Obfuscation of a Kerberos Golden Ticket Attack

Leon Vogel

Abstract. Sigma rules are an open-source, rule-based format for detecting malicious behavior. As such, they are subject to the attention of actors with both good and bad intentions. Adversaries can exploit the open-source nature to try to construct their attacks in a way that they are no longer detected by the rules, which raises the question of the rules' effectiveness and robustness. Previous research has already addressed this through a systematic approach, attempting to evade a subset of command-line related rules. In contrast, this study adopts a more practical approach by executing a real-world attack, analyzing the generated alerts, and subsequently attempting to evade them. The evaluation showed that approximately 90% of the alerts generated by the analyzed rules were evadable with minimal effort, and most of the remaining 10% with higher effort.

Keywords: Sigma rules · Intrusion detection · Obfuscation

1 Introduction

Cybercrime continues to increase both quantitatively as well as qualitatively, causing damage to businesses and organizations through ransomware attacks, industrial espionage and data extortion among others [1]. Consequently, businesses utilize systems like Security information and event management (SIEM) systems that "gives security teams a central place to collect, aggregate, and analyze volumes of data" [2] to detect and prevent the attacks. A way to leverage the centralized data collection of SIEM systems is to apply rules on the log data to generate alerts when malicious events are detected. One such ruleset is the Sigma ruleset [3], which is community-driven and open source. This allows people to review the rules and improve on them, but at the same time enables bad actors to find and abuse potential weaknesses by obfuscating their code to evade the detection. This raises the question of how robust and secure these rules are, and how well they can withstand both accidental and malicious changes.

In that work, however, only a subset of rules, that act on process creation events, was considered, and the evasion techniques focused mostly on the command line. In this work, the effectiveness and robustness of the Sigma rules are analyzed more broadly, based on the practical example of the Kerberos golden ticket attack. For this purpose, five categories of obfuscation techniques have been developed, partly building on the previous work, whilst also devising entirely new techniques that also consider script contents. Each of these categories are analyzed separately for their effectiveness in reducing alerts while also considering their ease of use. In a final run, all the separate techniques are combined to see how much the alerts could be reduced to.

The following report is structured as follows: In section 2 the test environment, the process of generating the logs and obtaining the alerts, and the attack setup is described. In section 3, each of the evasion techniques is explained in detail and illustrated by practical examples. The techniques are then applied and the results evaluated in section 4. Finally, in section 5, all findings are summarized and potential future work is discussed.

2 Test environment and attack chain

To obtain representative results, three desirable properties should be considered for this experiment [4]:

1. An environment where the tests can be executed repeatedly and in isolation
2. A way to capture and compile events that happen on multiple machines
3. A consistent attack chain that has the same behavior in all runs.

This chapter describes how each of these feats was achieved, what tools and scripts were used, what limitations there were, and what assumptions had to be made.

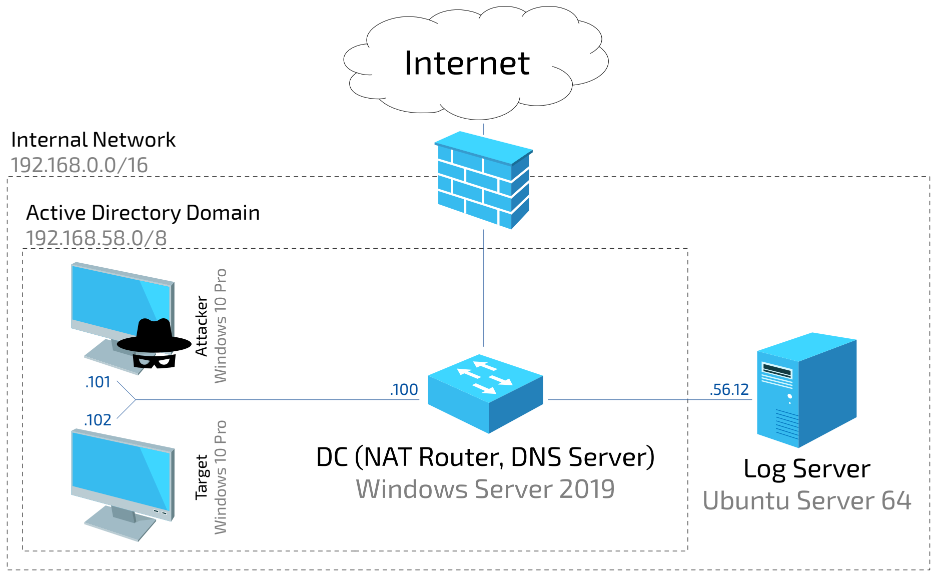

Fig. 1: Simplified overview of the test environment consisting out of two workstations, a Domain Controller (DC) and a log server.

2.1 Environment setup

The first step of setting up the test environment is setting up the virtual machines the experiment is run on. For the attack itself, three machines are needed [1]: a domain controller, and two workstations, one used as the attackers inital foothold and the other as the target of the attack. An abstract version of the architecture can be found in Figure 1.

Domain controller "A domain controller is a server that is running a version of the Windows Server operating system and has Active Directory Domain Services installed" [5]. In this experiment, the domain controller is running Windows Server 2019 build 17763.737 [6, 7], which is an early build from 2018, as later patches prevent the golden ticket attack, as it is described in subsection 2.3. For the purpose of this experiment, in addition to running the Active Directory, the domain controller acts as the DNS server for the attacker and target machines, and is configured as a NAT router. Since the interest of the experiment does not lie in obfuscating the attack from antivirus software, Windows Defender is turned off.

Attacker and target Both the attacker and target machine run Windows 10 Pro version 22H2 [8]. They are set up to use the domain controller as their default gateway and DNS server. Their computers are joined to the Active Directory domain and for each an account on the domain is created. As with the domain controller, Windows Defender is disabled on both machines. In addition, the firewall is disabled on the workstations and remote connections are allowed. Furthermore, it is assumed that the attacker has local administrator rights, which are required to execute the attack.

2.2 Log and alert generation

In addition to the steps described in the previous section, logging software is set up on both the domain controller as well as the attacker and target machine. The software generates the log data, which is subsequently used to generate the alerts using the Sigma rules. A more detailed explanation of both the logs and the alerts is provided in the following sections.

Log generation For the logging of the Windows system events, Microsoft's System Monitor (Sysmon) is used. Using a configuration file, it is possible to further specify what events exactly are logged. Evaluating this, however, is of no further interest for the experiment and is taken from [4] as is. Another way to get additional logging is by enabling module and script block logging, which logs the pipeline execution events for specific modules and, the processing of

[1] In the most basic form, only the domain controller and the attacker are required, but for demonstration purposes a third machine is used

scripts, respectively [9]. Both were enabled, with module logging enabled for all modules.

