. `SubmitDopplegangDetection`
3. `SubmitExploitData`
4. `SubmitMWACDetection`
5. `SubmitQuarantineRestoreItem`
6. `SubmitRansomwareDetection`

7. `SubmitRootkitDetection`

8. `SubmitShurikenDetection`

The first thing we are interested in to investigate is how samples are submitted. To this end, we executed a portable executable file infector and broke into the `send` function of `WS2_32.dll` to see how the buffer content looks like.



As you can see in the JSON structure of the previous image, the sample file is stored into the following location:

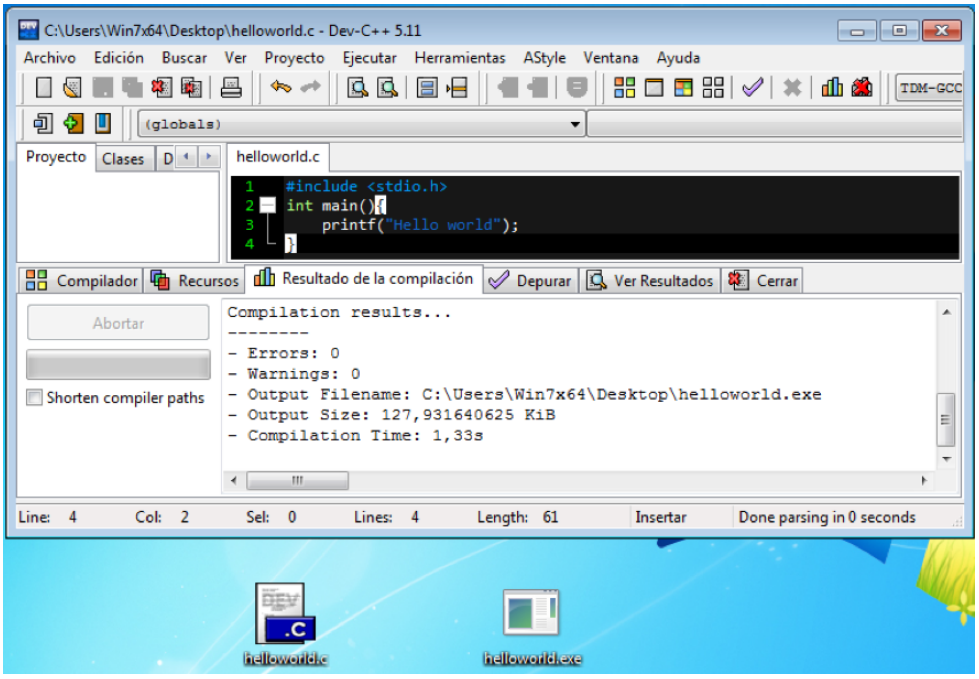
```
{C:\PROGRAMDATA\MALWAREBYTES\MBAMSERVICE\tmp\{hash_sha256}\{hash_sha256}.zip
```

This is a PKZIP file encrypted with the typical malware sample password: "infected"[31]. And it will be sent to the Amazon server

```
btoc-samples-prod.s3.amazonaws.com.
```

The hypothesis that files are submitted is confirmed. The other thing we are interested in to investigate is if `Malwarebytes` indiscriminately submits all files or not. Since a "hello world" program should never lead to a false positive, if it is submitted, it can be assumed that submissions are done indiscriminately.

The experiment result indicated that, when `helloworld.exe` was executed, some information was sent to `Malwarebytes` by using `WS2_32.dll` library `send` function with the call flow coming from somewhere inside `RTPControllerImpl.dll`. This library contains the following strings:



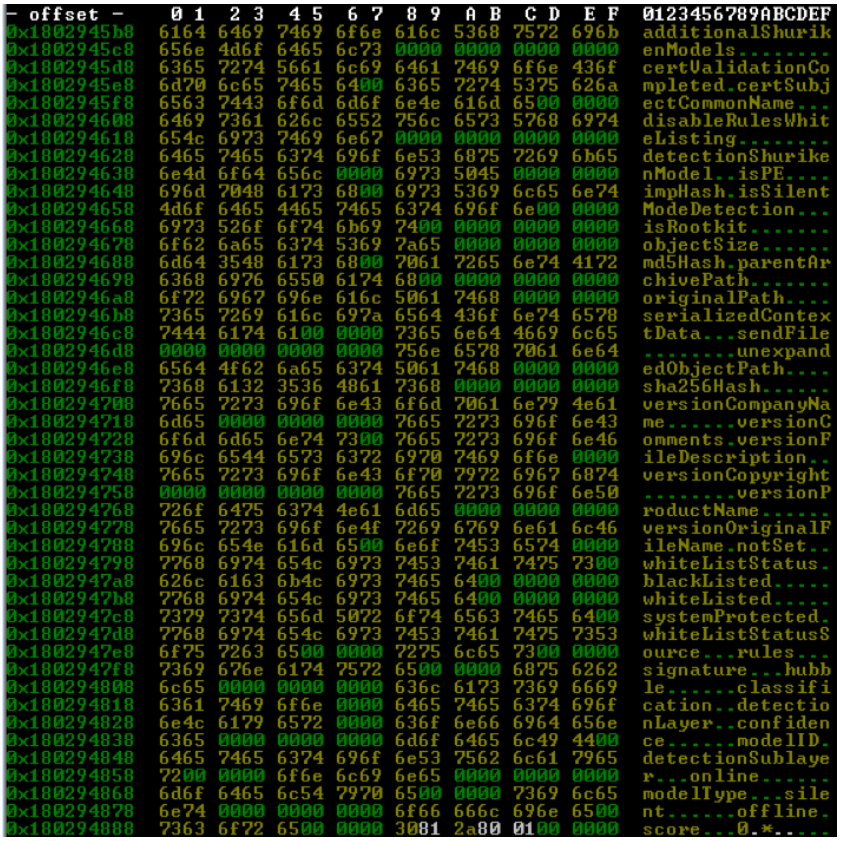
A function of `RTPControllerImpl` uses it to generate the JSON. Experiment JSON looks as in Figure 2.6. So, `Malwarebytes` Premium sends file samples only if it suspects a file could be infected (a `sample_upload_reason` field do exist into the JSON structure). If the file is not a suspicious one, `Malwarebytes` Premium sends information about the file (like the file path and something like this) but not the file content itself. Anyway, subjectively, in our opinion, executable files leak a lot of information about the user behavior.

2.2.2 Conclusions

The reverse engineering of the `Malwarebytes` antivirus products reveals that these are not especially intrusive. They are classical antiviruses with cloud features which send suspicious files and telemetry to the cloud server just for purposes of comparison.

Our analysis indicates that some files (but, with congratulations to `Malwarebytes`, not an indiscriminate massive volume of goodware ones) are sent to the company's cloud server powered by Amazon.

Sample submission is done by using a PKZIP file protected with the typical malware sample password: "infected"[31]. And metadata information (telemetry) is sent separately using a JSON format. On the other hand, the most important thing is that accessed goodware information is sent, the user is unambiguously identified and some information is collected apart of the



sample file itself. If such collected information were accessed, for instance, by a third party like an unethical employee or a government intelligence agency (which maybe collaborates with the antivirus company and could take advantage of this fact) they could track a specific user (or users, in general) and combine this information with other databases (including another antivirus products)[6]. `Malwarebytes` could substantially improve its system by removing both, the compressed samples system and the telemetry data. Instead, it could progressively add sample and telemetry support to the UMSE dynamic linking library and call to it before submitting samples.

2.3 Cyber threat hunting telemetry and samples submission

2.3.1 Analysis

Cyber threat hunting is defined as follows: "the process of proactively and iteratively searching through networks to detect and isolate advanced threats that evade existing security solutions".

.{...{. "detectedObjects" : [. {. "addit
ionalShurikenModels" : [. {. "
classification" : "Normal",. "confidence"
: 0,. "detectionLayer" : "2019011701",.
"detectionSublayer" : "22000",.
"modelID" : "0",. "modelType" : "sil
ent",. "score" : 0. }.
],. "certSubjectCommonName" : "",. "ce
rtValidationCompleted" : false,. "ddsSigFileVer
sion" : "",. "detectionShurikenModel" : {.
"classification" : "Normal",. "confid
ence" : 0,. "detectionLayer" : "2018031101",
. "detectionSublayer" : "22000",.
"modelID" : "0",. "modelType" : "online",.
"score" : 0. },. "disableHub
bleWhiteListing" : false,. "disableRulesWhiteLi
sting" : true,. "disableSignatureWhiteListing"
: false,. "impHash" : "CA9E505E8575285BA20E32E6
C80C12F7",. "isDDS" : false,. "isDopple
ganging" : false,. "isPE" : false,. "is
PUP" : false,. "isRootkit" : false,. "i
sShuriken" : true,. "isSilentModeDetection" : f
alse,. "md5Hash" : "",. "objectPath" :
"C:\\Users\\Win7x64\\Desktop\\helloworld.exe",.
"objectSize" : 131002,. "objectType" : "file",
. "originalPath" : "C:\\Users\\Win7x64\\Desktop
\\helloworld.exe",. "parentArchivePath" : "",.
"ruleID" : 0,. "ruleString" : "",.
"rulesVersion" : "2.2.3",. "sendFile" : fal
se,. "serializedContextData" : "",. "sh
a256Hash" : "BD5B930A7C1965991A52F2EDEE199A98D64A0DC81F
007716B10D6E5197D688CC",. "threatID" : -1,.
"threatName" : "",. "unexpandedObjectPath"
: "%DESKTOP%\\helloworld.exe",. "useDDA" : fal
se,. "versionComments" : "",. "versionC
ompanyName" : "",. "versionCopyright" : "",.
"versionFileDescription" : "",. "versionO
riginalFileName" : "",. "versionProductName" :
"",. "whiteListStatus" : "notSet",. "wh
iteListStatusSource" : "none". }.],. "scanTyp
e" : "threatScan",. "schemaVersion" : 11,. "sourceC
ontrollerType" : "rtp",. "sourceControllerVersion" :
"2.2.3".}..n.7.x.6.4.\..E8..... 8.....@r{.....S.\.U.T.

