Testing antivirus engines to determine their effectiveness as a security layer

Jameel Haffejee
Department Of Computer Science
Rhodes University
Grahamstown
JameelHaffejee@gmail.com

Barry Irwin
Department Of Computer Science
Rhodes University
Grahamstown
b.irwin@ru.ac.za

***Abstract*—This research has been undertaken to empirically test the assumption that it is trivial to bypass an antivirus application and to gauge the effectiveness of antivirus engines when faced with a number of known evasion techniques. A known malicious binary was combined with evasion techniques and deployed against several antivirus engines to test their detection ability. The research also documents the process of setting up an environment for testing antivirus engines as well as building the evasion techniques used in the tests. This environment facilitated the empirical testing that was needed to determine if the assumption that antivirus security controls could easily be bypassed. The results of the empirical tests are also presented in this research and demonstrate that it is indeed within reason that an attacker can evade multiple antivirus engines without much effort. As such while an antivirus application is useful for protecting against known threats, it does not work as effectively against unknown threats.**

Keywords : Malware, Antivirus, Defense

I. Introduction and Background

A. Background

While antivirus has existed as a security control layer to protect end users from malicious application for the past 15 years [1]. The basis of how these antivirus application exercises their method of protection has remained static [1].

This largely unchanged method of detection has led to a common belief that antivirus engines are an ineffective means of securing a system, because there is a tendency for them to provide a false sense of security. This belief is based on the assumption that it is trivial to bypass an antivirus engines method of detection if an attacker knows what to avoid.

Antivirus applications primarily rely on a database of known malicious security artifacts that are compared to information that is presented to the antivirus application for analysis. Information that is matched to a signature by the antivirus is then assumed to also be malicious.

This method of detection has a number of problems. By associating any application that matches a specific signature as malicious, antivirus applications commonly falsely flag applications as malicious. This method of detection also means that most malicious applications will only be detected after a researcher working for an antivirus company creates a signature for the malicious application. Until this signature is released to the end users, a users system will still be vulnerable to the malware. As previously when an attacker is aware that an antivirus engine is checking for a specific signature they are able to then craft a new malware application that avoids exhibiting this signature.

B. Defense in depth

Defense in depth is commonly recommend as a security system to protect networks and is included in the National Institute of Standards and Technology [2]. Yet we commonly find systems that do not implement it completely or when implemented they do not use the recommendations [3]. Most notable is the use of Antivirus applications as the primary means of host based defenses without maintaining the system [4]. Further, more when investigating the implementation of the defense in depth strategy we find that the same strategy can applied to the antivirus defense layer. This is determined since antivirus engines have to cover multiple areas of responsibility. Since these areas are only covered by a single application, there is a greater chance of failure [2].

Because of this single point of failure computer security defenses can be modeled as a zero sum system [5]. Where a breach in a system's defenses can allow an attacker complete control of the system given enough time. By modeling the system in this manner and then isolating the system into layers [6], the security system can be isolated and visualized for stricter control [5]. Each layer either prevents an attacker from gaining any access or alerting a system administrator that a potential breach is in progress.

With the usage of malware applications on the increase in recent years according to data collected in the 2014 Verizon Data Breach Investigation report [3], antivirus vendors are trying more than ever to to ensure their product of choice is the standard. While this in itself is not a problem, it is the false sense of security that these products provide that leads companies to be negligent on numerous other systems that should also been in place to protect the system. It should be no surprise as well that the antivirus solutions that are sold to companies to protect them from malware are also available to the malware authors. This resulted in the malware authors crafting malware that can evade an antivirus engine and bypass the end users protection completely. By the time a malware

978-1-4799-3383-9/14/$31.00 ©2014 IEEE

analyst does get to analyze a copy of the malware the damage has already been done.

C. What is an effective security layer

With this in mind it would be useful to define what constitutes an effective security layer. While the effectiveness of a security layer can vary depending on its use within the overall security structure, ideally the security layer should fail closed. This means that if a problem were to occur with the operations of the system that constitutes the security layer, the system would prevent all access. As stated previously, this does depend on the implementation of the system. For example, a monitoring layer can not entirely fail closed since increasing the notification limit can end up overloading the operator, which is an unintended side effect.

D. Antivirus Overview

Having largely covered what is expected from an effective security layer, it is worth giving a short overview of the tasks required from an antivirus engine. According to Microsoft's Safety and security center [7], an antivirus application is "A computer program that detects , prevents and takes action to disarm or remove malicious software programs".