Up until this point, the events are logged locally and individually for the different machines, however, the desired behavior is to gather all logs using a SIEM system. To achieve this *Elasticsearch* and *Kibana* from the Elastic Security stack [10, 11] were installed onto an Ubuntu server, which is the final component of the environment as seen in Figure 1. The shipping of the events to the log server was done using *winlogbeat*, which was also installed onto all three machines and is also a part of the Elastic stack [12].

Alert generation With the events collected at a centralized place, they can be used to generate the alerts. Kibana offers the option of applying rules directly to the logs and perform operations accordingly [13]. In this case, however, a separate non-automatic analysis is sufficient, which is why the logs are downloaded first from the log server. Afterwards, the Sigma rules can be applied to the logs using the *Chainsaw* tool [14] to generate the alerts.

The Sigma rules are released in different variations like *core rules*, *emerging threats* and *all rules*. In addition to the regularly published releases, the repository can be downloaded for an even greater amount of rules than the *all-rules*-variation. The additional rules, however, are often more experimental and cause more noise in the resulting alerts, which led to the decision to limit the experiment to the *all-rules*-variation of the latest release at that time (r2024-03-11) [15].

A shortened, example Sigma rule is shown in Listing 1 with the full rule in the appendix in Listing 2. The rule tries to detect the execution of Base64 encoded commands and consists mainly out of four components: metadata, the source of the log, how the malicious event is detected and classification of the severity. First off is the metadata. The metadata contains information such as the name and ID of the rule, a short description, references as to what malicious events this rule tries to catch, and information concerning the author and date the rule was created. Next is the source of the log, that the rule should be applied to. In the example it is specified that only process creation events within windows should be considered. The main component of a rule is the detection. In this context, the rule generates an alert, if the program that is executed is detected as PowerShell and is executed with some variation of the *encoding* flag. Along with detection, exceptions are also specified, which prevent the rule from generating an alert even if the condition is met. The last part of a Sigma rule is the classification regarding the severity and potential false positives. This enables those responsible for the security of these systems to assess the likelihood and extent to which an alert is related to an attack. While the severity is always specified, the rate of false positives is, as with this rule, not always provided.

The corresponding alert to the above explained rule can be found in Listing 3 in the appendix. The attribute that is of interest is the *document* attribute.

```
title: Suspicious Execution of Powershell with Base64
id: fb843269-508c-4b76-8b8d-88679db22ce7
status: test
description: Commandline to launch powershell with a base64 payload
author: frack113
date: 2022/01/02
logsource:
  category: process_creation
  product: windows
detection:
  selection:
    Image|endswith:
      - \powershell.exe
    CommandLine|contains:
      - ' -enc '
  condition: selection
falsepositives:
  - Unknown
level: medium
```

Listing 1: A shortened example of a Sigma rule that detects Base64 encoding

It contains information about the time of the event, the agent that caused it and the suspicious event itself. In the case of this example the command line arguments are included.

2.3 Golden ticket attack

Kerberos is an authentication protocol, that is used to authenticiate within an Windows Active Directory domain [16]. The authentication is done using *tickets* and it is distinguished between two kinds: the Ticket-Granting Ticket (TGT) and the service tickets. The TGT is issued by a centralized Key Distribution Center (KDC) instance and contains the identity of the requesting client. It can be "exchanged" for the second type of ticket, the service tickets, which can then actually be used to access different services. The full process of the Kerberos authentication is naturally more complex than what has been briefly outlined here, but for the purpose of this work this explanation will suffice. [17]

The golden ticket attack itself involves forging a TGT, which is entirely customizable and can contain the identity of any user in the domain [18]. To forge the TGT, the password hash of the *krbtgt* account, which is the service account corresponding to the KDC in Windows domains, is required [18, 19]. Obtaining this hash is not a trivial task and requires privileged access to the domain, such as through an already compromised system [18]. This requirement, combined with the ability to create non-expiring TGT, renders the golden ticket attack particularly effective for maintaining persistence in an already breached system, rather than initiating the breach itself [20].

This experiment follows part of the Advanced Persistent Threat (APT) 29 emulation plan by the Center for Threat-Informed Defense, which tries to recreate how the attack chain might have looked like and the tools that were used [21, 22]. The golden ticket attack, as described in the emulation plan, can be split into four main steps:

1. Extract information about the domain
2. Read the plaintext password from the domain administrator
3. Use mimikatz to dump the *krbtgt* credentials
4. Forge a golden ticket using the *krbtgt* hash

For the first step the PowerView script is used to obtain information about the domain [23]. The second step requires a domain administrator to have been recently logged into the attackers workstation. To dump the plaintext password, the mimikatz executable was used [24]. Using the domain administrators password the mimikatz executable is first copied to the domain controller and then used to dump the *krbtgt* credentials in step 3. The credentials are then used to forge the golden ticket using the Invoke-Mimikatz script [23]. Finally, a command is run on the targets workstation to ensure that the attack was successful. All steps were combined into an additional script to avoid deviations in the attack runs.

3 Obfuscation techniques

In the prior chapter, all prerequisites necessary for the attack and therefore also the following obfuscation have been outlined. As mentioned in the motivation, there has already been work done by Uetz et al. [25] focusing on command-line techniques. Where applicable to the golden ticket attack, these techniques have been adopted, resulting in a total of five distinct techniques, three of which were new and have been developed specifically for the experiment. This chapter describes each of these obfuscation techniques, including illustrative examples, what kind of Sigma rules they evade, and an assessment of their efficiency while also considering how much effort they involve.

3.1 File renaming

The first of the five techniques involved the renaming of specific script files. Since only *PowerView* and *Invoke-Mimikatz* are well-known scripts, with the rest of the used scripts being "helper scripts", only these were recognized and thus needed to be renamed.

There are three different kinds of rules that are evaded by this technique. The first type detects the creation of files with a suspicious name, which in this case happened when the payload was unzipped on the attackers workstation. The second type detects the scripts filename inside the *context information* of command invocations. When the *PowerView* and *Invoke-Mimikatz* scripts are imported, many commands are executed in the background. For each of these commands, there is a structure is that keeps information like information about the host the command was invoked on, the username and also the name of the script that is being imported. In addition to that there are rules that detect suspicious keywords, which are also detected when importing the scripts.

Overall, renaming files involves only a small amount of effort, depending on the number of scripts files involved, and can even happen without any malicious intent. At the same time the technique is efficient for what it does and makes it difficult for rules to detect as there is an almost infinite number of names to which a file can be renamed.

3.2 Command-line substitution

Substitution of commands and flags was one of the techniques that was adopted. There are three different types of substitutions, each of which has an example shown in Table 1: flag substitution, command substitution and flag prefix substitution. For flag substitutions either built-in aliases were used, or, as shown in the example, the autocomplete feature of PowerShell was abused, which infers the correct flag as long as it is unambiguous. The other substitutions relied completely on built-in aliases like `cp` to copy a file and a “slash” character instead of a “dash” as the prefix for flags.