Figure 2.6: Sample of the JSON experiment.

In practice, cyber threat hunting means to capture as many events as possible, correlate them and send reports of them all the time. All the magic can be summarized in one sentence: everything can be detected if everything is real-time reviewed.

For instance, Windows Defender Advanced Threat Protection captures the following information[32], as shown in Figure 2.7:[1]

1. Alerts on Microsoft Defender Security Center.
2. Machine information, including OS information.

[1] https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/advanced-hunting-schema-reference

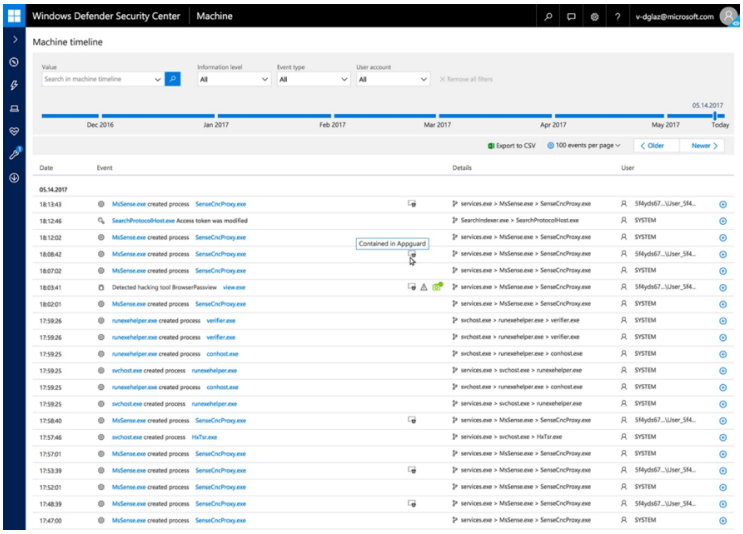


Figure 2.7: A screenshot of WindowsDefender.

3. Network properties of machines, including adapters, IP and MAC addresses, as well as connected networks and domains.
4. Process creation and related events.
5. Network connection and related events.
6. File creation, modification, and other file system events.
7. Creation and modification of registry entries.
8. Sign-ins and other authentication events.
9. DLL loading events.
10. Multiple event types, including events triggered by security controls such as Windows Defender Antivirus and exploit protection.

The idea is the same for all products. Event correlation is an important feature. Check Figure 2.8, as an example, taken out of from the `Carbon Black`[33] tool.

2.3.2 Conclusion

It is hard to imagine a more aggressive kind of security tools in terms of user data confidentiality. More detections but unjustifiably much less confidentiality. You can check pictures publicly available in Google of this kind of tools, most of them will be carefully chosen by the manufacturer (meaning that the aggressive behavior will be as hidden as possible) but if one watches the dashboard containing event logs of those tools, one will soon realize how powerful they are in terms of surveillance. You will see a steady stream of

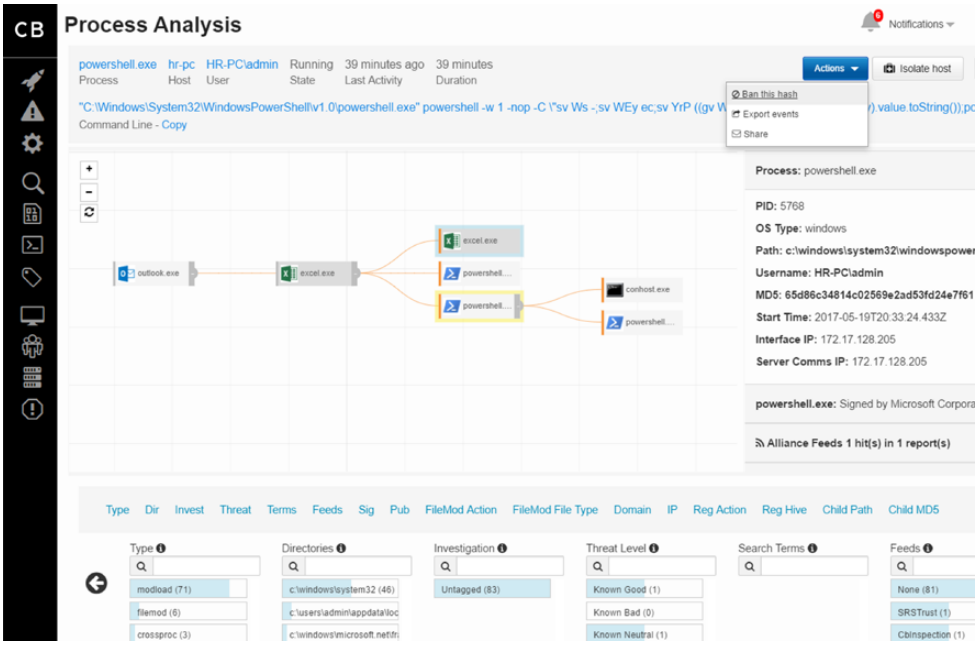


Figure 2.8: Source: https://www.carbonblack.com/wp-content/uploads/2017/04/BanHash-B.png

events unrelated to malware. Threat hunting tools could also substantially improve their system by adding events, files, processes, registry keys and support for system elements into the UMSE dynamic linking library, and by calling to its API before submitting collected data.

2.4 Operating system telemetry and samples submission

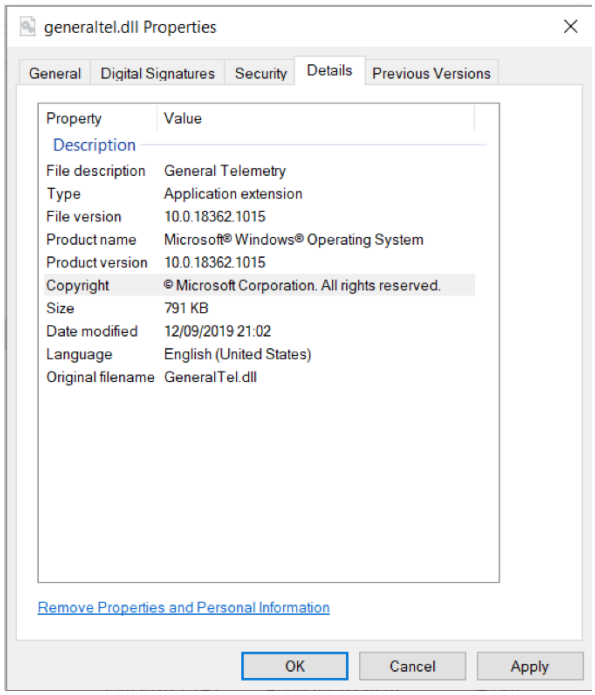
2.4.1 Analysis

Next, we want to explore what happens if there are no additional antivirus software installed in the computer. Must the user be worried about security products that maybe come built-in the operating system? And, if these are disabled, must the user be worried about other security products installed in the local area network (LAN) because of firewall telemetry and sample submission capabilities?

The Microsoft Windows telemetry DLL file is located in

```
C:\Windows\System32\generaltel.dll.
```

And you will readily notice that miscellaneous antivirus, antispyware and



firewall information is sent to Microsoft, where `Firewall information` means network information. It is possible to disable all the telemetry but maybe your LAN neighbor does not do that.

The cloud submission features of sample files also do exist and they are customizable by the user. It is possible to check easily (without reverse engineering nothing) what kind of information is sent to Microsoft because, due to open criticism, they released a tool named `Diagnostic Data Viewer`. You can use it immediately for those purposes (Figure 2.9).

