Applied Comparative Evaluation of the Metasploit Evasion Module

Peter Casey[†], Mateusz Topor[†], Emily Hennessy[†], Saed Alrabaee[*], Moayad Aloqaily[⋆], and Azzedine Boukerche[‡]
[*]College of IT, United Arab Emirates University (UAEU), Al Ain, UAE
[†]School of ECE and Computer Science, University of New Haven, NH, USA
[⋆]Gnowit Inc., Ottawa, ON, Canada
[‡]University of Ottawa, Ottawa, ON, Canada
[†]{pgrom1@unh.newhaven.edu, mtopo1@unh.newhaven.edu, ehenn2@unh.newhaven.edu}
[*]salrabaee@uaeu.ac.ae, [⋆]moayad@gnowit.com, [‡]boukerch@site.uottawa.ca

***Abstract*—The great revitalization of information and communication technologies has facilitated broad connectivity to the Internet. However, this convenience in terms of connectivity comes with costly caveats, including internet fraud, information damage or theft, and cybersecurity issues. Most individuals rely on anti-virus software for protection. This anti-virus software has long been a foe to malware authors, but there are brief moments when new techniques slip through the cracks, and even the most sophisticated engines sometimes fail. A new tool, namely Metasploits new evasion modules, claims to exploit that. In this study, we compare and evaluate legacy evasion techniques with the novel tactics presented by Metasploits lead researcher Wei Chen. We consider the benefits and pitfalls of each technique and evaluate the new modules successes (or *failures!*).**

***Index Terms*—Computer viruses, Anti-viruses, computer security, communication secuty, security methods.**

I. Introduction

With the endless stream of breaches and vulnerability announcements, system administrators are forced to view computer and network security from the perspective of *when* rather than *if* penetration is possible. An attacker who gains access to a system will inevitably leave their mark [1] [2] even if only in memory [3] [4]. Thus, anti-virus protection is an essential (and often default) line of defense against malicious actors and programs.

Anti-virus software is both low cost and convenient for most network and host configurations. Popular client operating systems, such as Windows and macOS, come pre-configured with such protection. Windows provides *Windows Defender* with Windows 8 and later, while versions of macOS starting with *Snow Leopard* is configured with XProtect [5]. The ubiquity of this defense ensures that anti-virus evasion will remain at the forefront of an attacker's concern, but it may also create a false sense of security for end users.

Anti-virus detection and evasion has long been a back-and-forth battle between attackers and defenders. Both malicious actors and researchers are continuously improving evasive tactics while anti-virus developers craft their retaliation. Previously, anti-virus software was exclusively responsive, implying that malware had to be observed before a signature could be identified; however recently multi-layered techniques even allow for protections against some zero-days. Furthermore, in an effort to reduce the forensic footprint, malware tends to remain exclusively in memory, forcing detection to occur at delivery or run-time. New advanced detection techniques include heuristic, behavioral and run-time analysis and many anti-virus packages allow for potential malware samples to be offloaded to a cloud infrastructure, providing a more robust analysis [6]. Despite these improvements, defenders will endlessly be forced to prepare for the unknown. Other efforts have been seen in the domain of deep learning and AI for network traffic filtering and protection [7] [8].

The attackers, who drive the progression of anti-virus, are now faced with evading not only signature-based detection, but a suite of sophisticated detection mechanisms. Simply creating a unique binary may defeat a signature-based detection; however, the functionality of the payload must now also be veiled. Phases 2 and 3 of the Lockheed Martin Cyber Kill Chain involve weaponization and delivery of a payload [9]; the phase in which the attacker must consider concealing and packaging their payload to avoid detection. Some of the techniques commonly employed include manual binary editing and custom source code, encoding, encryption, execution templates, and functionality to detect sand-boxing and debugging processes. Anti-virus developers are keenly aware of the 'script kiddies' tool-box, and can tailor their algorithms to identify the product of each.

To further complicate the scenario, testing newly generated payloads is representative of the Heisenberg uncertainty principle, where observation has an effect on the result [10]. Simply testing the effectiveness of a payload against an anti-virus will spoil the ambush, as the signature database will quickly reflect the new payload. As a direct consequence, successful tactics remain closely guarded by their developers.

Metasploit has long been a popular framework for reconnaissance, exploitation, and actions-on. In October 2018, a new module has been added to bleeding-edge technology that allows for payload developers to utilize many of the previous popular techniques with the flexibility of generating unique payloads. These add-ons have been dubbed evasion modules. The contribution of this study is to compare the effectiveness of previously popular tools against the newly announced Metasploit modules.