While this definition of an antivirus application does cover a number of a areas, for this paper we are primarily concerned with prevention and detection. These areas are most concerning to an attacker, as it would prevent a malicious application from running if it was detected. If an application is not flagged correctly, the attacker now has the potential to change a number of inconspicuous items on the system that would give them remote access or allow the attacker to extract information that can allow an attacker alternate means of access.

With the basic introduction and background covered above the rest of the paper is organized as follows. Section two covers the methodology that will be used for the testing in section three. Section three covers the baseline tests as well the final tests with the evasion techniques applied. Finally section four covers a summary of the results from section three and a direction to take future work.

II. Methodology

A. Overview

The work conducted tested how antivirus engines operate from a black box perspective. Furthermore, the approach taken in this paper is different from the approaches taken by Jon Oberheide [8], [9] and does not rely on setting up and maintaining multiple virtual machines. Instead the samples are collected locally and then sent for remote analysis to the VirusTotal web service. Once the analysis is completed remotely, the results are then stored locally for later analysis.

B. Virus Total

The choice to use VirusTotal for the testing process was a natural process as it would save both setup time and licensing costs. Furthermore, VirusTotal is the largest publicly available collection of malware reports which provides the researcher with the ability to flag false detections and skip the scanning process when the hash signature of the binary matches one of a previously scanned binaries.

While these options do make the VirusTotal ideal for research and testing, it does not allow for rapid scanning of millions of files in a short period of time as a regular desktop antivirus application would. The scanning of the binaries submitted to virustotal are scanned by the same antivirus engine that would run on a desktop personal computer, except the scan would be manually initiated versus a background scanner that would run on a desktop system.

C. Baseline Binary

Unlike previous research done in this area, the antivirus engines are not tested by simply scanning random malware samples collected from the Internet. Instead a binary that is known to be flagged as malicious but is not an inherently malicious application will be used for testing. This baseline binary that gets detected as malicious is required so that we have full control over the binary that is generated. It also provides a means to test whether the application is scanning for signatures only or if the actions of the binary are also being analyzed.

D. Baseline Tests

A basic baseline test will be completed with the baseline binary to determine if the binary is actually detected as malicious by VirusTotal. The baseline tests will be run with multiple binaries to determine which binary is the most suitable.

Once a baseline is determined, a number of known evasion techniques will be applied in an attempt to make the binary bypass detection by the antivirus. These tests will then be recorded and the results used to determine the effectiveness of an antivirus application at detecting known malicious applications.

E. Evasion Techniques

As mentioned in the previous subsection, once a baseline binary is selected after analyzing the different requirements for a baseline binary, the selected evasion techniques will be applied to it and then submitted for rescanning.

The evasion techniques selected for testing, are some of the initial evasion techniques used during the rise of antivirus engines. These techniques are then categorized as packers or crypters. These techniques offer a large margin when it comes to complexity. This means that it is possible to build custom implementations of each of the techniques that will still follow the same pattern and idea, but are implemented differently. This in turn means that we are able to analyze the effectiveness of techniques versus prior implementations which might have unique signatures that are present when used.

III. Baseline Testing Process

A. Baseline Testing Process

a) Testing Process: The testing process will proceed according to the following steps.

- Build Malware Sample. A base malware sample will be built and then wrapped for later submission to VirusTotal. This will be known as the baseline binary.
- Submit sample to VirusTotal. The sample will be submitted to VirusTotal using the web interface.
- Scan the baseline binary and record results for later analysis. Details of the sample such as the technique used , date scanned and number of Antivirus engines that detected the binary as malicious will be recorded for later analysis.
- Scan the baseline binary plus and evasion technique and record results for later analysis. Same as previous point.

1) Baseline binary properties: Before selecting a binary that will be used for the baseline testing, a number of properties will be defined that dictate if a binary is suitable for use in testing. For an application to be considered, it needs to have the following properties:

- Recognized by multiple antivirus applications as malicious or flagged as an anomaly for further investigation. The multiple detections are required to ensure that a single antivirus is falsely flagging an application.
- Can be executed safely. The effects of the application need to be carefully defined such that a researcher will not cause unknown harm to their system by running the application. This will allow the application to be run without the need for a sandboxed environment and also allows the researcher to recognize what triggers an antivirus engine [10].
- Must have source code available. While this is not a high priority for the tests in this paper, for continuity of future tests which require that an application re write itself, the source becomes important (as in the case of metamorphism).