In the lab computer used for this work, minimum telemetry is allowed which means that antimalware telemetry is disabled but nevertheless crashes allow Microsoft to follow the development of this thesis in real-time. Someone might think that this assessment is unrelated to malware but information about crashes is also used for this[34] (see Figure 2.10) purpose.[2]

2.4.2 Conclusion

Some operating systems contain built-in antimalware/firewall/threat hunting solutions. It is difficult for the common user to disable telemetry capabilities. A minimum telemetry is required by the operating system (and seems to be used also for malware purposes). Antivirus, antispyware and

[2] https://www.microsoft.com/security/blog/2019/11/07/the-new-cve-2019-0708-rdp-exploit-attacks-explained/

```
[0x180086520]> pd
            ;-- "InventoryMiscellaneousAntivirusInformation":
            ; DATA XREF from 0x1800280c3 (sub.InventoryMiscellaneousAntivirusInformation_f28)
            ; DATA XREF from 0x1800282e4 (sub.InventoryMiscellaneousAntivirusInformation_f28)
            0x180086520     .string "InventoryMiscellaneousAntivirusInformation" ; len=86
            ;-- hit0_1:
            0x18008654c      41006e00       add byte [r14], bpl
            0x180086550      7400           je 0x180086552
            0x180086552      690076006900   imul eax, dword [rax], 0x690076
            0x180086558      7200           jb 0x18008655a
            0x18008655a      7500           jne 0x18008655c
            0x18008655c      7300           jae 0x18008655e
            0x18008655e      49006e00       add byte [r14], bpl
            0x180086562      66006f00       add byte [rdi], ch
            0x180086566      7200           jb 0x180086568
            0x180086568      6d             insd dword [rdi], dx
            0x180086569      006100         add byte [rcx], ah
            0x18008656c      7400           je 0x18008656e
            0x18008656e      69006f006e00   imul eax, dword [rax], 0x6e006f
            0x180086574      0000           add byte [rax], al
            0x180086576      0000           add byte [rax], al
            ;-- hit4_32:
            ; DATA XREF from 0x1800284ca (sub.InventoryMiscellaneousAntispywareInformation_444)
            0x180086578     .string "InventoryMiscellaneousAntispywareInformationStartSync" ; len=54
            0x1800865ae      0000           add byte [rax], al
            ;-- hit4_33:
            ; DATA XREF from 0x1800284de (sub.InventoryMiscellaneousAntispywareInformation_444)
            ; DATA XREF from 0x1800286f8 (sub.InventoryMiscellaneousAntispywareInformation_444)
            0x1800865b0     .string "InventoryMiscellaneousAntispywareInformation" ; len=45
            0x1800865dd      0000           add byte [rax], al
            0x1800865df  ~   004900         add byte [rcx], cl
            ;-- "InventoryMiscellaneousAntispywareInformation":
            ; DATA XREF from 0x1800285df (sub.InventoryMiscellaneousAntispywareInformation_444)
            ; DATA XREF from 0x180028800 (sub.InventoryMiscellaneousAntispywareInformation_444)
            0x1800865e0     .string "InventoryMiscellaneousAntispywareInformation" ; len=90
            0x18008663a      0000           add byte [rax], al
            0x18008663c      0000           add byte [rax], al
            0x18008663e      0000           add byte [rax], al
            ;-- hit4_34:
            ; DATA XREF from 0x1800289e6 (sub.InventoryMiscellaneousFirewallInformation_960)
            0x180086640     .string "InventoryMiscellaneousFirewallInformationStartSync" ; len=51
            0x180086673      0000           add byte [rax], al
            0x180086675      0000           add byte [rax], al
            0x180086677  ~   00496e         add byte [rcx + 0x6e], cl
            ;-- hit4_35:
            ; DATA XREF from 0x1800289fa (sub.InventoryMiscellaneousFirewallInformation_960)
            ; DATA XREF from 0x180028c14 (sub.InventoryMiscellaneousFirewallInformation_960)
            0x180086678     .string "InventoryMiscellaneousFirewallInformation" ; len=42
```

firewall information is sent meaning that also LAN network information can be revealed. Operating system antimalware/firewall/threat hunting solutions could also substantially improve its system by adding events, files, processes, registry keys and system elements support into the UMSE dynamic linking library and calling to it before submitting collected data.

2.5 Intelligence products

2.5.1 Analysis

Malware intelligence tools store goodware in, maybe, greater volume than malware. `VirusTotal` is a service which allows you to check if a file is malware or not by querying a lot of antivirus engines. If you search, e.g.,"tutorial pdf" in Google, (a reasonable random goodware file), the first result,

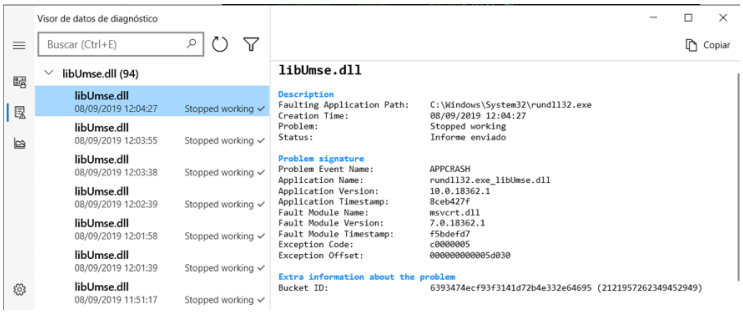


Figure 2.9: libUmse.dll

Microsoft security signals showed an increase in RDP-related crashes that are likely associated with the use of the unstable BlueKeep Metasploit module on certain sets of vulnerable machines. We saw:

- An increase in RDP service crashes from 10 to 100 daily starting on September 6, 2019, when the Metasploit module was released
- A similar increase in memory corruption crashes starting on October 9, 2019
- Crashes on external researcher honeypots starting on October 23, 2019

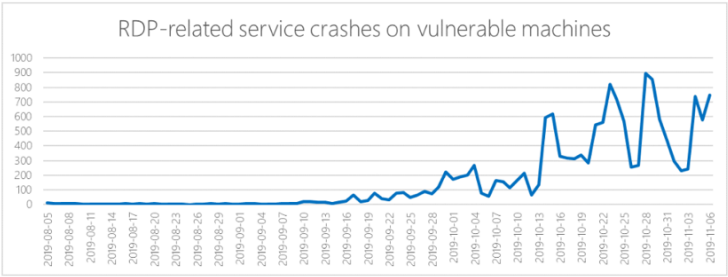


Figure 1. Increase in RDP-related service crashes when the Metasploit module was released

Figure 2.10:

in this case, is a Python Tutorial PDF. Finally, if you check if this file does exist in `VirusTotal`, you can see that it does (really, a lot of files are in `VirusTotal`, this can be checked straightforwardly). So, any person with a `VirusTotal` paid API account can download this file, make advanced searches in binaries (including Yara search) and easily locate files[35]. It is true that people who submits files accepts the EULA but, as you can see, if you have the paid `VirusTotal` API you can download books, software, movies .... Those old determined goodware files, some even signed, will be never removed from storage.

2.5.2 Conclusion

Malware intelligence tools store goodware in, maybe, greater volume than malware[36]. Those are not malware samples, those are simple, ordinary files. Intelligence products could also substantially improve its system by

adding encryption support for uncommon and copyrighted likely goodware files (because those files are not interesting for malware analysis in any way, not even to be compared with infected files) into the UMSE dynamic linking library. In this way, only files detected by at least one antimalware engine or a reliable human should be decrypted and opened to the world.

Chapter 3

Universal Malware Sample Encryption (UMSE)

3.1 Current malware sample shortcomings

It is no surprise, it makes sense. Neither the end user nor the best antivirus comparators[37][38], not even the most excellent and tested, have ever rated security tools in this key aspect: confidentiality. The amount of client computer leaked data by each accomplishes malware detection. When comparing them, they only take into account some other factors currently much more in dispute. We will refer to its table columns in the results: "blocked", "user dependent", "compromised", "protection rate" and "false alarms"[39].

	Blocked	User dependent	Compromised	PROTECTION RATE [Blocked % + (User dependent %)/2]*	False Alarms
Kaspersky	732	–	–	100%	0
Bitdefender	732	–	–	100%	6
VIPRE	731	–	1	99.9%	2
Microsoft	730	2	–	99.9%	24
Sophos	730	–	2	99.7%	5
McAfee	729	–	3	99.6%	5
K7	729	–	3	99.6%	10
Avast	728	–	4	99.5%	2
Panda	728	–	4	99.5%	19
CrowdStrike	725	–	7	99.0%	8

Therefore, all antiviruses are good in detection and this explains different

results drawn months apart when comparing them but, how good are they in confidentiality? Just note how important confidentiality is for the user when choosing between antimalware products.

Taking up the issue of what malware sample currently is, we should add that, sometimes, in order to mitigate risks in transport, storage and sample sharing, this functional file is wrapped in a compression format like PKZIP and, when supported by such format, maybe encrypted also by the de facto key, i.e., the symmetric but publicly known key: "infected"[31][40][35][42][43][44] The following picture shows a threat intelligence website in which files are downloaded in this way but the reader should know `Malwarebytes` uses this mechanisms because it was analyzed and shown in section **??**

This way of acquiring and storing malware samples, as is evident, was improvised but not seriously thought. In summary, the approach has the following shortcomings:

1. Identification shortcoming: Given a malware sample, it is not possible to know, a priori, if the file is a malware sample or not. It is, at every respect, simply a compressed file.

2. Universality shortcoming: A malware sample is not a file. Malware, according to UMSE, is an element or a set of elements, hardware, software and/or firmware which framed in a context and in a state of its life cycle is determinated malicious. Accordingly, a malware element can be "fileless" or, in a given moment of its life cycle, to be volatile, located only in principal memory (like it happens with `DoublePulsar`,

to name an example) and this must not be a problem when taking a malware sample. To put it differently, the sample must contain the element or the set of elements indicating its nature and all of those metainformation which is considered necessary.

3. Contextualization shortcoming: Context is lost. The sample is usually is a file without context. This lost will take unfortunate consequences in subsequent phases of the reverse engineering process.

4. Standardization shortcoming: The analyst does not know what kind of element is dealing with when he/she gets a sample, or how it came to be a sample. A real life example: During compression and decompression processes, it usually happens that, for instance, the compression format, being arbitrary, does not allow to store all file attributes and enters spurious values when decompressed.

