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

1 Introduction 1

2 Malware and malware samples 3
   2.1 The state of the art in malware samples 4
   2.2 Antivirus telemetry 7
       2.2.1 Analysis 7
       2.2.2 Conclusions 19
   2.3 Cyber threat hunting telemetry and samples submission 20
       2.3.1 Analysis 20
       2.3.2 Conclusion 22
   2.4 Operating system telemetry and samples submission 23
       2.4.1 Analysis 23
       2.4.2 Conclusion 24
   2.5 Intelligence products 25
       2.5.1 Analysis 25
       2.5.2 Conclusion 26

3 Universal Malware Sample Encryption (UMSE) 29
   3.1 Current malware sample shortcomings 29
   3.2 Universal Malware Sample Encryption 31
       3.2.1 UMSE universality 31
       3.2.2 UMSE as a sample format 32
       3.2.3 UMSE confidentiality 32
       3.2.4 UMSE authentication 32
   3.3 UMSE format specification 32
       3.3.1 Formal specification 32
       3.3.2 Overview 38
       3.3.3 UMSE header 39
Contents

3.3.4 Decryption table ........................................... 39
3.3.5 Entries ..................................................... 40
3.3.6 File properties ............................................. 40
3.3.7 Authentication header ...................................... 40
3.3.8 RSA private key ............................................ 41
3.4 UMSE implementation ........................................ 41
  3.4.1 UMSE C/C++ library ...................................... 41
3.5 UMSE agent .................................................... 45
  3.5.1 UMSE server ............................................... 46
  3.5.2 UMSE Shell ............................................... 47
  3.5.3 UMSE tools ............................................... 47

4 Conclusions and future work .................................. 49

Bibliography ...................................................... 51
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
1. **Introduction**

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
2. Malware and malware samples

Malware context is absolutely fundamental during analysis\[10]\[11] so, it must 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
2.1. The state of the art in malware samples

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).  

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.

---


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
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, TelemetryControllerImpl::ReportMalwareStream in this case:
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.
2. **Malware and malware samples**

![Telemetry API](image)

**Figure 2.4:** Malwarebytes telemetry API.

```sql
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
```
2.2. Antivirus telemetry

Since Microsoft Winsock functions are used for network communications, we can also break in Send and SendTo functions and show the 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 submitted 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:
2.2. Antivirus telemetry

Figure 2.5: Debugging Malwarebytes.
2. Malware and malware samples

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
2. Malware and malware samples

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
2.2. Antivirus telemetry

CloudControllerImpl.dll library file:

1. SubmitDDSSample
2. SubmitDoppelgangDetection
3. SubmitExploitData
4. SubmitMWACDetection
5. SubmitQuarantineRestoreItem
6. SubmitRansomwareDetection
2. Malware and malware samples

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:
2.2. Antivirus telemetry

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”.
2.3. Cyber threat hunting telemetry and samples submission

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.\(^\text{1}\)

1. Alerts on Microsoft Defender Security Center.

2. Machine information, including OS information.

---

2. Malware and malware samples

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
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\textsuperscript{34} (see Figure 2.10) purpose.\textsuperscript{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

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,
2. Malware and malware samples

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.

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.
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].

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

Therefore, all antiviruses are good in detection and this explains different
3. Universal Malware Sample Encryption (UMSE)

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,
3.2 Universal Malware Sample Encryption

3. Contextualization shortcoming: Context is lost. The sample is usually 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 Encryption

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-
3. Universal Malware Sample Encryption (UMSE)

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.

3.3 UMSE format specification

3.3.1 Formal specification

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

```plaintext
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"
```

33
3. Universal Malware Sample Encryption (UMSE)

```plaintext
- id: author_name
type: str
size: author_name_length
encoding: UTF-8
doc: "Author name"
decryption_table:
  seq:
    - id: level_of_confidentiality
type: u1
  - id: aes_wrapped
type: u1
repeat: expr
repeat-expr: 256
entry:
  seq:
    - id: size
type: u4
  - id: level_of_confidentiality
type: u1
  - id: encrypted_message
type: u1
repeat: expr
repeat-expr: size
  - id: num_metadata
type: u4
  - id: entry_metadata
type: entry_metadata
repeat: expr
repeat-expr: num_metadata
if: num_metadata > 0
entry_metadata:
  seq:
    - id: tag
type: u1
repeat: expr
repeat-expr: 8
  - id: length
type: u4
  - id: value
type: u1
repeat: expr
repeat-expr: length
```
3.3. UMSE format specification

```plaintext
file_properties:
seq:
- id: level_of_confidentiality
type: u1
- id: hash_value
type: u1
repeat: expr
repeat-expr: 32
- id: num_metadata
type: u4
- id: file_metadata
type: file_metadata
repeat: expr
repeat-expr: num_metadata
if: num_metadata > 0

file_metadata:
seq:
- id: tag
type: u1
repeat: expr
repeat-expr: 8
- id: length
type: u4
- id: value
type: u1
repeat: expr
repeat-expr: length

authentication_header:
seq:
- id: length
type: u4
doc: "HMAC length"
- id: hmac
type: u1
repeat: expr
repeat-expr: length
doc: "HMAC"

rsa_private_key:
seq:
- id: length
type: u4
if: not _io.eof
doc: "RSA private key length"
- id: rsa_private_key
type: u1
repeat: expr
repeat-expr: length
if: not _io.eof
doc: "RSA private key"
```
For practical purposes, an UMSE format structure parser template was also developed:

```c
//--- 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
// ID Bytes: 55 4D 53 45 //UMSE
// History:

//------------------------------------------------

struct ENCRYPTED_SAMPLE {
    struct HEADER {
        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">;
    }

    decryptionTable [ header.numRecordsDecTable ];
};
```

