699.3

Lateral Movement Emulation & Detection

SEC699.3 | Advanced Purple Team Tactics

SANS

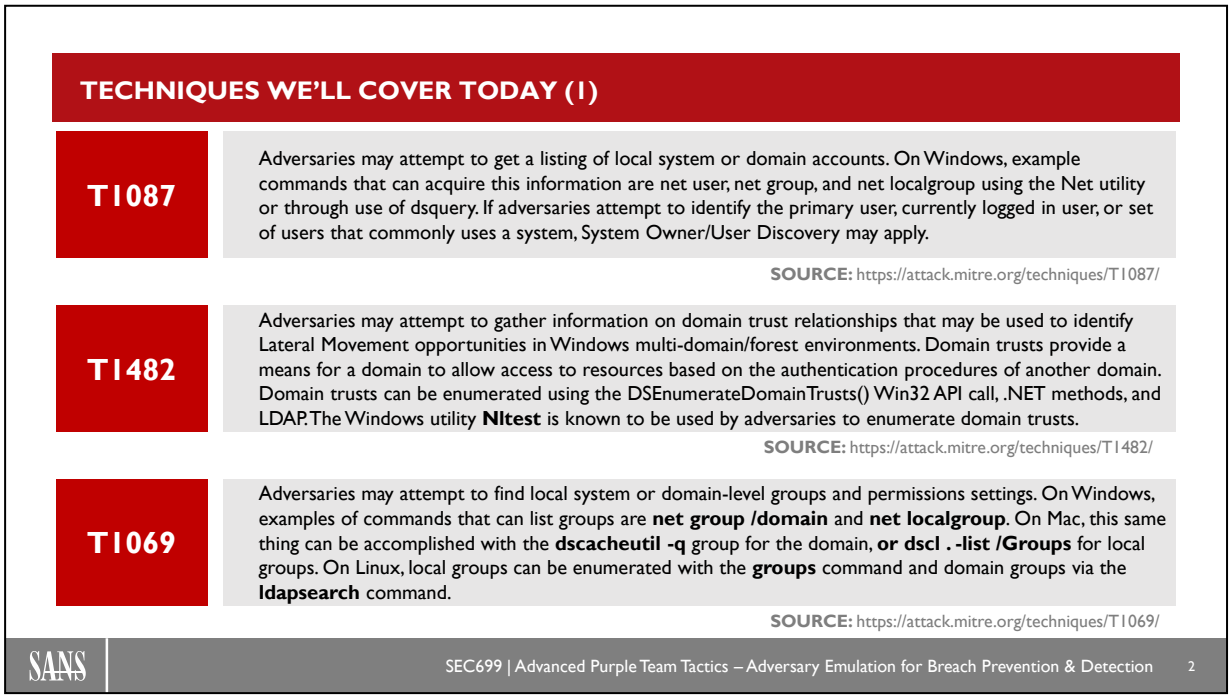
Lateral Movement Emulation & Detection

Welcome to SANS Security SEC699: Advanced Purple Team Tactics – Adversary Emulation for Breach Prevention & Detection.

During today's section, we will zoom in on Lateral Movement.

For any remarks, please reach out to the authors:

Erik Van Buggenhout
evanbuggenhout@nviso.eu
www.nviso.eu
Update: G01_01



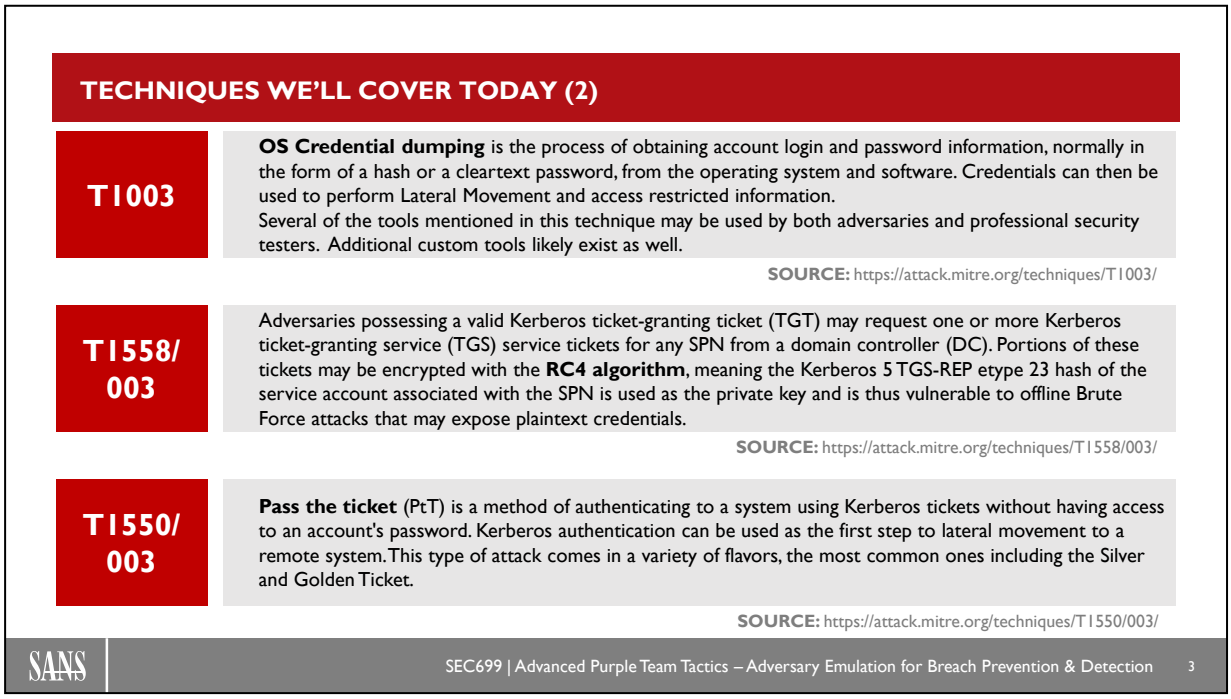
Techniques We'll Cover Today (1)
Some of the techniques we'll cover today include:

T1087 – Account Discovery
Adversaries may attempt to get a listing of local system or domain accounts. On Windows, example commands that can acquire this information are net user, net group, and net localgroup using the Net utility or through use of dsquery. If adversaries attempt to identify the primary user, currently logged in user, or set of users that commonly uses a system, System Owner/User Discovery may apply.
Source: https://attack.mitre.org/techniques/T1087/

T1482 – Domain Trust Discovery
Adversaries may attempt to gather information on domain trust relationships that may be used to identify Lateral Movement opportunities in Windows multi-domain/forest environments. Domain trusts provide a means for a domain to allow access to resources based on the authentication procedures of another domain. Domain trusts can be enumerated using the DSEnumerateDomainTrusts() Win32 API call, .NET methods, and LDAP. The Windows utility Nltest is known to be used by adversaries to enumerate domain trusts.
Source: https://attack.mitre.org/techniques/T1482/

T1069 – Permissions Group Discovery
Adversaries may attempt to find local system or domain-level groups and permissions settings. On Windows, examples of commands that can list groups are net group /domain and net localgroup. On Mac, this same thing can be accomplished with the dscacheutil -q group for the domain, or dscl . -list /Groups for local groups. On Linux, local groups can be enumerated with the groups command and domain groups via the ldapsearch command.
Source: https://attack.mitre.org/techniques/T1069/

As we did previously, we will start by explaining these techniques in a lot more detail and review opportunities for prevention and detection.



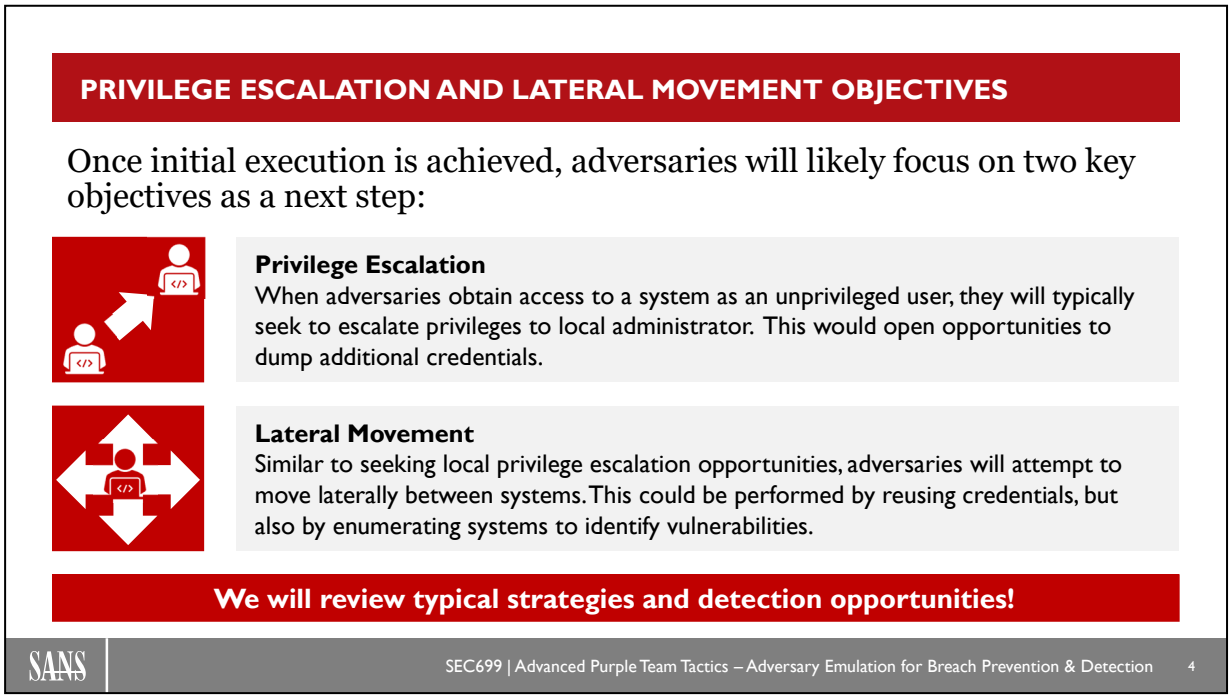
Techniques We'll Cover Today (2)
Some of the techniques we'll cover today include:

T1003 – Credential Dumping
Credential dumping is the process of obtaining account login and password information, normally in the form of a hash or a cleartext password, from the operating system and software. Credentials can then be used to perform Lateral Movement and access restricted information. Several of the tools mentioned in this technique may be used by both adversaries and professional security testers. Additional custom tools likely exist as well.
Source: https://attack.mitre.org/techniques/T1003/

T1558/003 – Kerberoasting
Adversaries possessing a valid Kerberos ticket-granting ticket (TGT) may request one or more Kerberos ticket-granting service (TGS) service tickets for any SPN from a domain controller (DC). Portions of these tickets may be encrypted with the RC4 algorithm, meaning the Kerberos 5 TGS-REP etype 23 hash of the service account associated with the SPN is used as the private key and is thus vulnerable to offline Brute Force attacks that may expose plaintext credentials.
Source: https://attack.mitre.org/techniques/T1558/003/

T1550/003 – Pass-the-Ticket
Pass the ticket (PtT) is a method of authenticating to a system using Kerberos tickets without having access to an account's password. Kerberos authentication can be used as the first step to lateral movement to a remote system. This type of attack comes in a variety of flavors, the most common ones including the Silver and Golden Ticket.
Source: https://attack.mitre.org/techniques/T1550/003/

As we did previously, we will start by explaining these techniques in a lot more detail and review opportunities for prevention and detection.



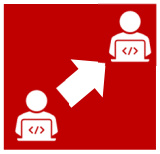
Privilege Escalation and Lateral Movement Objectives

Once initial execution is achieved, adversaries will likely focus on two key objectives as a next step:

- Privilege Escalation: When adversaries obtain access to a system as an unprivileged user, they will typically seek to escalate privileges to local administrator. This would open opportunities to dump additional credentials. These issues could occur as a result of missing software patches (known vulnerabilities) or misconfigurations.
- Lateral Movement: Similar to seeking local privilege escalation opportunities, adversaries will attempt to move laterally between systems. This could be performed by reusing credentials, but also by enumerating systems to identify vulnerabilities. Given the prevalence of Microsoft systems, deep enumeration of active directory is essential.

We will review typical strategies and detection opportunities.

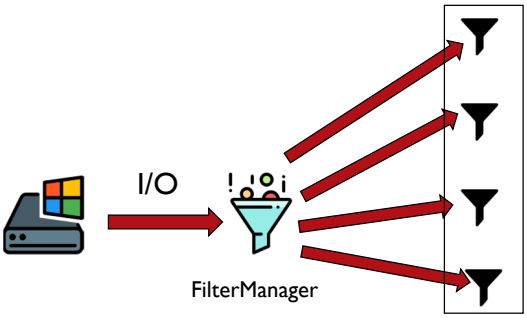
LOCAL ADMINISTRATOR PERKS – SILENCING SYSMON – UNLOADING (1)



Importance of privilege escalation
Adversaries understandably attempt to obtain as many privileges as they can in a target environment. Even local administrator access is fundamentally important to adversaries, as it will allow them to start **tampering with security controls**. We will now introduce some techniques that are aimed at **silencing security monitoring telemetry**.

Security programs (such as AV / EDR tools) need to be able to act fast. This means that the number of "layers" between the AV and the kernel should be as low as possible. To facilitate this, Microsoft introduced the Filter Manager and the concept of Minifilters.

For any system as of Windows XP, the filter manager attaches itself to the filesystem. The filter manager only becomes active, however, if it gets a registration of a Minifilter.

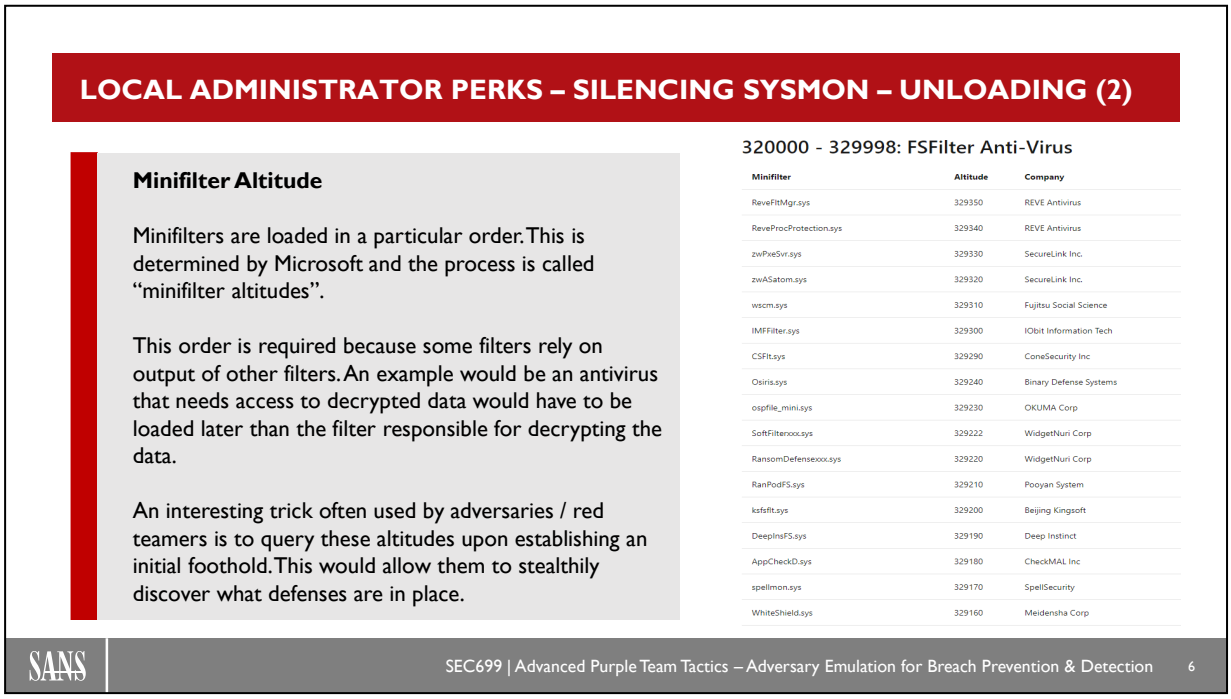


Local Administrator Perks – Silencing Sysmon – Unloading (1)
We'd like to start this section with explaining the importance of gaining administrative privileges. Adversaries understandably attempt to obtain as many privileges as they can in a target environment. Even local administrator access is fundamentally important to adversaries, as it will allow them to start tampering with security controls. We will now introduce some techniques that are aimed at silencing security monitoring telemetry.

We will start with a first attack strategy aimed at silencing Sysmon. Security programs (such as AV / EDR tools) need to be able to act fast. This means that the number of "layers" between the AV and the kernel should be as low as possible. To facilitate this, Microsoft introduced the Filter Manager and the concept of Minifilters. For any system as of Windows XP, the filter manager attaches itself to the filesystem. The filter manager only becomes active however, if it gets a registration of a Minifilter.

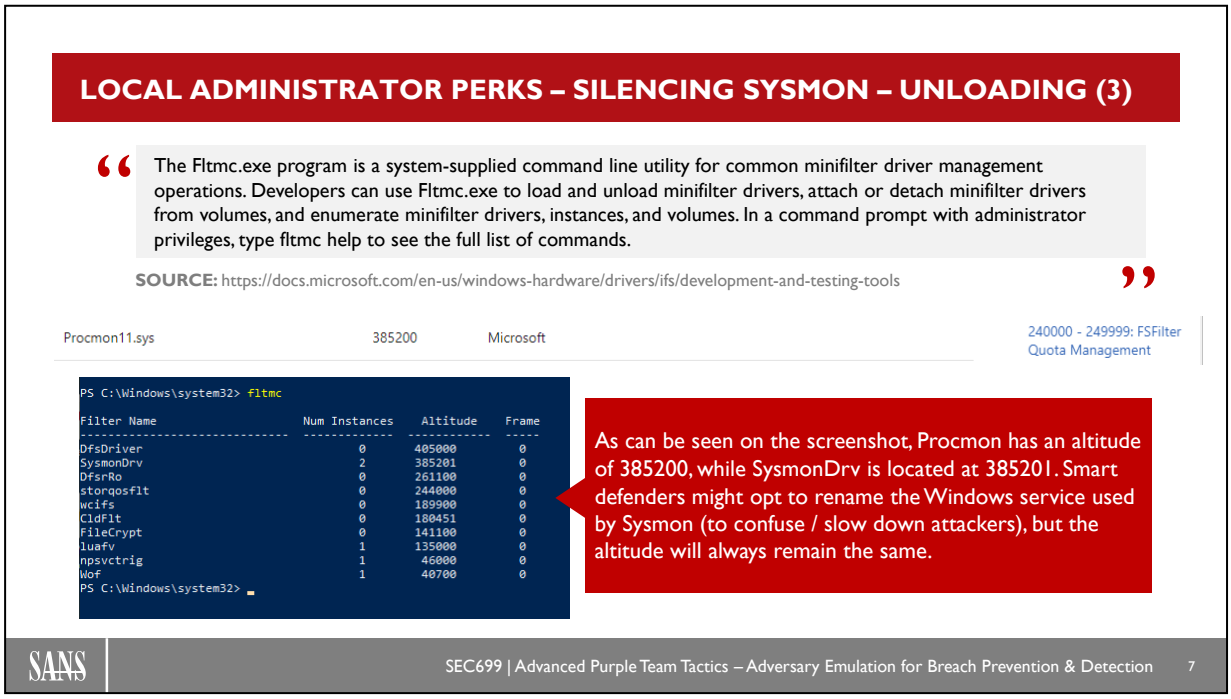
This is the reason why most AVs (if not all of them) install a driver onto your OS, which allows them to get information from the kernel directly, without having to go up the stack. Obviously, the operating system has an enormous amount of I/O to deal with, not all of which are interesting to an antivirus. Inspecting all of this I/O would cost a tremendous amount of overhead and would clutter the execution time drastically. This is why Microsoft introduced the Filter Manager and the concept of Minifilters.

Reference:
https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts



Local Administrator Perks – Silencing Sysmon – Unloading (2)
Minifilters are loaded in a particular order. This is determined by Microsoft and the process is called "minifilter altitudes". A full overview of altitudes can be found on the Microsoft knowledgebase (https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/allocated-altitudes).

This order is required because some filters rely on output of other filters. An example would be an antivirus that needs access to decrypted data would have to be loaded later than the filter responsible for decrypting the data. An interesting trick often used by adversaries / red teamers is to query these altitudes upon establishing an initial foothold. This would allow them to stealthily discover what defenses are in place.



Local Administrator Perks – Silencing Sysmon – Unloading (3)
In order to manage Minifilters, Microsoft provides a tool called fltmc.exe:

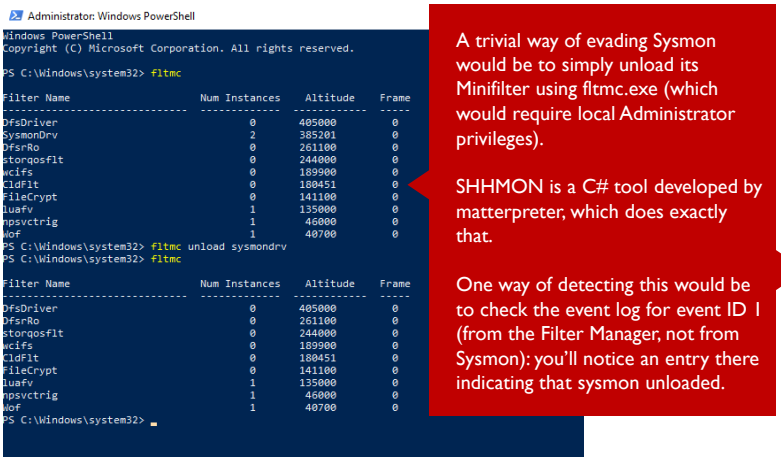
"The Fltmc.exe program is a system-supplied command line utility for common minifilter driver management operations. Developers can use Fltmc.exe to load and unload minifilter drivers, attach or detach minifilter drivers from volumes, and enumerate minifilter drivers, instances, and volumes. In a command prompt with administrator privileges, type fltmc help to see the full list of commands."

As can be seen on the screenshot, Procmon has an altitude of 385200, while SysmonDrv is located at 385201. Smart defenders might opt to rename the Windows service used by Sysmon (to confuse / slow down attackers), but the altitude will always remain the same.

Reference:

https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/development-and-testing-tools

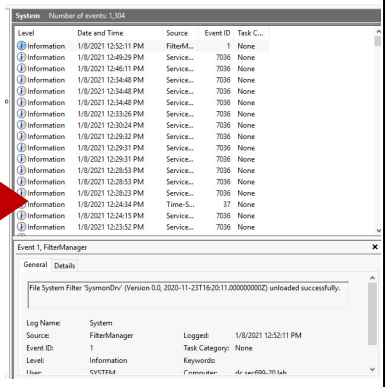
LOCAL ADMINISTRATOR PERKS – SILENCING SYSMON – UNLOADING (4)



A trivial way of evading Sysmon would be to simply unload its Minifilter using fltmc.exe (which would require local Administrator privileges).

SHHMON is a C# tool developed by matterpreter, which does exactly that.

One way of detecting this would be to check the event log for event ID 1 (from the Filter Manager, not from Sysmon): you'll notice an entry there indicating that sysmon unloaded.



Local Administrator Perks – Silencing Sysmon – Unloading (4)
A trivial way of evading Sysmon would be to simply unload it its Minifilter using fltmc.exe (which would require local Administrator privileges). SSHMON is a C# tool developed by matterpreter, which does exactly that. One way of detecting this would be to check the event log for event ID 1 (from the Filter Manager, not from Sysmon); you'll notice an entry there indicating that Sysmon unloaded.

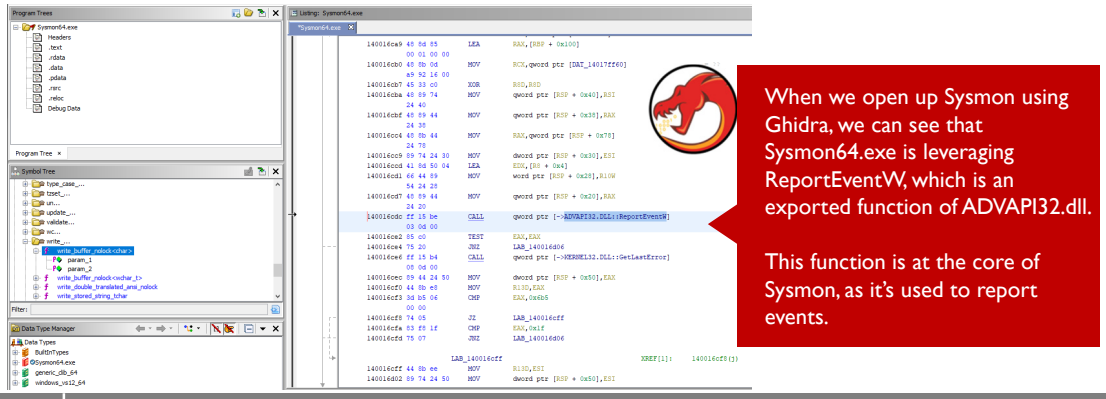
References:

https://medium.com/bugbountywriteup/unloading-the-sysmon-minifilter-driver-86f4541fa55a

https://github.com/matterpreter/Shhmon

LOCAL ADMINISTRATOR PERKS – SILENCING SYSMON – HOOKING (1)

Is there a stealthier way of bypassing Sysmon than simply unloading the Sysmon driver?
Put on your reverse-engineering caps and let's find out!



Local Administrator Perks – Silencing Sysmon – Hooking (1)
While unloading Sysmon sure is effective, it's not that stealth, as defenders could:

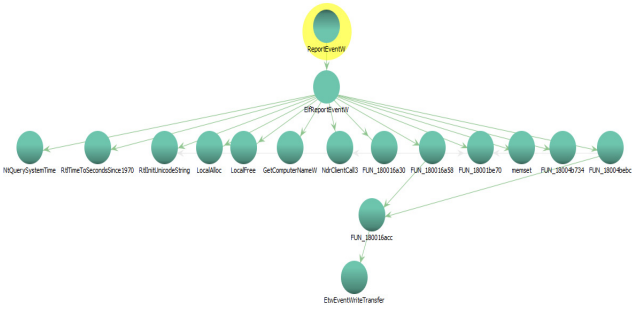
- Identify that Sysmon is no longer returning any events
- Identify the event that highlights Sysmon being unloaded

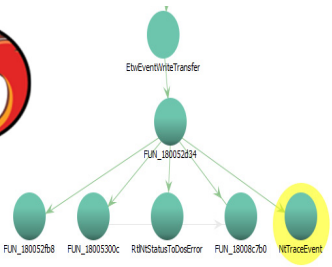
A more stealth approach was described by @Batsec, who works as a security researcher for MDSec. He took a deep-dive in Sysmon internals to better understand how it works. His deep dive yielded results, as he found a more stealth way of evading Sysmon. Let's have a look! When we open up Sysmon using Ghidra, we can see that Sysmon64.exe is leveraging ReportEventW, which is an exported function of ADVAPI32.dll. This function is at the core of Sysmon, as it's used to report events.

Reference:
https://blog.dylan.codes/evading-sysmon-and-windows-event-logging/

LOCAL ADMINISTRATOR PERKS – SILENCING SYSMON – HOOKING (2)

Inspecting the ReportEventW call further in Ghidra using the function call trees, we can determine that ReportEventW is calling EtwEventWriteTransfer and EtwEventWriteTransfer, in its turn, is calling NtTraceEvent, which is communicating with the kernel. If we want to tamper with it, we'll need a kernel-level read/write.



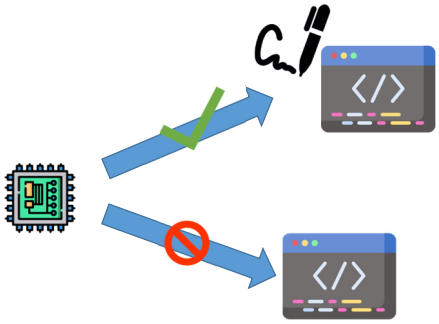


Local Administrator Perks – Silencing Sysmon – Hooking (2)
Inspecting the ReportEventW call further in Ghidra using the function call trees, we can determine that ReportEventW is calling EtwEventWriteTransfer and EtwEventWriteTransfer, in its turn, is calling NtTraceEvent, which is communicating with the kernel. If we want to tamper with it, we'll thus need kernel level read/write.