5. Confidentiality shortcoming: Potentially confidential data stored in the samples is not protected in any way. Therefore, this data will be revealed to analysts, companies and intelligence organizations and/or partners or whoever malware sample is shared with. This data far exceeds the "need to know" principle, that is, the amount of information the professional really needs in order to do his/her job. We can put an easy-to-understand example of this risk. If an office file has aroused malware suspicion and it has macros, it is common sense to examine the macros at first and, if the code in macros code is not sufficient then, using the concept of encrypted malware sample proposed in this work, to decrypt other parts of the file content usually more confidential and less malware-prone but keeping a log of this decryption operations.

6. Authentication shortcoming: It is not possible to determine the authenticity of a sample if it was modified after acquisition. Note that authentication is also necessary to support encryption in order to guarantee confidentiality.

3.2 Universal Malware Sample Encryprtion

Universal Malware Sample Encryption format, (henceforth, UMSE), is a specific universal file format for malware samples, making efforts in efficient acquisition, transport and storage which preserves confidentiality, authenticity and makes possible the potential samples sensitive data access registration.

3.2.1 UMSE universality

This format is declared to be **universal** because it supports the storage of any kind of potential malware sample (contextualized and metadata en-

riched) covering any possible state of its life cycle. Expanding, therefore, the current malware sample concept[31] (limited to files) to a more general concept which allows the right storage of any element software, hardware and/or firmware. Understanding by elements software, hardware and/or firmware: system processes, program parameters, environment variables, register entries, network traffic captures, memory content, file paths and other file system information, integrated circuits photos and schemes, etc.

3.2.2 UMSE as a sample format

As UMSE is a format purposely designed for malware samples, **any file given in such format can be identified as malware sample**. Those malware samples can be easily identified by its magic ASCII value located at the very beginning of the file: UMSE. It is also specifically **stored in a standard format, minimizing the loss of sample context information by itself and its parts. While keeping the maximum storage and sharing security**.

3.2.3 UMSE confidentiality

UMSE format allows also to preserve potentially sensible data in the samples by confidentiality. This is achieved through a symmetric cipher. Symmetric keys are randomly generated (and IVs are also randomly generated) and wrapped with a public RSA key. Each symmetric key is associated to a confidentiality level. Each sample is divided in parts and each part is evaluated to assign a confidentiality level to it. Therefore, only a sufficiently privileged analyst will be able to decrypt a sample part and, every time a key is revealed to him by RSA private key decryption, the event could be registered.

3.2.4 UMSE authentication

Each sample is HMAC authenticated following an Encrypt-then-MAC (EtM) scheme[47]. HMAC key is derived from the symmetric keys sequential list, initialization vectors and associated confidentiality levels. Therefore, only people who knows all the symmetric keys and initialization vectors (this fact means knowing the entire decrypted sample) could be able to modify it. This protects the sample from "downgrade" attacks to the confidentiality level of its parts, from "decryption oracle attacks", etc.

3.3 UMSE format specification

3.3.1 Formal specification

UMSE structure is formally specified using Kaitai Struct language[27].

```
meta:
  id: umse
  file-extension: umse
  endian: le

seq:
  - id: umse_header
    type: header
  - id: decryption_table
    type: decryption_table
    repeat: expr
    repeat-expr: umse_header.num_records_dec_table
  - id: entry
    type: entry
    repeat: expr
    repeat-expr: umse_header.num_file_entries
  - id: file_properties
    type: file_properties
  - id: authentication_header
    type: authentication_header

types:
  header:
    seq:
      - id: magic
        contents: UMSE
        doc: "Magic of: Universal Malware Sample Encryption"
      - id: version
        type: str
        encoding: UTF-8
        size: 4
        doc: "Magic of: Universal Malware Sample Encryption"
      - id: num_records_dec_table
        type: u4
        doc: "Number of Records in Decryption Table"
      - id: num_file_entries
        type: u4
        doc: "Number of file entries or encrypted byte chunks"
      - id: author_name_length
        type: u4
        doc: "Author name length"
```

```
42      - id: author_name
43        type: str
44        size: author_name_length
45        encoding: UTF-8
46        doc: "Author name"
47  decryption_table:
48    seq:
49      - id: level_of_confidentiality
50        type: u1
51      - id: aes_wrapped
52        type: u1
53        repeat: expr
54        repeat-expr: 256
55  entry:
56    seq:
57      - id: size
58        type: u4
59      - id: level_of_confidentiality
60        type: u1
61      - id: encrypted_message
62        type: u1
63        repeat: expr
64        repeat-expr: size
65      - id: num_metadata
66        type: u4
67      - id: entry_metadata
68        type: entry_metadata
69        repeat: expr
70        repeat-expr: num_metadata
71        if: num_metadata > 0
72  entry_metadata:
73    seq:
74      - id: tag
75        type: u1
76        repeat: expr
77        repeat-expr: 8
78      - id: length
79        type: u4
80      - id: value
81        type: u1
82        repeat: expr
83        repeat-expr: length
```

```
84   file_properties:
85     seq:
86       - id: level_of_confidentiality
87         type: u1
88       - id: hash_value
89         type: u1
90         repeat: expr
91         repeat-expr: 32
92       - id: num_metadata
93         type: u4
94       - id: file_metadata
95         type: file_metadata
96         repeat: expr
97         repeat-expr: num_metadata
98         if: num_metadata > 0
99   file_metadata:
100    seq:
101       - id: tag
102         type: u1
103         repeat: expr
104         repeat-expr: 8
105       - id: length
106         type: u4
107       - id: value
108         type: u1
109         repeat: expr
110         repeat-expr: length
111   authentication_header:
112     seq:
113       - id: length
114         type: u4
115         doc: "HMAC length"
116       - id: hmac
117         type: u1
118         repeat: expr
119         repeat-expr: length
120         doc: "HMAC"
121  rsa_private_key:
122    seq:
123      - id: length
124        type: u4
125        if: not _io.eof
126        doc: "RSA private key length"
127      - id: rsa_private_key
128        type: u1
129        repeat: expr
130        repeat-expr: length
131        if: not _io.eof
142        doc: "RSA private key"
```

For practical purposes, an UMSE format structure parser template was also developed:

```
//------------------------------------------------
//--- 010 Editor v8.0.1 Binary Template
//
//      File: umse.bt
//   Authors: David Alvarez Perez
//   Version: 0.1
//   Purpose: Template for Universal Malware Sample Encryption
//  Category: Misc
// File Mask: umse
//  ID Bytes: 55 4D 53 45 //UMSE
//   History:
//------------------------------------------------

struct ENCRYPTED_SAMPLE {
   struct HEADER
       char    magic[4] <comment="Magic of: Universal Malware Sample
                                  Encryption">;
       char    version[4] <comment="Version of: Universal Malware
                                    Sample Encryption format">;
       int     numRecordsDecTable <comment="Number of Records in
                                           Decryption Table">;
       int     numFileEntries <comment="Number of file entries or
                                        encrypted byte chunks">;
       int     authorNameLength <comment="Author name length">;
       char    authorNameValue[header.authorNameLength]
                                  <comment="Author name">;
    header;

   struct DECRYPTION_TABLE
       char    levelOfConfidentiality <comment="Level of
                                               confidentiality">;
       char    aesWrapped[256] <comment="IV and AES256 Key
                                        (wrapped with Public RSA) both
                                        necessaries to decrypt chunks
                                        of this level">;
    decryptionTable [ header.numRecordsDecTable ];

```

```
29     struct ENTRY
30         int     size <comment="Size of this entry">;
31         char    levelOfConfidentiality <comment="Level of
                                           confidentiality of this entry">;
32         char    encryptedMessage[entry.size] <comment="Encrypted chunk">;
33         int     numMetadata <comment="Number of public metadata entries
                                       of this chunk">;
34         struct ENTRY_METADATA
35             char    tag[8] <comment="TAG of this metadata">;
36             int     length <comment="Size of this metadata content">;
37             char    value[ length ] <comment="metadata value">;
38          entry_meta [ entry. numMetadata ]<optimize=false>;
39      entry [ header.numFileEntries ]<optimize=false>;
40
41     struct FILE_PROPERTIES
42         char    levelOfConfidentiality <comment="File confidentiality level">;
43         char    hashValue[ 32 ] <comment="Hash value">;
44         int     numMetadata <comment=" Number of public metadata entries
                                        of this file">;
45         struct FILE_METADATA
46             char    tag[8] <comment="TAG of this flag">;
47             int     length <comment="Size of this metadata content">;
48             char    value[ file_metadata.length ] <comment="metadata value">;
49          file_metadata [ file_properties.numMetadata ]<optimize=false>;
50      file_properties;
51
52     struct AUTHENTICATION_HEADER
53         int     authenticationLength  <comment="Authentication length">;
54         char    authentication[authentication_header.authenticationLength]
             <comment="HMAC SHA256 Authentication. The key is the sha256 of
                       the encryption table but after entire decrypted">;
55      authentication_header;
56
57     if(!FEof()) {
58         struct RSA_PRIVATE_KEY {
59             int     rsaPrivateKeyLength <comment=" RSA Key Length">;
60             char    rsaPrivateKey[ rsa_private_key.rsaPrivateKeyLength ]
                  <comment="RSA Key">;
61         } rsa_private_key <comment="RSA Key (This is an optional field)">;
62     }
63 } encryptedSample;
64
```

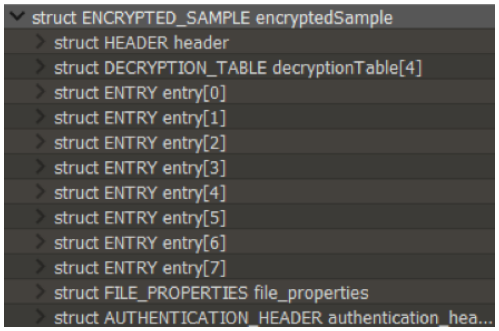
```
65
66  // Main
67
68  int isUMSE(void)
69  {
70      local TFindResults countUmse;
71      countUmse = FindAll("UMSE");
72      return countUmse.count;
73  }
74
75
76  isUMSE();
77  if (isUMSE()==0)
78  {
79      Printf("Keywords not found, not a UMSE file!\n");
80      return;
81  }
```

3.3.2 Overview

A sample given in UMSE format consists in the following sub-structures:

- UMSE header.
- Decryption table.
- Entries.
- File properties.
- Authentication header.
- RSA Private key (this is an optional field).