II. Anti-virus Evasion Techniques

This section will discuss methods used by malware creators to attempt to evade anti-virus detection.

A. Manual Binary Editing

The goal of manual binary editing is to identify the signature that is causing the binary to be flagged and change that portion without having to recompile the entire executable. This technique helps to evade signature-based detection; however, it does not attempt to mask the payload functionality. Using tools such as *Dsplit*, the signature can be quickly identified [11]. Dsplit can be used to break files into smaller segments, which are then fed into the anti-virus software to be tested against. Each smaller portion that is still flagged by the anti-virus can be assumed to contain a malware signature and further segmented. This trial and error process can be repeated until the segment sizes are manageable and closely reflect the signature size. Traversing a segment byte-by-byte, producing constant size segments can determine the signatures start and stop location [11]. Unfortunately, directly modifying a binary is likely to have adverse effects on the functionality of the executable. Inserting new opcodes, even those that produce no significant operations, is likely to offset any following absolute jump instructions. The simplest modification that would not have any adverse impact would be changing the value of a string. Of course this only applies if a string is present in the signature.

B. Polymorphic Code

Polymorphic code is used to evade anti-virus software by mutating each time it runs. While the code may be changing, the algorithm itself is kept intact. This helps the malware hide itself. Encryption is commonly used to hide malicious code, and when this technique is used the main body of the written code will appear meaningless. There will be a decryption function added to the code that will execute before running the main body of the code. In addition, the encryption and decryption pair will be mutated together with each copy of mutated code.

III. Apparatus

The workstation for the experimental phase was as Windows 10 desktop computer, with details provided in Table I.

TABLE I
Workstation Details

Device	Details
Processor	Intel Core i7-6700 CPU
System Type:	64-bit OS, x64 processor
Graphics Card	NVDIA GeForce GTx 1070
Manufacturer	iBUYPOWER
Installed Memory (RAM)	16.00 GB

Three virtual machines (VMs) were utilized; the first for payload creation (Kali Linux), testing Windows Defender (Win1) and testing Norton Security by Symantec (Win2). The details of these machines are provided in Table II, with the only difference between VM Win1 and Win2 being the anti-virus provider. All virtual machines and software was fully updated during preparation for this research. The tools presented in Table III were installed on Kali Linux VM for the purpose of creating or modifying a payload to evade anti-virus detection.

TABLE II
Virtual Machine Details

	Kali Linux	Win1 and Win2
Operating System	Kali Linux 2018.4	Windows 10 Home
Version	4.2018.0-kali2-amd64	10.0.17134.376
Cores	2	2
Memory	2GB	8GB
HTTP Server	Apache 2.4.35 (Debian)	N/A

Each application was either acquired directly from the developers repository, or for those shipping with Kali Linux, updated prior to use.

TABLE III
Evasion Application Details

Application	Version
Hyperion	1.2
MSFVenom	4.11.4
peCloaky.py	N/A
Veil Framework	3.1.11
Metasploit	v5.0.0-dev

IV. Methodology

We begin our exploration of anti-virus evasion with an evaluation of traditional and legacy techniques. A selection of both common and arbitrary payload types will be tested with the popular evasion techniques described in Section II. Furthermore, we will be comparing the detection rate of each for both Windows Defender and Norton Security by Symantec Norton. Three payloads were selected for testing with each tool and anti-virus: windows/meterpreter/reverse_tcp, windows/powershell_reverse_tcp, windows/shell/reverse_tcp

These payloads were selected because of their popularity among penetration testers, and to cover the different styles of payloads and stagers.

A. Payload Generation

Each payload was generated using MSFvenom with the following parameters: -arch x86, -platform windows, LHOST = 192.168.0.2, LPORT = 6666, -format exe, -out payload-XXX.exe.

The payload was then modified using each of the following techniques and transferred to the Windows test VM via HTTP which by then scanned using the respective anti-virus. In the following we describe each legacy evasion technique employed.

1) Encoding: As previously discussed, the purpose of encoding is to remove problematic characters that will not survive transport or placement into memory. Encoding and specifying problematic characters can be specific at the time of payload generation in MSFvenom. At the time of writing, MSFvenom offered forty-two different methods of encoding, the most popular being `shikata_ga_nai` and `powershell_base64`. The payload representative of the encoding technique was again generated with MSFvenom with these additional parameters: -encoder x86/shikata_ga_nai and -iterations 1.