By using a combination of all of these different substitutions, many of the rules that check for specific commands, flags or a combination of these can be evaded, since they do not check for all possible variations.

Similarly to the renaming of files, obfuscations of this nature might also happen on accident, as most of these are also down to a personal preference. The effort is generally quite low, but depends on the extensiveness of the list of variations of the rules.

Table 1: Example for each type of command-line substitution

Original command	Obfuscated command
`Set-ExecutionPolicy Bypass -Force`	`Set-ExecutionPolicy B -Force`
`Copy-Item src dst`	`cp src dst`
`powershell.exe -enc [Base64]`	`powershell.exe /enc [Base64]`

3.3 Command-line insertion

Another technique that was adopted, is the insertion of characters into commands. Characters that were inserted are quotation marks within flags like “`powershell.exe -""enc [Base64]`”, which generally works in PowerShell, because they are ignored when the command is actually executed.

Inserting an empty set of quotation marks, can be used for all rules that check for specific flags or values, which effectively renders them ineffective.

Unlike the prior techniques, obfuscations of this type are most likely done with malicious intent and are unlikely to occur by chance. The effort, however, was comparable to the prior techniques, and therefore quite low.

3.4 Customized scripts

Apart from the file renaming technique, this is the first technique, that does not consider the command-line. The technique itself is further split into three sub-techniques: code removal, identifier renaming and code splitting, which build upon one another. The decision to combine these techniques was made deliberately because, on the one hand, a separate consideration would go beyond the scope of this work and, on the other hand, because attackers would most likely proceed in a similar way.

The first of the sub-techniques was the removal of code that was not used for the golden ticket attack. Especially PowerView has a lot of code that was not needed for the attack, generating lots of alerts for unused lines of code. By removing all unused code, approximately 99.7% of the PowerView script and 7.3% of the Invoke-Mimikatz script was removed, based on lines of code.

Following the removal of code was the renaming of identifiers detected by rules. Identifiers include function names, variable names and object member names. As most of the code in PowerView was already removed at this point only changes in Invoke-Mimikatz were made. An example for the changes is the renaming of *Win32* functions that were loaded using `Get-ProcAddress` and stored inside an object for later use. Renaming, however, was only possible for the object members and not the parameter to the `Get-ProcAddress` invocation, thus only partly being usable to remove the alerts.

The final step in the script customization is the splitting of code into several blocks. This is possible because PowerShell allows the use of strings to create new objects, create variables, and other functionalities. By splitting suspicious keywords into two strings and concatenating them in the e.g. `New-Object` invocation they are no longer detected by the rules.

There is a variety of rules that can be evaded by these techniques. First, by removing unused code, virtually any rule can be evaded, if they are detected by unused code. Secondly, by renaming variables, functions and more, rules that detect suspicious keywords can be evaded, as long as the keywords are not built-in type definitions or functions. For the cases of built-in keywords, the code splitting can be applied, allowing to split the keywords into as many substrings as needed to evade the rules.

The effort of the subtechniques is gradually increasing. The removal of unused code and the renaming of identifiers involved relatively low effort, as it only required to identify unused code and perform search-and-replace on suspicious keywords. In contrast, code splitting required significantly more manual effort. Regardless, this technique proves to be efficient at removing alerts, as it is effectively impossible for rules to detect suspicious keywords if they are either renamed or split up.

3.5 Customized executables

The final new technique that was devised, was specifically for evading the detection of mimikatz modules and commands. Mimikatz uses commands such as `sekurlsa::logonpasswords`, to show credentials of the logged-on users, which are detected as malicious by certain rules. Since mimikatz is open source, its commands can be modified to evade detection, and the executable can be recompiled to create a customized version. Using the example from before, the command was changed to `seklsa::logonpw` and therefore no longer detected.

Given that the technique specifically targets mimikatz, the number of rules that can be evaded is correspondingly limited. Furthermore, it is also the technique that required the most effort. Nevertheless, the rules that can be evaded are rendered ineffective as a result.

4 Evaluation

To evaluate the effectiveness of the developed techniques at obfuscating the attack, it is first necessary to determine how well the Sigma rules detect it. To achieve this, the attack was first run without any of the techniques applied. For this and all further analysis, it is important to note that some alerts have been removed for the analysis as they are not caused by the attack. A complete list of alerts generated by each technique can be found in the appendix in the section Technique alerts. Moreover, it is important to note that there may be deviations in the number of alerts independent of the obfuscation, due to the fact that there is randomness involved in the script logging [26].

Running the unobfuscated attack resulted in a total of 1564 alerts generated from 27 distinct rules. Examining the detection method, 16 of the rules use script-based detection, 5 use command-line-based detection, 1 uses context information, and 5 use other means such as the parent image of a program. In Figure 2 is a more detailed view in regard to the severity of the alerts, revealing that 1497 of the alerts are of high severity and only 67 of medium. This indicates that the Sigma rules, without any obfuscation applied, work generally well at detecting the golden ticket attack.

In Table 2 the source of the alerts can be better analyzed. There are four different sources for alerts that result from the attack. By a large majority, the PowerShell module is the most common source with 1375 alerts all coming from one rule. The second most alerts come from PowerShell script logging and amounted for 176 alerts. Only a small fraction originated in process creations and file events with 11 and 2 alerts respectively. The process creation was detected for applications used for remote session creations and also more specifically for the mimikatz executable. The file events captured the creation of both the PowerView and Invoke-Mimikatz script. Of special interest is the fact, that the logging for the two major log sources had to be enabled additionally, as described in section 2, showing the importance of these.

4.1 Results

The technique that evaded the least amount of alerts is the command-line substitution. In comparison to the baseline it reduced the alerts by only 15 resulting in 1549 of 1564 alerts. At the same time, however, the number of distinct rules was reduced to 21 from the baselines 27. Also considering the severity of the alerts as seen in Figure 2, it is visible that both medium and high severity alerts are reduced. Additionally, in Table 2 it can be seen that actually the PowerShell scripts category is the most evaded rule with some from process creation, which is due to the fact that command-line input is also considered in the script logging. Ultimately, the technique is constrained by the amount of command-line usage in the experiment. However, its effectiveness is still demonstrated, considering the distinct rules that could be evaded and the effort required, showing that the technique works well when applicable.