Depending on your familiarity with recent Microsoft controls, you might know that Kernel R/W access doesn't only require local admin rights: Microsoft also introduced tools such as Driver Signing Policies and Patch Guard.

Reference:
https://blog.dylan.codes/evading-sysmon-and-windows-event-logging/

LOCAL ADMINISTRATOR PERKS – SILENCING SYSMON – HOOKING (3)



Driver Signing Policy

04/20/2017 • 3 minutes to read •

Note

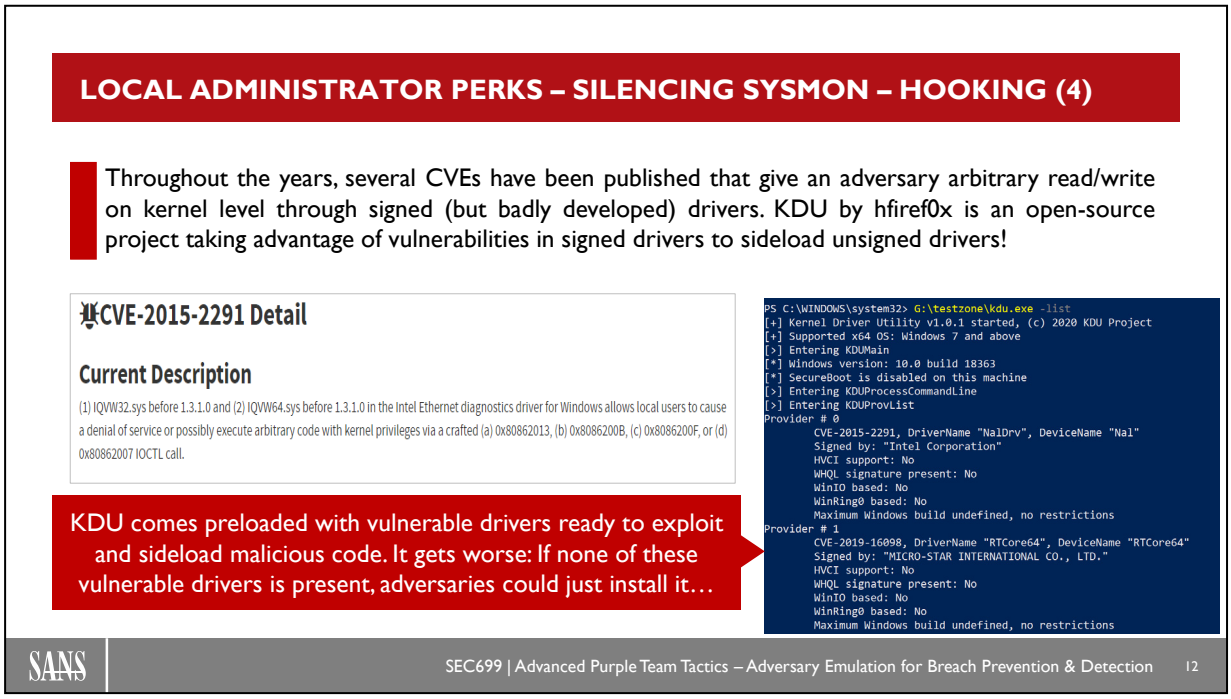
Starting with Windows 10, version 1607, Windows will not load any new kernel-mode drivers which are not signed by the Dev Portal. To get your driver signed, first **Register for the Windows Hardware Dev Center program**. Note that an **EV code signing certificate** is required to establish a dashboard account.

There are many different ways to submit drivers to the portal. For production drivers, you should submit HLK/HCK test logs, as described below. For testing on Windows 10 client only systems, you can submit your drivers for attestation signing, which does not require HLK testing. Or, you can submit your driver for Test signing as described on the Create a new hardware submission page.

Getting a driver signed by Microsoft is not a trivial fact. For adversaries, it's going to be difficult to thus add their own drivers… Could they, however, **abuse existing drivers?**

Local Administrator Perks – Silencing Sysmon – Hooking (3)

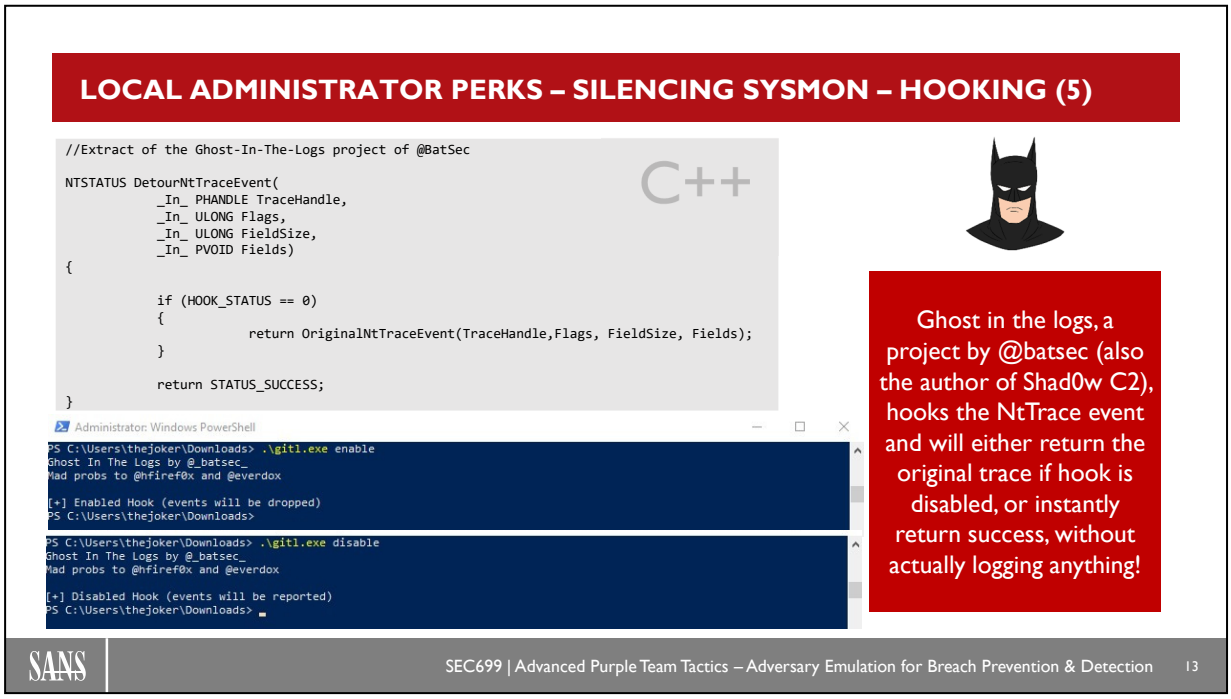
The Driver Signing Policy in Microsoft forces developers to register for the "Windows Hardware Dev Center" program, after which they can have their code /drivers signed by Microsoft. As you can imagine, this could pose problems for red teamers or actual threat actors, who don't particularly like their attacks to be attributed to them. Instead of creating and signing their own drivers, couldn't they abuse existing drivers?



Local Administrator Perks – Silencing Sysmon – Hooking (4)
Until further notice (when AI takes over globally) developers are usually humans, and humans make mistakes. Throughout the years, several CVEs have been published that give an adversary arbitrary read/write on kernel level through signed (but badly developed) drivers. KDU by hfiref0x is an open-source project taking advantage of vulnerabilities in signed drivers to sideload unsigned drivers!

This allows an attacker to effectively bypass the Driver Signing Policy and thus to sideload the malicious driver needed to patch the kernel! As a defender, we could say: "OK, let's review all signed Kernel drivers for vulnerabilities." However, this is not sufficient, as attackers could leverage local Administrator rights to install and subsequently exploit vulnerable drivers.

Reference:
https://github.com/hfiref0x/KDU



Local Administrator Perks – Silencing Sysmon – Hooking (5)
Now that we know how this all works, we understand that if we have a driver that hooks the ETW event, we can successfully patch it and blind it.

This is exactly what "Ghost-in-the-logs" does, a project by batsec.

As seen in the code extract on the slide, NtTraceEvent is being detoured. If the hook is placed, NtTraceEvent will return status_success, which will contain no additional data, resulting in seemingly OK trace events, but they are in fact completely empty!

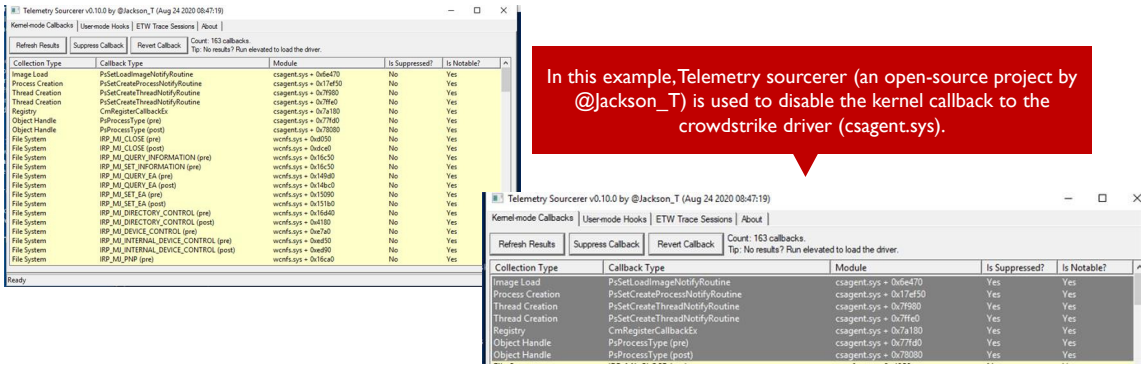
In case the hook is disabled, the detoured function just acts as a pass-through and passes execution back to the original NtTraceEvent call.

This allows for easy toggling the hook on and off as an adversary sees fit. The adversary could thus only "disable" visibility when they are about to do something malicious…

Reference:
https://github.com/bats3c/Ghost-In-The-Logs

LOCAL ADMINISTRATOR PERKS – SILENCING SYSMON – HOOKING (6)

The same methodology can also be applied to be a ghost for EDRs. If you can load an arbitrary kernel driver, you can hook other kernel drivers and disable them.



Local Administrator Perks – Silencing Sysmon – Hooking (6)
It should come to no surprise that the EDR tools that use kernel callback functions can also be circumvented using the same techniques that ghost-in-the-logs uses.

A well-known project for this is TelemetrySourcer (by Jackson_T). This also relies on exploiting a vulnerable driver to sideload the telemetrydriver.

Once loaded, it can enumerate any kernel-mode and user-mode callback hooks as well as mute specific ETW Trace Sessions. This can be used to disable callbacks to EDRs and thus effectively bypass their telemetry, making adversaries' lives significantly easier as the EDR is rendered blind.

Reference:

https://github.com/jthuraisamy/TelemetrySourcerer

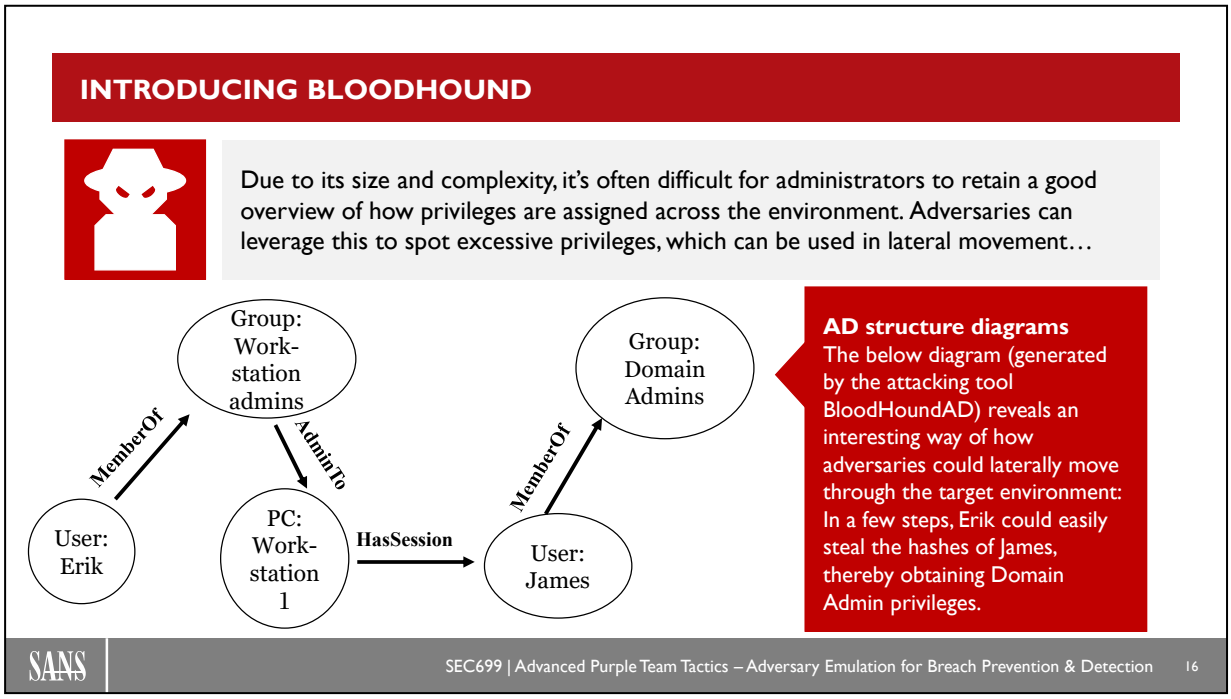
Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

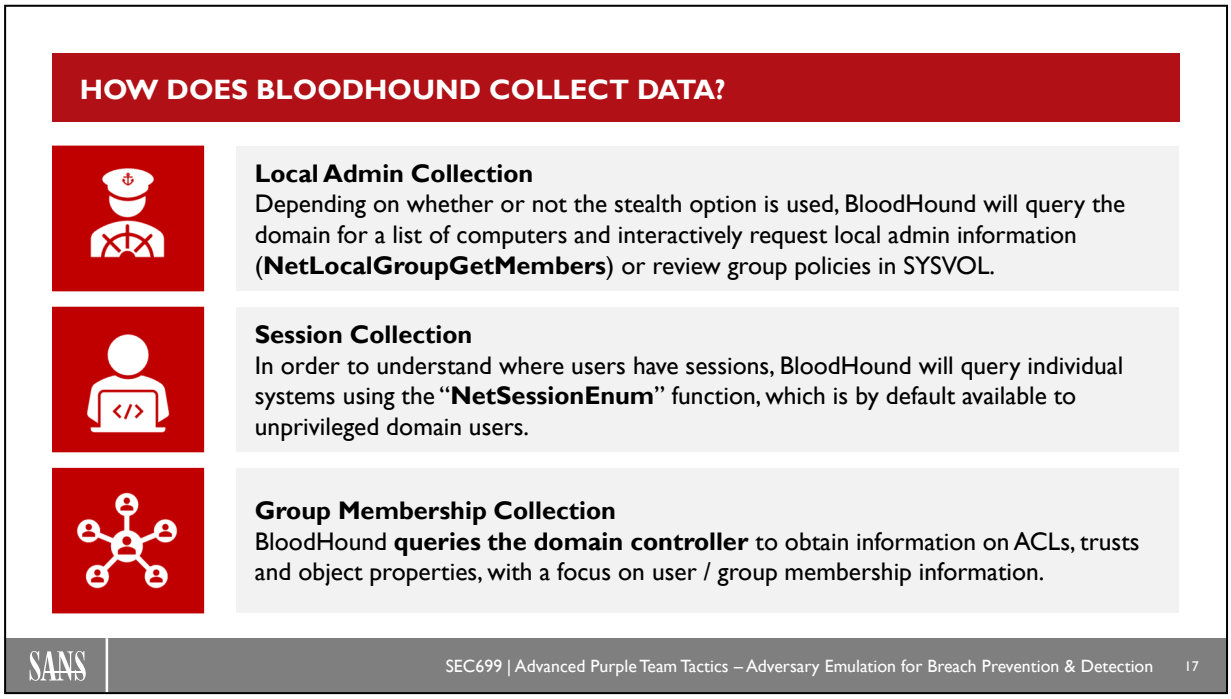
This page intentionally left blank.



Introducing BloodHound
Due to its size and complexity, it's often difficult for administrators to retain a good overview of how privileges are assigned across the environment. Adversaries can leverage this to spot excessive privileges, which can be used in lateral movement…

Once (limited) administrator privileges are obtained (e.g., on all workstations but not on servers), adversaries can start hopping from one system to the other in an attempt to steal credentials from different hops, thereby escalating privileges as they go along. An example would be a Domain Administrator that is authenticated to one of the workstations under the control of the adversary. The adversary could go to this workstation and dump the credentials from memory using Mimikatz.

A tool that facilitates this attack is BloodHoundAD, which generates a diagram of active sessions and relationships in Active Directory. On the slide above, we can see an example of such a diagram. I



How Does BloodHound Collect Data?
The graphs generated by BloodHound are a highly effective way of doing in-depth analysis by blue and red teams alike. In order to generate the graphs, however, a lot of information is to be collected from the overall domain environment. There are three main collections that need to take place:

Local Admin Collection

Depending on whether or not the stealth option is used, BloodHound will query the domain for a list of computers and interactively request local admin information (NetLocalGroupGetMembers) or review group policies in SYSVOL.

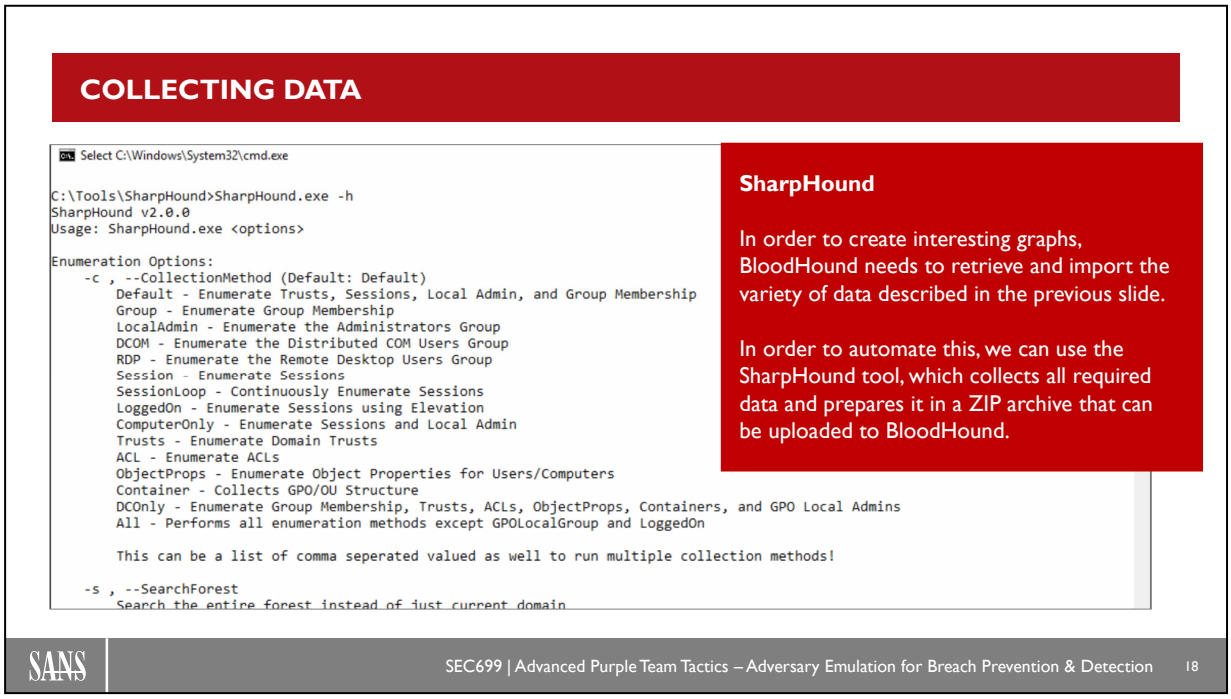
Session Collection

In order to understand where users have sessions, BloodHound will query individual systems using the "NetSessionEnum" function, which is by default available to unprivileged domain users.

Group Membership Collection

In order to map what users are part of what groups, BloodHound queries the domain controller to obtain information on ACLs, trusts and object properties. As part of the object properties, user / group membership information is enumerated.

References:
https://github.com/BloodHoundAD/Bloodhound/wiki
https://www.pentestpartners.com/security-blog/bloodhound-walkthrough-a-tool-for-many-tradecrafts/



Collecting Data

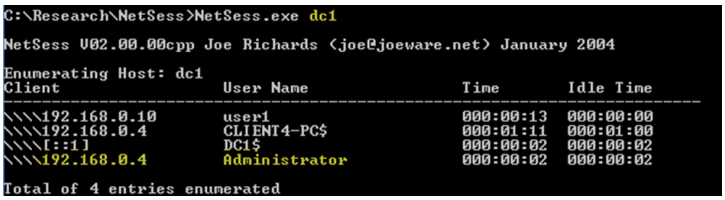
In order to create interesting graphs, BloodHound needs to retrieve and import the variety of data described in the previous slide. In order to automate this, we can use the SharpHound tool, which collects all required data and prepares it in a ZIP archive that can be uploaded to BloodHound.

In the screenshot above, we can see the SharpHound help feature, where a variety of different options is presented:

- Default
- Group
- LocalAdmin
- DCOM
- RDP
- Session
- SessionLoop
- …

Note that, depending on the level of stealth required, users can use a variety of enumeration options. The more stealth required, the less easy it becomes to enumerate data.

CAN YOU REALLY ENUMERATE SESSIONS WITHOUT PRIVILEGES?



SOURCE: https://docs.microsoft.com/en-us/samples/browse/?redirectedfrom=TechNet-Gallery

A question we hear often is whether or not we can enumerate logon / session information without administrative access to the target machine. In fact, by default, this is possible, as domain users can execute the NetSessionEnum. By default, this means they can query the following information:

The **name/IP address** of the computer.
The **name of the user** who established the session.
The number of seconds the **session has been active**. (since the query)
The number of seconds the **session has been idle**. (since the query)

Can You Really Enumerate Sessions without Privileges?
A question we hear often is whether or not we can enumerate logon / session information without administrative access to the target machine. In fact, by default, this is possible, as domain users can execute the NetSessionEnum method. By default, this means they can query the following information:

- The name/IP address of the computer.
- The name of the user who established the session.
- The number of seconds the session has been active. (since the query)
- The number of seconds the session has been idle. (since the query)

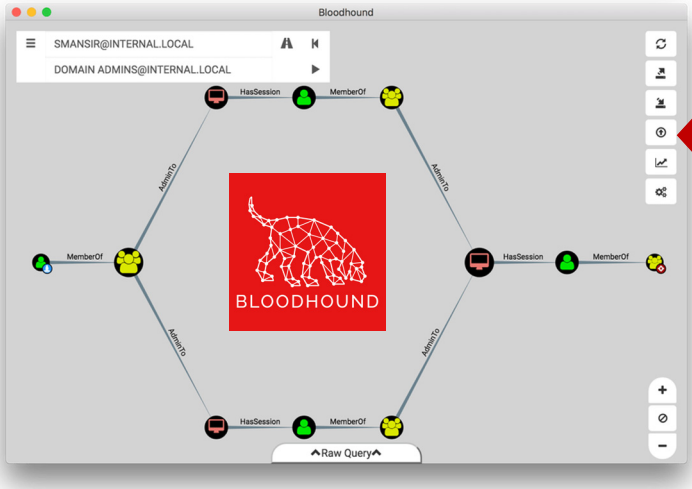
A tool that supports this is NetSess.exe (http://www.joeware.net/freetools/tools/netsess/index.htm).

The permissions required to enumerate sessions through SMB are handled by a registry key in HKLM\SYSTEM\CurrentControlSet\Services\LanmanServer\DefaultSecurity. Net Cease is a PowerShell script developed by Itai Grady, available on Microsoft TechNet. Net Cease will harden SMB session enumeration on target systems by modifying the “SrvsvcSessionInfo” registry key.

Note that, next to SMB enumeration, BloodHound can still collect logged on information through querying the remote registry (analyze HKEY_USERS).

For full information, please refer to https://gallery.technet.microsoft.com/Net-Cease-Blocking-Net-1e8dcb5b.

BLOODHOUND IN ACTION: GRAPH INTERFACE



BloodHound Graph Interface

In the graph to the left, we can see the result of one of the queries.

In this specific case, a user attempted to find the shortest path to getting Domain Administrator. It appears there are two available paths, which are clearly described.

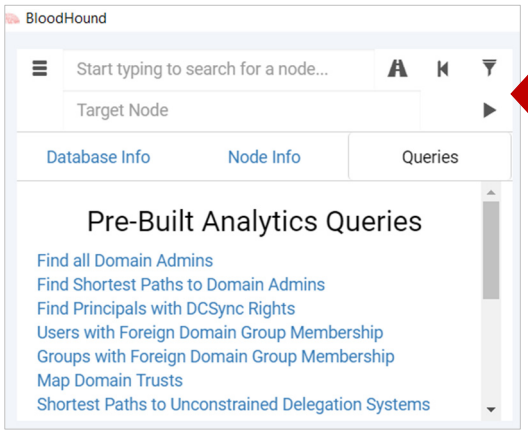
A real-life graph of an enterprise environment will likely look more complex!

BloodHound in Action: Graph Interface

In the graph on the slide, we can see the result of one of the queries. BloodHound uses a neo4j database to store all of the information and provides it in a visual web interface for analysis. In this specific case, a user attempted to find the shortest path to getting Domain Administrator. It appears there are two available paths, which can be clearly seen in the diagram.

As a next step, the attacker would now authenticate to one of the first machines in the graph and attempt to steal credentials of the next user in the attack path. Note that in this specific case, there are only two computer hops; a real-life graph of an enterprise environment will likely look more complex!

BLOODHOUND BASIC QUERIES



BloodHound Queries

Once BloodHound has ingested data (which can be collected using different tools), it can start creating graphs for further analysis. You can develop your own queries or use one of the many readily available built-in queries:

- Find all Domain Admins
- Find shortest paths to Domain Admins
- Find principals with DCSync rights
- …

One of the most interesting (and most well-known) is the "Find Shortest Paths to Domain Admins".

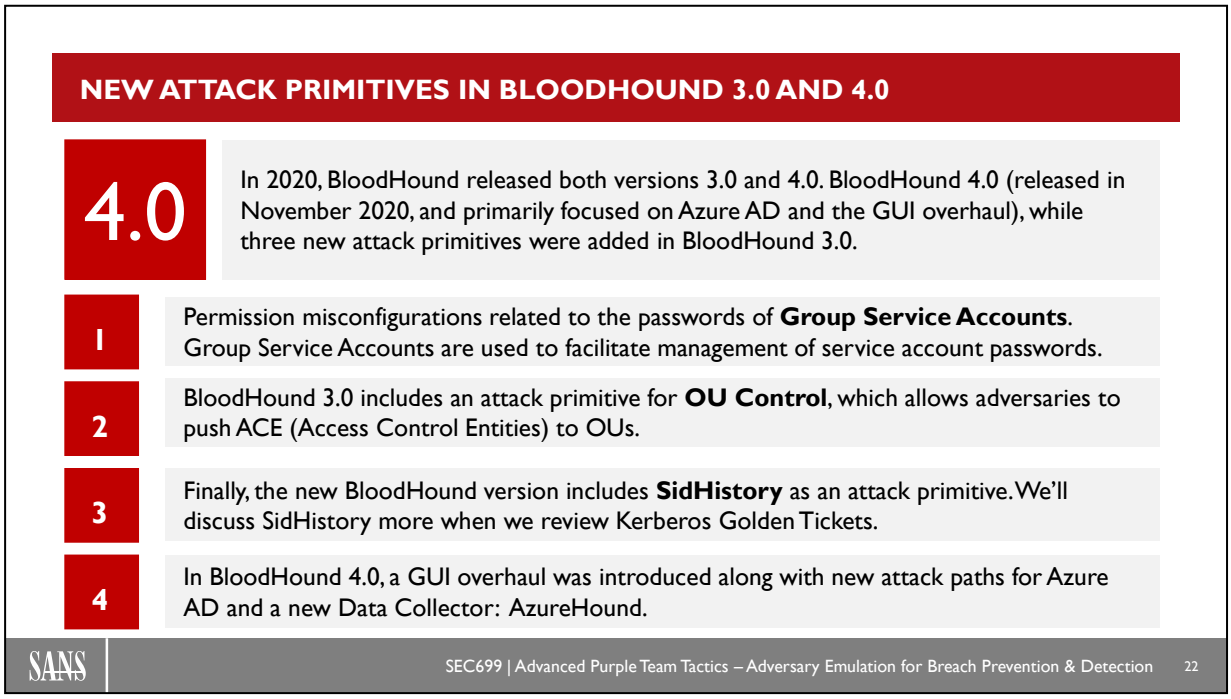
BloodHound Basic Queries

Once BloodHound has ingested data (which can be collected using different tools), it can start creating graphs for further analysis. BloodHound provides multiple tools for data collection (called ingestors). A good example is SharpHound, which is a C# ingestor, the primary ingestor to run on Windows systems.