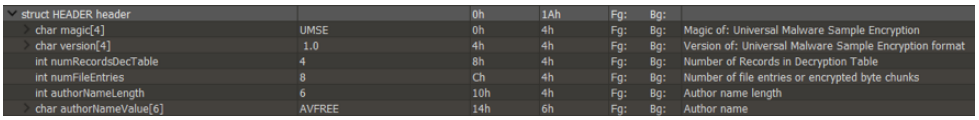
In the following picture an example of UMSE file format structure is shown.



3.3.3 UMSE header

As you can appreciate, this format is first formed by a `HEADER` header consisting of the following fields:

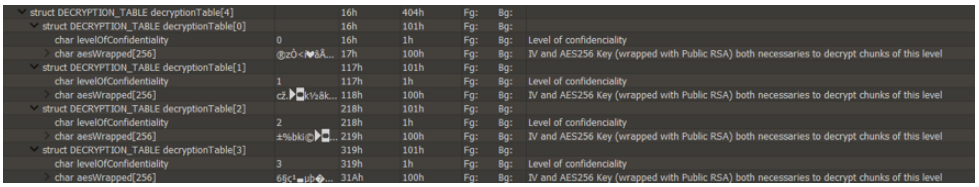
- The `Magic` field. Whose value is always `UMSE`. This value allows to easily identify that it is an UMSE format file.
- The `Version` field. A string of four characters indicating UMSE version. The current version string is: `1.0`.
- The `numRecordsDecTable` field. This field indicates, by using an integer of 4 bytes, the number of entries of which `Decryption Table` is composed, I mean, the number of AES keys stored in the UMSE file (an AES key for each possible confidentiality level).
- The `numFileEntries` field. This field indicates, by using an integer of 4 bytes, the number of chunks stored in the UMSE sample.
- The `authorNameLength` field indicates, by using an integer of 4 bytes, the size of the next field (`authorNameValue`).
- The field `authorNameValue` the name of the malware samples author.



struct HEADER header		0h	1Ah	Fg:	Bg:	
char magic[4]	UMSE	0h	4h	Fg:	Bg:	Magic of: Universal Malware Sample Encryption
char version[4]	1.0	4h	4h	Fg:	Bg:	Version of: Universal Malware Sample Encryption format
int numRecordsDecTable	4	8h	4h	Fg:	Bg:	Number of Records in Decryption Table
int numFileEntries	8	Ch	4h	Fg:	Bg:	Number of file entries or encrypted byte chunks
int authorNameLength	6	10h	4h	Fg:	Bg:	Author name length
char authorNameValue[6]	AVFREE	14h	6h	Fg:	Bg:	Author name

3.3.4 Decryption table

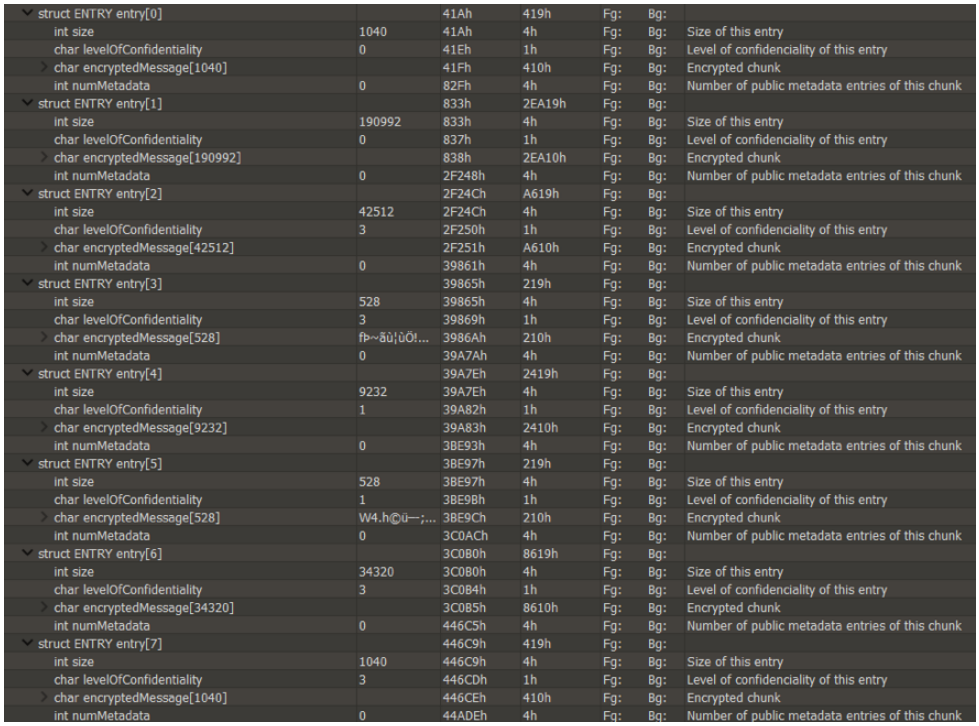
Next, there is the `Decryption table`. This table contains AES keys and IVs asociated to each confidentiality level. This keys are used to encrypt chunks regarding the confidentiality level. Each table entry contains an AES key and IV, both wrapped with RSA public key and stored into the `aesWrapped` field. And corresponding confidentiality level is stored into `levelOfConfidentiality` field.



struct DECRYPTION_TABLE decryptionTable[4]		16h	404h	Fg:	Bg:	
struct DECRYPTION_TABLE decryptionTable[0]		16h	101h	Fg:	Bg:	
char levelOfConfidentiality	0	16h	1h	Fg:	Bg:	Level of confidenciality
char aesWrapped[256]	@zÖ<…	17h	100h	Fg:	Bg:	IV and AES256 Key (wrapped with Public RSA) both necessaries to decrypt chunks of this level
struct DECRYPTION_TABLE decryptionTable[1]		117h	101h	Fg:	Bg:	
char levelOfConfidentiality	1	117h	1h	Fg:	Bg:	Level of confidenciality
char aesWrapped[256]	cž…k½ák…	118h	100h	Fg:	Bg:	IV and AES256 Key (wrapped with Public RSA) both necessaries to decrypt chunks of this level
struct DECRYPTION_TABLE decryptionTable[2]		218h	101h	Fg:	Bg:	
char levelOfConfidentiality	2	218h	1h	Fg:	Bg:	Level of confidenciality
char aesWrapped[256]	±%bki©…	219h	100h	Fg:	Bg:	IV and AES256 Key (wrapped with Public RSA) both necessaries to decrypt chunks of this level
struct DECRYPTION_TABLE decryptionTable[3]		319h	101h	Fg:	Bg:	
char levelOfConfidentiality	3	319h	1h	Fg:	Bg:	Level of confidenciality
char aesWrapped[256]	6§c¹…uÞ…	31Ah	100h	Fg:	Bg:	IV and AES256 Key (wrapped with Public RSA) both necessaries to decrypt chunks of this level

3.3.5 Entries

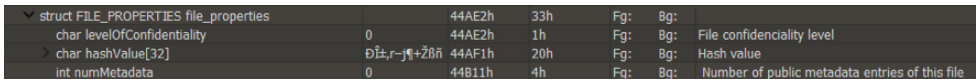
Each sample chunk is stored (encrypted) as an entry regarding its confidentiality level. Each entry has: a field named `size` indicating the size of the encrypted content, a field named `levelOfConfidentiality` indicating the chunk encrypted confidentiality level, a field named `encryptedMessage` containing the encrypted chunk, a field named `numMetadata` indicating the number of metadata entries associated to the chunk.