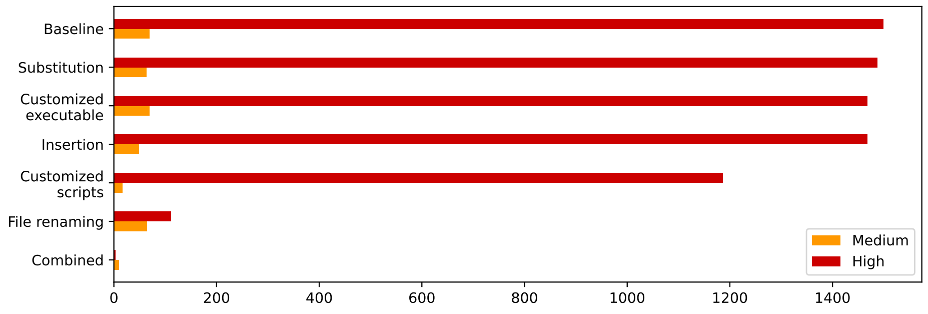

Fig. 2: Results of the obfuscation techniques divided by their severity level

Slightly more effective than the substitution, is the usage of the customized mimikatz executable. Using it, the alerts were reduced to 1529 out of 1564, 35 fewer alerts than the baseline were generated, but also only one distinct rule was evaded. Especially the high severity alerts were reduced by a decent amount, while the medium alerts are actually more than the substitution technique. Comparing the effectiveness based on the category it is evident where the difference to the prior technique is. It has roughly the same rules evaded for process creation with a difference of just 1, but about 20 more alerts evaded in the PowerShell script category. As the mimikatz executable is called within the scripts this aligns with the prior expectations. While the technique is only the

Table 2: Results of the obfuscation techniques divided into the separate alert categories

Techniques	File Event	Process Creation	PS Module	PS Script
Baseline	2	11	1375	176
Substitution	2	8	1375	164
Customized executable	2	9	1375	143
Insertion	2	8	1375	130
Customized scripts	2	11	1175	13
File renaming	0	11	0	161
Combined	0	6	0	4

fourth most effective with regard to the number of alerts evaded, it is of high significance in this experiment, as it is the sole technique that evades one of the mimikatz specific rules.

The insertion technique reduced the alerts by 49, resulting in 1515 alerts, with 21 distinct rules left. Generally, based on the results, this technique appears to be similar to the substitution technique just more effective in reducing the alerts with 20 medium severity alerts and 29 high severity alerts evaded. As for the categories, like the substitution technique, most of the alerts are from the PowerShell script category and some from the process creation. Comparing the actual rules that are evaded, they are the same for both techniques. In this case the difference in alerts is solely based on the aforementioned randomness of the script logging.

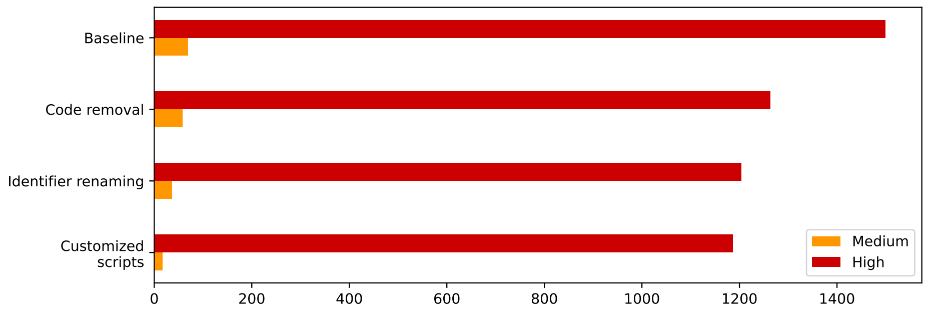

Fig. 3: Results of the script customization techniques further divided into the subtechniques and severity levels

Next is the script customization technique with 1201 out of 1564 alerts and 16 out of 27 distinct rules remaining, making it highly effective. As mentioned in section 3, the technique is split into three further subtechniques. The severity of the alerts of the individual subtechniques is displayed separately in Figure 3.

With the code removal alone, the number of alerts has already decreased to 1315. After applying the identifier renaming it is further reduced to 1232, and finally, with code splitting is reaches the 1201 alerts. The medium severity alerts decrease steadily at about 10-20 per subtechnique, while the high severity alerts are mostly evaded using the code removal. Taking also the categories into consideration which can be seen in Table 3, it follows the same pattern already visible in Figure 3 concerning the number of alerts for the different subtechniques. In general, this technique is the first that evaded some PowerShell module alerts. As code is removed from the scripts, the amount of code imported and logged by the module logging is naturally reduced, resulting in fewer alerts. Overall, this technique is highly effective, significantly reducing the number of alerts, especially considering that it fully obfuscates some rules that are not or only partly obfuscated by other techniques.

Table 3: Results of the separate customized scripts subtechniques divided into the separate alert categories

Techniques	File Event	Process Creation	PS Module	PS Script
Baseline	2	11	1375	176
Code removal	2	11	1240	62
Identifier renaming	2	11	1186	33
Code splitting	2	11	1175	13

The final technique was the file renaming. This technique required the least effort, but at the same time results in the lowest number of remaining alerts, with 172. It does, however, only evade two distinct rules. Both rules that are evaded are of high severity, with most of the rules originating from PowerShell module logging and the rest from file events. It fully evades both rules, while the remaining alerts remain are the same as the baseline. For the obfuscation of the golden ticket attack, it is essential, as it is the only technique that fully evades these two distinct techniques.

By combining all the techniques, only a minimal number of alerts remain. There are 8 medium severity alerts and 2 high severity alerts remaining, split up onto 5 distinct rules. These 10 alerts fall either into the process creation or PowerShell script category. The process creation alerts are all remote sessions that had to be created to e.g. upload mimikatz to the domain controller, which is necessary for the attack and due to the detection method unevadable. The PowerShell script alerts that are detected, originate from built-in Windows scripts executed on the domain controller and can therefore not be evaded by any of the techniques. All in all, the obfuscation was a success, especially considering that the remote sessions could also be benign behavior which is also mentioned in the rules.

5 Conclusion

This work proposed five techniques to be used for the obfuscation of Sigma rules based on the practical example of the golden ticket attack and evaluated the effectiveness of both the rules and the techniques by conducting an experiment. The results from section 4 showed that even basic changes, such as renaming a file, which do not have to be done with malicious intent, can remove about 90% of all alerts. Remaining alerts can then be removed using more complex techniques that do, however, require a much higher invested amount of effort.

However, it is also important to contextualize these results by considering the assumptions made about the execution environment of the attack. The workstations had to have a bad configuration, an outdated version of Windows Server was used, and Windows Defender had to be turned off among other factors.

In summary, while the Sigma rules are generally able to detect the attack, they are prone to evasion. Actors with malicious intent are unlikely to be stopped by Sigma rules on their own. As part of a defense-in-depth strategy, Sigma rules are still a valuable additional layer of security to detect malicious behavior.

Future research into this topic could use systems like Adaptive Misuse Detection System (AMIDES) [25], which aims at detecting the obfuscations, to additionally evaluate the effectiveness of the techniques against such systems.