In the next section each of the baseline applications that will be tested will be run through the checks detailed in the items above to evaluate if they meet the requirements for testing an evasion technique. Once a suitable application is selected, it will be combined with an evasion technique and submitted to virustotal for testing.

IV. Baseline Results

A. Netcat

Based on the properties defined in section 3, the Netcat application was initially selected for testing. The Netcat application is a network utility application commonly used to debug network issues. However the application was commonly distributed with a number of malicious dropper binaries that allowed an attacker to gain remote access.

B. Property Testing

By running through the properties defined, we can test that Netcat fits all the requirements. To validate that the application is detected as malicious by multiple antivirus engines, we scan the original that is available for download from the following location[17] and the download can be validated with the following sha1 hash "2d3026b4630789247abf07aa3986d7a697cf4cd".

a) Multiple Malicious Detections: Once the application is downloaded, a compiled copy of the binary can be acquired by extracting the zip file and running the test with the following command from the command line:

"python scan.py nc.exe"

This python script simply automates the process of uploading the file to VirusTotal and saving the results to file on the local disk. After executing the script and loading the results, the following information will be found :

```
SHA256: 7379c5f5989be9b790d07
1481ee4fdfaeeb0dc7c4566cad836
3cb016acc8145e
File name : nc.exe
Detection ratio : 21 / 46
Analysis date : 2013-09-10
```

The results confirm that the binary is indeed detected by multiple antivirus engines as malicious. It also provides us with a sha256 hash which is more accurate. This will also allow us to lookup results on VirusTotal in future and extract the results from the report above.

b) Safe Execution: The next property that needs to be tested is the safe execution of a binary. By default, when compiled from source the Netcat application does not have any special actions. It simply connects to a remote port or opens a remote port locally, which others can connect to. Unfortunately without significant work into reverse engineering the application or running the application through multiple tests in a sandboxed environment that records all environmental actions, it can not be said the application is not performing other unknown actions.

Since we have access to the source code of the application, we can compile a copy of Netcat that we can ensure is not performing any actions aside from what it was originally intended to perform. The compilation of the application is covered in the documents distributed with the application and as such will not be covered here.

Once the application is compiled it can then reliably be said that the application will perform as expected without modification to its original intent. At this point it would be advisable to perform the previous test again to ensure the custom compiled version of our application still meets the previous requirement.

After re running the scan with the same command as before:

"python scan.py nc.exe"

The command will result in the following output.

```
SHA256 : 087a3c776bde51857c7
4897dc0e3b0f8a6725ab124edcf1
4e4a80d7c454fa4cd
File name : nc.exe
Detection ratio : 1 / 46
Analysis date : 2013-09-15 18:31:02 UTC
```

These results are unfortunately not ideal for the tests that

we want to perform. If simply recompiling the application is enough to bypass an antivirus engine it points to the antivirus engine depending on very simple signature verification for its detection routines.

c) Source Code: The Netcat application is primarily distributed via source code in a zip archive, so getting access to the source is not a problem.

C. Discontinuation of Netcat

While the initial results were promising, the issue raised by simple recompilation is unfortunately a problem that can not be ignored. As such an alternative base application was found.

D. Alternative Binary

With the Netcat application ruled out the Metasploit Payload binary will be tested for validity. The Metasploit payload binary is distributed with Metasploit framework and is used as a shell to deliver a users payload upon execution on a targets system.

When analyzing this scenario it can be compared very closely to the process that malware authors tend to follow when trying to gain access to a remote system. As such this means that the application will more closely reflect an attack in the wild and its detection or evasion will similarly reflect scans in the wild.

E. Testing Metasploit Binary

The Metasploit payload binary is not distributed as a single binary, instead the application inserts the payload the user wants into a template binary. This template binary is what will be used for testing. The template combined with a simple payload to launch a calculator application will be embedded to ensure that the application is considered active and executable. Further, only the x86 version will be tested, as this is the most common architecture that is available for testing. The specific file that will be used for testing can be found in the Metasploit distribution under the "exe_templates" folder and is named "template_x86_windows.exe".

d) Multiple Detections: Submitting the application to VirusTotal for scanning with the command :

"python scan.py template_x86_windows.exe"

Will result in the following results.

```
SHA256: 640fc87b5754d9191
a6d5326d73e105c01bb3ada21
033e3f54bbde250fb59a15
File name:  ad21c93553af2
3ecec319c0ea5f11b755acc3342
Detection ratio: 13 / 47
Analysis 2013-09-15 2013-07-09 11:56:24 UTC
```

While the template did not get flagged by as many antivirus engines as the Netcat application, it is still enough for use in the tests. It should be noted though that the higher detection rate is still most likely attributed to signature comparison.

e) Safe execution: The template binary is an inert application and does not have any functionality without a payload being embedded. As such until a user inserts a payload, the application can not cause any side affects on the system that it is executed on.

f) Source Code: The source code for all the templates is distributed along with the rest of the Metasploit framework.