2) Custom Execution Template: MSFvenom can also be used to generate a payload with a specific execution template. For testing, the executable `notepad.exe` was arbitrarily selected. The executable was collected from a Windows 10 workstation and transferred (scp) to our Kali Linux VM. During payload generation, the switch `--template notepad.exe` was also supplied.

3) Hyperion - Encryption: Hyperion was acquired from *NullSecurity* and installed according to [12]. The default payloads were then copied into the Hyperion working directory. Hyperion encrypted each payload using the default parameters (random 6 bytes AES key) with the following example command:

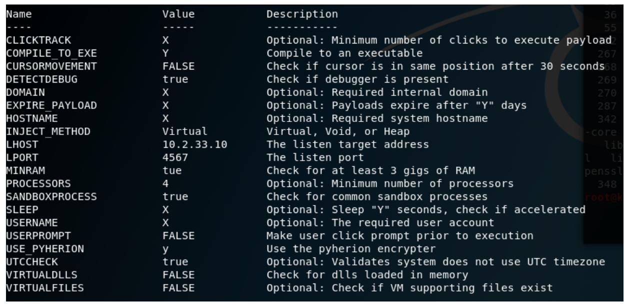
```
wine hyperion.exe meterpreter.exe
meterpreter-enc.exe
```

4) peCloak.py: Owing to compatibility issues and custom packages required for Python, a *peCloak* package was acquired from ‘The anti-virus hacker handbook’. The main script, however, was acquired directly from the author [13]. The payloads were then copied into the peCloak working directory. No command line parameters were passed, and each payload was modified with the default parameters.

5) Veil-Evasion: The default installation of Veil Evasion that ships with Kali Linux was utilized for this portion. The framework was fully updated and ensured to match the latest release in accordance with the source repository [14]. There was one dependency for the compilation of the Python source code, thus we installed the package `winbind`. The manner in which the Veil-Framework generates payloads differs slightly from that of MSFvenom. The payload is compiled directly from source code in several languages such as Python, C, Go, Perl, Ruby, and PowerShell. The meterpreter reverse tcp payload was available directly from the source and the Python version was chosen (payload 28). The Shell and PowerShell payloads were indirectly available through MSFvenom using the AES-encrypt payload module (payload 29). To showcase features that are unique to Veil-Evasion, we will apply some optional evasion techniques during payload creation. Changes from the default payload options are presented in Figure 1. Finally, to generate the executable from the malicious source code, the default tool pyInstaller was utilized.

B. Testing

A back-up of each payload on the Kali Linux machine was arranged prior transferring to the Windows VMs. To simulate a typical delivery, the payloads were placed in an Apache2 web directory and served to the Windows machines on a local network. On both Windows machines, a web browser (Internet Explorer) was opened and navigated to the relevant web directory. The payload was then downloaded and scanned with the respective anti-virus software. The payload was often automatically detected, and in these cases manually initiating the scan was foregone. When detection occurred, details were recorded and logged.



```
Name                Value          Description
----                -----          -----------
CLICKTRACK          X              Optional: Minimum number of clicks to execute payload
COMPILE_TO_EXE      Y              Compile to an executable
CURSORMOVEMENT      FALSE          Check if cursor is in same position after 30 seconds
DETECTDEBUG         true           Check if debugger is present
DOMAIN              X              Optional: Required internal domain
EXPIRE_PAYLOAD      X              Optional: Payloads expire after "Y" days
HOSTNAME            X              Optional: Required system hostname
INJECT_METHOD       Virtual        Virtual, Void, or Heap
LHOST               10.2.33.10     The listen target address
LPORT               4567           The listen port
MINRAM              tue            Check for at least 3 gigs of RAM
PROCESSORS          4              Optional: Minimum number of processors
SANDBOXPROCESS      true           Check for common sandbox processes
SLEEP               X              Optional: Sleep "Y" seconds, check if accelerated
USERNAME            X              Optional: The required user account
USERPROMPT          FALSE          Make user click prompt prior to execution
USE_PYHERION        y              Use the pyherion encrypter
UTCCHECK            true           Optional: Validates system does not use UTC timezone
VIRTUALDLLS         FALSE          Check for dlls loaded in memory
VIRTUALFILES        FALSE          Optional: Check if VM supporting files exist
```

Fig. 1. Options used for payload generation in Veil-Evasion

V. Results and Analysis

In this section we present the detection rates of each of the payloads against the various anti-virus software. The binary results against Windows Defender and Norton Security and the detection summary from Virus Total are presented in Table IV.