You can develop your own queries or use one of the many readily available built-in queries. Some examples of available queries include:

- Find all Domain Admins.
- Find Shortest Paths to Domain Admins.
- Find Principals with DCSync Rights.
- …

One of the most interesting (and most well-known) is the "Find Shortest Paths to Domain Admins"!



New Attack Primitives in BloodHound 3.0 and 4.0

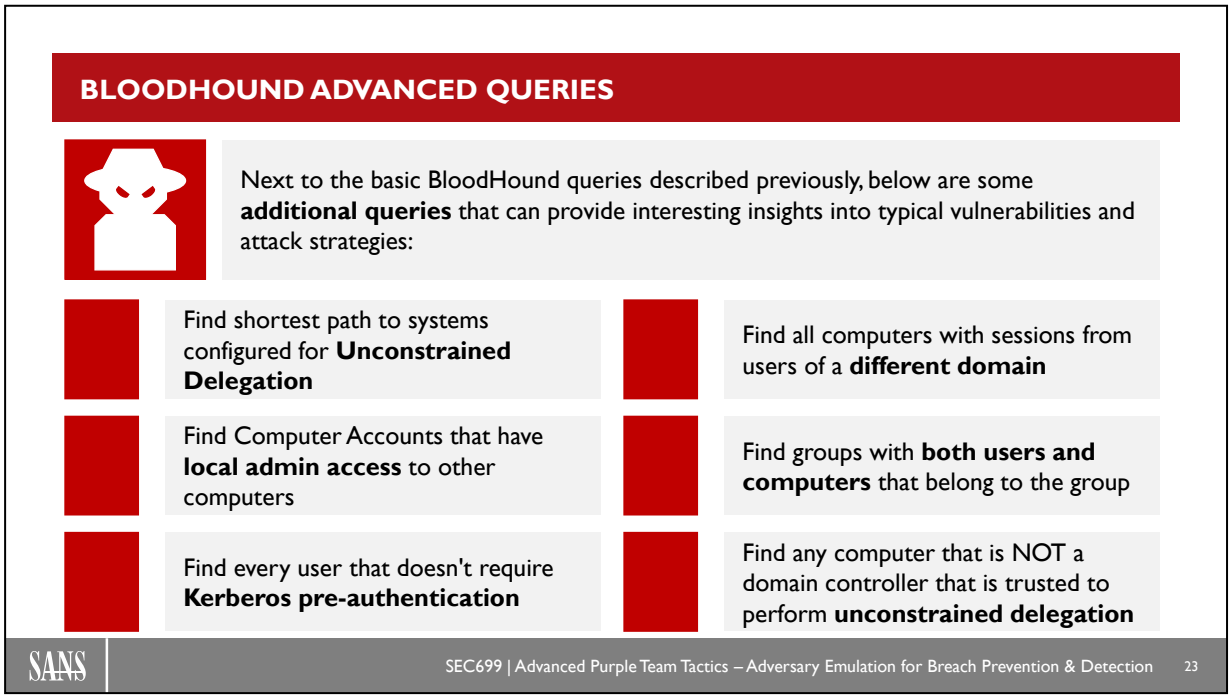
In February 2020, BloodHound 3.0 was released, which included three new interesting attack primitives that further enhance BloodHound's capabilities:

1. Permission misconfigurations related to the passwords of Group Service Accounts. Group Service Accounts are used to facilitate management of service account passwords. The goal here is to identify badly configured service accounts where adversaries could be able to read the cleartext passwords set.
2. BloodHound 3.0 includes an attack primitive for OU Control, which allows adversaries to push ACE (Access Control Entities) to OUs.
3. Finally, the new BloodHound version includes SidHistory as an attack primitive. We'll discuss SidHistory more when we review Kerberos Golden Tickets.

You can find additional information on the features in BloodHound 3.0 release here:
https://posts.specterops.io/introducing-bloodhound-3-0-c00e77ff0aa6

Furthermore, BloodHound 4.0 is tailored to teach the hound some cloud tricks, a new data collector was introduced called AzureHound. Along with AzureHound, a ton of new attack paths where added, all tailored to AzureAD.

Additional information on BloodHound 4.0 can be found here:
https://posts.specterops.io/introducing-bloodhound-4-0-the-azure-update-9b2b26c5e350

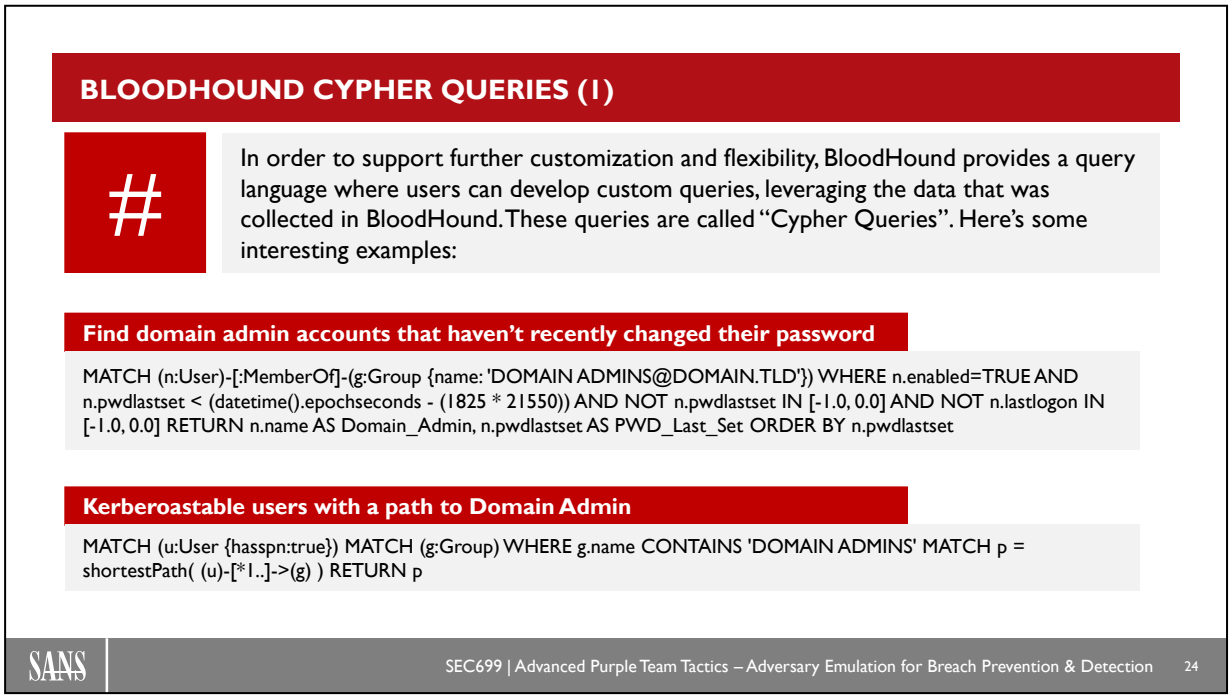


BloodHound Advanced Queries

Next to the basic BloodHound queries described previously, below are some additional queries that can provide interesting insights into typical vulnerabilities and attack strategies:

- Find shortest path to systems configured for Unconstrained Delegation
- Find Computer Accounts that have local admin access to other computers
- Find every user that doesn't require Kerberos pre-authentication
- Find all computers with sessions from users of a different domain
- Find groups with both users and computers that belong to the group
- Find any computer that is NOT a domain controller that is trusted to perform unconstrained delegation

It's a good idea to visit the BloodHound query gallery hosted here:
https://github.com/BloodHoundAD/BloodHound/wiki



BloodHound Cypher Queries (1)
In order to support further customization and flexibility, BloodHound provides a query language where users can develop custom queries, leveraging the data that was collected in BloodHound. These queries are called "Cypher Queries". Here's some interesting examples:

Find domain admin accounts that haven't recently changed their password

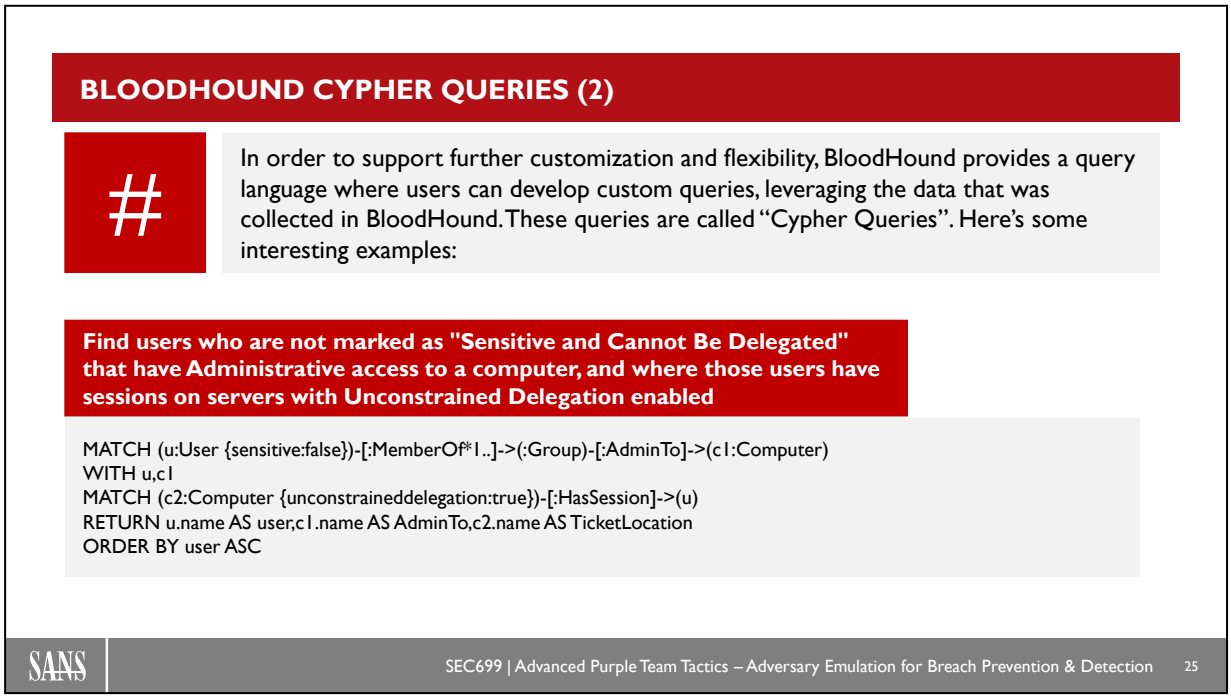
MATCH (n:User)-[:MemberOf]-(g:Group {name: 'DOMAIN ADMINS@DOMAIN.TLD'}) WHERE n.enabled=TRUE AND n.pwdlastset < (datetime().epochseconds - (1825 * 21550)) AND NOT n.pwdlastset IN [-1.0, 0.0] AND NOT n.lastlogon IN [-1.0, 0.0] RETURN n.name AS Domain_Admin, n.pwdlastset AS PWD_Last_Set ORDER BY n.pwdlastset

Kerberoastable users with a path to Domain Admin

MATCH (u:User {hasspn:true}) MATCH (g:Group) WHERE g.name CONTAINS 'DOMAIN ADMINS' MATCH p = shortestPath((u)-[*1..]->(g)) RETURN p

For additional query inspiration, please find some good references below:

https://ernw.de/download/BloodHoundWorkshop/ERNW_DogWhispererHandbook.pdf
https://blog.cptjesus.com/posts/introtocypher
https://hausec.com/2019/09/09/bloodhound-cypher-cheatsheet/
https://github.com/BloodHoundAD/BloodHound/wiki
BloodHound Slack #cypher_queries channel.

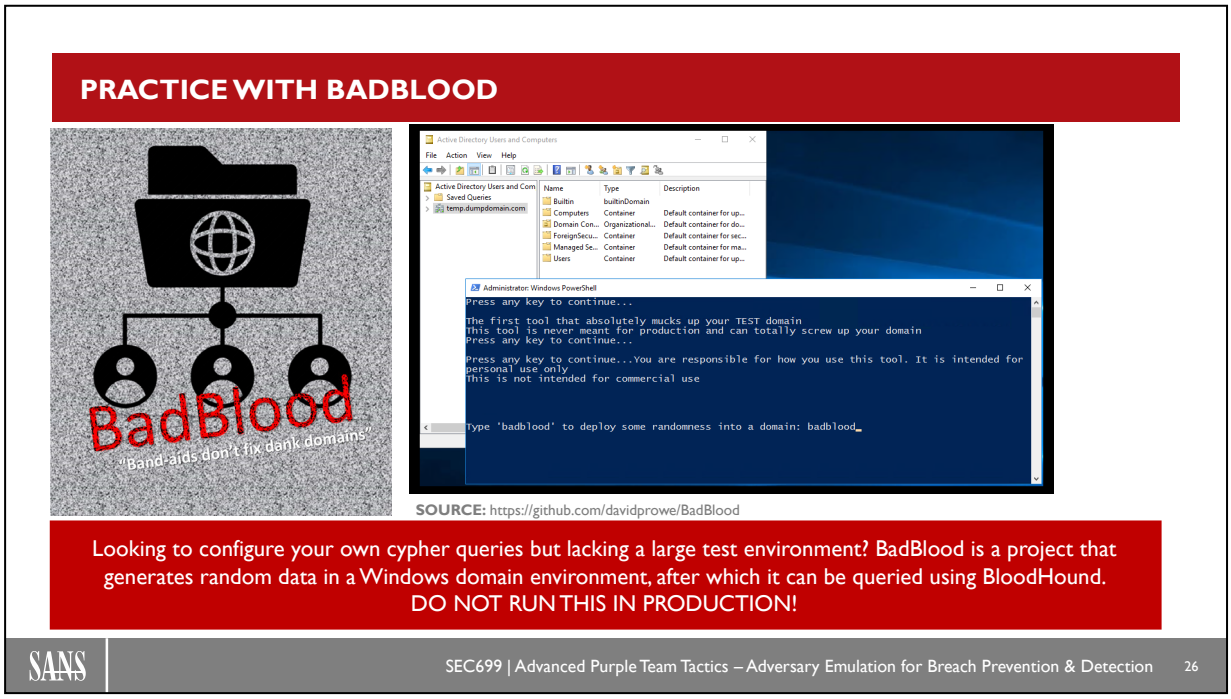


BloodHound Cypher Queries (2)

A final example of an interesting query can be found below:

Find users who are not marked as "Sensitive and Cannot Be Delegated" that have Administrative access to a computer, and where those users have sessions on servers with Unconstrained Delegation enabled

```
MATCH (u:User {sensitive:false})-[:MemberOf*1..]->(:Group)-[:AdminTo]->(c1:Computer)
WITH u,c1
MATCH (c2:Computer {unconstraineddelegation:true})-[:HasSession]->(u)
RETURN u.name AS user,c1.name AS AdminTo,c2.name AS TicketLocation
ORDER BY user ASC
```

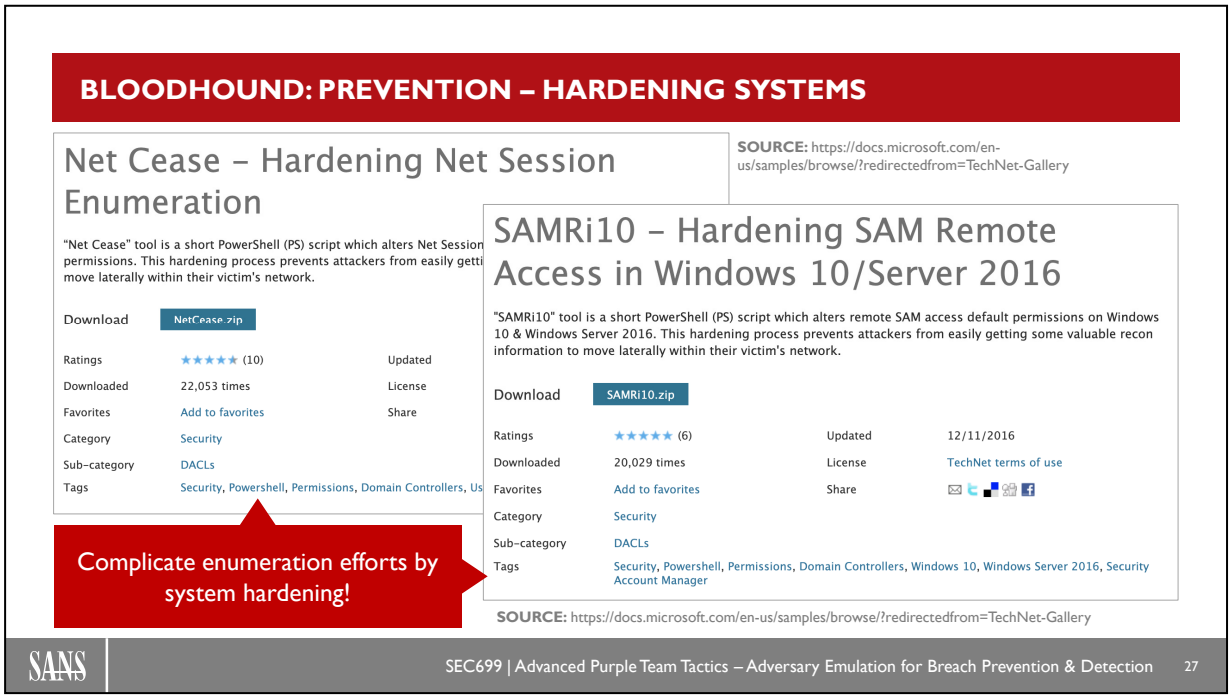


Practice with BadBlood
Although it's relatively easy to set up a Windows domain environment with some base users, computers, and groups, it can prove harder to create a realistic domain environment. BadBlood is an open-source tool that can help facilitate this. From their GitHub page:

"BadBlood by Secframe fills a Microsoft Active Directory Domain with a structure and thousands of objects. The output of the tool is a domain similar to a domain in the real world. After BadBlood is ran on a domain, security analysts and engineers can practice using tools to gain an understanding and prescribe to securing Active Directory. Each time this tool runs, it produces different results. The domain, users, groups, computers and permissions are different. Every. Single. Time."

It should be obvious that this script should NOT be run in a production environment, as it will create a wide variety of users, groups, computers, and associated permissions.

Reference:
https://github.com/davidprowe/BadBlood

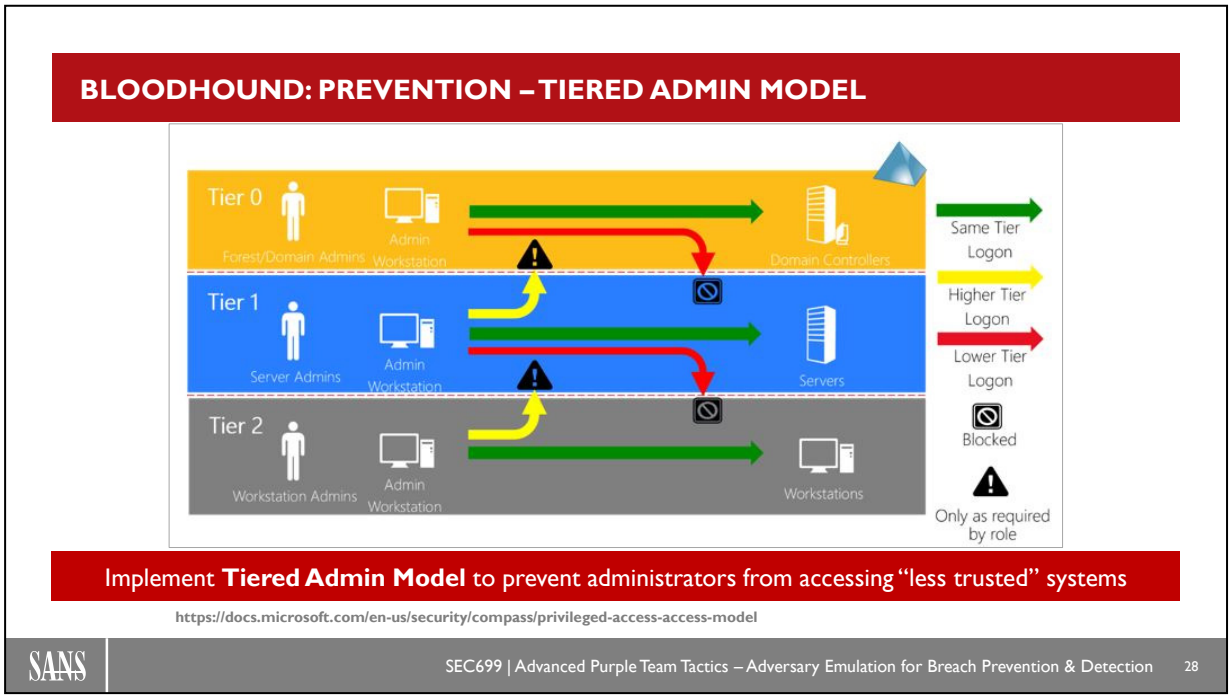


BloodHound: Prevention – Hardening Systems
How could we stop BloodHound?

BloodHound relies on a series of built-in Microsoft Windows mechanisms to enumerate information about the domain. One opportunity to hinder enumeration efforts is to harden the Windows system to provide less information. Two highly interesting tools were written and published on Microsoft TechNet by Itai Grady:

- We can limit access to the NetSessionEnum method, which is used for session enumeration over SMB. The NetCease script hardens the access to the NetSessionEnum method by removing the execute permission for the Authenticated Users group and adding permissions for interactive, service and batch logon sessions. This will allow any administrator, system operator, and power user to remotely call this method, and any interactive/service/batch logon session to call it locally. Standard enumeration using an unprivileged user account will no longer be feasible though.
- We can limit access to the remote SAM (via the SAM-Remote) protocol. The SAMRi10 script hardens the remote access to the SAM by only giving permission for members of the Administrators group or the newly created group named "Remote SAM Users" (this group is created by the SAMRi10 script). This will allow any administrator, or any service/user account added to the "Remote SAM Users" local group, to remotely access the SAM on the hardened machine. Standard enumeration using an unprivileged user account will no longer be feasible though.

References:
https://gallery.technet.microsoft.com/Net-Cease-Blocking-Net-1e8dcb5b
https://gallery.technet.microsoft.com/SAMRi10-Hardening-Remote-48d94b5b



BloodHound: Prevention – Tiered Admin Model
Instead of trying to prevent BloodHound from enumerating information, we could try to tackle the problems it tries to identify; administrative users that have access to systems we could possibly compromise.

Microsoft developed the "administrative tiered model" (standard three tiers), which applies to administrative accounts (not normal user accounts). The purpose of the model is to protect key identity systems (e.g., domain controllers) from more high-risk systems such as workstations (which are frequently compromised).

The defined tiers are:

- Tier 0: Administrative access to directory services such as domain controllers, where central management is performed. Typical profiles include domain administrators.
- Tier 1: Administrative access to enterprise servers and applications, where sensitive data is typically centralized. Typical profiles include server administrators.
- Tier 2: Administrative access to workstations, where possibly sensitive data can be stolen. Typical profiles include help/service desk personnel.

The full description can be found on Microsoft's knowledge base:

https://docs.microsoft.com/en-us/windows-server/identity/securing-privileged-access/securing-privileged-access-reference-material

The "buffers" can be enforced using group policies. As a simple example, domain administrators should not be able to authenticate to tier 1 or tier 2 systems (e.g., member servers or workstations). Additional documentation can be found here:

https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/appendix-f--securing-domain-admins-groups-in-active-directory

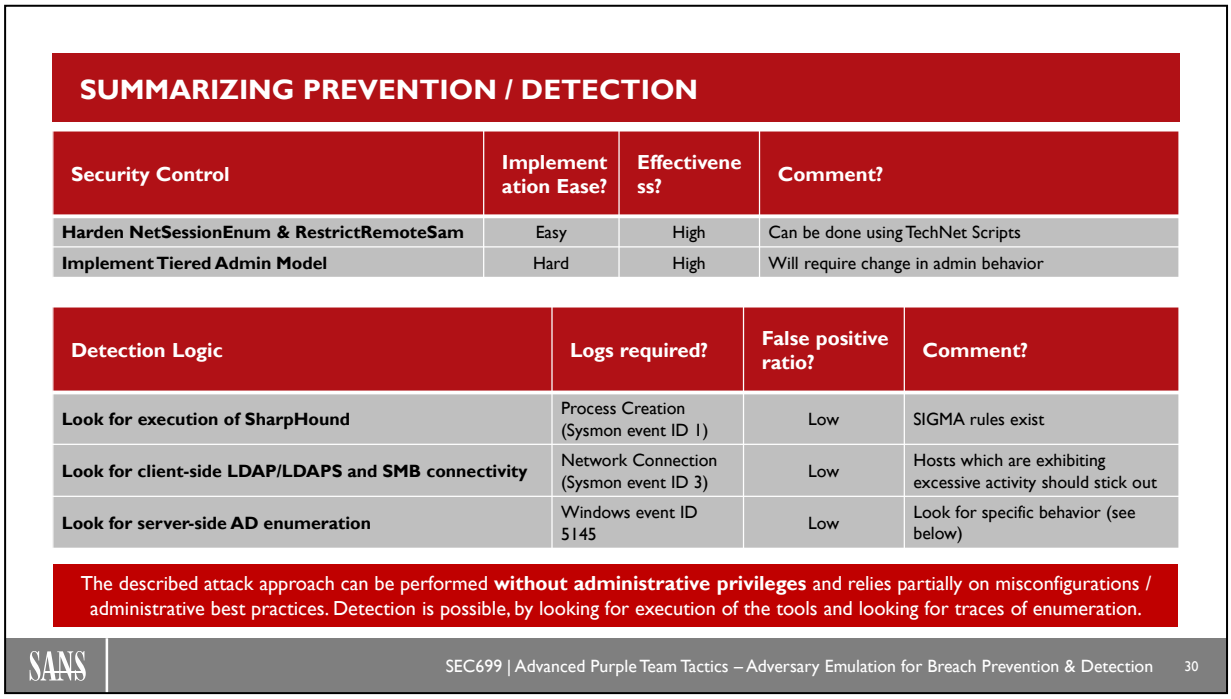
BLOODHOUND: DETECTION – EXAMPLE SIGMA RULE

```
title: Bloodhound and Sharphound Hack Tool
id: f376c8a7-a2d0-4ddc-aa0c-16c17236d962
description: Detects command line parameters used by Bloodhound
and Sharphound hack tools
author: Florian Roth
references:
    - https://github.com/BloodHoundAD/BloodHound
    - https://github.com/BloodHoundAD/SharpHound
date: 2019/12/20
modified: 2019/12/21
tags:
    - attack.discovery
    - attack.t1087
logsource:
    category: process_creation
    product: windows
detection:
    selection1:
        Image|contains:
            - '\Bloodhound.exe'
            - '\SharpHound.exe'
    selection2:
        CommandLine|contains:
            - ' -CollectionMethod All '
            - '.exe -c All -d '
            - 'Invoke-Bloodhound'
            - 'Get-BloodHoundData'
 selection3:
        CommandLine|contains|all:
            - ' -JsonFolder '
            - ' -ZipFileName '
    selection4:
        CommandLine|contains|all:
            - ' DCOnly '
            - ' --NoSaveCache '
    condition: 1 of them
falsepositives:
    - Other programs that use these command line option and
accepts an 'All' parameter
level: high
```

A good example of tool-based detection is the rule above that looks for typical **command line arguments used by BloodHound / SharpHound**.