3. Universal Malware Sample Encryption (UMSE)
3.3. UMSE format specification

```c
struct ENTRY
int size <comment="Size of this entry">;
char levelOfConfidentiality <comment="Level of confidentiality of this entry">;
char encryptedMessage[entry.size] <comment="Encrypted chunk">;
int numMetadata <comment="Number of public metadata entries of this chunk">;

struct ENTRY_METADATA
char tag[8] <comment="TAG of this metadata">;
int length <comment="Size of this metadata content">;
char value[length] <comment="metadata value">;
entry_meta[entry.numMetadata] <optimize=false>;
entry[header.numFileEntries] <optimize=false>;

struct FILE_PROPERTIES
char levelOfConfidentiality <comment="File confidentiality level">;
char hashValue[32] <comment="Hash value">;
int numMetadata <comment="Number of public metadata entries of this file">;

struct FILE_METADATA
char tag[8] <comment="TAG of this flag">;
int length <comment="Size of this metadata content">;
char value[file_metadata.length] <comment="metadata value">;
file_metadata[file_properties.numMetadata] <optimize=false>;
file_properties;

struct AUTHENTICATION_HEADER
int authenticationLength <comment="Authentication length">;
char authentication[authentication_header.authenticationLength] <comment="HMAC SHA256 Authentication. The key is the sha256 of the encryption table but after entire decrypted">;
authentication_header;

if(!FEof()) {
struct RSA_PRIVATE_KEY {
int rsaPrivateKeyLength <comment="RSA Key Length">;
char rsaPrivateKey[rsa_private_key.rsaPrivateKeyLength] <comment="RSA Key">;
} rsa_private_key <comment="RSA Key (This is an optional field)">;
}

encryptedSample;
```
3. Universal Malware Sample Encryption (UMSE)

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. UMSE format specification

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.

3.3.4 Decryption table

Next, there is the Decryption table. This table contains AES keys and IVs associated 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.
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.

![Structured entry example]

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

![Structured file properties example]

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
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\cite{48}\cite{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 wants 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\cite{51} (see Figure 3.2):

- **bin**: 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.
3.4. UMSE implementation

The project content can be summarized as follows (Figure 3.2):

- 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.
### 3. Universal Malware Sample Encryption (UMSE)

<table>
<thead>
<tr>
<th>Description</th>
<th>This function converts a chunk to Universal Malware Sample Encryption format.</th>
</tr>
</thead>
</table>
| **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.

<table>
<thead>
<tr>
<th>Description</th>
<th>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.</th>
</tr>
</thead>
</table>
| **Input parameters** | chunkLength  
chunk  
entryToDecrypt  
accessLevel  
rsaPrivateKey |
| **Output parameters** | decryptedEntryLength  
decryptedEntry |
| **Return value** | -2 if integrity check is 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.
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.

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

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.
3. Universal Malware Sample Encryption (UMSE)

![Figure 3.3: Structure of the UMSE server.](image)

### 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_seh-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).
3.5. UMSE agent

- 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.

![Universal Malware Sample Encryption](image)

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:

```
class MyCmd(base):
    prompt = PROMPT

    def do_decrypt(self, args):
    def do_login(self, self, args):
    def do_list(self, self, args):
    def do_download(self, self, args):
    def do_clear(self, self, args):
    def do_exit(self, 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 benefitting himself by using a specific format to share malware files and, in a future
3. Universal Malware Sample Encryption (UMSE)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>do_list</td>
<td>List samples stored in the UMSE server.</td>
</tr>
<tr>
<td>do_download</td>
<td>Download samples from the server.</td>
</tr>
<tr>
<td>do_login</td>
<td>Login into the server.</td>
</tr>
<tr>
<td>do_decrypt</td>
<td>Decrypt an entry of an UMSE sample (login is required).</td>
</tr>
<tr>
<td>do_clear</td>
<td>Clear screen.</td>
</tr>
<tr>
<td>do_exit</td>
<td>Exit the program.</td>
</tr>
</tbody>
</table>

A 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\cite{10}\cite{11}\cite{13}\cite{14}\cite{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\cite{53}, but maybe they collaborate with a third party who also uses intelligence for purposes unrelated to malware analysis\cite{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 \texttt{xxxToUmse.cpp}. For instance, \texttt{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


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

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

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


[16] Microsoft Portable Executable and Common Object File Format Speci-

dlls/dynamic-link-libraries?redirectedfrom=MSDN

[18] NIST (April, 2013). Rebecca M. Blank and Patrick D. Gallagher. Se-
curity and Privacy Controls for Federal Information Systems and Orga-
SpecialPublications/NIST.SP.800-53r4.pdf


gov/glossary/term/malware

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

[22] Lubold, Gordon; Harris, Shane (6 October 2017). “Russian Hackers
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


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