References

[1] Bundeskriminalamt (BKA). *Bundeslagebild Cybercrime 2023.* 2023. URL: https://www.bka.de/SharedDocs/Downloads/DE/Publikationen/JahresberichteUndLagebilder/Cybercrime/cybercrimeBundeslagebild2023.html.

[2] Microsoft. *What is SIEM | Microsoft Security.* URL: https://www.microsoft.com/en-us/security/business/security-101/what-is-siem (visited on 08/09/2024).

[3] SigmaHQ. *Sigma - SIEM Detection Format | The shareable detection format for security professionals.* URL: https://sigmahq.io/ (visited on 08/09/2024).

[4] Rafael Uetz et al. “Reproducible and adaptable log data generation for sound cybersecurity experiments”. In: *Proceedings of the 37th Annual Computer Security Applications Conference.* 2021, pp. 690–705.

[5] Microsoft. *Domain Controller Roles: Active Directory.* URL: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc786438(v=ws.10) (visited on 08/23/2024).

[6] Internet Archive. *Windows Server 2019 (build 17763.737).* URL: https://www.microsoft.com/en-us/evalcenter/download-windows-server-2019 (visited on 08/23/2024).

[7] Microsoft. *Windows Server 2019 | Microsoft Evaluation Center.* URL: https://www.microsoft.com/en-us/evalcenter/download-windows-server-2019 (visited on 08/23/2024).

[8] Microsoft. *Download Windows 10 Disc Image (ISO File).* URL: https://www.microsoft.com/de-de/software-download/windows10ISO (visited on 08/23/2024).

[9] Microsoft. *About Logging Windows PowerShell.* URL: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_logging_windows?view=powershell-7.4 (visited on 08/30/2024).

[10] elastic. *Elasticsearch: The Official Distributed Search & Analytics Engine | Elastic.* URL: https://www.elastic.co/elasticsearch (visited on 08/30/2024).

[11] elastic. *Kibana: Explore, Visualize, Discover Data | Elastic.* URL: https://www.elastic.co/kibana (visited on 08/30/2024).

[12] elastic. *Winlogbeat: Analyze Windows Event Logs | Elastic.* URL: https://www.elastic.co/beats/winlogbeat (visited on 08/30/2024).

[13] elastic. *Kibana Alerting: Alerts & Actions for Elasticsearch data | Elastic.* URL: https://www.elastic.co/kibana/alerting (visited on 09/01/2024).

[14] WithSecure Labs. *Chainsaw: Rapidly Search and Hunt through Windows Forensic Artefacts.* URL: https://github.com/WithSecureLabs/chainsaw (visited on 08/16/2024).

[15] Ben Montour et al. *Sigma rules GitHub - Release r2024-03-11.* Mar. 2024. URL: `https://github.com/SigmaHQ/sigma/releases/tag/r2024-03-11`.

[16] Robin Hardwood et al. *Windows Authentication Overview.* URL: `https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/windows-authentication-overview` (visited on 09/10/2024).

[17] C. Neuman et al. *The Kerberos Network Authentication Service (V5).* RFC 4120 (Proposed Standard). Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113, 6649, 6806. Internet Engineering Task Force, July 2005. URL: `http://www.ietf.org/rfc/rfc4120.txt`.

[18] Itamar Mizrahi and Cymptom. *Steal or Forge Kerberos Tickets: Golden Ticket.* 2020. URL: `https://attack.mitre.org/versions/v15/techniques/T1558/001/`.

[19] Daniel Simpson et al. *Active Directory Accounts.* URL: `https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-default-user-accounts` (visited on 09/04/2024).

[20] Miguel Soria-Machado et al. "Kerberos golden ticket protection". In: *Mitigating Pass-the-Ticket on Active Directory, CERT-EU Security Whitepaper* 7 (2014), p. 2016.

[21] Daniyal Naeem et al. *APT29.* 2017. URL: `https://attack.mitre.org/versions/v15/groups/G0016/`.

[22] Center for Threat-Informed Defense. *APT29 Adversary Emulation.* 2022. URL: `https://github.com/center-for-threat-informed-defense/adversary_emulation_library/tree/4a57b3dd5d28ad1bd79e927e04b20fd4d66934a0/apt29`.

[23] PowerShellMafia. *Release PowerSploit v.3.0.0.* 2015. URL: `https://github.com/PowerShellMafia/PowerSploit/releases/tag/v3.0.0`.

[24] Benjamin Delpy. *Mimikatz Release 2.2.0 20220919 Djoin parser & Citrix SSO Extractor.* 2022. URL: `https://github.com/gentilkiwi/mimikatz/releases/tag/2.2.0-20220919`.

[25] Rafael Uetz et al. "You Cannot Escape Me: Detecting Evasions of SIEM Rules in Enterprise Networks". In: *arXiv preprint arXiv:2311.10197* (2023).

[26] Aditya Patwardhan. *GitHub PowerShell / CompiledScriptBlock.cs.* URL: `https://github.com/PowerShell/PowerShell/blob/7ec8e4ed8f47e81e70de5353500f8a01d5fe396c/src/System.Management.Automation/engine/runtime/CompiledScriptBlock.cs#L1451-L1454` (visited on 09/11/2024).

Appendix

Example Sigma rule and resulting alert

```yaml
title: Suspicious Execution of Powershell with Base64
id: fb843269-508c-4b76-8b8d-88679db22ce7
status: test
description: Commandline to launch powershell with a base64 payload
references:
    - https://github.com/redcanaryco/atomic-red-team/blob/
    f339e7da7d05f6057fdfcdd3742bfcf365fee2a9/atomics/T1059.001/T1059.001.
    md#atomic-test-20---powershell-invoke-known-malicious-cmdlets
    - https://unit42.paloaltonetworks.com/
    unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks/
    - https://mikefrobbins.com/2017/06/15/
    simple-obfuscation-with-powershell-using-base64-encoding/
author: frack113
date: 2022/01/02
modified: 2023/01/05
tags:
    - attack.execution
    - attack.t1059.001
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image|endswith:
            - \powershell.exe
            - \pwsh.exe
        CommandLine|contains:
            - ' -e '
            - ' -en '
            - ' -enc '
            - ' -enco'
            - ' -ec '
    filter_encoding:
        CommandLine|contains: ' -Encoding '
    filter_azure:
        ParentImage|contains:
            - 'C:\Packages\Plugins\Microsoft.GuestConfiguration.ConfigurationforWindows\'
            - '\gc_worker.exe'
    condition: selection and not 1 of filter_*
falsepositives:
    - Unknown
level: medium
```