BloodHound: Detection – Example Sigma Rule
A good example of tool-based detection is the rule above that looks for typical command-line arguments used by BloodHound / SharpHound. It leverages Sysmon Event ID 1, ProcessCreate!

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma



Summarizing Prevention / Detection

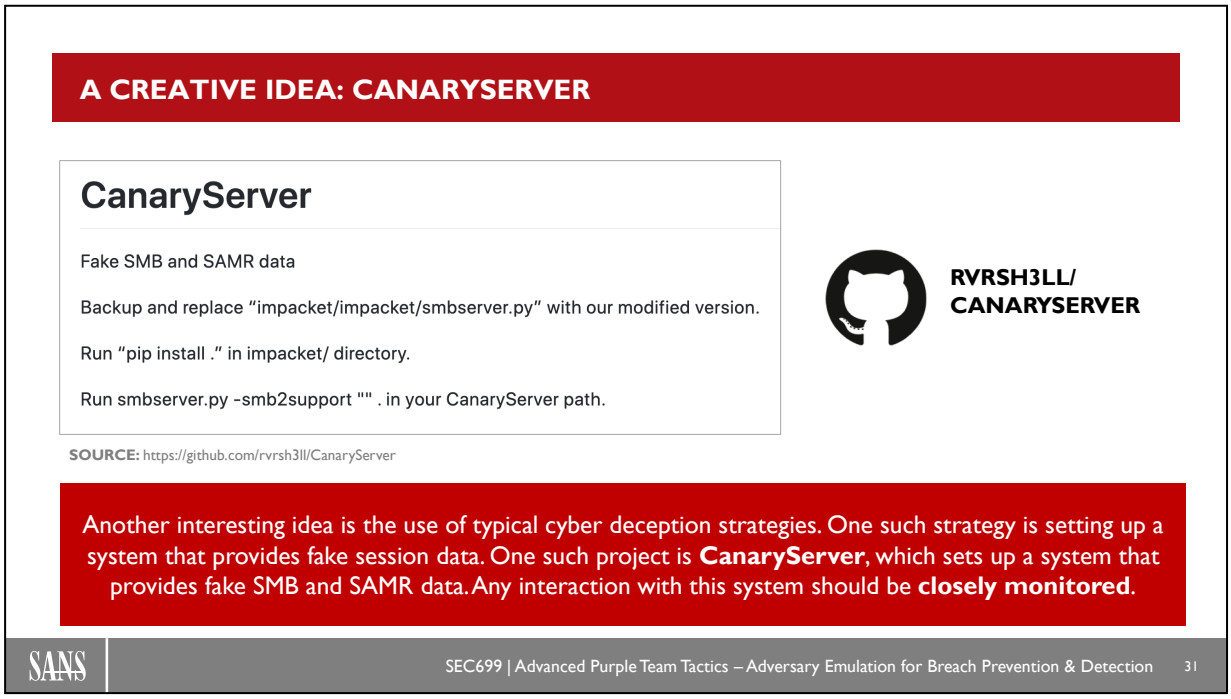
Enumeration of a Windows domain environment is a built-in capability. The options for prevention are:

- Harden the Windows systems to limit enumeration opportunities (using NetSessionEnum and RestrictRemoteSam)
- Implement the Tiered Admin Model to prevent administrators from logging in

For there's a few strategies to consider:

- We can detect SharpHound execution using Sysmon event ID 1 (Process Creation). Example SIGMA rules exist
- We can look for LDAP/LDAPS and SMB connectivity toward the workstations:
  - Multiple connections to LDLDAPS (389/636) and SMB (445) TCP ports
  - Multiple connection to named pipes "srvsvc" and "lsass"
- We can look for artifacts on the DC:
  - AP/Connections to named pipes srvsvc, lsarpc and samr (apply to "default" and "all" scan modes)
  - Connections to named pipe srvsvc and access to share relative target name containing "Groups.xml" and "GpTmpl.inf" (apply to --Stealth scan mode)

An interesting article is here: https://blog.menasec.net/2019/02/threat-hunting-7-detecting.html

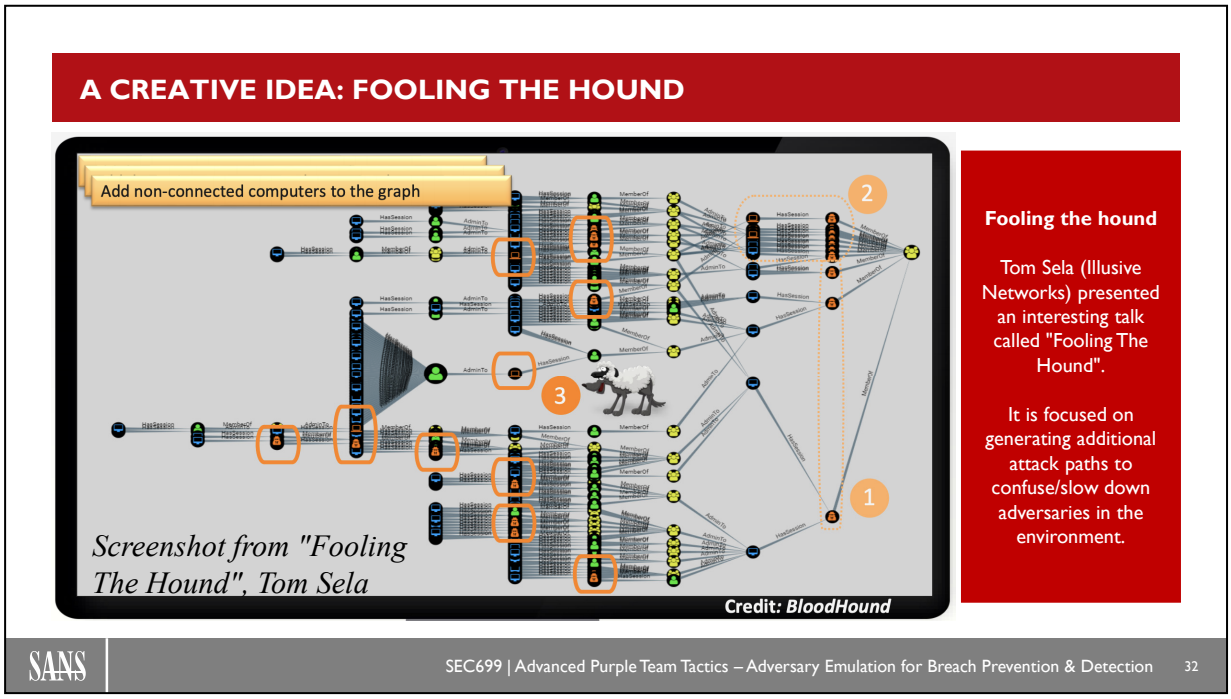


A Creative Idea: CanaryServer
While we can definitely review logs to identify SharpHound activity in the environment, this might be tricky.

Can we maybe lure the adversaries to us, or trick them? This is where cyber deception strategies come into play. One such strategy is setting up a system that provides fake session data. One such project is CanaryServer, which sets up a system that provides fake SMB and SAMR data. This is something a typical BloodHound / SharpHound scan would fall for!

Any interaction with this system should be closely monitored…

Full details can be found on https://github.com/rvrsh3ll/CanaryServer.



A Creative Idea: Fooling the hound
Another good example of a creative deception strategy was presented by Tom Sela (Illusive Networks). He created a talk called "Fooling The Hound—Deceiving Domain Admin Hunters". The graph on the slide is an extract from the presentation.

Instead of setting up one server (like CanaryServer), he plants fake information across the environment.

Fake paths are generated by providing fake responses to:

- RegEnumK (used to enumerate HKEY_USERS hive)
- NetWkstaUserEnum (similar to honeyhashes described previously)
- NetSessionEnum (used to enumerate user sessions)
- NetLocalGroupGetMembers (used to enumerate group memberships)

The full presentation can be found here:
https://cdn.shopify.com/s/files/1/0177/9886/files/phv2017-tsela.pdf

Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.

EXERCISE: ANALYZING BLOODHOUND ATTACK CHAINS

Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

Course Roadmap

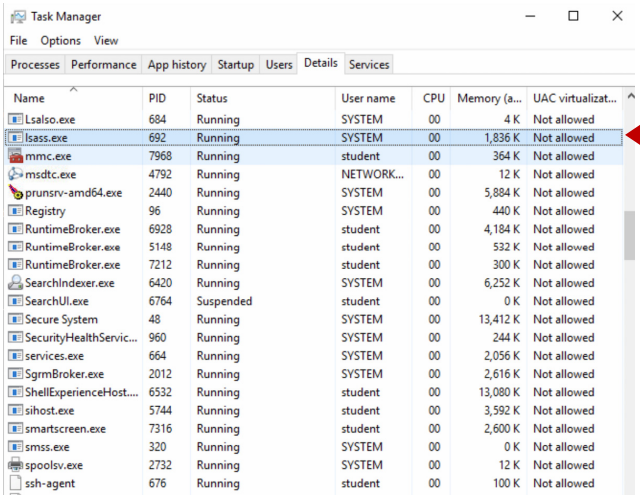
- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.

LSASS CREDENTIAL STEALING TECHNIQUES



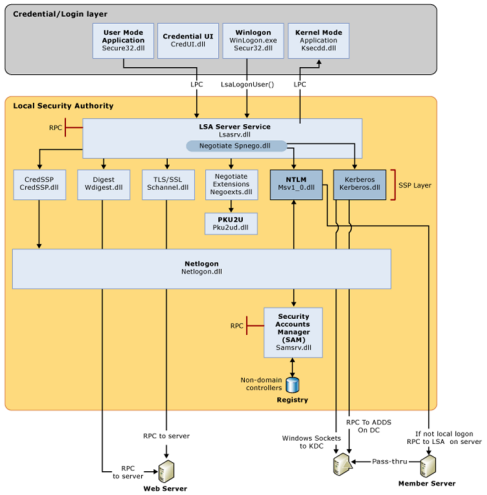
Once adversaries have obtained local administrative access to a machine, they will often attempt to dump credentials that would enable further lateral movement / privilege escalation in the rest of the environment. A key target is the LSASS process!

LSASS Credential Stealing Techniques
In the screenshot on the slide, we can see the lsass.exe process running in the task manager. It's running under the Username "SYSTEM" and is a core Windows process.

LSASS is widely known as the "holy grail" when attacking a Windows workstation or server… Once adversaries have obtained local administrative access to a machine, they will often attempt to dump credentials that would enable further lateral movement / privilege escalation in the rest of the environment. A key target for such an attack strategy is the LSASS process!

But why is it that this process has such tremendous value? Let's investigate!

WHAT IS LSASS?



LSASS (Local Security Authority Subsystem Service) is one of the core Microsoft Windows processes that is responsible for **enforcing security policy** on systems.

Security Support Providers (SSPs) are DLLs that are loaded into LSASS. They handle authentication packages and can typically store credentials in memory. Example SSPs include NTLM, Kerberos, and Digest.

SOURCE: https://docs.microsoft.com/en-us/windows-server/security/windows-authentication/credentials-processes-in-windows-authentication

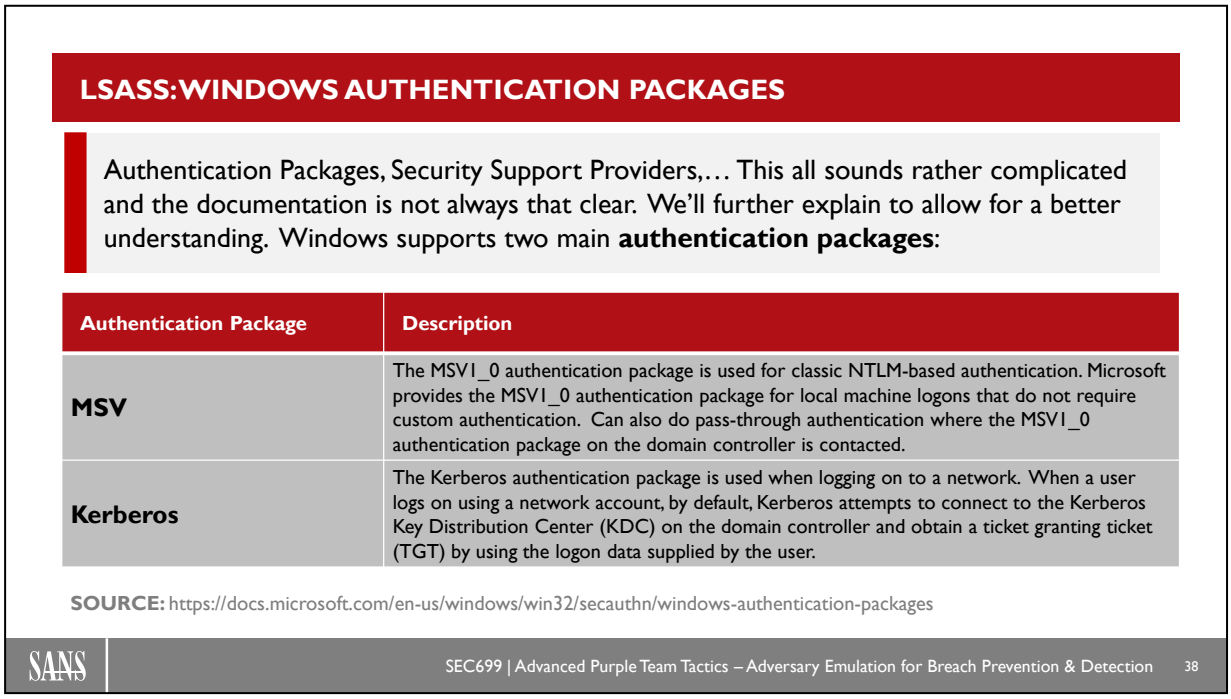
What Is LSASS?

So, what is lsass.exe? LSASS (Local Security Authority Subsystem Service) is one of the core Microsoft Windows processes that is responsible for enforcing security policy on systems. Some more facts:

- It is started upon boot
- It has wininit.exe as a parent process
- It runs as NT AUTHORITY\SYSTEM
- It is launched from %Systemroot%\System32\lsass.exe
- It should never spawn child processes

A key component of LSASS are security support providers (SSPs), which support the different authentication options in Windows (e.g., NTLM, Kerberos or Digest). In practice, Security Support Providers (SSPs) are DLLs that are loaded into LSASS. They handle authentication packages and can typically store credentials in memory. They are thus often the target of attacks, such as Mimikatz' attempts to steal cleartext credentials from Digest.

If you're interested, additional, in-depth information can be found in Microsoft's documentation: https://docs.microsoft.com/en-us/windows-server/security/windows-authentication/credentials-processes-in-windows-authentication



LSASS: Windows Authentication Packages

As we saw in the previous slide, the overall LSASS process and function is rather complex. When doing a bit of research, you may have encountered several terms being used "in the wild": Authentication packages, security support providers,… It can sometimes become a little bit daunting AND these concepts are actually not that far from one another.

Let's start with the basics. For simplicity's sake, there are two main authentication packages in Microsoft Windows:

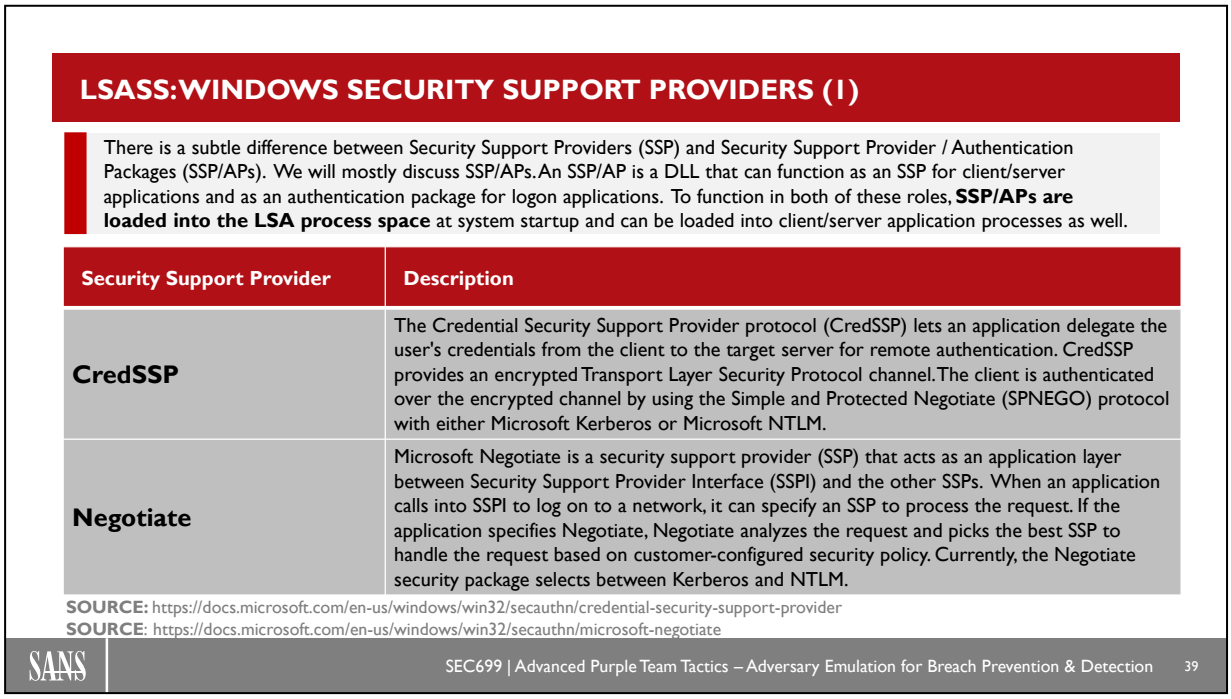
MSV

The MSV1_0 authentication package is used for classic NTLM-based authentication. Microsoft provides the MSV1_0 authentication package for local machine logons that do not require custom authentication. Can also do pass-through authentication where the MSV1_0 authentication package on the domain controller is contacted.

Kerberos

The Kerberos authentication package is used when logging on to a network. When a user logs on using a network account, by default, Kerberos attempts to connect to the Kerberos Key Distribution Center (KDC) on the domain controller and obtain a ticket granting ticket (TGT) by using the logon data supplied by the user.

Full documentation can be found here:
https://docs.microsoft.com/en-us/windows/win32/secauthn/windows-authentication-packages



LSASS: Windows Security Support Providers (1)
So, what are these Security Support Providers?

There is a subtle difference between Security Support Providers (SSP) and Security Support Provider / Authentication Packages (SSP/APs). We will mostly discuss SSP/APs. An SSP/AP is a DLL that can function as an SSP for client/server applications and as an authentication package for logon applications. To function in both of these roles, SSP/APs are loaded into the LSA process space at system startup and can be loaded into client/server application processes as well. We will walk through the different SSPs that are built in by Microsoft.

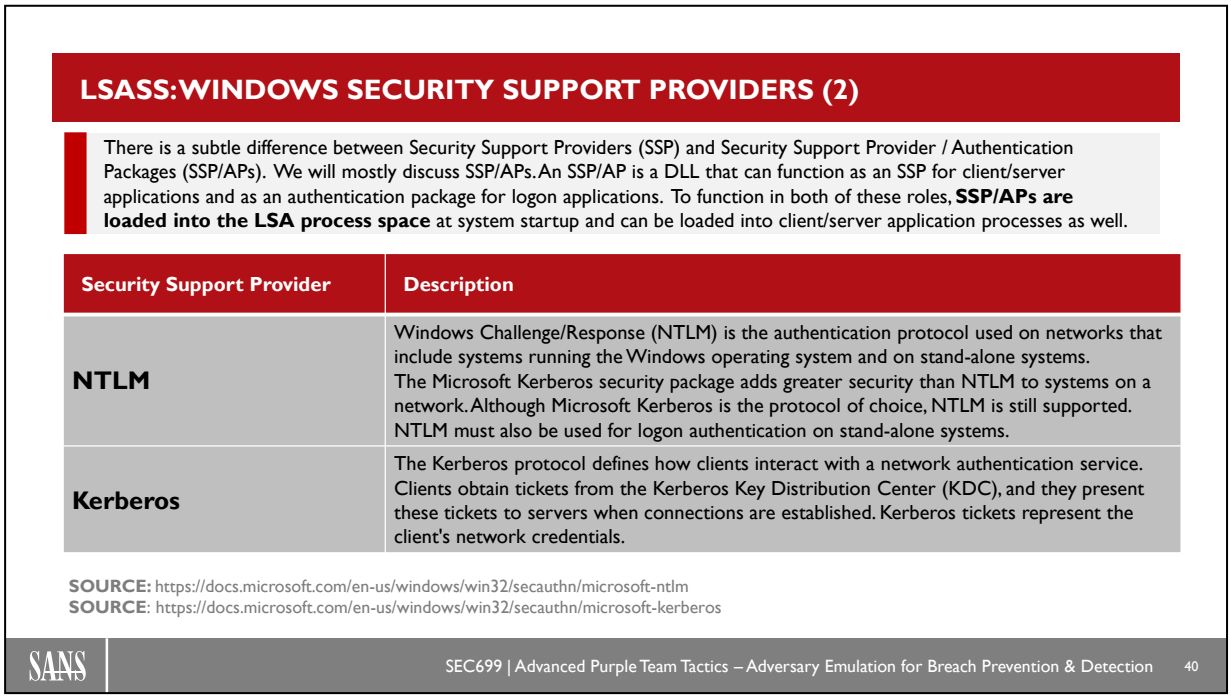
CredSSP

The Credential Security Support Provider protocol (CredSSP) lets an application delegate the user's credentials from the client to the target server for remote authentication. CredSSP provides an encrypted Transport Layer Security Protocol channel. The client is authenticated over the encrypted channel by using the Simple and Protected Negotiate (SPNEGO) protocol with either Microsoft Kerberos or Microsoft NTLM.

Negotiate

Microsoft Negotiate is a security support provider (SSP) that acts as an application layer between Security Support Provider Interface (SSPI) and the other SSPs. When an application calls into SSPI to log on to a network, it can specify an SSP to process the request. If the application specifies Negotiate, Negotiate analyzes the request and picks the best SSP to handle the request based on customer-configured security policy. Currently, the Negotiate security package selects between Kerberos and NTLM.

Full documentation for these SSPs can be found below:
https://docs.microsoft.com/en-us/windows/win32/secauthn/credential-security-support-provider
https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-negotiate



LSASS: Windows Security Support Providers (2)
Let's continue our journey through Security Support Providers with NTLM and Kerberos, which likely ring a bell:

NTLM

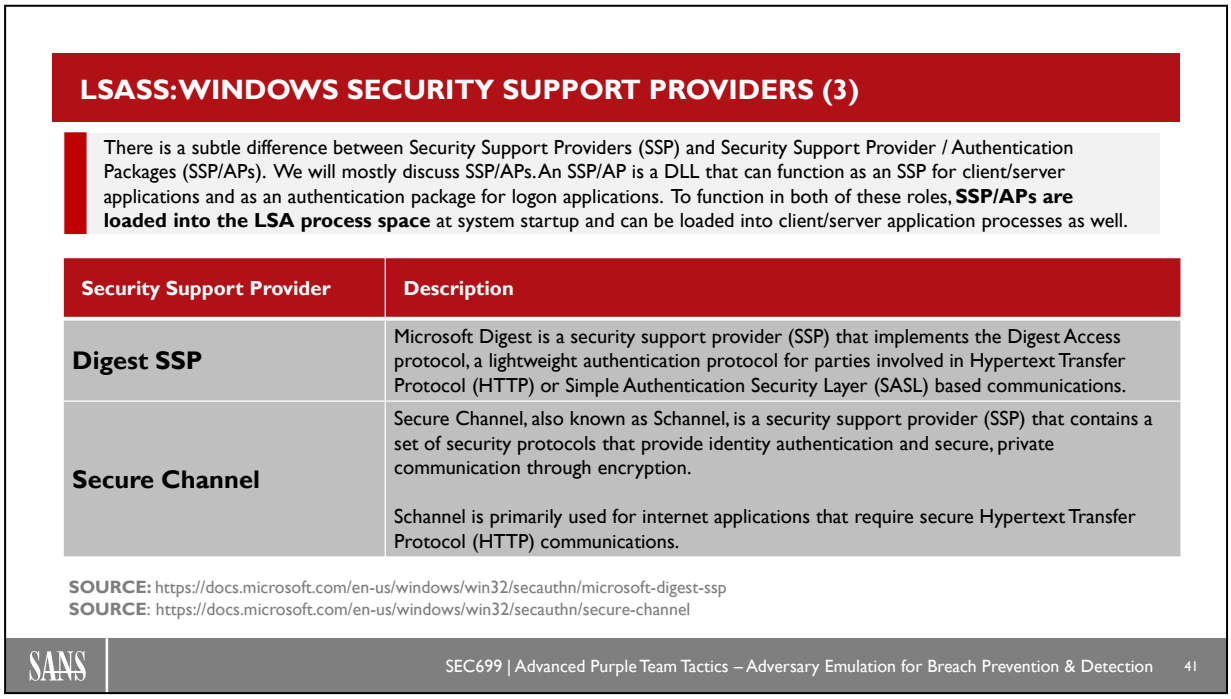
Windows Challenge/Response (NTLM) is the authentication protocol used on networks that include systems running the Windows operating system and on stand-alone systems.

The Microsoft Kerberos security package adds greater security than NTLM to systems on a network. Although Microsoft Kerberos is the protocol of choice, NTLM is still supported. NTLM must also be used for logon authentication on stand-alone systems.

Kerberos

The Kerberos protocol defines how clients interact with a network authentication service. Clients obtain tickets from the Kerberos Key Distribution Center (KDC), and they present these tickets to servers when connections are established. Kerberos tickets represent the client's network credentials.

Full documentation for these SSPs can be found below:
https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-ntlm
https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-kerberos



LSASS: Windows Security Support Providers (3)
Let's continue our journey through Security Support Providers with Digest SSP and Secure Channel. You may have heard of the infamous "Digest" authentication:

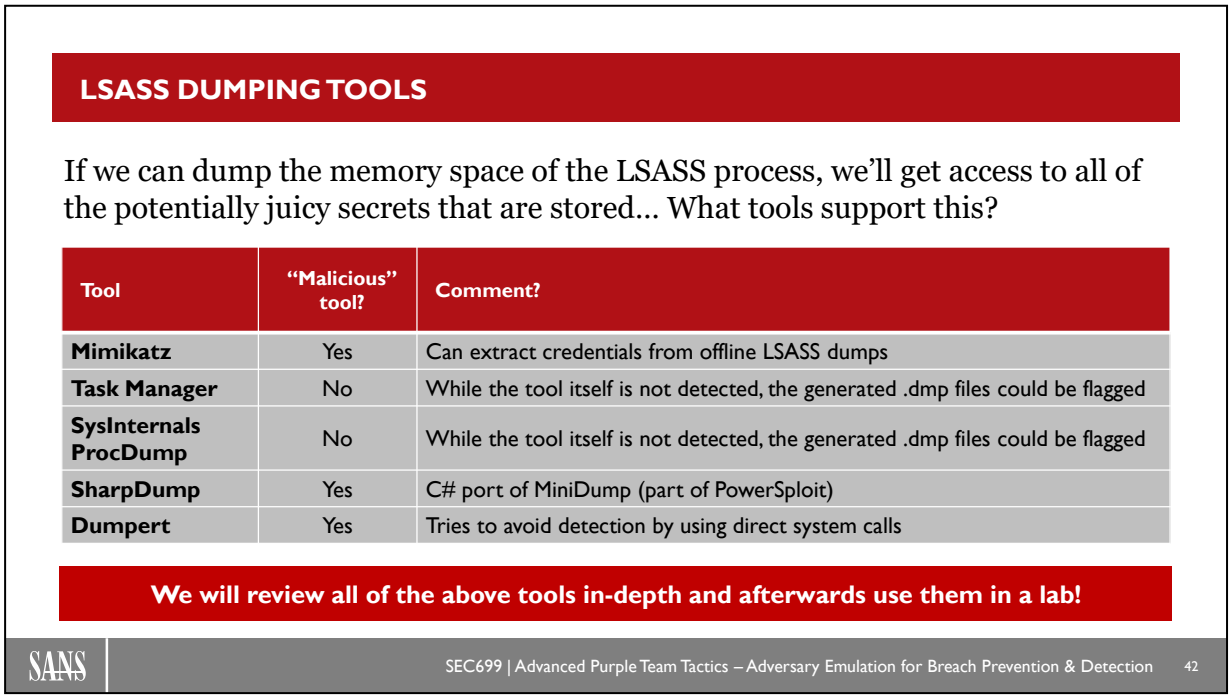
Digest

Microsoft Digest is a security support provider (SSP) that implements the Digest Access protocol, a lightweight authentication protocol for parties involved in Hypertext Transfer Protocol (HTTP) or Simple Authentication Security Layer (SASL) based communications.