As we demonstrated with the original Netcat custom build from source we need to compile and rescan a local build to ensure that the previous results were not based of a static signature.

Firstly it is recommended that the build be completed using mingw, while cygwin does provide similar tools it is more error prone. For the specific instructions on how to install mingw refer to the section in which Netcat was built. The current version of the tests are being run on a Windows 7 pc with mingw as described earlier.

Next download the template file from [11] and save it to a file called template.c on disk. There is no make file to build the template as the original indicating how the author may have built the binary, as such simply running the following command will build the template binary.

gcc template.c -o template_gcc.exe

Results For Template Scan :

```
SHA256: 76de5d51b259a52045de
8571995d359c8a398b8487862811
d2d31b8af8b5df6f
File name:  template_gcc.exe
Detection ratio:     2 / 47
Analysis 2013-12-08 2013-12-08 11:49:32 UTC:
```

While the basic compilation with gcc does provide some favorable results , the number of Antivirus engines that detected the binary as malicious is low (2/47). This is most likely due to the fact that the application does nothing at all besides crash when run. We will next add some basic operation codes which will be used to launch the calculator application. This will allow the application to at least run without crashing and possibly trigger more detection routines.

The opcode template compiles to the same binary as would be generated by the msfpayload application when using the template_x86_windows.exe binary. Note that the binaries will not be identical to the byte level as they are compiled on different computers.

The template file with the opcodes added can be found in template_op.c. Compile it with the following command

gcc template_op.c -o template_x86_windows_op.exe

Once the application is compiled, the binary can be submitted to virus total again. This will generate the following results :

```
SHA256: 1874c340ba2e140aa4c
c825459bf727bd4decb68cffc90
0501f971033a32dafb
File name:  template_gcc_op.exe
Detection ratio:     6 / 47
```

```
Analysis 2013-12-08
```

With these results we get slightly better detection rates (6/47) than the initial version of the scan. These slightly lower detection rates work in the favor of future analysis as future scans with an evasion wrapper but a higher detection rate means that the AV engines are detecting the wrapper instead of the base application.

The test application can not be compiled with the Microsoft compiler as the compiled version prevents any potential exploit code from running.

F. Scanning baseline binaries with known evasion techniques

This last section will scan the baseline binary one more time with each of the evasion techniques custom implementations applied. Kevin Roundy and Barton Miller [12] discuss a number of common packers and encrypters that were tested as well as their techniques that assist in antivirus evasion. Based on the detailed analysis performed it is safe to assume that antivirus engines have performed similar analysis and as such have signatures to detect these packers or encrypters. To ensure that the antivirus engines are not simply detecting known signatures from these applications, a custom packer and a custom encrypter will be built for the tests.

g) Packer Implementation: A summary of the technique commonly used for packing a binary is a follows. The author selects a method for compressing the binary. Depending on the level of sophistication the author will either build a utility application that will pack the target application or simply write a once off script that will pack the target application.

The dropper application which is the output of the utility application mentioned in the previous paragraph. The dropper application is comprised of an unpacker executable is pre-built and injected with a target application. The target application in this case is any application that the attacker wishes to use to bypass an antivirus engine.

The platform and language selected to implement both the packer and dropper is the Microsoft dot Net framework and the C# language. The packer and dropper have both been implemented as discussed above. The full details of the implementation are available on request and have not been included here due to space requirements.

h) Encrypter Implementation: A summary of the technique used for encrypting a binary which will be used later is a follows. The author of the binary selects and encryption algorithm or builds their own. Next an encrypter as well as dropper is built in the same way as the packer binary. While this is not the only method to build an encrypter or the most sophisticated, it is the simplest and allows for a large amount of randomization with regards to the parts that are used. As with the packer the encrypter is built using the .Net framework using C#. The encryption used in the implementation is AES. Further the payload is encrypted directly and is not zipped before encryption.

i) Evasion Tests: For each of the tests (the packer and encrypter) we build a binary by wrapping the Metasploit payload with one of the evasion techniques. This will result in a dropper application that when executed will either extract the Metasploit payload or decrypt it and then subsequently run the payload.