struct ENTRY entry[0]		41Ah	419h	Fg:	Bg:	
int size	1040	41Ah	4h	Fg:	Bg:	Size of this entry
char levelOfConfidentiality	0	41Eh	1h	Fg:	Bg:	Level of confidenciality of this entry
char encryptedMessage[1040]		41Fh	410h	Fg:	Bg:	Encrypted chunk
int numMetadata	0	82Fh	4h	Fg:	Bg:	Number of public metadata entries of this chunk
struct ENTRY entry[1]		833h	2EA19h	Fg:	Bg:	
int size	190992	833h	4h	Fg:	Bg:	Size of this entry
char levelOfConfidentiality	0	837h	1h	Fg:	Bg:	Level of confidenciality of this entry
char encryptedMessage[190992]		838h	2EA10h	Fg:	Bg:	Encrypted chunk
int numMetadata	0	2F248h	4h	Fg:	Bg:	Number of public metadata entries of this chunk
struct ENTRY entry[2]		2F24Ch	A619h	Fg:	Bg:	
int size	42512	2F24Ch	4h	Fg:	Bg:	Size of this entry
char levelOfConfidentiality	3	2F250h	1h	Fg:	Bg:	Level of confidenciality of this entry
char encryptedMessage[42512]		2F251h	A610h	Fg:	Bg:	Encrypted chunk
int numMetadata	0	39861h	4h	Fg:	Bg:	Number of public metadata entries of this chunk
struct ENTRY entry[3]		39865h	219h	Fg:	Bg:	
int size	528	39865h	4h	Fg:	Bg:	Size of this entry
char levelOfConfidentiality	3	39869h	1h	Fg:	Bg:	Level of confidenciality of this entry
char encryptedMessage[528]	fÞ~ãù¦ùÖ!...	3986Ah	210h	Fg:	Bg:	Encrypted chunk
int numMetadata	0	39A7Ah	4h	Fg:	Bg:	Number of public metadata entries of this chunk
struct ENTRY entry[4]		39A7Eh	2419h	Fg:	Bg:	
int size	9232	39A7Eh	4h	Fg:	Bg:	Size of this entry
char levelOfConfidentiality	1	39A82h	1h	Fg:	Bg:	Level of confidenciality of this entry
char encryptedMessage[9232]		39A83h	2410h	Fg:	Bg:	Encrypted chunk
int numMetadata	0	3BE93h	4h	Fg:	Bg:	Number of public metadata entries of this chunk
struct ENTRY entry[5]		3BE97h	219h	Fg:	Bg:	
int size	528	3BE97h	4h	Fg:	Bg:	Size of this entry
char levelOfConfidentiality	1	3BE9Bh	1h	Fg:	Bg:	Level of confidenciality of this entry
char encryptedMessage[528]	W4.h©ú—;...	3BE9Ch	210h	Fg:	Bg:	Encrypted chunk
int numMetadata	0	3C0ACh	4h	Fg:	Bg:	Number of public metadata entries of this chunk
struct ENTRY entry[6]		3C0B0h	8619h	Fg:	Bg:	
int size	34320	3C0B0h	4h	Fg:	Bg:	Size of this entry
char levelOfConfidentiality	3	3C0B4h	1h	Fg:	Bg:	Level of confidenciality of this entry
char encryptedMessage[34320]		3C0B5h	8610h	Fg:	Bg:	Encrypted chunk
int numMetadata	0	446C5h	4h	Fg:	Bg:	Number of public metadata entries of this chunk
struct ENTRY entry[7]		446C9h	419h	Fg:	Bg:	
int size	1040	446C9h	4h	Fg:	Bg:	Size of this entry
char levelOfConfidentiality	3	446CDh	1h	Fg:	Bg:	Level of confidenciality of this entry
char encryptedMessage[1040]		446CEh	410h	Fg:	Bg:	Encrypted chunk
int numMetadata	0	44ADEh	4h	Fg:	Bg:	Number of public metadata entries of this chunk

3.3.6 File properties

An additional entry indicating properties which applies to the entire file do exist.

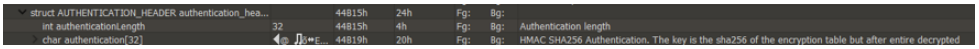


struct FILE_PROPERTIES file_properties		44AE2h	33h	Fg:	Bg:	
char levelOfConfidentiality	0	44AE2h	1h	Fg:	Bg:	File confidenciality level
char hashValue[32]	ÐÏ±,r~j¶+Žßñ	44AF1h	20h	Fg:	Bg:	Hash value
int numMetadata	0	44B11h	4h	Fg:	Bg:	Number of public metadata entries of this file

3.3.7 Authentication header

Finally, an HMAC checksum of the file is generated. The key used to generate this HMAC checksum is the SHA256 hash of the RSA decrypted

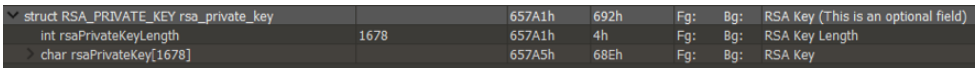
`decryption table`.



Authentication header is absolutely necessary. For instance, an analyst who has not sufficient privilege to decrypt an UMSE entry, will not be able to generate another UMSE file with the entry confidentiality level downgraded, in order to decrypt it. It protects from padding oracle attacks[48][49] as well. This block is complementary to encryption.

3.3.8 RSA private key

This is an optional (and generally not recommended) field used to embed the private RSA key into the file. It can be useful in some cases when confidentiality is not an issue but the analyst want to exploit the UMSE format capabilities (see section: 12.5 UMSE tools).



3.4 UMSE implementation

3.4.1 UMSE C/C++ library

A C/C++ library was developed to work more easily with UMSE. As most of security products are developed in C/C++, the UMSE library was developed in these languages but note that it can be used with independence of the programming language, for instance, the author integrated it with Python and, obviously, it worked. The library project follows a standard C project structure[51] (see Figure 3.2:

- `bin` directory: Compiled resulting binaries are putted inside this directory.
- `build`: Building files are putted in this directory.
- `doc` Documentation directory.
- `include`: Includes are here. A subfolder named `include` also exists and it is used to store third party includes.
- `lib`: Library files are stored here.
- `src`: The program source code.
- `test`: Some code to test the library.

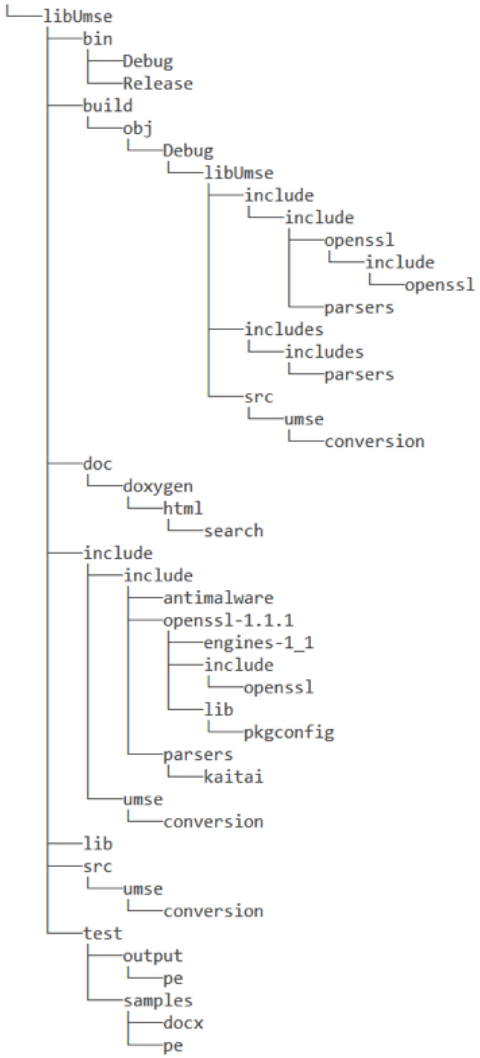


Figure 3.1: Structure of the UMSE library.

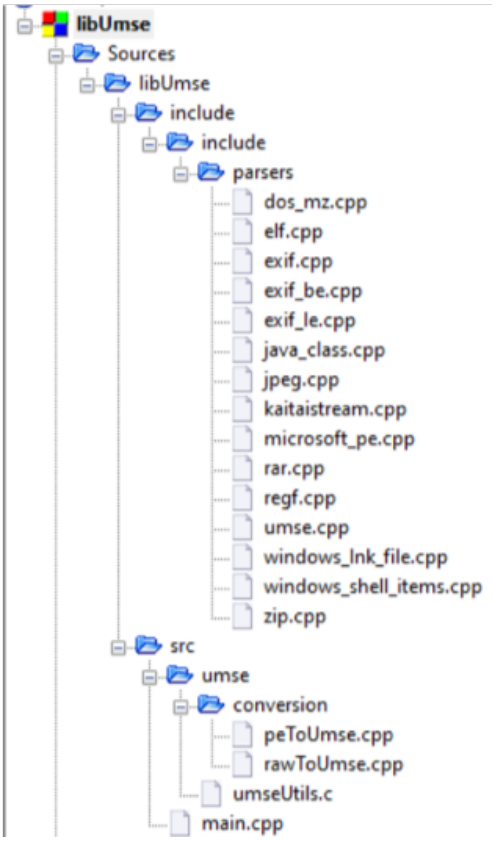


Figure 3.2: Organization of the UMSE project.

The project content can be summarized as follows (Figure **??**):

- the `parsers` directory contains a lot of common file type parsers for C++. These parsers were generated automatically by Kaitai Struct. Parsers folder includes also `umse.cpp` in order to demonstrate that UMSE specification works as expected and it is implemented.
- the `conversion` directory implements the conversion from every specific file format or chunk structure to UMSE. In file headers of each file format/chunk structure converter are defined parameters to calculate the corresponding confidentiality level of each file part.
- the `umseUtils.c` all necessary functions to deal with UMSE file format are developed here. It is not dependent from Kaitai Struct in order to allow you to re-use it in other projects.
- the `main.cpp` library exported functions are written here and some testing/debugging code is also written here.

The library exported functions are displayed in Tables 3.1-3.3.

Description	This function converts a chunk to Universal Malware Sample Encryption format.
Input parameters	`chunkLength` `chunk`
Output parameters	`umseSize` `umse`
Return value	`void`

Function header	`void DLL_EXPORT ChunkToUmse(`
	`  unsigned int chunkLength,`
	`  unsigned char* chunk,`
	`  unsigned int* umseSize,`
	`  unsigned char** umse`
	`);`

Table 3.1: Function f1.