Secure Channel

Secure Channel, also known as Schannel, is a security support provider (SSP) that contains a set of security protocols that provide identity authentication and secure, private communication through encryption. Schannel is primarily used for internet applications that require secure Hypertext Transfer Protocol (HTTP) communications.

Full documentation for these SSPs can be found below:
https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-digest-ssp
https://docs.microsoft.com/en-us/windows/win32/secauthn/secure-channel



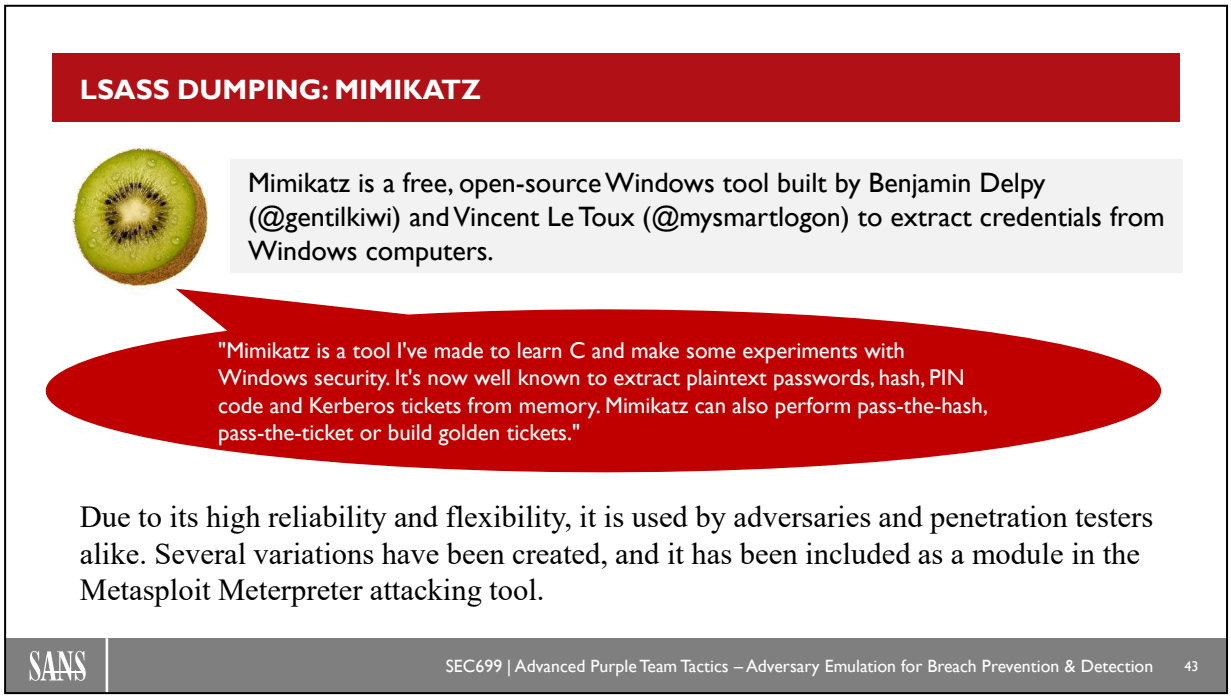
LSASS Dumping Tools
If we can dump the memory space of the LSASS process, we'll get access to all of the potentially juicy secrets that are stored…

This has been a strategy that many penetration testers and security researchers have used over the past couple of years.

What tools support this, however? The list below is by no means exhaustive, but it's a nice overview of modern techniques:

- Mimikatz
- Task Manager: The Windows Task Manager can create a "mini-dump" of the lsass.exe process
- SysInternals ProcDump: Similar to the Task Manager (although not built-in in Windows), it can create a "mini-dump" of the lsass.exe process
- SharpDump: A C# port of the infamous MiniDump tool, which is part of PowerSploit
- Dumpert: A relatively new tool developed by OutFlankNL. It tries to avoid detection by using direct system calls

We will review all of the above tools in-depth and afterwards use them in a lab!



LSASS Dumping: Mimikatz

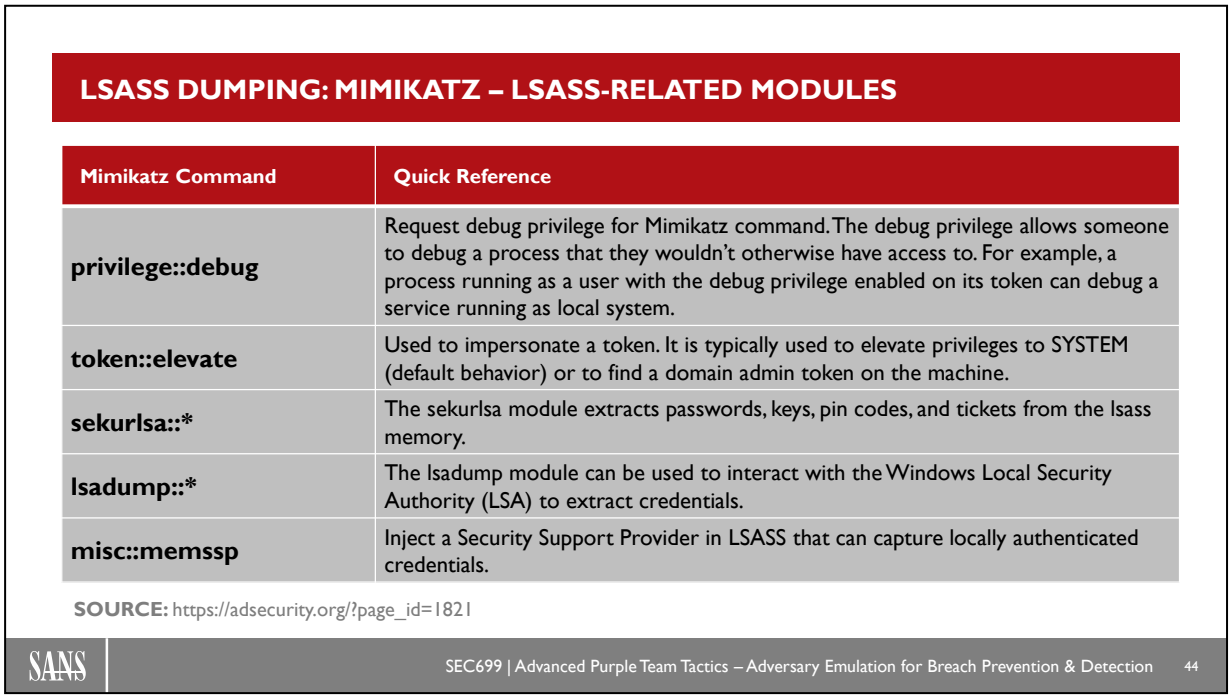
Mimikatz is a tool that has many features and functions, for example, extracting hashes from the LSA process lsass.exe. It is a free, open-source Windows tool, developed by Benjamin Delpy (@gentilkiwi) and Vincent Le Toux (@mysmartlogon). It has many features:

- Extracting hashes
- Extracting passwords
- Extracting tickets
- Executing pass-the-hash attacks
- Executing pass-the-ticket attacks
- Generating golden tickets
- …

Several of these features will be explained later. Because of all these features and constant updates with new features and support for new Windows versions, Mimikatz has become the most popular credential tool used by Red Teams and adversaries. Mimikatz has three components (in 32-bit and 64-bit versions):

1. Mimikatz.exe: This is the executable and the console that interacts with the user.
2. Mimilib.dll: This is the dll.
3. Mimidrv.sys: This is the kernel driver, necessary for features that require access or modifications to kernel data.

Because it is very powerful and open source, it has been transformed by hackers and malware authors for various purposes. A lot of antivirus programs detect the mimikatz.exe executable. Because this poses a problem to pen testers, fileless versions have been developed that launch directly into memory from various scripting platforms, like PowerShell.



LSASS Dumping: Mimikatz – LSASS-related Modules
So how does Mimikatz interact with LSASS? It's important to note that Mimikatz has a wide variety of different modules that can be used to achieve a variety of different attack strategies.

Here are some of the most popular ones, recognizing that the list is not exhaustive:

- **privilege::debug**
  Request debug privilege for Mimikatz command. The debug privilege allows someone to debug a process that they wouldn't otherwise have access to. For example, a process running as a user with the debug privilege enabled on its token can debug a service running as local system.
- **token::elevate**
  Used to impersonate a token. It is typically used to elevate privileges to SYSTEM (default behavior) or to find a domain admin token on the machine.
- **sekurlsa::***
  The sekurlsa module extracts passwords, keys, pin codes, and tickets from the lsass memory.
- **lsadump::***
  The lsadump module can be used to interact with the Windows Local Security Authority (LSA) to extract credentials.
- **misc::memssp**
  Inject a Security Support Provider in LSASS that can capture locally authenticated credentials.

For further information, an excellent blog post was written by Sean Metcalf:
https://adsecurity.org/?page_id=1821

LSASS DUMPING: MIMIKATZ – ZOOM IN ON LSADUMP

Mimikatz Command	Quick Reference
lsadump::sam	Attempts to dump credentials from the SAM (Security Account Manager). Can run both online or offline (against dumped SAM file). Does not interact with lsass.exe memory.
lsadump::lsa	Dumps credentials from the LSA (Local Security Authority). Can use the /patch and /inject flags, which interact with lsass.exe. In more recent versions of Mimikatz, the "sekurlsa" modules are typically preferred above "lsadump::lsa", among others, due to better stealth.
lsadump::cache	Dumps cached credentials from the local system.
lsadump::dcsync	Abuses MS-DRS (Directory Replication Service) to dump secrets (password hashes, Kerberos encryption keys,…) from a domain controller.

SOURCE: https://adsecurity.org/?page_id=1821
SOURCE: https://github.com/gentilkiwi/mimikatz/wiki/module-~-lsadump

LSASS Dumping: MimiKatz – Zoom in on lsadump
Let's zoom in on the lsadump module. Mimikatz' documentation can be a little daunting (and sometimes even in French ☺), so we've taken some time to document the use case of some of these commands:

- **lsadump::sam**
  Attempts to dump credentials from the SAM (Security Account Manager). This module can run both online or offline (against a dumped SAM file).
  The module does not interact with lsass.exe memory, but is limited to local credential dumping.
- **lsadump::lsa**
  Dumps credentials from the LSA (Local Security Authority). Can use the /patch and /inject flags, which interact with lsass.exe. In more recent versions of Mimikatz, the "sekurlsa" modules are typically preferred above "lsadump::lsa", among others, due to better stealth.
- **lsadump::cache**
  The module dumps cached credentials from the local system (MSCACHE format). These credentials cannot be directly reused, as they need to be brute forced.
- **lsadump::dcsync**
  Abuses MS-DRS (Directory Replication Service) to dump secrets (password hashes, Kerberos encryption keys,…) from a domain controller.

References:
https://github.com/gentilkiwi/mimikatz/wiki/module-~-lsadump
https://adsecurity.org/?page_id=1821

LSASS DUMPING: MIMIKATZ – ZOOM IN ON SEKURLSA

Mimikatz Command	Quick Reference
sekurlsa::pth	Performs the "Pass-The-Hash" attack by starting a process with a fake identity, after which the fake information (NTLM hash of a fake password) is substituted with valid information (NTLM hash we are passing).
sekurlsa::logonPasswords	Extracts all available credentials in memory, including all different Security Support Providers (Digest, MSV, tspkg,…). ***One of the most commonly used commands in Mimikatz!***
sekurlsa::tickets	Extracts all Kerberos tickets currently in memory.
sekurlsa::ekeys	Extracts all Kerberos encryption keys currently in memory.
sekurlsa::dpapi	Extracts all dpapi keys from memory for users currently logged on.
sekurlsa::minidump	Loads a minidump file that was previously dumped for offline extraction.

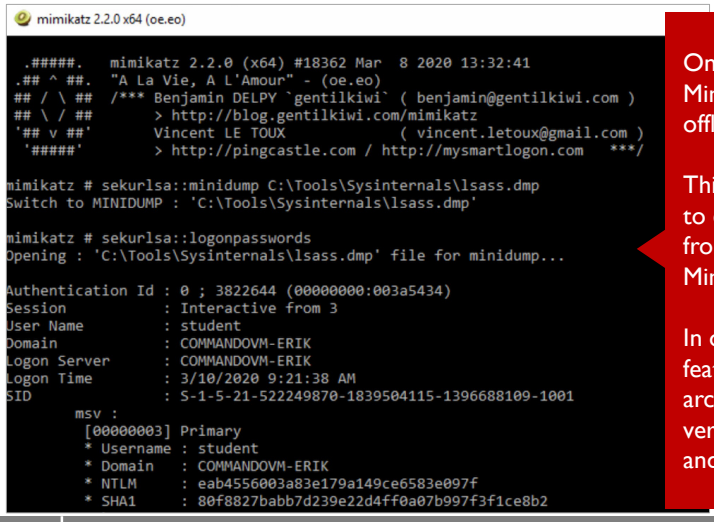
SOURCE: https://adsecurity.org/?page_id=1821
SOURCE: https://github.com/gentilkiwi/mimikatz/wiki/module-~-sekurlsa

LSASS Dumping: MimiKatz – Zoom in on Sekurlsa
The "sekurlsa" module is currently one of the most infamous modules in Mimikatz. It's very well known for its credential dumping techniques, especially "sekurlsa::logonPasswords".
Let's have a look at some of the more interesting commands:

- **sekurlsa::pth**
  Performs the "Pass-The-Hash" attack by starting a process with a fake identity, after which the fake information (NTLM hash of a fake password) is substituted with valid information (NTLM hash we are passing).
- **sekurlsa::logonPasswords**
  Extracts all available credentials in memory, including all different Security Support Providers (Digest, MSV, tspkg,…). One of the most commonly used commands in Mimikatz!
- **sekurlsa::tickets**
  Extracts all Kerberos tickets currently in memory.
- **sekurlsa::ekeys**
  Extracts all Kerberos encryption keys currently in memory.
- **sekurlsa::dpapi**
  Extracts all dpapi keys from memory for users currently logged on.
- **sekurlsa::minidump**
  Loads a minidump file that was previously dumped for offline extraction.

References:
https://github.com/gentilkiwi/mimikatz/wiki/module-~-sekurlsa
https://adsecurity.org/?page_id=1821

LSASS DUMPING: MIMIKATZ – WORKING OFFLINE



One of the interesting features in Mimikatz is the ability to work with an offline lsass memory dump.

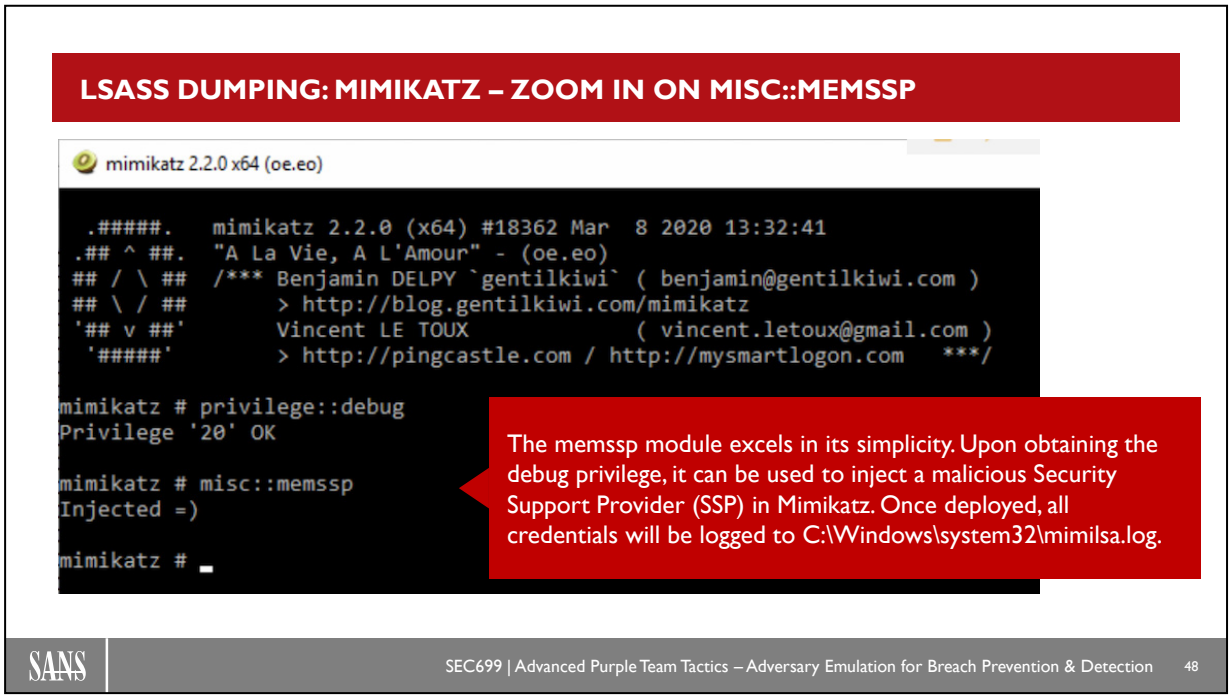
This can be highly useful when we want to emulate stealth adversaries that refrain from touching a target machine using Mimikatz.

In order to use the "sekurlsa::minidump" feature, we do need to make sure the architecture and the major Windows version of the machine running Mimikatz and the victim machine are the same.

LSASS Dumping: MimiKatz – Working Offline

In the screenshot on the slide, we can see the "sekurlsa::logonpasswords" command being used not against the live system, but against a previously created lsass memory dump. As indicated on the previous slide, Mimikatz has the ability to work with an offline lsass memory dump. This can be highly useful when we want to emulate stealth adversaries that refrain from touching a target machine using Mimikatz.

In order to use the "sekurlsa::minidump" feature, we do need to make sure the architecture and the major Windows version of the machine running Mimikatz, and the victim machine are the same. This has frustrated the course author on numerous occasions and has led to a delayed compromise in some interesting red team scenarios. ☺

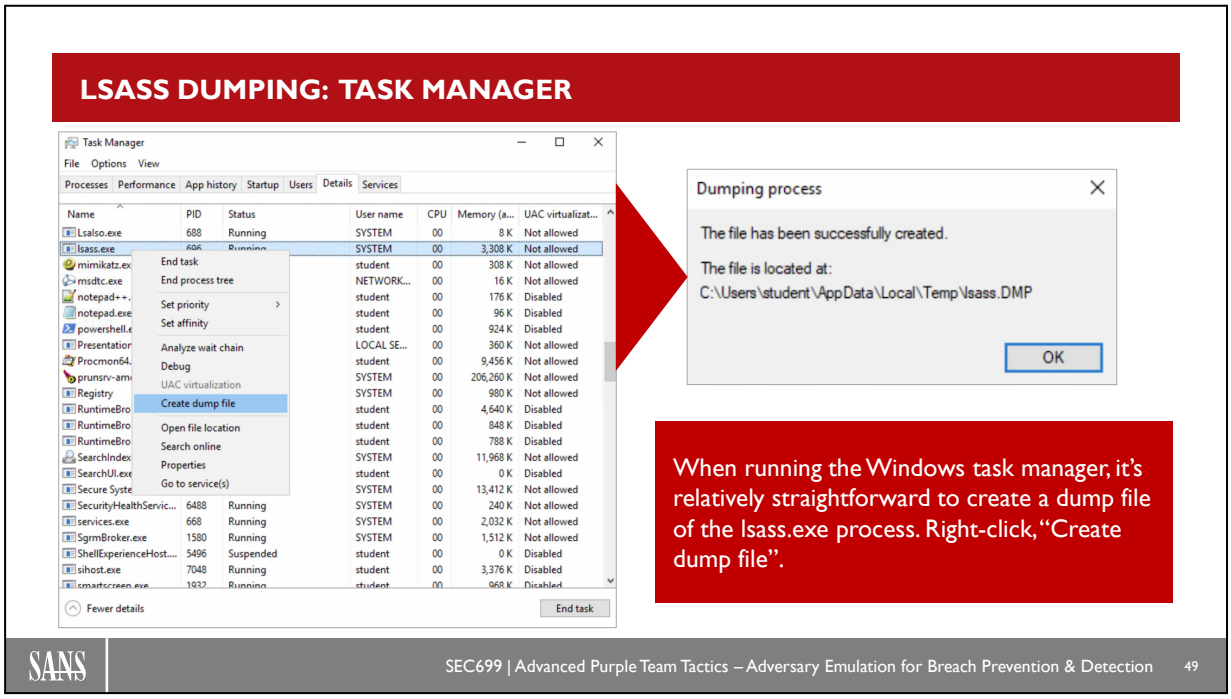


LSASS Dumping: MimiKatz – Zoom in on misc::memssp
As CredentialGuard was introduced, Benjamin Delpy made it a habit to demonstrate alternative ways of obtaining credentials using Mimikatz. One of these was an interesting trick called “memssp”.

The memssp module excels in its simplicity. Upon obtaining the debug privilege (using privilege::debug), Mimikatz can easily inject a malicious Security Support Provider (SSP) in Mimikatz (using misc::memssp).

Once deployed, all credentials will be logged to C:\Windows\system32\mimilsa.log. It’s important to note that the Security Support Provider is injected in LSASS memory and thus disappears after a reboot.

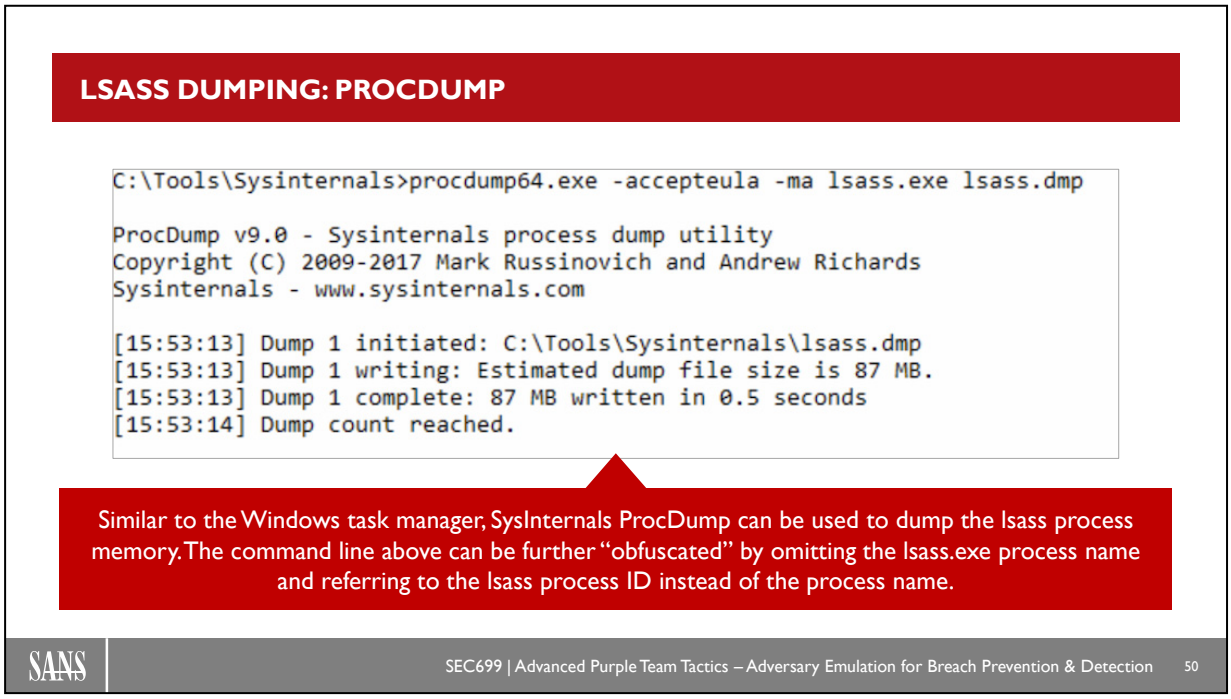
An interesting article fully dedicated to the memssp can be found here:
https://stealthbits.com/blog/stealing-credentials-with-a-security-support-provider-ssp/



LSASS Dumping: Task Manager

Let's move on to another tool that can be used to dump lsass.exe. How about our very own, built-in, task manager? Often overlooked, but when it is executed with local administrator privileges, the Windows Task Manager can easily create a memory dump of the lsass.exe process (called a mini dump file).

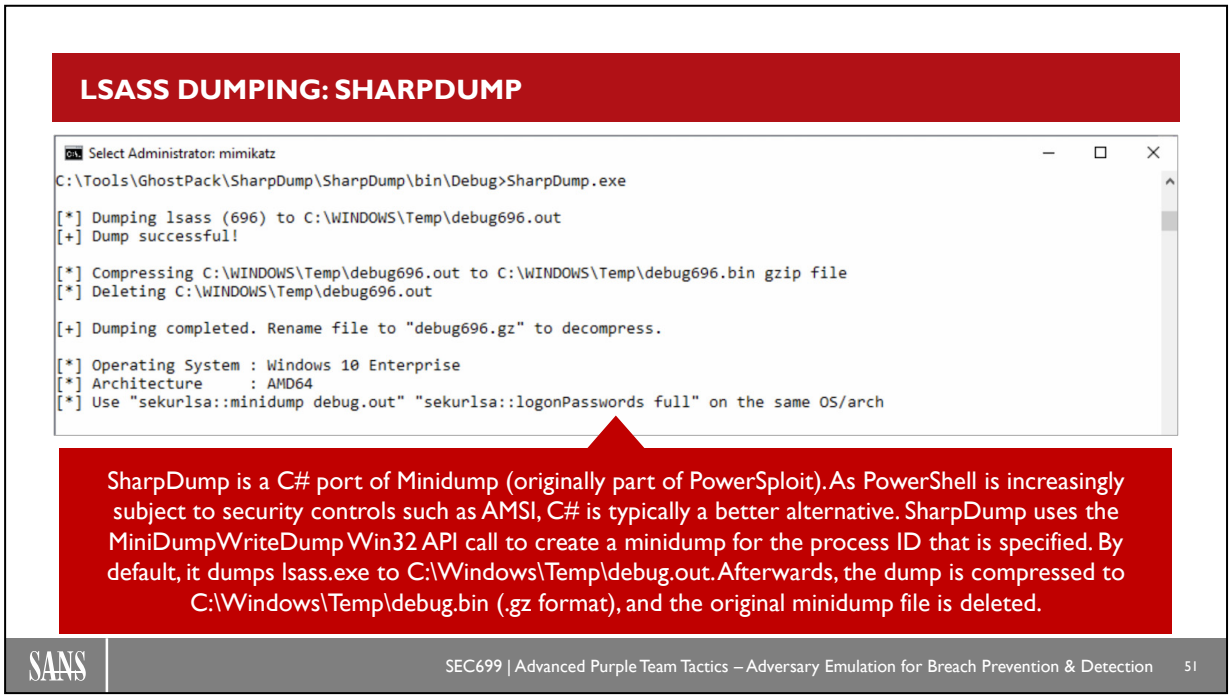
When running the Windows task manager, it's relatively straightforward to create a dump file of the lsass.exe process. Right-click, "Create dump file". This will automatically write <PROCESSNAME>.DMP to the user Temp folder.



LSASS Dumping: ProcDump
Next up, we have the similar, yet different ProcDump. Similar to the Windows task manager, SysInternals ProcDump can be used to dump the lsass process memory. The command line above can be further "obfuscated" by omitting the lsass.exe process name and referring to the lsass process ID instead of the process name.