The scan results from uploading the packer binary can be found below, the link to the scan and its details can be found here [13].

```
SHA256: df06d7bc21f629b255ece
e954fabf668761b20390ef3fa2612
cb1621cde91826
File name:  SelfExtractor.exe
Detection ratio:     2 / 49
```

Antivirus	Result	Update
Avast	Win32:Malware-gen	20140318
Malwarebytes	Backdoor.Bot.gen	20140318

As can be seen only two applications have detected the program as malicious.

The scan results from uploading the encrypter binary can be found below, the link to the scan and its details can be found here [14].

```
SHA256: 4aad495e717686faeb47b92
49545482d0831f2d1eaafbff637128e
a90ef60d86
File name:  SelfDecrypter_OP.exe
Detection ratio:     0 / 49
```

Unlike with the previous tests none of the antivirus engines detected the binary as malicious.

V. Conclusion

As can be seen from running two of the simplest tests using the latests signatures from the all the antivirus engines, once a malicious application is wrapped with a basic antivirus evasion technique it is able to bypass a significant number of known antivirus engines at the time.

Considering the simplicity involved with creating and then modifying these implementations of each of the investigated evasion technniques it should be reasonable to consider that a malware author with a financial incentive can create a similar or better implementation of these techniques.

With this in mind any institute that deals with financials of others should not consider antivirus engines an effective means of protection.

While these tests were performed with VirusTotal they could have been performed with a number of other companies that provide a similiar service. Performing these tests on other systems and comparing the results is left as future work. Furthermore by performing the tests with a provider that has a application programming interface (API) future research can re run these tests and compare the results to those that were done previously. This provides a way to compare past results and base future tests against previously completed work.

REFERENCES

[1] C. Nachenberg, "Computer virus. coevolution." http://www.cs.bgu.ac.il/~dsec121/wiki.files/j7b.pdf.

[2] R. Harrison, "The antivirus defense-in-depth guide." http://www.bitdefender.com/files/KnowledgeBase/file/Antivirus_Defense-in-Depth_Guide.pdf.

[3] Verizon, "Verizon data breach investigations report." http://www.verizonenterprise.com/DBIR/2014/reports/rp_Verizon-DBIR-2014_en_xg.pdf, 2014.

[4] K. R. Straub, "Information security managing risk with defense in depth." http://www.sans.org/reading-room/whitepapers/infosec/information-security-managing-risk-defense-in-depth-1224.

[5] O. O. Ibidunmoye E.O., Alese B.K., "Modeling attacker-defender interaction as a zero-sum stochastic game." http://pubs.sciepub.com/jcsa/1/2/3/.

[6] I. Systems, "Security system and internet security." http://pic.dhe.ibm.com/infocenter/iseries/v7r1m0/index.jsp?topic=%2Frzaj4%2Frzaj40a0internetsecurity.htm.

[7] Microsoft, "What is antivirus software." http://www.microsoft.com/security/resources/antivirus-whatis.aspx.

[8] F. J. Jon Oberheide, Evan Cooke, "Cloudav: N-version antivirus in the network cloud." http://static.usenix.org/events/sec08/tech/full_papers/oberheide/oberheide_html/.

[9] J. C. H. Orathai Sukwong, Hyong S. Kim, "Commercial antivirus software effectiveness: An empirical study." http://www.computer.org/csdl/mags/co/2011/03/mco2011030063.pdf.

[10] A. Mushtaq, "The dead giveaways of vm-aware malware." http://www.fireeye.com/blog/technical/malware-research/2011/01/the-dead-giveaways-of-vm-aware-malware.html.

[11] Rapid7, "Metasploit template base." https://github.com/rapid7/metasploit-framework/blob/b6458d2bfa54fa33801da1f62e418ba000e45477/data/templates/src/pe/exe/template.c.

[12] B. M. Kevin Roundy, "Effectiveness of antivirus in detecting metasploit payloads." ftp://ftp.cs.wisc.edu/pub/paradyn/papers/Roundy12-Packers.pdf.

[13] J. Haffejee, "Virustotal packer scan results." https://www.virustotal.com/en/file/ea39f07cf2200fbf315f2294922ede4aa261d3edbddfc73de4186e29b2515b89/analysis/1395166900/.

[14] J. Haffejee, "Virustotal encrypter scan results." https://www.virustotal.com/en/file/4aad495e717686faeb47b9249545482d0831f2d1eaafbff637128ea90ef60d86/analysis/1395170705/.