Description	If UMSE integrity check is successful, this function decrypts an entry of a given Universal Malware Sample Encryption file chunk and its RSA Private key.

Input parameters	`chunkLength`
	`chunk`
	`entryToDecrypt`
	`accessLevel`
	`rsaPrivateKey`

Output parameters	`decryptedEntryLength` `decryptedEntry`
Return value	-2 if integrity check iss not successful -1 if `entryToDecrypt` does not exist 0 if the function ends without errors.

Function header	`int DLL_EXPORT DecryptUmse(`
	`  unsigned int umseLength,`
	`  unsigned char* umse,`
	`  unsigned int entryToDecrypt,`
	`  unsigned int accessLevel,`
	`  char* rsaPrivateKey,`
	`  unsigned int* decryptedEntryLength,`
	`  unsigned char** decryptedEntry`
	`);`

Table 3.2: Function f2.

Description	This function checks UMSE integrity.

Input parameters	`chunkLength`
	`chunkLength unsigned int`
	`chunk unsigned char *`

Output parameters	`umseSize unsigned int *` `umse unsigned char **`
Return value	True if integrity check is successful, otherwise it returns false

Function header	`bool DLL_EXPORT CheckUmseIntegrity(`
	`unsigned int umseLength,`
	`unsigned char* umse,`
	`char* rsaPrivateKey`
	`);`

Table 3.3: Function f3.

3.5 UMSE agent

The UMSE Agent is an extremely simple Python program to demonstrate how to collect system elements, transform it to UMSE, and sent the resulting UMSE samples to a central intelligence server.

Files	Purpose
`aboutform.py`	A simple form showing information about the program.
`av_agent.py`	PyQt5 System Tray Icon entrypoint.
`intelligenceclient.py`	REST API functions for communicating with UMSE Server.
`scanfiles.py`	Element collector, UMSE conversion and sending via `libUmse.dll`.
`trayicon.py`	System Tray Icon implementation.

The agent must to be launched as follows:

```
start /D . C:\Python\Python37\pythonw.exe av_agent.py
```

and this is how the agent looks like. Options are self-explanatory but notice that the `Collect malware samples` option means collect samples, convert it to UMSE and automatically send it to the UMSE server.

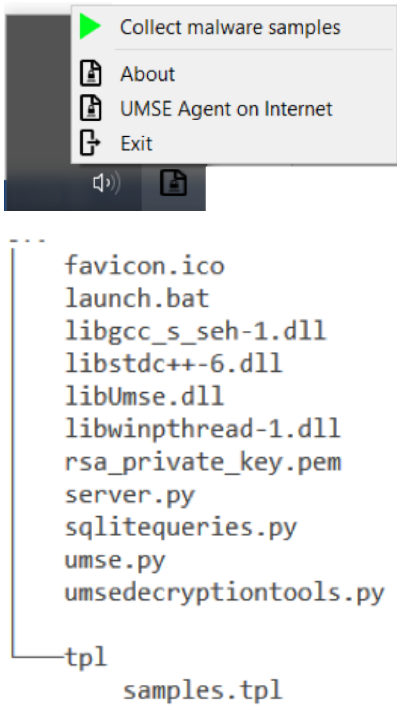


Figure 3.3: Structure of the UMSE server.

3.5.1 UMSE server

The UMSE server is composed of the following files (see Figure 3.3):

- `favicon.ico`: The web application favicon.
- `launch.bat`: Simple batch file to start the program.
- `libgcc_s_sheh-1.dll`, `libstdc++-6.dll`, `libwinpthread-1.dll`: Some `libUmse.dll` dependencies.
- `libUmse.dll`: The compiled UMSE C/C++ library.
- `rsa_private_key.pem`: RSA Private key which allows UMSE sample parts decryption.
- `server.py`: A Cherrypy server implementing an extremely simple intelligence webpage and REST API to allow the agent to operate with samples. Samples are stored in a SQLite database.
- `sqlitequeries.py`: SQLite queries are defined here.
- `umse.py`: Kaitai Struct UMSE file format parser.
- `umsedecryptiontools.py`: Python functions allowing to decrypt an UMSE entry (functionality is implemented by `libUmse.dll`. It is only a bridge to it).

- `tpl/samples.tpl`: HTML with JAVASCRIPT template used by Cherrypy.

This is how intelligence panel looks like. It is extremely simple. Only sample listing (- and + are shortcut keys to go back and forward), sample search, and sample downloading features are implemented. But note that REST API requires authentication in order to decrypt samples, and every decryption operation is logged into SQLite database access logging table.

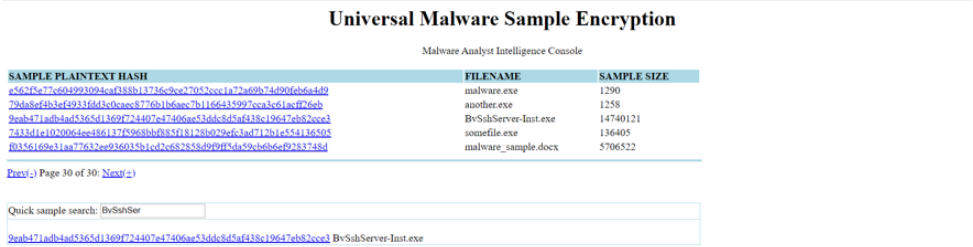


Figure 3.4: Some caption.

3.5.2 UMSE Shell

An extremely simple command line tool implemented in Python which allows you to interact with the UMSE server. All implementation remains into the `umse_shell.py` file which looks as follows:

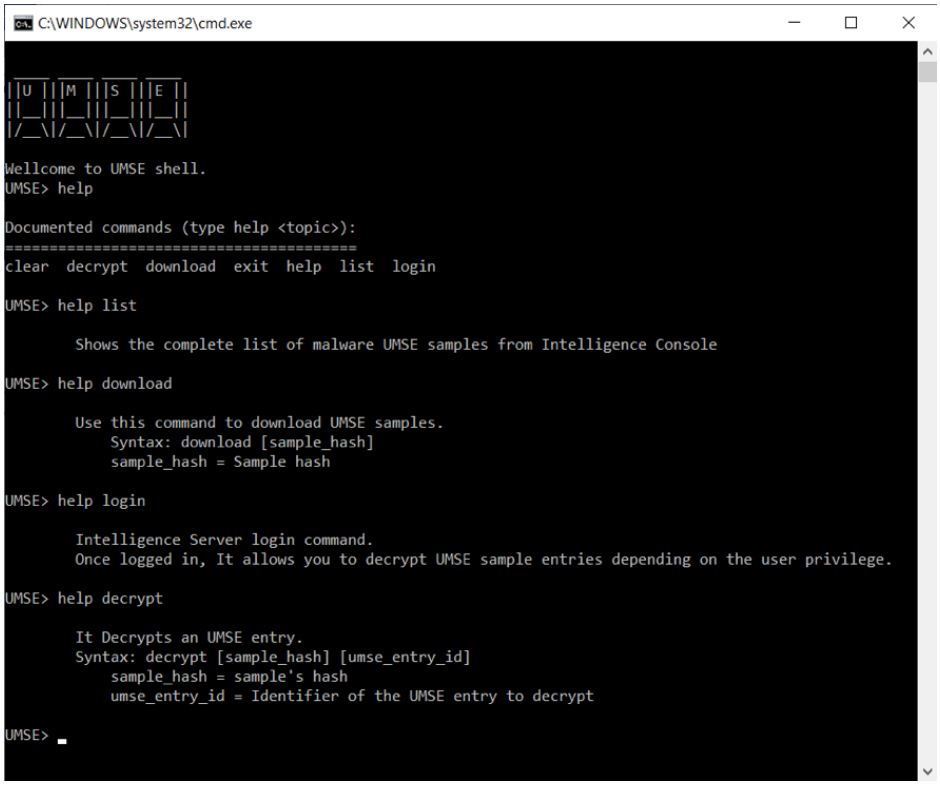
```
25  class MyCmd(Cmd):
26
27      prompt = PROMPT
28
29      def do_decrypt(self, args): ...
66
67      def do_login(self, args): ...
89
90      def do_list(self, args): ...
104
105     def do_download(self, args): ...
130
131     def do_clear(self, args): ...
136
137     def do_exit(self, args): ...
```

And this is how this console looks like.

3.5.3 UMSE tools

The UMSE format RSA encryption is a key feature but, in some situations like when sharing malware between colleagues encryption can be a hindrance. In this situations you can embed the RSA private key into the sample and dont care about encryption anyway. In this way, the analyst is benefiting himself by using a specific format to share malware files and, in a future

Method	Description
`do_list`	List samples stored in the UMSE server.
`do_download`	Download samples from the server.
`do_login`	Login into the server.
`do_decrypt`	Decrypt an entry of an UMSE sample (login is required).
`do_clear`	Clear screen.
`do_exit`	Exit the program.



version of this simple python tool, of metadata and heterogeneous element per sample appending capabilities of UMSE format. You can use a simple Python tool (`tool_for_single_users.py`) developed for this purpose:

Chapter 4

Conclusions and future work

The UMSE malware sample format improves malware reverse engineering processes[10][11][13][14][15] and also allows security products (including those which comes built-in the operating system) to be polite with potentially confidential elements and information about its clients and users. Antivirus enterprises do not seem to be interested in spying the user[53], but maybe they collaborate with a third party who also uses intelligence information for purposes unrelated to malware analysis[6]. By using UMSE, antivirus products can take the malware sample treatment control in a transparent way and also protect users' data from a hypothetical malicious third party. With the arrival of cyber threat hunting products (extremely aggressive products against confidentiality), antivirus comparison tools must take seriously into account the issues of confidentiality. The rest of features are already sufficiently competed. Therefore, clients and users must take confidentiality into account while choosing between them. It is possible to keep the protection ratio and users confidentiality at the same time if UMSE is used to acquire, transport and store the samples.