Note that the SysInternals toolsuite is not built-in by default, but at least the tools will typically not trigger AV alerts.

Additional information can be found here:
https://docs.microsoft.com/en-us/sysinternals/downloads/procdump

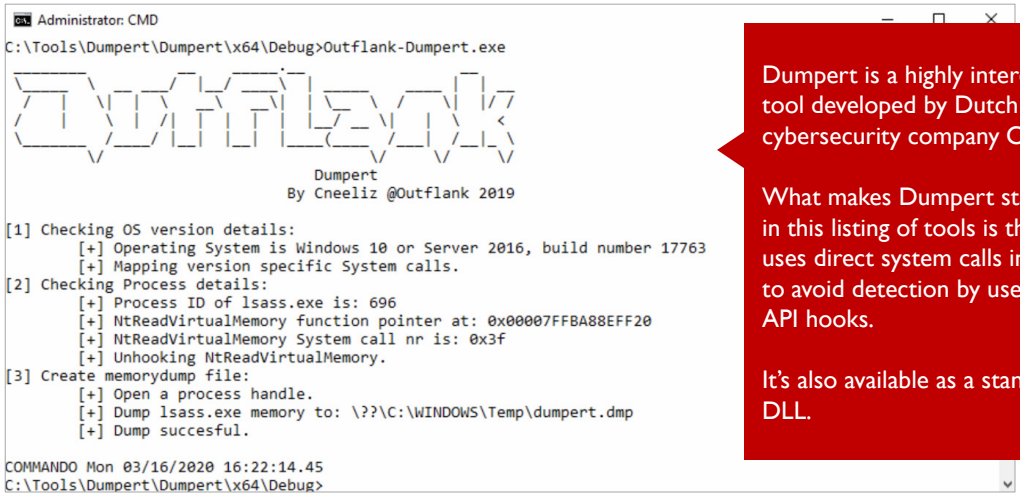


LSASS Dumping: SharpDump
Back to some of the more "malicious" credential dumping tools…

SharpDump is a C# port of Minidump (originally part of PowerSploit). As PowerShell is increasingly subject to security controls such as AMSI, C# is typically a better alternative. SharpDump uses the MiniDumpWriteDump Win32 API call to create a minidump for the process ID that is specified. By default, it dumps lsass.exe to C:\Windows\Temp\debug.out. Afterwards, the dump is compressed to C:\Windows\Temp\debug.bin (.gz format), and the original minidump file is deleted.

Additional information can be found here:
https://github.com/GhostPack/SharpDump

LSASS DUMPING: DUMPERT



Dumpert is a highly interesting tool developed by Dutch cybersecurity company OutFlank.

What makes Dumpert stand out in this listing of tools is that it uses direct system calls in order to avoid detection by user-mode API hooks.

It's also available as a stand-alone DLL.

LSASS Dumping: Dumpert

A relatively new player in this field is Dumpert. Dumpert is a highly interesting tool developed by Dutch cybersecurity company OutFlank. What makes Dumpert stand out in this listing of tools is that it uses direct system calls in order to avoid detection by user-mode API hooks. We discussed this evasion technique in-depth yesterday, so it's nice to see an example of an actual tool implementing it.

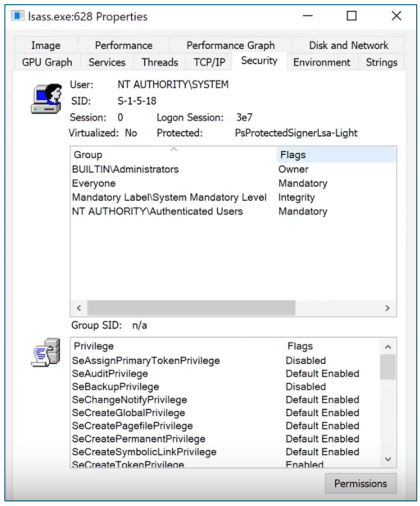
Dumpert is open-source and can thus be easily compiled. Both a .EXE and a .DLL version are available!

Additional information can be found here:
https://github.com/outflanknl/Dumpert

PREVENTING LSASS DUMPING: PROTECTED PROCESSES (1)

In order to prevent hash dumping attacks aimed at the LSA process, Microsoft introduced "Protected Processes" as of Windows 8 and Windows Server 2012.

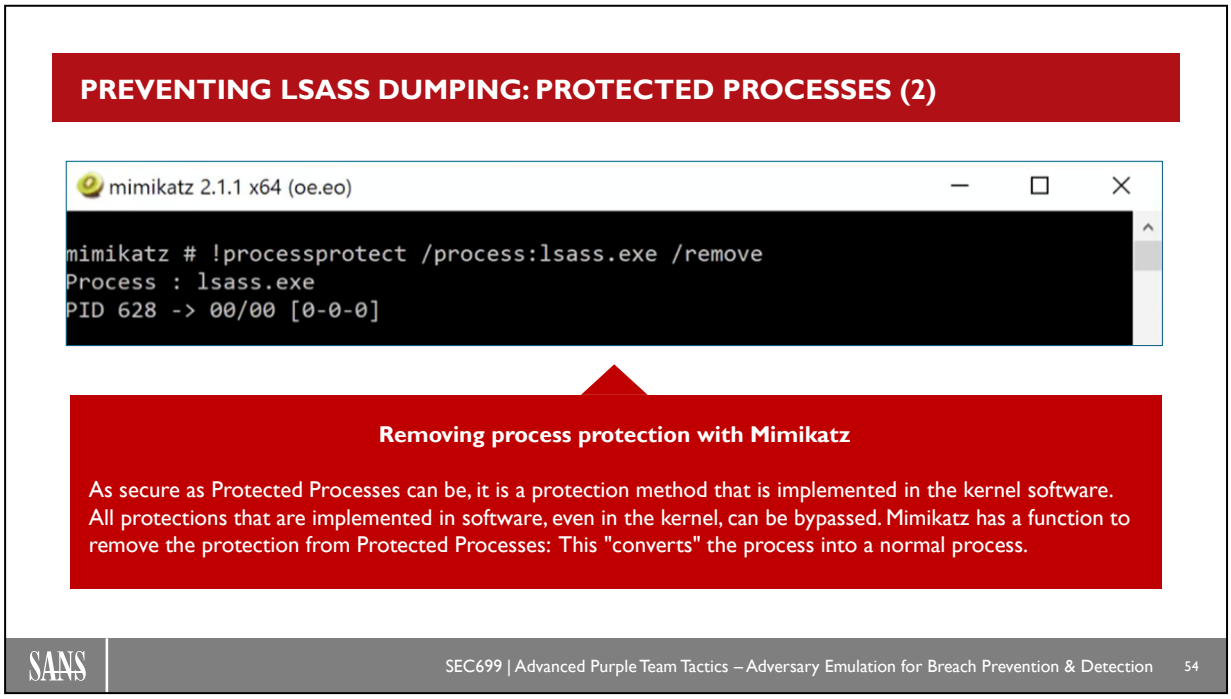
- Protected Processes were first introduced in Windows Vista for DRM (Digital Rights Management) purposes, but were adapted for "security purposes" in Windows 8
- The screenshot on the right provides an example of the lsass.exe process running as a "protected process"
- Protected Processes are implemented in the Kernel software and can thus be defeated…



Preventing LSASS Dumping: Protected Processes (1)
With Windows Server 2012 (and Windows 8), the LSA can be configured to have its lsass.exe process hosted as a protected process. Under Windows, with the correct privileges, accounts can access the memory of any Windows process. This is not the case with Protected Processes. Protected Processes (and their memory) cannot be accessed by other processes, regardless of the account they run with. Protected Processes were introduced with Windows Vista for DRM purposes (to protected media players), but Protected Processes were repurposed for security when Windows 8 was introduced.

By running the lsass.exe process as a protected process, tools like Mimikatz cannot access the process to extract credentials. In the screenshot above, we can see that the lsass.exe process running on this machine is protected: Process Explorer's security tab indicates that the process is protected (PsProtectedSignerLsa-Light).

Protected Processes are implemented in the Kernel software and can thus be defeated… Mimikatz has a function to remove the protection from Protected Processes: This "converts" the process into a normal process. To do this, Mimikatz requires its kernel driver to be installed. Installation of this kernel driver can, however, be detected and responded to.

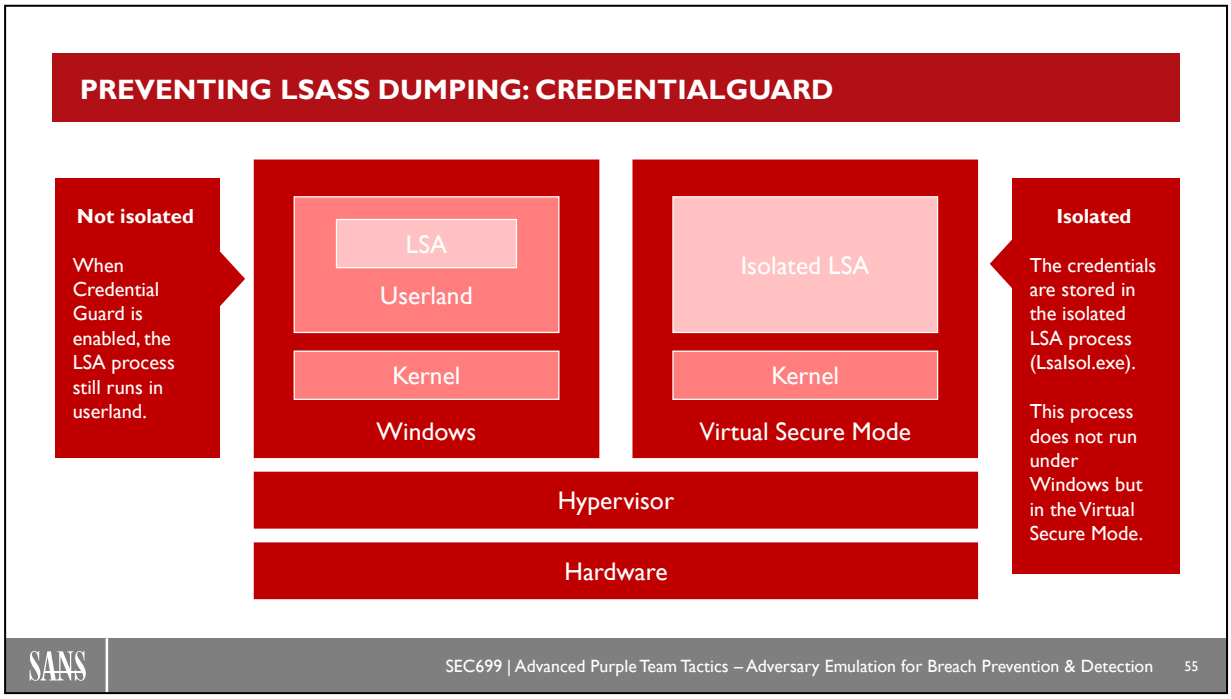


Preventing LSASS Dumping: Protected Processes (2)
As secure as Protected Processes can be, it is a protection method that is implemented in the kernel software. All protections that are implemented in software, even in the kernel, can be bypassed. Mimikatz has a function to remove the protection from Protected Processes: This "converts" the process into a normal process. To do this, Mimikatz requires its kernel driver to be installed.

This can be done with the command **!+**.

The next step is to use the kernel command processprotect to remove the protection of the lsass.exe process: **!processprotect /process:lsass.exe /remove**.

When this is done, Mimikatz (or other tools) can be used to extract the credentials, as the memory of the LSA process is no longer protected.

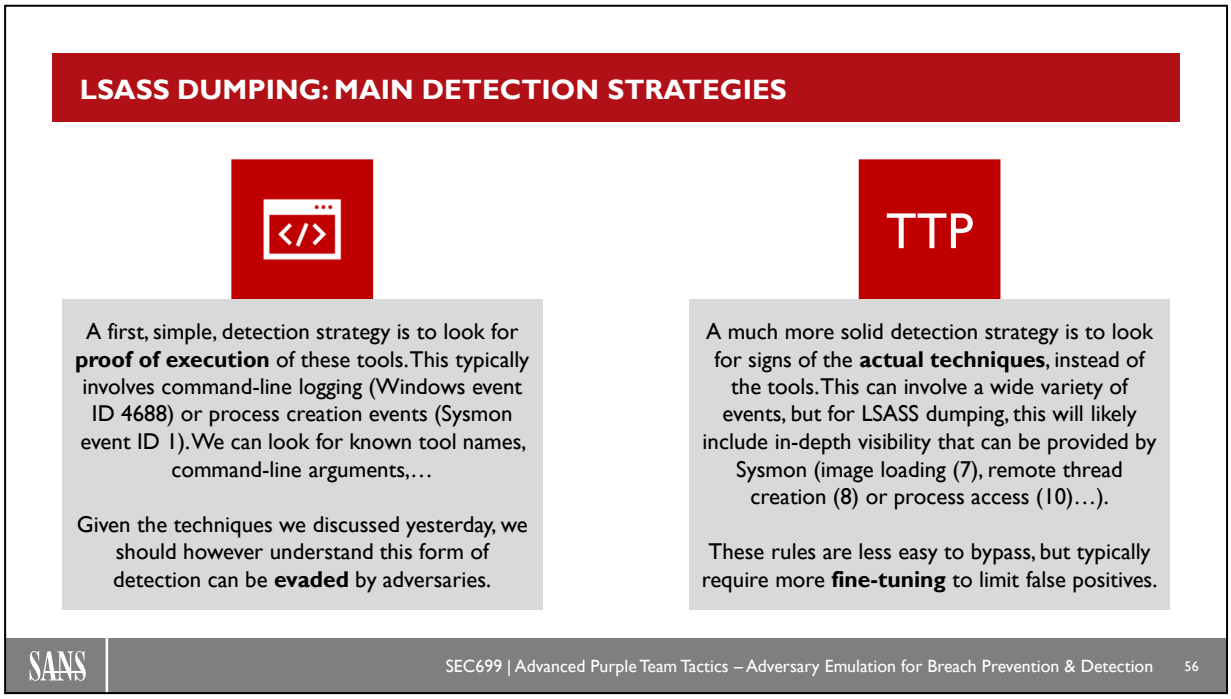


Preventing LSASS Dumping: CredentialGuard
Credential Guard was introduced with Windows Server 2016 and Windows 10 enterprise editions. It requires modern CPUs that provide virtualization functionality.

When Credential Guard is enabled, Windows still runs on top of the hypervisor and the hardware, and the LSA process still runs in userland. The difference, however, is that the credentials are no longer stored inside this LSA process (lsass.exe).

With Credential Guard, the credentials are stored in the isolated LSA process (LsaIso.exe). This process does not run under Windows but in the Virtual Secure Mode. This is a separate, virtualized environment that is separated from the other environments (like Windows) via hardware.

It is impossible for processes in the Windows environment to access processes in the Virtual Secure Mode environment, even by manipulating kernel data structures. All operations that require credentials, like checking NTML hashes, are done by the isolated LSA upon request of the LSA. The credentials never leave the isolated LSA.



LSASS Dumping: Main Detection Strategies
In order to detect LSASS dumping, there's two main detection strategies we can leverage:

Detect execution of the tools

A first, simple, detection strategy is to look for proof of execution of these tools. This typically involves command-line logging (Windows event ID 4688) or process creation events (Sysmon event ID 1). We can look for known tool names, command-line arguments,… Given the techniques we discussed yesterday, we should however understand this form of detection can be evaded by adversaries.

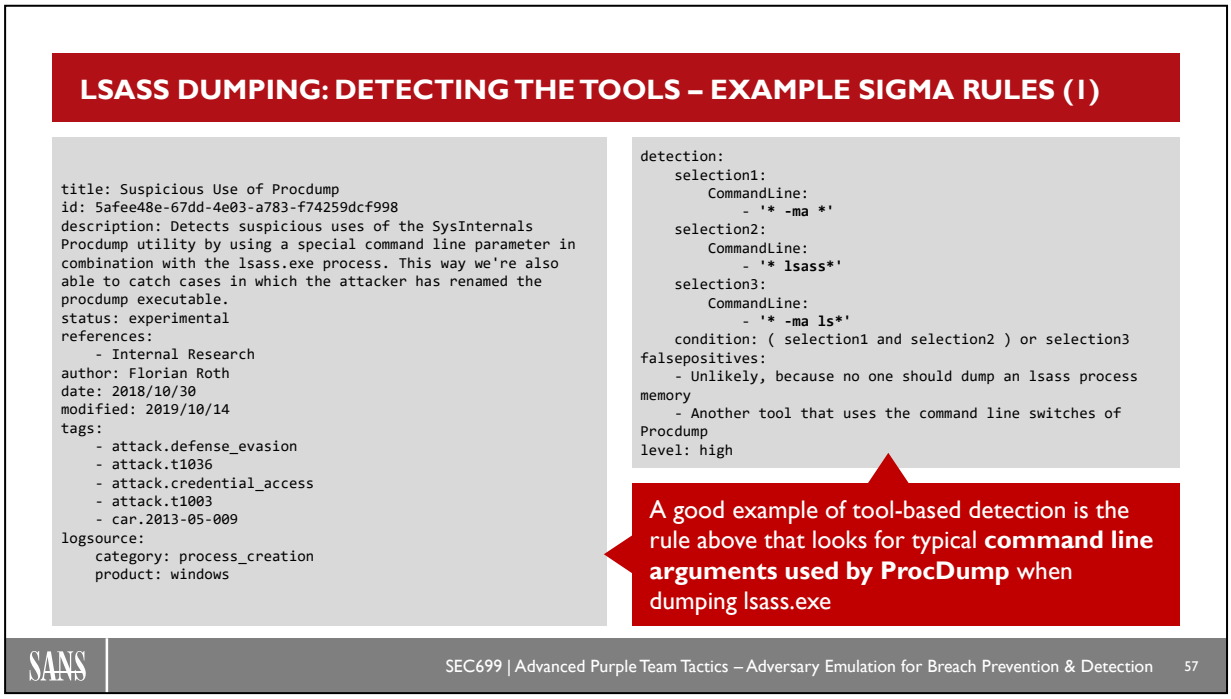
Detect execution of the techniques

A much more solid detection strategy is to look for signs of the actual techniques, instead of the tools. This can involve a wide variety of events, but for LSASS dumping, this will likely include in-depth visibility that can be provided by Sysmon (image loading (7), remote thread creation (8) or process access (10)…). These rules are less easy to bypass, but typically require more fine-tuning to limit false positives.

An excellent presentation that covers both strategies was presented at ZeroNights in 2017 by Teymur Kheirhabarov:

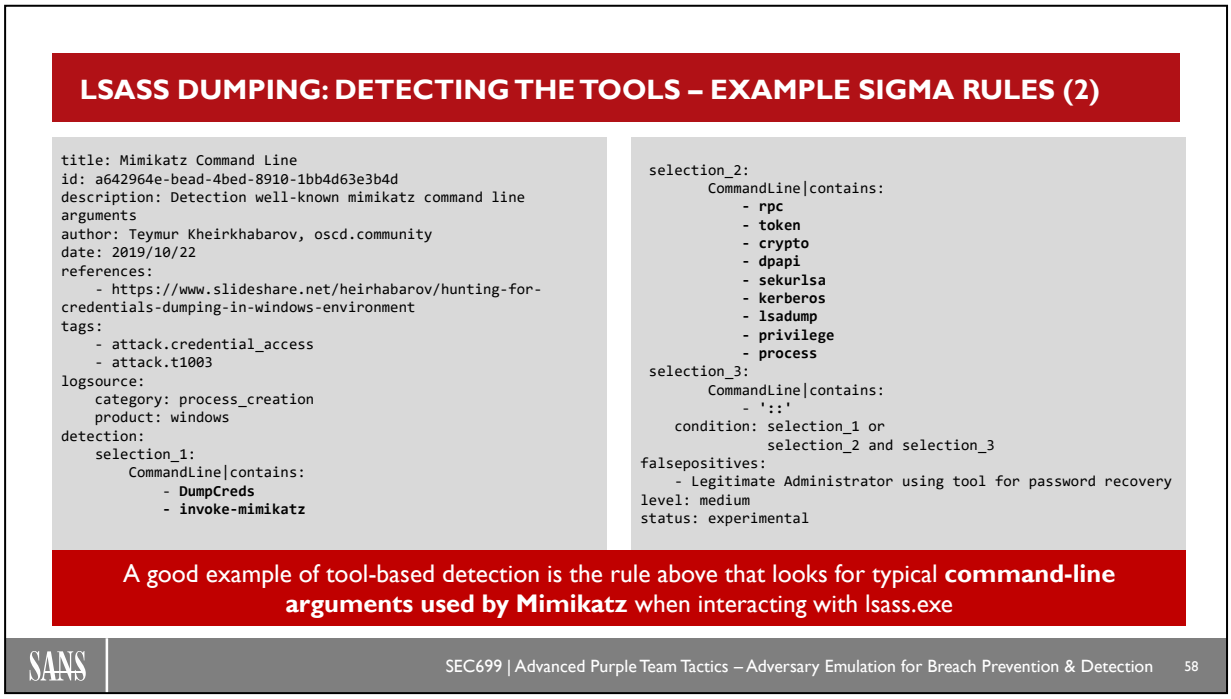
https://2017.zeronights.org/wp-content/uploads/materials/ZN17_Kheirkhabarov_Hunting_for_Credentials_Dumping_in_Windows_Environment.pdf

We will review these two strategies in this section of the course!



LSASS Dumping: Detecting the Tools – Example SIGMA Rules (1)
A good example of tool-based detection is the rule above that looks for typical command-line arguments used by ProcDump when dumping lsass.exe. It leverages Sysmon Event ID 1, ProcessCreate!

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma



LSASS Dumping: Detecting the Tools – Example SIGMA Rules (2)
A good example of tool-based detection is the rule above that looks for typical command-line arguments used by Mimikatz when interacting with lsass.exe. It leverages Sysmon Event ID 1, ProcessCreate!

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma

LSASS DUMPING: DETECTING THE TOOLS

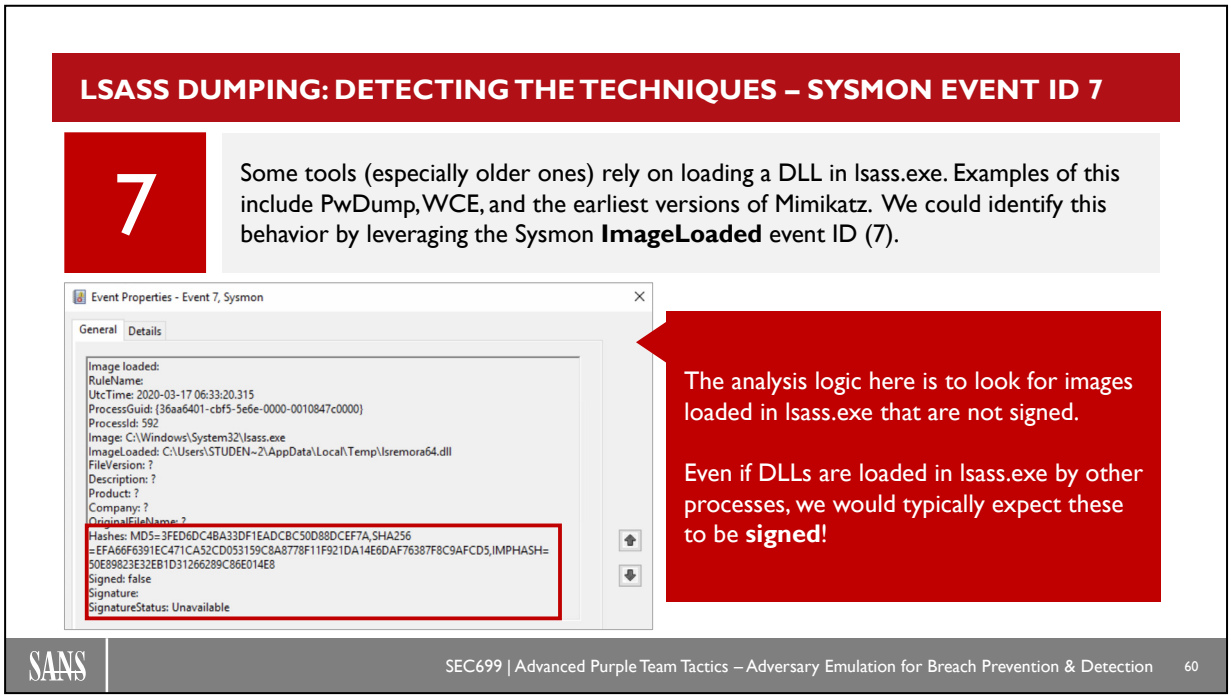
```
action: global
title: Dumpert Process Dumper
id: 2704ab9e-afe2-4854-a3b1-0c0706d03578
description: Detects the use of Dumpert process dumper, which
dumps the lsass.exe process memory
author: Florian Roth
references:
    - https://github.com/outflanknl/Dumpert
    - https://unit42.paloaltonetworks.com/actors-still-
exploiting-sharepoint-vulnerability/
date: 2020/02/04
tags:
    - attack.credential_access
    - attack.t1003
logsource:
    product: windows
    service: sysmon
falsepositives:
    - Very unlikely
level: critical
```

```
---
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Imphash: '09D278F9DE118EF09163C6140255C690'
    condition: selection
---
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 11
        TargetFilename: C:\Windows\Temp\dumpert.dmp
    condition: selection
```

A good example of tool-based detection is the rule above that looks at the Dumpert **imphash** or the default Dumpert dump file (**C:\Windows\Temp\dumpert.dmp**) being created.

LSASS Dumping: Detecting the Tools
A good example of tool-based detection is the rule above that looks at the Dumpert imphash or the default Dumpert dump file (C:\Windows\Temp\dumpert.dmp) being created. It leverages Sysmon Event ID 11, FileCreate!

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma

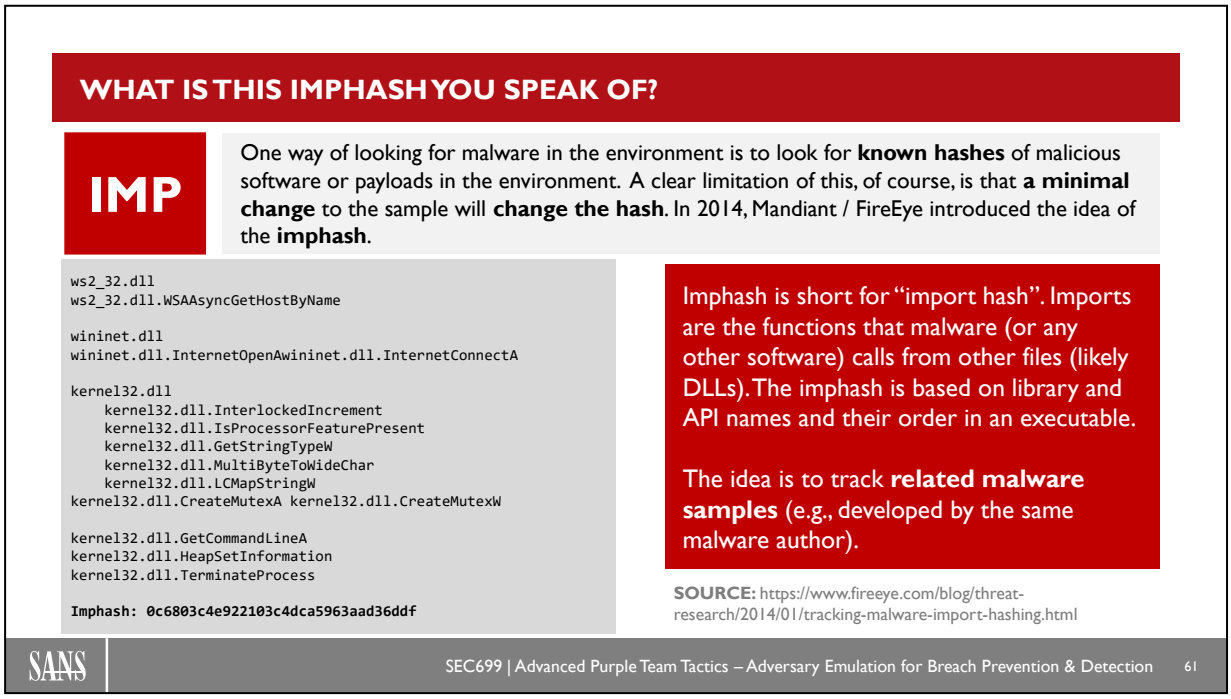


LSASS Dumping: Detecting the Techniques – Sysmon Event ID 7
Some tools (especially older ones) rely on loading a DLL in lsass.exe. Examples of this include PwDump, WCE, and the earliest versions of Mimikatz. We could identify this behavior by leveraging the Sysmon ImageLoaded event ID (7). The analysis logic here is to look for images loaded in lsass.exe that are not signed. Even if DLLs are loaded in lsass.exe by other processes, we would typically expect these to be signed!