Listing 2: Example Sigma rule that detects Base64 encoding

```
{
    "group": "Sigma",
    "kind": "individual",
    "document": {
        "kind": "json",
        "path": "logs/baseline.jsonl",
        "data": {
            "@timestamp": "2024-08-28T18:42:02.320Z",
            "agent": {
                "ephemeral_id": "5b05db5d-64fa-4dde-af33-69f81c43657f",
                "hostname": "ALICE",
                "id": "0129b102-48b3-446e-9609-35d247958315",
                "name": "ALICE",
                "type": "winlogbeat",
                "version": "7.10.1"
            },
            "ecs": {
                "version": "1.5.0"
            },
            "event": {
                "action": "Process Create (rule: ProcessCreate)",
                "category": [
                    "process"
                ],
                "code": 1,
                "created": "2024-08-28T18:42:04.137Z",
                "kind": "event",
                "module": "sysmon",
                "provider": "Microsoft-Windows-Sysmon",
                "type": [
                    "start",
                    "process_start"
                ]
            },
            "hash": {
                "imphash": "88cb9a420410bda787e305b65518a934",
                "md5": "bcf01e61144d6d6325650134823198b8",
                "sha256": "b4e7bc24bf3f5c3da2eb6e9ec5ec10f90099defa91b820f2f3fc70dd9e4785c4"
            },
            "host": {
                "name": "ALICE.LAB.local"
            },
            "log": {
                "level": "information"
            },
            "message": "Process Create:\nRuleName: -\nUtcTime: 2024-08-28 18: 42: 02.320\nPr
        }
            \nProcessId: 5280\nImage:
            C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.
            exe\nFileVersion: 10.0.19041.2913 (WinBuild.160101.0800)
            \nDescription: Windows PowerShell\nProduct: Microsoft\u00ae
            Windows\u00ae Operating System\nCompany: Microsoft
            Corporation\nOriginalFileName: PowerShell.EXE\nCommandLine:
            powershell.exe -enc [...
        ]\nCurrentDirectory:
            C:\\Windows\\system32\\\nUser: LAB\\Alice\nLogonGuid: {
            1aecbc69-6f0c-66cf-c924-140000000000
        }\nLogonId: 0x1424C9\nTerminalSessionId: 2\nIntegrityLevel: High\nHashes:
            MD5=BCF01E61144D6D6325650134823198B8,
            SHA256=B4E7BC24BF3F5C3DA2EB6E9EC5EC10F90099DEFA91B820F2F3FC70DD9E478
            5C4,IMPHASH=88CB9A420410BDA787E305B65518A934\nParentProcessGuid: {
            1aecbc69-6f30-66cf-1d01-000000001600
        }\nParentProcessId: 11016\nParentImage: C:\\Windows\\System32\\wbem\\WmiPrvSE.
            exe\nParentCommandLine: C:\\Windows\\system32\\wbem\\wmiprvse.exe
            -secured -Embedding\nParentUser: NT AUTHORITY\\NETWORK SERVICE","process": {"arg
        "[...]"
    ],
    "command_line": "powershell.exe -enc [...]",
    "entity_id": "{1aecbc69-6f7a-66cf-4a01-000000001600}",
    "executable": "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe",
    "hash": {
        "md5": "bcf01e61144d6d6325650134823198b8",
        "sha256": "b4e7bc24bf3f5c3da2eb6e9ec5ec10f90099defa91b820f2f3fc70dd9e4785c4"
    },
    "name": "powershell.exe",
    "parent": {
        "args": [
            "C:\\Windows\\system32\\wbem\\wmiprvse.exe",
            "-secured",
            "-Embedding"
        ],
        "command_line": "C:\\Windows\\system32\\wbem\\wmiprvse.exe -secured -Embedding",
        "entity_id": "{1aecbc69-6f30-66cf-1d01-000000001600}",
        "executable": "C:\\Windows\\System32\\wbem\\WmiPrvSE.exe",
        "name": "WmiPrvSE.exe",
        "pid": 11016
    },
    "pe": {
        "imphash": "88cb9a420410bda787e305b65518a934"
    },
    "pid": 5280,
    "working_directory": "C:\\Windows\\system32\\"
},
"related": {
    "hash": [
        "bcf01e61144d6d6325650134823198b8",
        "b4e7bc24bf3f5c3da2eb6e9ec5ec10f90099defa91b820f2f3fc70dd9e4785c4",
        "88cb9a420410bda787e305b65518a934"
    ],
    "user": "Alice"
},
"user": {
    "domain": "LAB",
    "name": "Alice"
},
"winlog": {
    "api": "wineventlog",
    "channel": "Microsoft-Windows-Sysmon/Operational",
    "computer_name": "ALICE.LAB.local",
    "event_data": {
        "Company": "Microsoft Corporation",
        "Description": "Windows PowerShell",
        "FileVersion": "10.0.19041.2913 (WinBuild.160101.0800)",
        "IntegrityLevel": "High",
        "LogonGuid": "{1aecbc69-6f0c-66cf-c924-140000000000}",
        "LogonId": "0x1424c9",
        "OriginalFileName": "PowerShell.EXE",
        "ParentUser": "NT AUTHORITY\\NETWORK SERVICE",
        "Product": "Microsoft\u00ae Windows\u00ae Operating System",
        "RuleName": "-",
        "TerminalSessionId": "2"
    },
    "event_id": 1,
    "opcode": "Info",
    "process": {
        "pid": 3484,
        "thread": {
            "id": 5008
        }
    },
    "provider_guid": "{5770385f-c22a-43e0-bf4c-06f5698ffbd9}",
    "provider_name": "Microsoft-Windows-Sysmon",
    "record_id": 10779,
    "task": "Process Create (rule: ProcessCreate)",
    "user": {
        "domain": "NT AUTHORITY",
        "identifier": "S-1-5-18",
        "name": "SYSTEM",
        "type": "User"
    },
    "version": 5
}
}
},
"name": "Suspicious Execution of Powershell with Base64",
"timestamp": "2024-08-28T18:42:02.320+00:00",
"authors": [
"frack113"
],
"level": "medium",
"source": "sigma",
"status": "experimental",
"falsepositives": [
"Unknown"
],
"id": "fb843269-508c-4b76-8b8d-88679db22ce7",
"logsource": {
"category": "process_creation",
"product": "windows"
},
"references": [
  "https://github.com/redcanaryco/atomic-red-team/blob/
  f339e7da7d05f6057fdfcdd3742bfcf365fee2a9/atomics/T1059.001/T1059.001.
  md#atomic-test-20---powershell-invoke-known-malicious-cmdlets",
  "https://unit42.paloaltonetworks.com/
  unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks/",
  "https://mikefrobbins.com/2017/06/15/
  simple-obfuscation-with-powershell-using-base64-encoding/"
],
"tags": [
"attack.execution",
"attack.t1059.001"
]
}
```

Listing 3: An alert generated from the Sigma rule in Listing 2. The actual encoded command has been omitted for reasons of compactness.