There of course some developments and features which were not implemented in UMSE so far. among the possible extensions that the tool could integrate, we would like to mention the following ideas for future work:

- UMSE version 1.0 does not specify malware samples metadata. In order to be as flexible as possible, it lets this definition absolutely open to security tools which implements UMSE. The future work consist in to develop a sufficiently general malware UMSE metadata standard to reduce security tools UMSE implementation curve.
- Current UMSE dynamic linking library includes some target format to UMSE converters `xxxToUmse.cpp`. For instance, `peToUMSE.cpp` was developed. We recognize that the mentioned converter file is too simple (special cases could lead to errors) and does not take advantage

of Portable Executable granularity (for instance, it does not encrypt each binary resource separately). We let this improvement and the incorporation of new converters for a future work.

- The main purpose of this work is to uncover some shortcomings in the current techniques for treating malware samples, and to provide a solution to them. Unfortunately, for reasons of time, we could not incorporate this mechanism to any existing open source solution for antimalware, like Clamav[1] for instance, letting it also open for a future work.
- Improve and add support of metadata and heterogeneous element per sample to "Simple UMSE tool for single users" mentioned in section XXX.

[1] https://github.com/Cisco-Talos/clamav-devel

Bibliography

[1] Paet, U. (25/05/2018). European Parliament: REPORT on cyber defence. http://www.europarl.europa.eu/doceo/document/A-8-2018-0189_EN.pdf?redirect

[2] Annemans, G. (13/06/2018). European Parliament: Designating programmes and companies as 'dangerous' from the point of view of cyber defence http://www.europarl.europa.eu/doceo/document/P-8-2019-001206_EN.pdf

[3] Lab., K. (2019). Kaspersky Lab. Retrieved from Kaspersky Lab. https://products.s.kaspersky-labs.com/homeuser/kav2020/20.0.14.1085abc/english-INT-0.2007.0/3231373433327c44454c7c4e554c4c/eula_en.txt

[4] McAfee (April, 2018). McAfee. Retrieved from McAfee. https://www.mcafee.com/enterprise/en-us/assets/legal/end-user-license-agreements-en-gb.pdf

[5] Malwarebytes (2020). Malwarebytes. Retrieved from Malwarebytes: https://www.malwarebytes.com/eula/

[6] Lab., K. (2019). Kaspersky Lab. Retrieved from Kaspersky Lab. blog: https://www.kaspersky.com/blog/the-boundaries-of-trust/

[7] Threat Hunting For Dummies Carbon Black Special Edition. Peter H. Gregory

[8] PROACTIVE HUNTING The Last Line of Defense Against the Mega Breach

[9] Hardware malware. [S.l.]: Morgan Claypool. ISBN 9781627052528.

[10] Analysis and classification of context-based malware behavior. Mohammadhadi Alaeiyan, Saeed Parsa, Mauro Conti

[11] Learning from Context: Exploiting and Interpreting File Path Information for Better Malware Detection. Adarsh Kyadige, Ethan M. Rudd, Konstantin Berlin

[12] Richard F. Schmidt, in Software Engineering, 2013

[13] Sharma, S (2013). "Terminate and Stay Resident Viruses" (PDF). International Journal of Research in Information Technology. 1 (11): 201210.

[14] "Fileless attacks against enterprise networks". Secure List. Secure List. Retrieved 20 February 2017.

[15] "Advanced volatile threat: New name for old malware technique?". CSO. CSO. Retrieved 20 February 2017.

[16] Microsoft Portable Executable and Common Object File Format Specification. Revision 6.0 February 1999. Microsoft Corporation.

[17] Corporation, M. (31/05/2018) Microsoft Corporation. Retrieved from Microsoft Docs. https://docs.microsoft.com/es-es/windows/win32/dlls/dynamic-link-libraries?redirectedfrom=MSDN

[18] NIST (April, 2013). Rebecca M. Blank and Patrick D. Gallagher. Security and Privacy Controls for Federal Information Systems and Organizations. Retrieved from NIST. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

[19] RFC (August, 2007). R. Shirey. Obtained from Internet Engineering Task Force. https://tools.ietf.org/html/rfc4949

[20] NIST (2020). NIST. Retrieved from NIST glossary. https://csrc.nist.gov/glossary/term/malware

[21] A Layered Detection Method for Malware Identification. Ting Liu, Xiaohong Guan, Yu Qu, Yanan Sun

[22] Lubold, Gordon; Harris, Shane (6 October 2017). "Russian Hackers Stole NSA Spy Secrets". The Wall Street Journal. New York City. pp. 1, 4. Retrieved 12 October 2017

[23] Harris, Shane; Lubold, Gordon (2017-10-11). "Russia Has Turned Kaspersky Software Into Tool for Spying". Wall Street Journal. ISSN 0099-9660. Retrieved 2017-10-19.

[24] Lab., K. (2020). Kaspersky Lab. Retrieved from Kaspersky Lab. https://www.kaspersky.com/transparency-center

[25] Kaspersky, E. (30/06/2017). Eugene Kaspersky. Retrieved from Eugene Kaspersky blog. https://eugene.kaspersky.com/2017/06/30/keeping-cybersecurity-separate-from-geopolitics/

[26] McAfee (June, 2019). McAfee. Retrieved from McAfee. https://www.mcafee.com/enterprise/en-us/assets/legal/end-user-license-agreements-en-us.pdf

[27] Struct, K. (2020). Kaitai Struct. Retrieved from Kaitai Struct documentation. https://doc.kaitai.io/

[28] Malwarebytes for Windows User Guide. Version 3.6.1. 19 September 2018. Copyright 2018 Malwarebytes. All rights reserved.

[29] Amazon (2020). Amazon Web Services, Inc. Retrieved from Amazon S3 documentation. https://docs.aws.amazon.com/es_es/AmazonS3/latest/dev/RESTAPI.html

[30] Malwarebytes (2020). Malwarebytes. Retrieved from Malwarebytes. https://es.malwarebytes.com/premium/

[31] Zeltser, L. (24/03/2018) Lenny Zeltser. Retrieved from Lenny Zeltser. https://zeltser.com/share-malware-with-researchers/

[32] Corporation, M. (01/14/2020) Microsoft Corporation. Retrieved from Microsoft Docs. https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/advanced-hunting-schema-reference

[33] CarbonBlack (April, 2017). CarbonBlack. Retrieved from CarbonBlack. https://www.carbonblack.com/wp-content/uploads/2017/04/BanHash-B.png

[34] Corporation, M. (07/11/2019) Microsoft Corporation. Retrieved from Microsoft blog site. https://www.microsoft.com/security/blog/2019/11/07/the-new-cve-2019-0708-rdp-exploit-attacks-explained/

[35] VirusTotal (2020). VirusTotal. Retrieved from VirusTotal file search: https://www.virustotal.com/intelligence/help/file-search/

[36] Intrusion Detection Networks: A Key to Collaborative Security. Carol Fung, Raouf Boutaba

[37] AV-Comparatives (2020). AV-Comparatives. Retrieved from AV-Comparatives: https://www.av-comparatives.org/

[38] AV-Test (2020). AV-Test. Retrieved from AV-Test. https://www.av-test.org/

[39] AV-Comparatives (July, 2019). AV-Comparatives. Retrieved from AV-Comparatives. https://www.av-comparatives.org/tests/business-security-test-2019-march-june/

[40] VXVault (2020). VXVault. Retrieved from VXVault. http://vxvault.net/ViriList.php

[41] VirusTotal (2020). VirusBay. Retrieved from VirusTotal. https://www.virustotal.com/gui/home/upload

[42] MalShare (2020). MalShare. Retrieved from MalShare. https://malshare.com/

[43] VirusShare (2020). VirusShare. Retrieved from VirusShare. https://virusshare.com/

[44] VirusBay (2020). VirusBay. Retrieved from VirusBay. https://beta.virusbay.io/sample/browse

[45] VirusTotal (2020). VirusTotal. Retrieved from VirusTotal Terms of Service. https://support.virustotal.com/hc/en-us/articles/115002145529-Terms-of-Service

[46] NIST (2020). NIST. Retrieved from NIST glossary. https://csrc.nist.gov/glossary/term/firmware

[47] Bellare M., Namprempre C. (2000) Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: Okamoto T. (eds) Advances in Cryptology ASIACRYPT 2000. ASIACRYPT 2000. Lecture Notes in Computer Science, vol 1976. Springer, Berlin, Heidelberg

[48] Serge Vaudenay (2002). Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS... EUROCRYPT 2002.

[49] Manger, James. "A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS 1 v2.0". Telstra Research Laboratories.

[50] "Information technology – Security techniques – Authenticated encryption". 19772:2009. ISO/IEC. Retrieved March 12, 2013.

[51] Canonical Project Structure. Boris Kolpackov 2018-10-08

[52] ClamAV (2020). ClamAV. Retrieved from ClamAV Github site. https://github.com/Cisco-Talos/clamav-devel

[53] BBC (22/07/2019). Joe Tidy. Retrieved from BBC News. https://www.bbc.com/news/business-49015609