In the screenshot on the slide, we can see that the following DLL was loaded:

C:\Users\student_ladm\AppData\Local\Temp\lsremora64.dll

We can see that it's obviously not signed, while we also get the hashes of the DLL that was injected.



What Is This Imphash You Speak Of?
One way of looking for malware in the environment is to look for known hashes of malicious software or payloads in the environment. A clear limitation of this, of course, is that a minimal change to the sample will change the hash. In 2014, Mandiant / FireEye introduced the idea of the imphash. Imphash is short for "import hash". Imports are the functions that malware (or any other software) calls from other files (likely DLLs). The imphash is based on library and API names and their order in an executable. The idea is to track related malware samples (e.g., developed by the same malware author).

The full concept is explained by FireEye on their website:
https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html

On their web page, they use the following code as an example:

```
#include
#include
#include
#include
#pragma comment(lib, "ws2_32.lib")
#pragma comment(lib, "wininet.lib")
int makeMutexA()
{
```

```
CreateMutexA(NULL, FALSE, "TestMutex");
    return 0;
}
int makeMutexW()
{
    CreateMutexW(NULL, FALSE, L"TestMutex");
    return 0;
}
int makeUserAgent()
{
    HANDLE hInet=0, hConn=0;
    char buf[sizeof(struct hostent)] = {0};
    hInet = InternetOpenA("User-Agent: (Windows; 5.1)",
INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
    hConn = InternetConnectA(hInet, "www.google.com", 443, NULL, NULL,
INTERNET_SERVICE_HTTP, 0, 0);
    WSAAsyncGetHostByName(NULL, 3, "www.yahoo.com", buf, sizeof(struct hostent));
    return 0;
}

int main(int argc, char *argv[])
{
    makeMutexA();
    makeMutexW();
    makeUserAgent();
    return 0;
}
```

When compiling this simple piece of code, the import table and hash looks as follows:

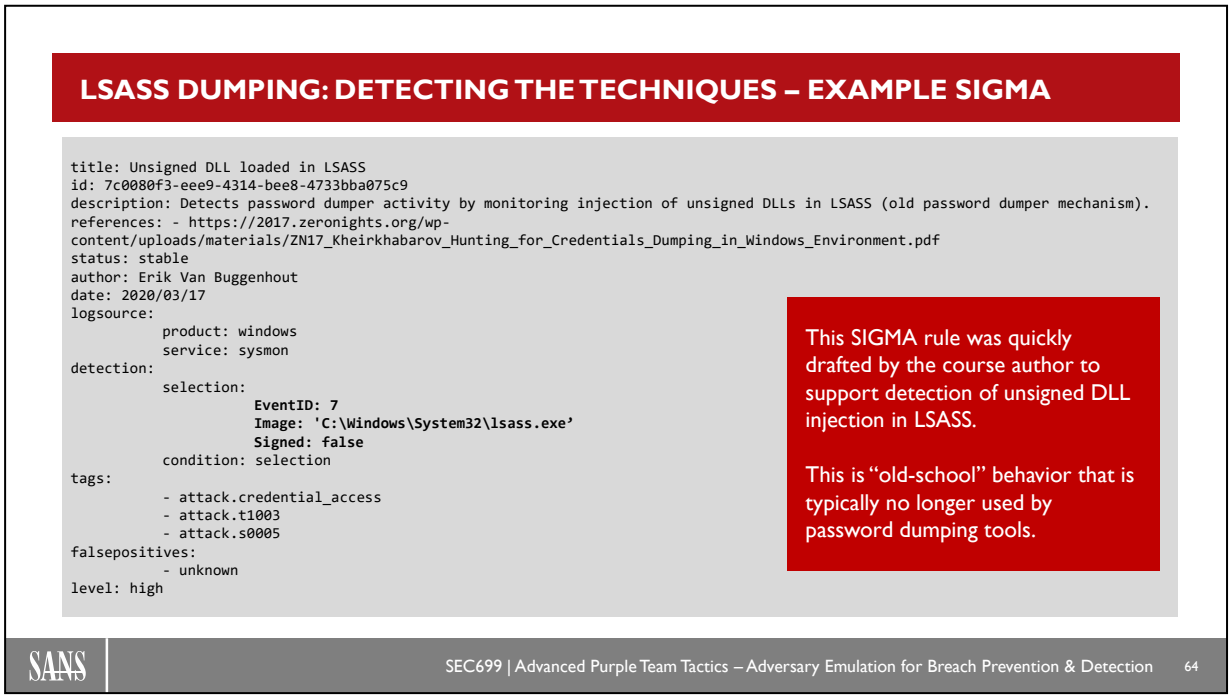
```
ws2_32.dll
ws2_32.dll.WSAAsyncGetHostByName

wininet.dll
wininet.dll.InternetOpenAwininet.dll.InternetConnectA

kernel32.dll
    kernel32.dll.InterlockedIncrement
    kernel32.dll.IsProcessorFeaturePresent
    kernel32.dll.GetStringTypeW
    kernel32.dll.MultiByteToWideChar
    kernel32.dll.LCMapStringW
kernel32.dll.CreateMutexA kernel32.dll.CreateMutexW

kernel32.dll.GetCommandLineA
kernel32.dll.HeapSetInformation
kernel32.dll.TerminateProcess
```

Imphash: 0c6803c4e922103c4dca5963aad36ddf

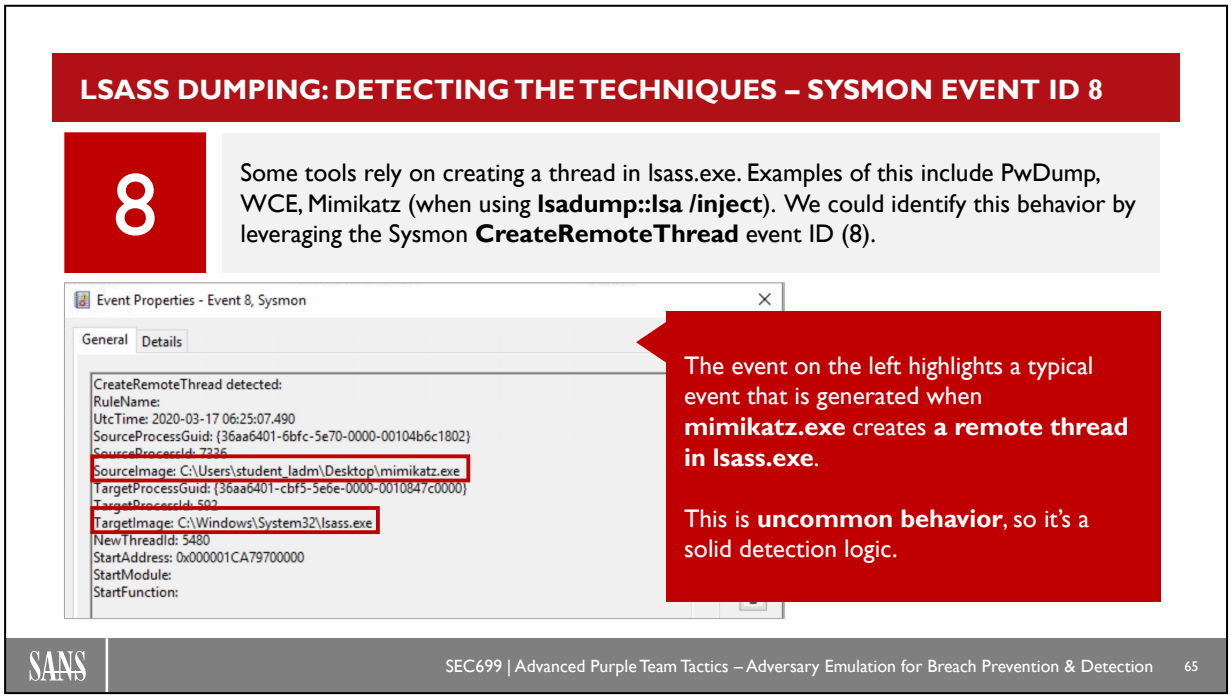


LSASS Dumping: Detecting the Techniques – Example SIGMA

This SIGMA rule was quickly drafted by the course author to support detection of unsigned DLL injection in LSASS. In the rule, we use the following detection logic:

- Event ID: 7 (Sysmon) Image Loaded)
- Image: C:\Windows\system32\lsass.exe
- Signed: false

This is "old-school" behavior that is typically no longer used by modern password dumping tools. It's still worth keeping an eye on this, as DLL injection in LSASS is always of interest.

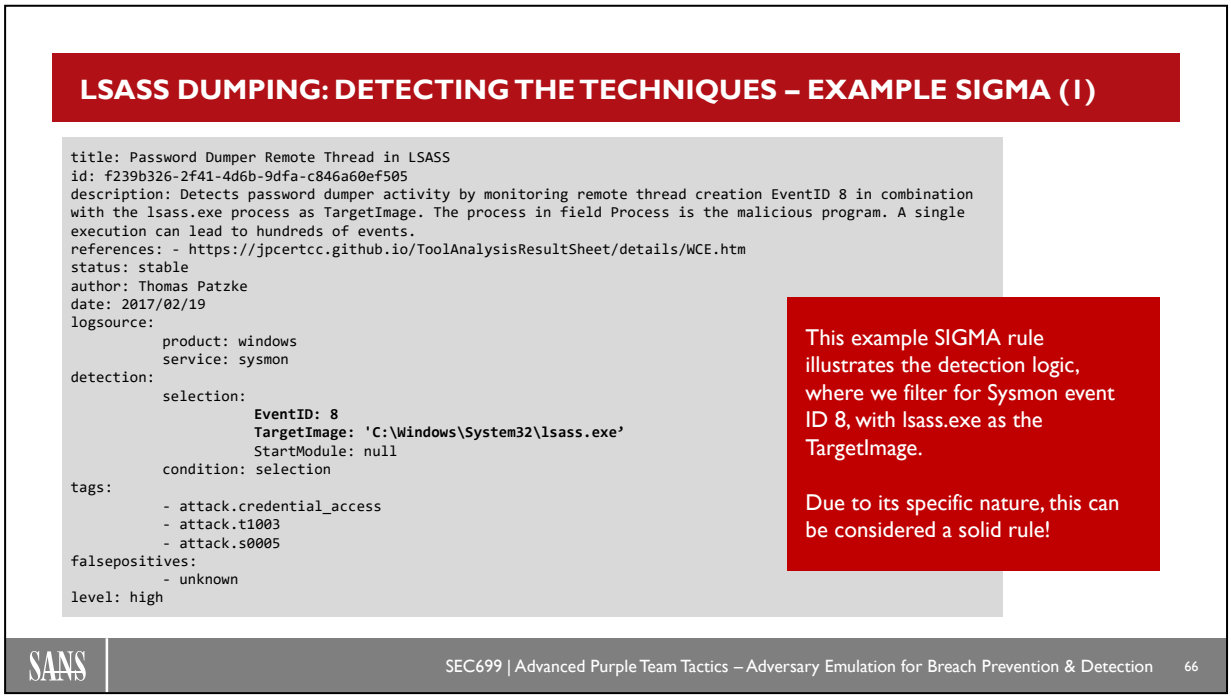


LSASS Dumping: Detecting the Techniques – Sysmon Event ID 8
The previous detection logic was based on the loading of images in LSASS (using event ID 7); however, there are other ways of interacting with lsass.exe. One of these methods is to create a thread in lsass.exe, to allow code execution. Examples of tools that exhibit this behavior include PwDump, WCE, and Mimikatz (when using lsadump::lsa /inject). We could identify this behavior by leveraging the Sysmon CreateRemoteThread event ID (8).

The event on the screenshot highlights a typical event that is generated when mimikatz.exe creates a remote thread in lsass.exe:

- The source image is mimikatz.exe
- The target image is lsass.exe

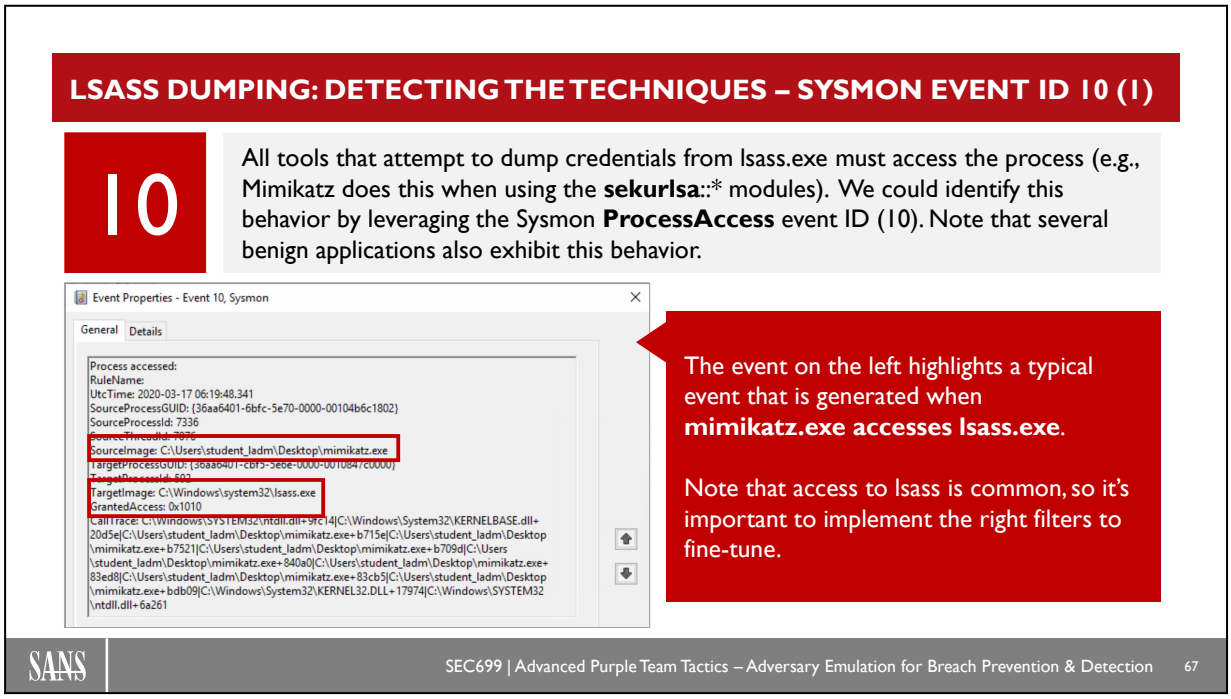
This is uncommon behavior, so it's a solid detection logic.



LSASS Dumping: Detecting the Techniques – Example SIGMA (1)
This example SIGMA rule illustrates the detection logic, where we filter for Sysmon event ID 8, with lsass.exe as the TargetImage. Due to its specific nature, this can be considered a solid rule!

It leverages Sysmon Event ID 8, CreateRemoteThread.

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma



LSASS Dumping: Detecting the Techniques – Sysmon Event ID 10 (1)
All tools that attempt to dump credentials from lsass.exe must access the process (e.g., Mimikatz does this when using the sekurlsa::* modules). We could identify this behavior by leveraging the Sysmon ProcessAccess event ID (10). The event on the slide highlights a typical event that is generated when mimikatz.exe accesses lsass.exe:

- The source image mimikatz.exe
- The target image is lsass.exe
- The grantedaccess field is 0x1010 (more on this later)

Note that access to lsass (and to other processes for that matter) is common, so it's important to implement the right filters to fine-tune.

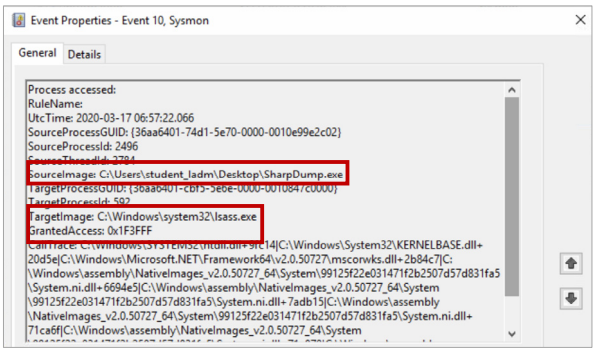
So, what does ProcessAccess mean? From Microsoft's documentation:

"The process accessed event reports when a process opens another process, an operation that's often followed by information queries or reading and writing the address space of the target process." (source: https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon).

LSASS DUMPING: DETECTING THE TECHNIQUES – SYSMON EVENT ID 10 (2)

10

All tools that attempt to dump credentials from lsass.exe must access the process (e.g., Mimikatz does this when using the **sekurlsa**::* modules). We could identify this behavior by leveraging the Sysmon **ProcessAccess** event ID (10). Note that several benign applications also exhibit this behavior.



The event on the left highlights a typical event that is generated when **SharpDump.exe accesses lsass.exe**.

Note the different access mask that is used by SharpDump (0x1F3FFF instead of 0x1010).

LSASS Dumping: Detecting the Techniques – Sysmon Event ID 10 (2)

The event on the slide highlights a typical event that is generated when SharpDump.exe accesses lsass.exe:

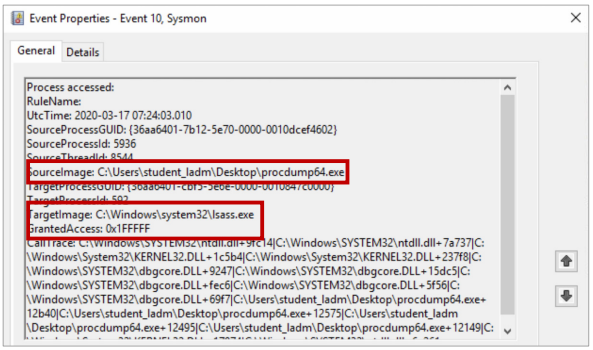
- The source image mimikatz.exe
- The target image is lsass.exe
- The grantedaccess field is 0x1F3FFF (more on this later)

Note the different access mask that is used by SharpDump (0x1F3FFF instead of 0x1010).

LSASS DUMPING: DETECTING THE TECHNIQUES – SYSMON EVENT ID 10 (3)

10

All tools that attempt to dump credentials from lsass.exe must access the process (e.g., Mimikatz does this when using the **sekurlsa**::* modules). We could identify this behavior by leveraging the Sysmon **ProcessAccess** event ID (10). Note that several benign applications also exhibit this behavior.



The event on the left highlights a typical event that is generated when **ProcDump accesses lsass.exe**.

Note the different access mask that is used (in this case, 0X1FFFFF).

LSASS Dumping: Detecting the Techniques – Sysmon Event ID 10 (3)

The event on the slide highlights a typical event that is generated when ProcDump.exe accesses lsass.exe:

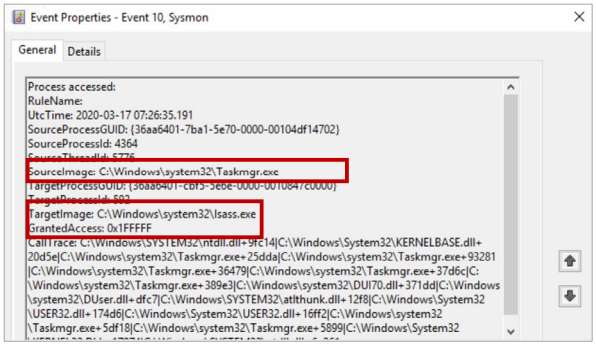
- The source image mimikatz.exe
- The target image is lsass.exe
- The grantedaccess field is 0x1FFFFF (more on this later)

Note the different access mask that is used by ProcDump (0x1FFFFF).

LSASS DUMPING: DETECTING THE TECHNIQUES – SYSMON EVENT ID 10 (4)

10

All tools that attempt to dump credentials from lsass.exe must access the process (e.g., Mimikatz does this when using the **sekurlsa**::* modules). We could identify this behavior by leveraging the Sysmon **ProcessAccess** event ID (10). Note that several benign applications also exhibit this behavior.



The event on the left highlights a typical event that is generated when **Task Manager accesses lsass.exe**.

Note that Task Manager uses the same access mask (0X1FFFFF) as ProcDump.

LSASS Dumping: Detecting the Techniques – Sysmon Event ID 10 (4)

The event on the slide highlights a typical event that is generated when Task Manager accesses lsass.exe:

- The source image mimikatz.exe
- The target image is lsass.exe
- The grantedaccess field is 0x1FFFFF (more on this later)

Note the access mask that is used is the same as the one used by ProcDump.

LSASS DUMPING: DETECTING THE TECHNIQUES – EXAMPLE SIGMA (2)

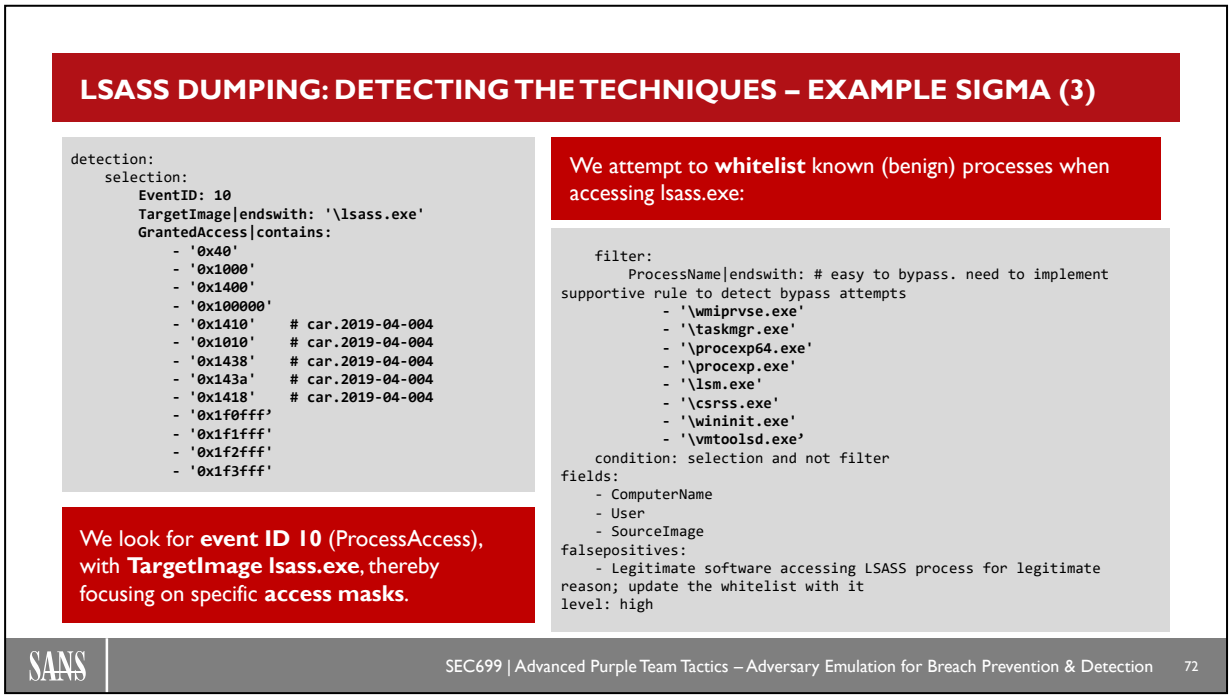
```
title: Credentials Dumping Tools Accessing LSASS Memory
id: 32d0d3e2-e58d-4d41-926b-18b520b2b32d
status: experimental
description: Detects process access LSASS memory which is typical for credentials dumping tools
author: Florian Roth, Roberto Rodriguez, Dimitrios Slamaris, Mark Russinovich, Thomas Patzke, Teymur Kheirkhabarov, Sherif Eldeeb,
James Dickenson, Aleksey Potapov,
    oscd.community (update)
date: 2017/02/16
modified: 2019/11/08
references:
    - https://onedrive.live.com/view.aspx?resid=D026B4699190F1E6!2843&ithint=file%2cpptx&app=PowerPoint&authkey=!AMvCRTKB_V1J5ow
    - https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for_22.html
    - https://www.slideshare.net/heirhabarov/hunting-for-credentials-dumping-in-windows-environment
    - http://security-research.dyndns.org/pub/slides/FIRST2017/FIRST-2017_Tom-Ueltschi_Sysmon_FINAL_notes.pdf
tags:
    - attack.t1003
    - attack.s0002
    - attack.credential_access
    - car.2019-04-004
logsource:
    product: windows
    service: sysmon
```

The above rule is an example SIGMA rule that highlights different ways of detecting process access in lsass.exe. We will walk through the rule step-by-step in the next slides.

LSASS Dumping: Detecting the Techniques – Example SIGMA (2)
The above rule is an example SIGMA rule that highlights different ways of detecting process access (event ID 10) in lsass.exe. We will walk through the rule step-by-step in the next slides.

The rule has gone through different iterations and is maintained by a series of community researchers (Florian Roth, Roberto Rodriguez, Dimitrios Slamaris, Mark Russinovich, Thomas Patzke, Teymur Kheirkhabarov, Sherif Eldeeb, James Dickenson, Aleksey Potapov, oscd.community).

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma



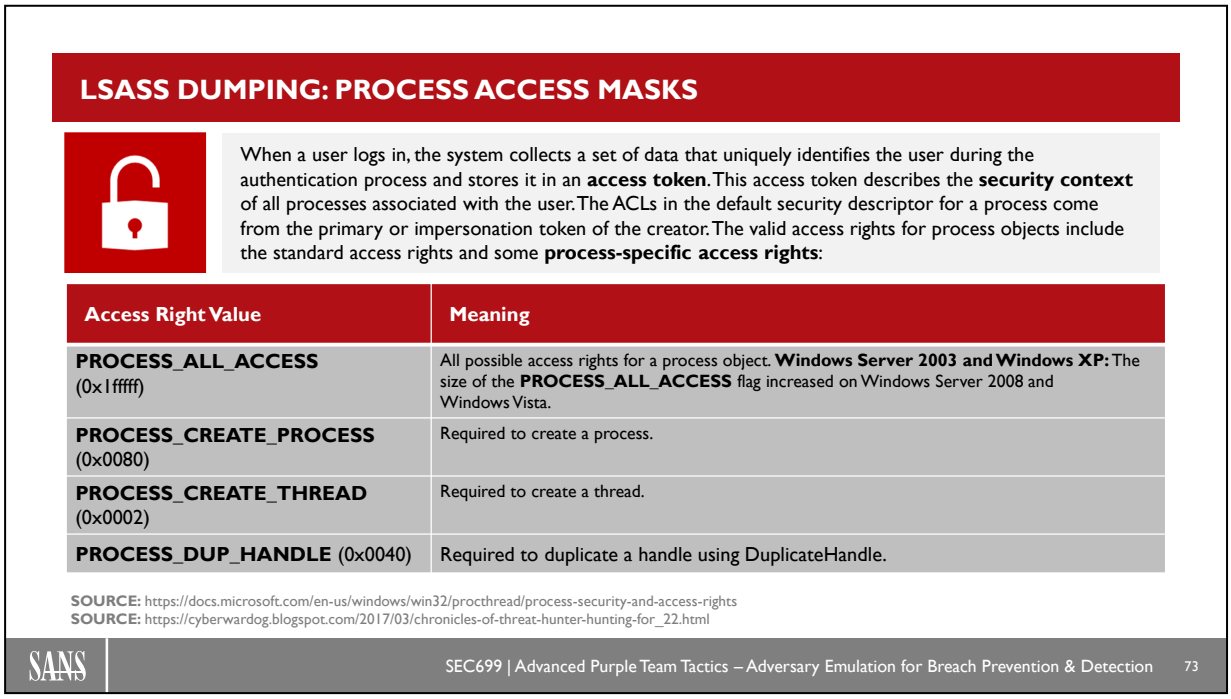
LSASS Dumping: Detecting the Techniques – Example SIGMA (3)
As we are looking for ProcessAccess events, we look for event ID 10 (ProcessAccess), with lsass.exe as the target image.