We immediately identify notice that the anti-virus software is achieves overwhelmingly successful over against our evasion tactics in all but a few trials. Accordingly, the unmodified (default) payloads were uniformly detected uniformly and set a baseline for comparison with regard toin regards to Virus Total . This was surpassed in a single trial only by encoding the Sshell payload. A comparison of the detection rates of Virus Total is presented in Figure 2.

Surprisingly, the mean detection rate forof both the encoding and encryption (Hyperion) categories were greater than for the default payloads. However, tThis difference is only slight only marginal (*less than %0.1*), and should not be interpreted construed as significant. Therefore, we can conclude that these two methods did nothing to improve the payloads chances of anti-virus evasion for the payloads. On the other handBy contrast, two methods did show lead to a significant improvement above over the other payload categories. The custom execution template ($M = 0.176, SD = 0.062$) was found to be detected less often than the default payloads ($M = 0.694, SD = 0.006$), $t = 0.517, p = 0.002$. Likewise, the Windows Defender JS HTA module ($M = 0.0798, SD = 0.0061$) had the best performanceed the greatest compared to against the default, $t = 0.614, p < 0.001$. Across all trials, only three payload categories had binaries that were not detected outright by the one of the anti-virus’s, namely the custom execution template, Veil-Evasion, and the Metasploit module Windows Defender JS HTA. This however does not

TABLE IV
ANTIVIRUS DETECTION

Payload	Anti-Virus Results		
	Windows Defender	**Norton Security**	**Virus Total**
Default	Trojan:Win32/Meterpreter.O	Packed.Generic.347	46/66
	Trojan:Win32/Meterpreter.O	Packed.Generic.347	46/67
	Trojan:Win32/Meterpreter.O	Packed.Generic.347	46/66
Encoding	Trojan:Win32/Meterpreter.O	Packed.Generic.347	46/66
	Trojan:Win32/Meterpreter.O	Packed.Generic.347	45/66
	Trojan:Win32/Meterpreter.O	Packed.Generic.347	47/66
Execution Template	No Threats Found	WS.Reputation.1	11/66
	No Threats Found	WS.Reputation.1	8/67
	Trojan:Win32/Meterpreter.gen!C	Meterpreter	16/66
Hyperion (Encryption)	TrojanDownloader:Win32/Banload	Packed.Generic.508	48/69
	TrojanDownloader:Win32/Banload	Packed.Generic.508	46/66
	TrojanDownloader:Win32/Banload	Packed.Generic.508	46/66
peCloak	Trojan:Win32/Feury.B!cl	Heur.AdvML.B	35/66
	Trojan:Win32/Feury.B!cl	Heur.AdvML.B	34/66
	Trojan:Win32/Feurboos.D!cl	Heur.AdvML.B	35/66
Veil-Evasion	Trojan:Win32/Feurboos.D!cl	Hacktool.Veil	26/66
	Trojan:Win32/Feurboos.D!cl	No Threats Found	28/67
	Trojan:Win32/Feurboos.d!cl	WS.Reputation.1	29/67
MSF Module Windows Defender EXE	Trojan:Win32/Meterpreter.Pl	Huer.AdvML.B	22/55
	Trojan:Win64/Meterpreter.O	Huer.AdvML.B	27/66
	Trojan:Win64/Meterpreter.B	Huer.AdvML.B	18/66
MSF Module: Windows Defender JS HTA	No Threats Found*	Web Attack: Metasploit Payload Download Activity	4/54
	No Threats Found*	Web Attack: Metasploit Payload Download Activity	4/54
	No Threats Found*	Web Attack: Metasploit Payload Download Activity	4/55

Payload ordering: **Top:** Meterpreter **Middle** Powershell **Bottom:** Shell
* - Payload was detected as malicious when executed

guarantee that the payload would be entirely successful. Run-time and behavioral analysis, included in both tested anti-virus's, may potentially detect malicious activities. We feel that it is necessary to specify that the HTA module was detected after execution because of the way the payload operates. The HTA file contains the source code for the payload, which was encrypted using RC4. Upon execution the payload is decrypted and compiled directly on the target computer. The result is written to disk and the staging payload is executed. Because the payload creates another, non-volatile signature, this cannot be overlooked as system calls for read and write from disk are monitored by the anti-virus software.