Technique alerts

Table 4: Alerts generated from the baseline

Rule	Count
Change PowerShell Policies to an Insecure Level - PowerShell	1
CurrentVersion Autorun Keys Modification	2
Execute Invoke-command on Remote Host	10
HackTool - Mimikatz Execution	2
HackTool - Rubeus Execution - ScriptBlock	2
Malicious Nishang PowerShell Commandlets	3
Malicious PowerShell Commandlets - ScriptBlock	57
Malicious PowerShell Keywords	28
Malicious PowerShell Scripts - FileCreation	2
Malicious PowerShell Scripts - PoshModule	1375
Manipulation of User Computer or Group Security Principals Across AD	5
NTFS Alternate Data Stream	2
Potential In-Memory Execution Using Reflection.Assembly	1
Potential Invoke-Mimikatz PowerShell Script	6
Potential Suspicious PowerShell Keywords	10
Potential WMI Lateral Movement WmiPrvSE Spawned PowerShell	1
Potential WinAPI Calls Via PowerShell Scripts	7
PowerShell Remote Session Creation	2
PowerView PowerShell Cmdlets - ScriptBlock	37
Powershell Install a DLL in System Directory	1
Powershell Timestomp	2
Remote PowerShell Session Host Process (WinRM)	4
Request A Single Ticket via PowerShell	1
Suspicious Encoded PowerShell Command Line	1
Suspicious Execution of Powershell with Base64	1
Suspicious PowerShell Encoded Command Patterns	1
Usage Of Web Request Commands And Cmdlets - ScriptBlock	1
Windows Defender Threat Detected	2
WmiPrvSE Spawned A Process	1

Table 5: Alerts generated from the file renaming technique

Rule	Count
Change PowerShell Policies to an Insecure Level - PowerShell	1
CurrentVersion Autorun Keys Modification	3
Disable Windows Defender Functionalities Via Registry Keys	1
Execute Invoke-command on Remote Host	9
HackTool - Mimikatz Execution	2
HackTool - Rubeus Execution - ScriptBlock	1
Malicious Nishang PowerShell Commandlets	3
Malicious PowerShell Commandlets - ScriptBlock	51
Malicious PowerShell Keywords	24
Malicious PowerShell Scripts - FileCreation	0
Malicious PowerShell Scripts - PoshModule	0
Manipulation of User Computer or Group Security Principals Across AD	5
Microsoft Defender Blocked from Loading Unsigned DLL	0
NTFS Alternate Data Stream	2
Potential In-Memory Execution Using Reflection.Assembly	1
Potential Invoke-Mimikatz PowerShell Script	6
Potential Suspicious PowerShell Keywords	10
Potential WMI Lateral Movement WmiPrvSE Spawned PowerShell	1
Potential WinAPI Calls Via PowerShell Scripts	7
PowerShell Remote Session Creation	2
PowerView PowerShell Cmdlets - ScriptBlock	34
Powershell Install a DLL in System Directory	1
Powershell Timestomp	2
Remote PowerShell Session Host Process (WinRM)	4
Request A Single Ticket via PowerShell	1
Suspicious Encoded PowerShell Command Line	1
Suspicious Execution of Powershell with Base64	1
Suspicious PowerShell Encoded Command Patterns	1
Usage Of Web Request Commands And Cmdlets - ScriptBlock	1
Use Short Name Path in Command Line	0
WmiPrvSE Spawned A Process	1

Table 6: Alerts generated from the command-line substitution technique

Rule	Count
Change PowerShell Policies to an Insecure Level - PowerShell	0
CurrentVersion Autorun Keys Modification	2
Execute Invoke-command on Remote Host	9
HackTool - Mimikatz Execution	2
HackTool - Rubeus Execution - ScriptBlock	1
Malicious Nishang PowerShell Commandlets	3
Malicious PowerShell Commandlets - ScriptBlock	53
Malicious PowerShell Keywords	28
Malicious PowerShell Scripts - FileCreation	2
Malicious PowerShell Scripts - PoshModule	1375
Manipulation of User Computer or Group Security Principals Across AD	4
NTFS Alternate Data Stream	2
Potential In-Memory Execution Using Reflection.Assembly	1
Potential Invoke-Mimikatz PowerShell Script	6
Potential Suspicious PowerShell Keywords	11
Potential WMI Lateral Movement WmiPrvSE Spawned PowerShell	1
Potential WinAPI Calls Via PowerShell Scripts	3
PowerShell Remote Session Creation	0
PowerView PowerShell Cmdlets - ScriptBlock	38
Powershell Install a DLL in System Directory	0
Powershell Timestomp	2
Remote PowerShell Session Host Process (WinRM)	4
Request A Single Ticket via PowerShell	2
Suspicious Encoded PowerShell Command Line	0
Suspicious Execution of Powershell with Base64	0
Suspicious PowerShell Encoded Command Patterns	0
Usage Of Web Request Commands And Cmdlets - ScriptBlock	1
Windows Defender Threat Detected	0
WmiPrvSE Spawned A Process	1

Table 7: Alerts generated from the command-line insertion technique

Rule	Count
Change PowerShell Policies to an Insecure Level - PowerShell	0
CurrentVersion Autorun Keys Modification	2
Execute Invoke-command on Remote Host	8
HackTool - Mimikatz Execution	2
HackTool - Rubeus Execution - ScriptBlock	1
Malicious Nishang PowerShell Commandlets	1
Malicious PowerShell Commandlets - ScriptBlock	45
Malicious PowerShell Keywords	18
Malicious PowerShell Scripts - FileCreation	2
Malicious PowerShell Scripts - PoshModule	1375
Manipulation of User Computer or Group Security Principals Across AD	4
NTFS Alternate Data Stream	2
Potential In-Memory Execution Using Reflection.Assembly	1
Potential Invoke-Mimikatz PowerShell Script	4
Potential Suspicious PowerShell Keywords	7
Potential WMI Lateral Movement WmiPrvSE Spawned PowerShell	1
Potential WinAPI Calls Via PowerShell Scripts	4
PowerShell Remote Session Creation	0
PowerView PowerShell Cmdlets - ScriptBlock	31
Powershell Install a DLL in System Directory	0
Powershell Timestomp	2
Remote PowerShell Session Host Process (WinRM)	4
Request A Single Ticket via PowerShell	1
Suspicious Encoded PowerShell Command Line	0
Suspicious Execution of Powershell with Base64	0
Suspicious PowerShell Encoded Command Patterns	0
Usage Of Web Request Commands And Cmdlets - ScriptBlock	1
Windows Defender Threat Detected	0
WmiPrvSE Spawned A Process	1