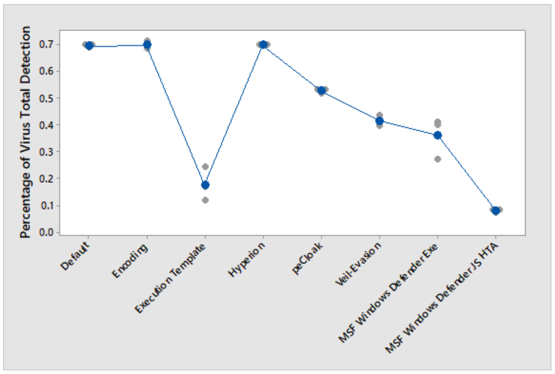


Fig. 2. Detection Percentage of Virus Total for each payload category. Blue indicates mean values.

VI. Discussion

This section will discuss the results and analyze the drivers behind observed trends. We first must remind the reader that no unique conditions, aside from randomly generated keys, were produced in our methodology. As this was an analysis of the tools themselves, to remove any bias based on the skills of the user, we only used default or generally representative settings. This undoubtedly led to conditions that were easily predictable from the perspective of the anti-virus. Exploring each of the branches of these tools would require a study far beyond the scope of this work. Notwithstanding, our work is still characteristic of the largest population of malicious actors, *script kiddies*.

A. Encoding and Encryption

As discussed in Section II, encoding is often mischaracterized as an evasion technique, when it is instead a way of removing unwanted characters. This was clearly evident in our results (Figure 2), where the rate of detection was on par with the default payload. Rarely will situations occur where multiple rounds of encoding improves the success of an attack, especially considering that anti-virus have signatures from payloads encoded far beyond the capability of a script kiddy. A common web-attack involves exploiting the way browsers handle URL encoding, where attackers can benefit from multiple rounds of encoding [6]. Regardless, we would not expect the results to improve based on any other encoding technique or by increasing the number of iterations.

By contrast, the results for the encryption-based technique are surprising similar. Because each Hyperion utilizes randomly generated keys of a generous size, we can safely assume that the payload portion of the binary will be unique and never seen before by the anti-virus. As with encoded payloads, the entrance point of the binary must include the key and instructions to decode the payload. Here we expect the anti-virus to identify the executable as malicious. Considering that the only unique part of the decryption process may be the key, we expect this to be a significant point of failure.

The two techniques employed have used some form of obfuscation and execution flow randomization in the decoder or decryption stub. This involves randomized code, functions and registers, to generate unique machine instructions. This is further obfuscated by inserting *junk code* between malicious segments and simply jumping over them. Although this may produce unique signatures, this has the ill-effect of creating an irregular and suspicious execution entry routine. If the anti-virus can simply follow the jumps dynamically, the instruction for RWX memory allocation will still be found [15]. Despite the payload itself being unrecognizable, this functionality will inevitably be detected by a heuristic anti-virus. This is the reason why both encoding and encryption-based techniques fail.

B. Custom Execution Templates

The flaws presented by encoding and encryption are addressed by custom execution templates. Rather than creating a tangled and suspicious execution entry routine, custom execution templates rely on known safe programs [16]. Where the anti-virus might have detected the decoder and RWX stub, the signature and functionality of a benign application will remain. Furthermore, the memory allocated to the payload will belong to a legitimate application. As this is one of the most difficult phases of payload staging, we suspect that directly addressing this problem contributed to the demonstrated success of this technique.

Aside from anti-virus evasion, this tactic also has valuable social engineering and deceptive benefits. The functionality of the carrier executable is preserved. This results in the creation of a trojan, where foul play is less likely to be suspected if launched directly by the victim as a result of social engineering. The payload will inevitably be executed; however, specifying an execution template does nothing to address heuristic detection and run-time analysis. We suspect that the anti-virus's that successfully detected the threat used these techniques. Combining this tactic with other evasive techniques may prove very effective.

C. Combined Techniques

Veil Evasion, peCloak, and the MSF evasion modules all employ a variety of evasion techniques. These include encryption, junk code, sandbox detection and heuristic run-time analysis protection. The MSF module uses a flaw in the logic found in the Window's Defender emulation engine that allows for the detection of a sandbox [17]. Similarly, peCloak, introduces reciprocal instructions to confuse run-time heuristics [18], while Veil Evasion uses many approaches, such as detecting emulation acceleration, or lost clock cycles owing to the anti-virus. Utilizing Window's APIs, the payload can check whether a debugger is present as an indication of emulation. Another powerful method Veil Evasion employs to detect sandboxing is to check for minimal hardware. Should the memory or number of processors present be lower than that expected for a typical user machine, the payload will remain benign.

These techniques improve the probability of success; thus, we observe a downtrend in detection rate when framework sophistication increases. Because anti-virus reporting is not entirely transparent regarding the actual detection engine that flagged the payload, it is difficult to determine which payload characteristic led to the detection. The above tactics are well known and likely to be mitigated in scenarios such as cloud-based analysis.

D. Metasploit Modules

Sadly, the performance of the Metasploit Evasion modules, specifically targeted at Windows Defender, was not on par with the author's claims. This is likely due to the public announcement of the tool, and further derailed by its documentation in the white-paper [17]. Although only released in early October 2018, the modules have been under development and publicly available since August. This certainly gave anti-virus vendors ample time to craft a defense.

The key benefit of the Windows Defender EXE module was the novel emulation detection. However, it is likely that the API calls used for run-time analysis detection have been patched to prevent this type of use of the logic. Aside from this feature, the module is rather simple. We suspect that the Virus Total results are likely due to slower responses from smaller anti-virus companies. We expect the detection rate compared to a complex framework, such as Veil, to increase with time.

By contrast, the HTA module was successful in evading the anti-virus initially; however it ended up being detected once the payload was compiled. This payload, having been written to disk, became subject to scanning. Should the payload be compiled and directly loaded into a code cave or a hollowed process, the secondary scan would be avoided.

E. Powershell Injection

Up to this point, this study has ignored a major trend in malware behavior and attack vectors. By and large, attackers have avoided touching non-volatile storage entirely and have loaded their payloads directly into RAM. If done correctly, this helps evade the anti-virus scanning that occurs during disk reads and writes, among other system calls. If the payload can be loaded and executed through some side-channels the chance of evasion increases dramatically. Additionally, from the perspective of anti-forensics, this has the benefit of leaving a far smaller footprint on the victims machine.

This attack vector and the strengths and capabilities of Microsoft's Powershell has led PowerShell script attacks to become very popular. In response, Windows Defender had to find a way to combat scripted and interpreted languages.

TABLE V
POWERSHELL INJECTION RESULTS

	Anti-Virus Results		
Payload	**Win Defender**	**Norton Security**	**Virus Total**
Meterpreter reverse TCP	X	X	5/57
Powershell reverse TCP	X	X	4/56
Shell Reverse TCP	X	X	4/57
Meterpreter reverse HTTPS	X	X	4/58

X- indicates the payload was detected

One solution was the antimalware scan interface application program interface (API) [19]. This call is accessible through PowerShell or any application to submit code snippets for scanning prior to execution [20]. Norton includes a similar feature that utilizes Symantec Online Network for Advanced Response (SONAR). If attackers have access to a victim PowerShell console, scripts can be executed without ever touching the disk, and will only remain in the log history for the session. It should be noted that the PowerShell payloads used in the experiment above utilized PowerShell to fetch the second stage payloads that were delivered as executables rather than in a script form.

F. Observer Effect

During testing, we made every effort to submit the payloads to each anti-virus in a time period that was as short as possible. Again, this was to prevent a signature being created by one anti-virus vendor influencing another. For example, it can be expected that a submission to Virus Total that has a highly malicious confidence will be accounted for in the next signature update for well-known anti-virus companies.

To *see* the observer effect in action, we delivered and scanned payloads that had previously successfully evaded the anti-virus after a ten-day waiting period. Two of the three custom execution template payloads were detected in the latter trial and no improvement was made to the MSF HTA payloads. It is possible that the HTA payloads did not have a high enough malicious confidence as reported in Virus Total. We do, however, suspect that the longer time period allowed for either signature updates or a more robust cloud-based analysis that was not possible by the host in the time allotted for the custom execution templates. Should this be the case, it would support the practice of researchers and malware authors of keeping their tactics secret.