Table 8: Alerts generated from the code removal technique

Rule	Count
Change PowerShell Policies to an Insecure Level - PowerShell	1
CurrentVersion Autorun Keys Modification	3
Disable Windows Defender Functionalities Via Registry Keys	1
Execute Invoke-command on Remote Host	3
HackTool - Mimikatz Execution	2
HackTool - Rubeus Execution - ScriptBlock	0
Malicious Nishang PowerShell Commandlets	0
Malicious PowerShell Commandlets - ScriptBlock	2
Malicious PowerShell Keywords	31
Malicious PowerShell Scripts - FileCreation	2
Malicious PowerShell Scripts - PoshModule	1240
Manipulation of User Computer or Group Security Principals Across AD	0
Microsoft Defender Blocked from Loading Unsigned DLL	0
NTFS Alternate Data Stream	2
Potential In-Memory Execution Using Reflection.Assembly	0
Potential Invoke-Mimikatz PowerShell Script	3
Potential Suspicious PowerShell Keywords	9
Potential WMI Lateral Movement WmiPrvSE Spawned PowerShell	1
Potential WinAPI Calls Via PowerShell Scripts	6
PowerShell Remote Session Creation	2
PowerView PowerShell Cmdlets - ScriptBlock	0
Powershell Install a DLL in System Directory	1
Powershell Timestomp	2
Remote PowerShell Session Host Process (WinRM)	4
Request A Single Ticket via PowerShell	0
Suspicious Encoded PowerShell Command Line	1
Suspicious Execution of Powershell with Base64	1
Suspicious PowerShell Encoded Command Patterns	1
Usage Of Web Request Commands And Cmdlets - ScriptBlock	0
Use Short Name Path in Command Line	0
Windows Defender Threat Detected	2
WmiPrvSE Spawned A Process	1

Table 9: Alerts generated from the identifier renaming technique

Rule	Count
Change PowerShell Policies to an Insecure Level - PowerShell	1
CurrentVersion Autorun Keys Modification	3
Disable Windows Defender Functionalities Via Registry Keys	1
Execute Invoke-command on Remote Host	3
HackTool - Mimikatz Execution	2
HackTool - Rubeus Execution - ScriptBlock	0
Malicious Nishang PowerShell Commandlets	0
Malicious PowerShell Commandlets - ScriptBlock	1
Malicious PowerShell Keywords	10
Malicious PowerShell Scripts - FileCreation	2
Malicious PowerShell Scripts - PoshModule	1186
Manipulation of User Computer or Group Security Principals Across AD	0
Microsoft Defender Blocked from Loading Unsigned DLL	2
NTFS Alternate Data Stream	2
Potential In-Memory Execution Using Reflection.Assembly	0
Potential Invoke-Mimikatz PowerShell Script	1
Potential Suspicious PowerShell Keywords	8
Potential WMI Lateral Movement WmiPrvSE Spawned PowerShell	1
Potential WinAPI Calls Via PowerShell Scripts	2
PowerShell Remote Session Creation	2
PowerView PowerShell Cmdlets - ScriptBlock	0
Powershell Install a DLL in System Directory	1
Powershell Timestomp	2
Remote PowerShell Session Host Process (WinRM)	4
Request A Single Ticket via PowerShell	0
Suspicious Encoded PowerShell Command Line	1
Suspicious Execution of Powershell with Base64	1
Suspicious PowerShell Encoded Command Patterns	1
Usage Of Web Request Commands And Cmdlets - ScriptBlock	0
Use Short Name Path in Command Line	0
Windows Defender Threat Detected	2
WmiPrvSE Spawned A Process	1

Table 10: Alerts generated from the code splitting technique

Rule	Count
Change PowerShell Policies to an Insecure Level - PowerShell	0
CurrentVersion Autorun Keys Modification	2
Disable Windows Defender Functionalities Via Registry Keys	0
Execute Invoke-command on Remote Host	0
HackTool - Mimikatz Execution	2
HackTool - Rubeus Execution - ScriptBlock	0
Malicious Nishang PowerShell Commandlets	0
Malicious PowerShell Commandlets - ScriptBlock	1
Malicious PowerShell Keywords	4
Malicious PowerShell Scripts - FileCreation	2
Malicious PowerShell Scripts - PoshModule	1175
Manipulation of User Computer or Group Security Principals Across AD	0
Microsoft Defender Blocked from Loading Unsigned DLL	0
NTFS Alternate Data Stream	2
Potential In-Memory Execution Using Reflection.Assembly	0
Potential Invoke-Mimikatz PowerShell Script	1
Potential Suspicious PowerShell Keywords	0
Potential WMI Lateral Movement WmiPrvSE Spawned PowerShell	1
Potential WinAPI Calls Via PowerShell Scripts	0
PowerShell Remote Session Creation	2
PowerView PowerShell Cmdlets - ScriptBlock	0
Powershell Install a DLL in System Directory	1
Powershell Timestomp	2
Remote PowerShell Session Host Process (WinRM)	4
Request A Single Ticket via PowerShell	0
Suspicious Encoded PowerShell Command Line	1
Suspicious Execution of Powershell with Base64	1
Suspicious PowerShell Encoded Command Patterns	1
Usage Of Web Request Commands And Cmdlets - ScriptBlock	0
Use Short Name Path in Command Line	0
WmiPrvSE Spawned A Process	1

Table 11: Alerts generated from the customized executable technique

Rule	Count
Change PowerShell Policies to an Insecure Level - PowerShell	1
CurrentVersion Autorun Keys Modification	2
Execute Invoke-command on Remote Host	9
Extracting Information with PowerShell	1
HackTool - Mimikatz Execution	0
HackTool - Rubeus Execution - ScriptBlock	1
Malicious Nishang PowerShell Commandlets	3
Malicious PowerShell Commandlets - ScriptBlock	41
Malicious PowerShell Keywords	26
Malicious PowerShell Scripts - FileCreation	2
Malicious PowerShell Scripts - PoshModule	1375
Manipulation of User Computer or Group Security Principals Across AD	4
Microsoft Defender Blocked from Loading Unsigned DLL	4
NTFS Alternate Data Stream	2
Potential In-Memory Execution Using Reflection.Assembly	1
Potential Invoke-Mimikatz PowerShell Script	3
Potential Suspicious PowerShell Keywords	11
Potential WMI Lateral Movement WmiPrvSE Spawned PowerShell	1
Potential WinAPI Calls Via PowerShell Scripts	3
PowerShell Remote Session Creation	2
PowerView PowerShell Cmdlets - ScriptBlock	29
Powershell Install a DLL in System Directory	1
Powershell Timestomp	2
Remote PowerShell Session Host Process (WinRM)	4
Request A Single Ticket via PowerShell	2
Suspicious Encoded PowerShell Command Line	1
Suspicious Execution of Powershell with Base64	1
Suspicious PowerShell Encoded Command Patterns	1
Usage Of Web Request Commands And Cmdlets - ScriptBlock	1
Windows Defender Threat Detected	0
Windows Defender Threat Detection Disabled - Service	2
WmiPrvSE Spawned A Process	1