VII. CONCLUSION

Anti-virus protection plays an important role in computer security and will continue to do so as a rampart against malicious attacks. Anti-virus detection techniques will continue to grow alongside the malware created to evade anti-virus detection. Metasploit's team created their evasion module with the goal of avoiding detection by common anti-virus products. We put this module to the test with a series of trials using various payloads. It appeared that, for the most part, anti-virus software is extremely successful at preventing these malicious attacks. Knowing that anti-virus software is sufficiently strong to prevent this Metasploit attack shows us as users that it can be relied upon to keep our computer systems very safe. Strong anti-virus software is what users will want on their systems to ensure data security and prevent the infestation of malware. However, does the strength of anti-virus software mean future malware will be even harder to detect? Software engineers will keep on testing anti-virus detection systems to watch for new malicious attacks.

REFERENCES

[1] Saed Alrabaee, Noman Saleem, Stere Preda, Lingyu Wang, and Mourad Debbabi. Oba2: An onion approach to binary code authorship attribution. *Digital Investigation*, 11:S94–S103, 2014.

[2] Saed Alrabaee, Mourad Debbabi, and Lingyu Wang. On the feasibility of binary authorship characterization. *Digital Investigation*, 28:S3–S11, 2019.

[3] Saed Alrabaee, Paria Shirani, Lingyu Wang, Mourad Debbabi, and Aiman Hanna. On leveraging coding habits for effective binary authorship attribution. In *European Symposium on Research in Computer Security*, pages 26–47. Springer, 2018.

[4] Saed Alrabaee, Paria Shirani, Lingyu Wang, and Mourad Debbabi. Fossil: a resilient and efficient system for identifying foss functions in malware binaries. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):8, 2018.

[5] Chris Hoffman. Xprotect explained: How your macs built-in anti-malware software works, 2015. https://www.howtogeek.com/217043/xprotect-explained-how-your-macs-built-in-anti-malware-works/.

[6] Moayad Aloqaily, Safa Otoum, Ismaeel Al Ridhawi, and Yaser Jararweh. An intrusion detection system for connected vehicles in smart cities. *Ad Hoc Networks*, 2019.

[7] S. Otoum, B. Kantarci, and H. T. Mouftah. Detection of known and unknown intrusive sensor behavior in critical applications. *IEEE Sensors Letters*, 1(5):1–4, Oct 2017.

[8] Safa Otoum, Burak Kantarci, and Hussein T Mouftah. On the feasibility of deep learning in sensor network intrusion detection. *IEEE Networking Letters*, 2019.

[9] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[10] mihi. Facts and myths about antivirus evasion with metasploit. http://schierlm.users.sourceforge.net/avevasion.html.

[11] Vishal S Sangwa. Hexing using dsplit to hide trojans from antivirus detection, 2012. http://vishalssangwa.blogspot.com/2012/09/hexing-using-dsplit-to-hide-trojans.html#axzz5Vc7JTudg.

[12] How to make hyperion work in kali? https://www.cybrary.it/forums/topic/how-to-make-hyperion-exe-work-in-kali/.

[13] Mike Czumak. pecloak.py an experiment in av evasion, 2015. https://www.securitysift.com/pecloak-py-an-experiment-in-av-evasion/.

[14] Veil-Framework. Veil-evasion. https://github.com/Veil-Framework/Veil-Evasion.

[15] scriptjunkie. Why encoding does not matter and how metasploit generates exes, 2011. https://www.scriptjunkie.us/2011/04/why-encoding-does-not-matter-and-how-metasploit-generates-exes/.

[16] David Maloney. The odd couple: Metasploit and antivirus solutions, December 2012. https://blog.rapid7.com/2012/12/14/the-odd-couple-metasploit-and-antivirus-solutions/.

[17] Wei Chen. Encapsulating antivirus (av) evasion techniques in metasploit framework. https://www.rapid7.com/globalassets/_pdfs/whitepaperguide/rapid7-whitepaper-metasploit-framework-encapsulating-av-techniques.pdf.

[18] Mike Czumak. pecloak.py - an experiment in av evasion, 2015. http://www.securitysift.com/pecloak-py-an-experiment-in-av-evasion/.

[19] Microsoft. Antimalware scan interface, 2018. https://docs.microsoft.com/en-us/windows/desktop/AMSI/antimalware-scan-interface-portal.

[20] Wei Chen. Hiding metasploit shellcode to evade windows defender, May 03 2018. https://blog.rapid7.com/2018/05/03/hiding-metasploit-shellcode-to-evade-windows-defender/.