Let's assume that we don't know these source images, as we want to detect lsass attacks originating from any process.

As indicated before, though, access to lsass.exe is not uncommon on a Windows system, so this alone is not sufficient. What other logic could we add?

We can leverage the access masks used by the different tools when they attempt to access lsass. In the rule on the slide, we can see a series of masks known to be used by different tools. We will further explain these masks in the next slides.

Next up, we can start whitelisting benign processes we expect to access lsass. Note that this rule includes the Task Manager in its whitelist…



LSASS Dumping: Process Access Masks

When a user logs in, the system collects a set of data that uniquely identifies the user during the authentication process and stores it in an access token. This access token describes the security context of all processes associated with the user. The ACLs in the default security descriptor for a process come from the primary or impersonation token of the creator. The valid access rights for process objects include the standard access rights and some process-specific access rights.

We will list the different access rights here one by one (from the Microsoft documentation):

PROCESS_ALL_ACCESS (0X1FFFFF)

All possible access rights for a process object. Windows Server 2003 and Windows XP: The size of the PROCESS_ALL_ACCESS flag increased on Windows Server 2008 and Windows Vista.

PROCESS_CREATE_PROCESS (0X0080)

Required to create a process.

PROCESS_CREATE_THREAD (0X0002)

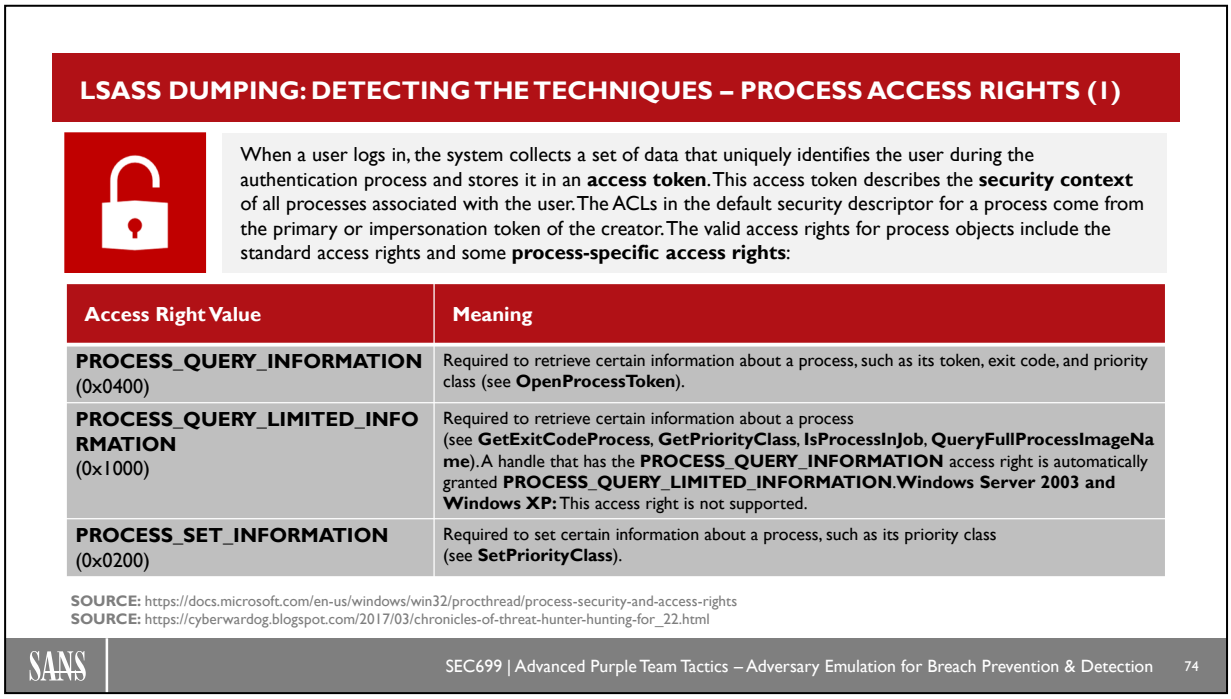
Required to create a thread.

PROCESS_DUP_HANDLE (0X0040)

Required to duplicate a handle using DuplicateHandle.

Two excellent resources:

https://docs.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights
https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for_22.html



LSASS Dumping: Detecting the Techniques – Process Access Rights (1)
When a user logs in, the system collects a set of data that uniquely identifies the user during the authentication process and stores it in an access token. This access token describes the security context of all processes associated with the user. The ACLs in the default security descriptor for a process come from the primary or impersonation token of the creator. The valid access rights for process objects include the standard access rights and some process-specific access rights.

We will list the different access rights here one by one (from the Microsoft documentation):

PROCESS_QUERY_INFORMATION (0x0400)

Required to retrieve certain information about a process, such as its token, exit code, and priority class (see OpenProcessToken).

PROCESS_QUERY_LIMITED_INFORMATION (0x1000)

Required to retrieve certain information about a process (see GetExitCodeProcess, GetPriorityClass, IsProcessInJob, QueryFullProcessImageName). A handle that has the PROCESS_QUERY_INFORMATION access right is automatically granted PROCESS_QUERY_LIMITED_INFORMATION. Windows Server 2003 and Windows XP: This access right is not supported.

PROCESS_SET_INFORMATION (0x0200)

Required to set certain information about a process, such as its priority class (see SetPriorityClass).

Two excellent resources:
https://docs.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights
https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for_22.html

LSASS DUMPING: DETECTING THE TECHNIQUES – PROCESS ACCESS RIGHTS (2)

When a user logs in, the system collects a set of data that uniquely identifies the user during the authentication process and stores it in an **access token**. This access token describes the **security context** of all processes associated with the user. The ACLs in the default security descriptor for a process come from the primary or impersonation token of the creator. The valid access rights for process objects include the standard access rights and some **process-specific access rights**:

Access Right Value	Meaning
PROCESS_SET_QUOTA (0x0100)	Required to set memory limits using **SetProcessWorkingSetSize**.
PROCESS_SUSPEND_RESUME (0x0800)	Required to suspend or resume a process.
PROCESS_TERMINATE (0x0001)	Required to terminate a process using **TerminateProcess**.
PROCESS_VM_OPERATION (0x0008)	Required to perform an operation on the address space of a process (see **VirtualProtectEx** and **WriteProcessMemory**).

SOURCE: https://docs.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights
SOURCE: https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for_22.html

LSASS Dumping: Detecting the Techniques – Process Access Rights (2)
When a user logs in, the system collects a set of data that uniquely identifies the user during the authentication process and stores it in an access token. This access token describes the security context of all processes associated with the user. The ACLs in the default security descriptor for a process come from the primary or impersonation token of the creator. The valid access rights for process objects include the standard access rights and some process-specific access rights.

We will list the different access rights here one by one (from the Microsoft documentation):

PROCESS_SET_QUOTA (0x0100)

Required to set memory limits using SetProcessWorkingSetSize.

PROCESS_SUSPEND_RESUME (0x0800)

Required to suspend or resume a process.

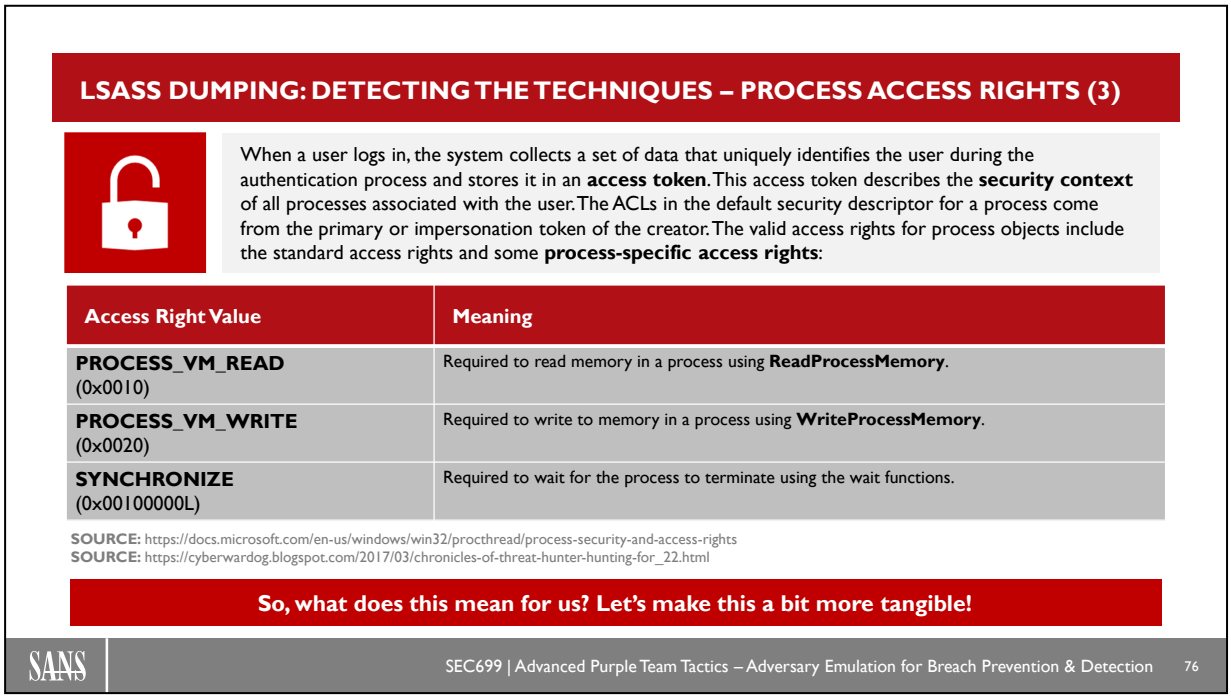
PROCESS_TERMINATE (0x0001)

Required to terminate a process using TerminateProcess.

PROCESS_VM_OPERATION (0x0008)

Required to perform an operation on the address space of a process
(see VirtualProtectEx and WriteProcessMemory).

Two excellent resources:
https://docs.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights
https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for_22.html



LSASS Dumping: Detecting the Techniques – Process Access Rights (3)
When a user logs in, the system collects a set of data that uniquely identifies the user during the authentication process and stores it in an access token. This access token describes the security context of all processes associated with the user. The ACLs in the default security descriptor for a process come from the primary or impersonation token of the creator. The valid access rights for process objects include the standard access rights and some process-specific access rights.

We will list the different access rights here one by one (from the Microsoft documentation):

PROCESS_VM_READ (0x0010)

Required to read memory in a process using ReadProcessMemory.

PROCESS_VM_WRITE (0x0020)

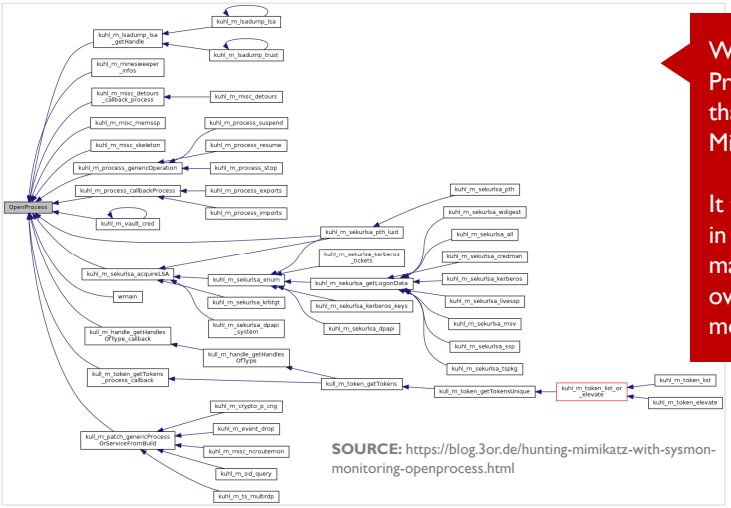
Required to write to memory in a process using WriteProcessMemory.

SYNCHRONIZE (0x00100000L)

Required to wait for the process to terminate using the wait functions.

Two excellent resources:
https://docs.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights
https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for_22.html

LSASS DUMPING: DETECTION – MIMIKATZ & OPENPROCESS



SOURCE: https://blog.3or.de/hunting-mimikatz-with-sysmon-monitoring-openprocess.html

We've explained the concept of ProcessAccess and the access masks that are involved. But when does Mimikatz "open" lsass.exe?

It turns out that a lot of the modules in Mimikatz exhibit this behavior. The mapping to the left provides a nice overview of how the different modules rely on OpenProcess!

LSASS Dumping: Detection – Mimikatz & OpenProcess
We've explained the concept of ProcessAccess and the access masks that are involved. But when does Mimikatz "open" lsass.exe?

It turns out that a lot of the modules in Mimikatz exhibit this behavior. The mapping on the slides provides a nice overview of how the different modules in Mimikatz rely on OpenProcess!

A well-written blog post by Dimitrios Slamaris is available here: https://blog.3or.de/hunting-mimikatz-with-sysmon-monitoring-openprocess.html

So, what type of access do they require to lsass.exe? Let's investigate!

LSASS DUMPING: MIMIKATZ PROCESS ACCESS MASKS

Mimikatz Command	OpenProcess caller function	Destination	Access Rights	Access Mask
lsadump::lsa /patch	kuhl_m_lsadump_lsa_getHandle()	SamSs	PROCESS_VM_READ PROCESS_VM_WRITE PROCESS_VM_OPERATION PROCESS_QUERY_INFORMATION	0x1438
lsadump::lsa /inject	kuhl_m_lsadump_lsa_getHandle()	SamSs	PROCESS_VM_READ PROCESS_VM_WRITE PROCESS_VM_OPERATION PROCESS_QUERY_INFORMATION PROCESS_CREATE_THREAD	0x143a
lsadump::trust /patch	kuhl_m_lsadump_lsa_getHandle()	SamSs	PROCESS_VM_READ PROCESS_VM_WRITE PROCESS_VM_OPERATION PROCESS_QUERY_INFORMATION	0x1438

SOURCE: https://blog.3or.de/hunting-mimikatz-with-sysmon-monitoring-openprocess.html

LSASS Dumping: Mimikatz Process Access Masks

So how do the different Mimikatz modules access lsass? The blog post that was previously mentioned (https://blog.3or.de/hunting-mimikatz-with-sysmon-monitoring-openprocess.html) has an excellent overview, using the different Mimikatz commands as a basis:

lsadump::lsa /patch

OpenProcess caller function: kuhl_m_lsadump_lsa_getHandle()

Destination: SamSs

Access rights: PROCESS_VM_READ, PROCESS_VM_WRITE, PROCESS_VM_OPERATION, PROCESS_QUERY_INFORMATION

Access mask: 0x1438

lsadump::lsa /inject

OpenProcess caller function: kuhl_m_lsadump_lsa_getHandle()

Destination: SamSs

Access rights: PROCESS_VM_READ, PROCESS_VM_WRITE, PROCESS_VM_OPERATION, PROCESS_QUERY_INFORMATION, PROCESS_CREATE_THREAD

Access mask: 0x143a

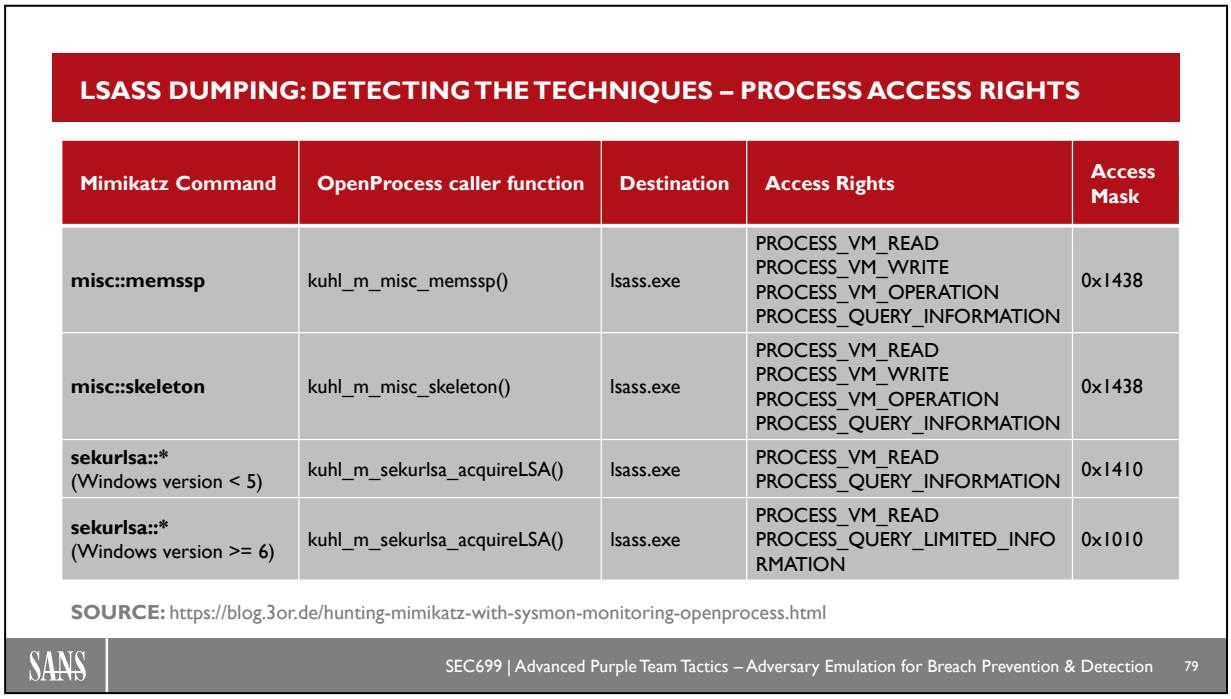
lsadump::trust /patch

OpenProcess caller function: kuhl_m_lsadump_lsa_getHandle()

Destination: SamSs

Access rights: PROCESS_VM_READ, PROCESS_VM_WRITE, PROCESS_VM_OPERATION, PROCESS_QUERY_INFORMATION

Access mask: 0x1438



LSASS Dumping: Detecting the Techniques – Process Access Rights

So how do the different Mimikatz modules access lsass? The blog post that was previously mentioned (https://blog.3or.de/hunting-mimikatz-with-sysmon-monitoring-openprocess.html) has an excellent overview, using the different Mimikatz commands as a basis:

misc::memssp

OpenProcess caller function: kuhl_m_misc_memssp()

Destination: lsass.exe

Access rights: PROCESS_VM_READ, PROCESS_VM_WRITE, PROCESS_VM_OPERATION, PROCESS_QUERY_INFORMATION

Access mask: 0x1438

misc::skeleton

OpenProcess caller function: kuhl_m_misc_skeleton()

Destination: lsass.exe

Access rights: PROCESS_VM_READ, PROCESS_VM_WRITE, PROCESS_VM_OPERATION, PROCESS_QUERY_INFORMATION

Access mask: 0x1438

sekurlsa::*

OpenProcess caller function: kuhl_m_sekurlsa_acquireLSA()

Destination: lsass.exe

Access rights: PROCESS_VM_READ, PROCESS_QUERY_INFORMATION

Access mask: 0x1410 (Windows version < 5)

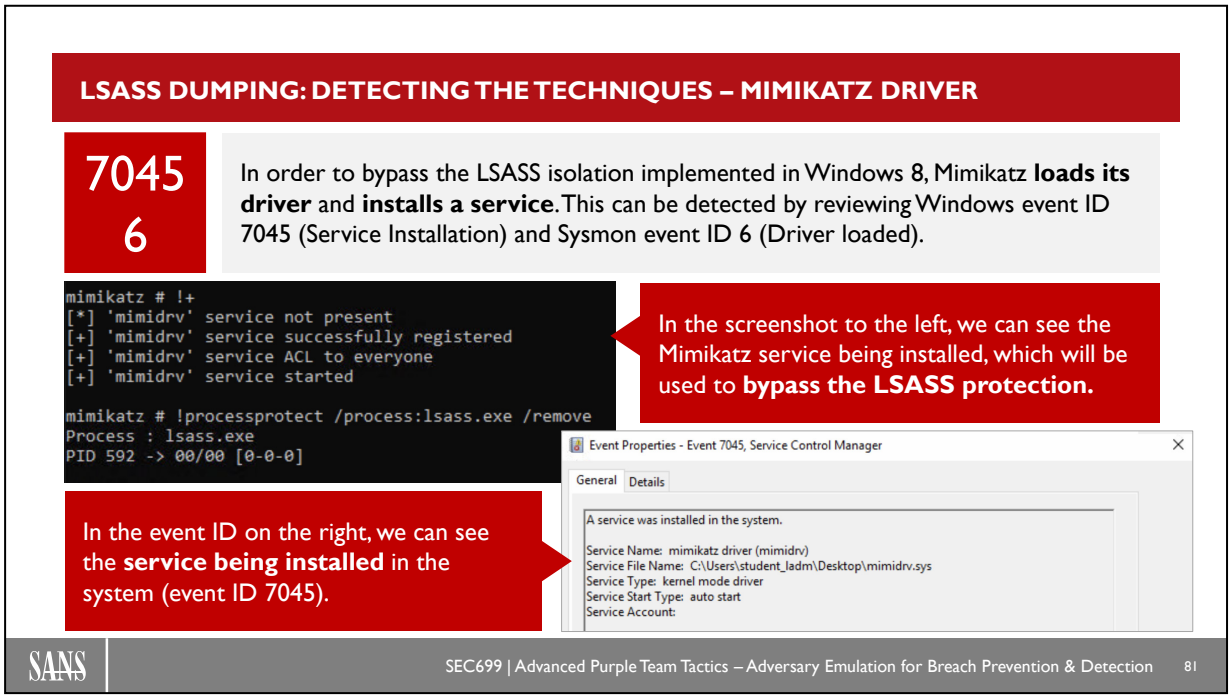
sekurlsa::*

OpenProcess caller function: kuhl_m_sekurlsa_acquireLSA()

Destination: lsass.exe

Access rights: PROCESS_VM_READ, PROCESS_QUERY_LIMITED_INFORMATION

Access mask: 0x1010 (Windows version >= 6)

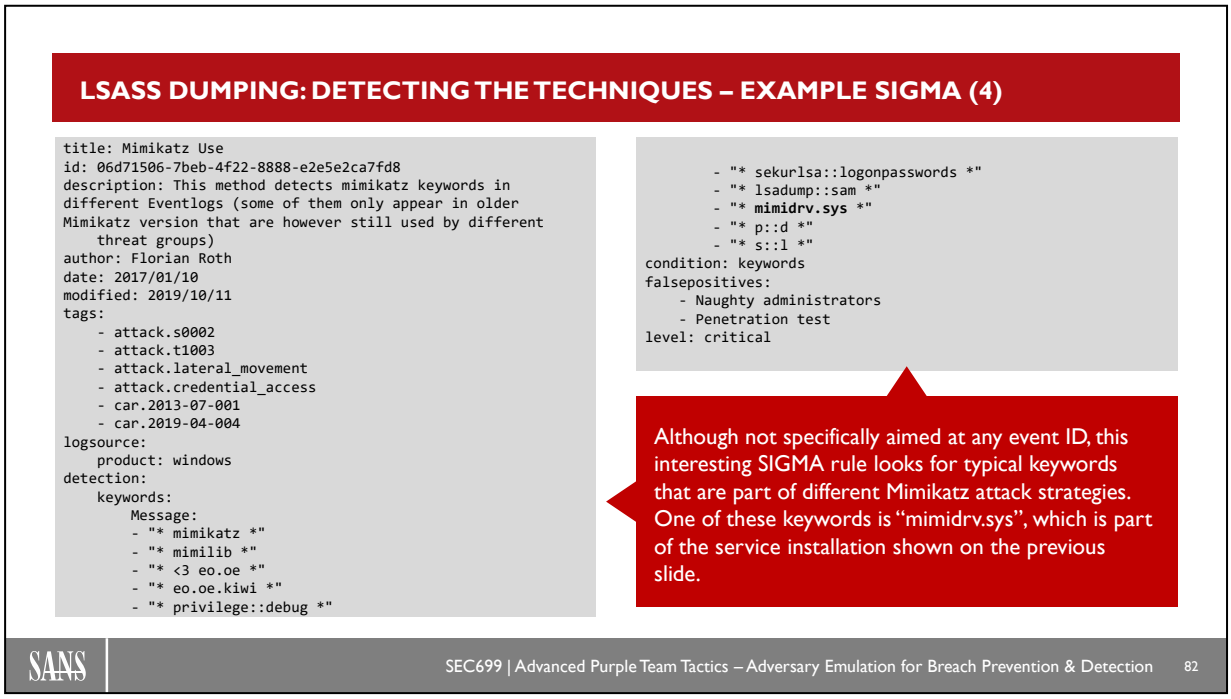


LSASS Dumping: Detecting the Techniques – Mimikatz Driver
In order to bypass the LSASS isolation implemented in Windows 8, Mimikatz loads its driver and installs a service. Right off the bat, you can imagine that this is not a very stealth approach. ☺

Indeed, we can detect such behavior by looking at two artifacts:

- The installation of the service (Windows event ID 7045)
- The loading of the Mimikatz driver (Sysmon event ID 6)

In the screenshot above, we can see the Mimikatz commands being executed and the service being installed!

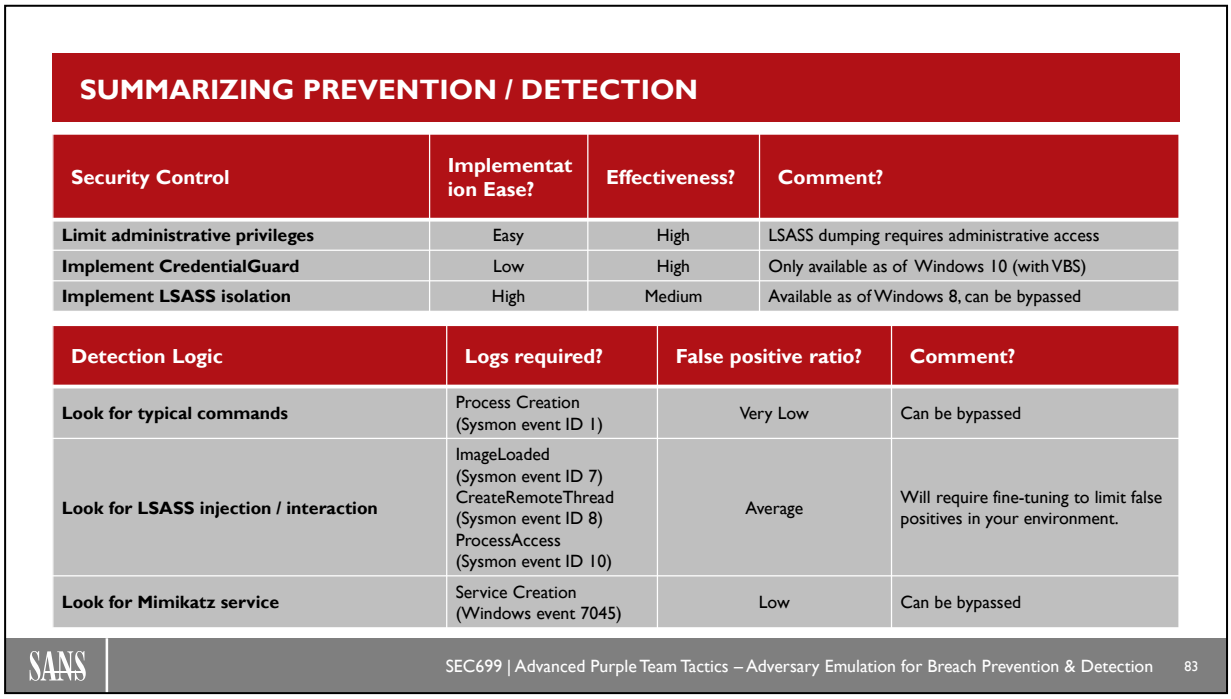


LSASS Dumping: Detecting the Techniques – Example SIGMA (4)
Although not specifically aimed at any event ID, this interesting SIGMA rule looks for typical keywords that are part of different Mimikatz attack strategies. One of these keywords is "mimidrv.sys", which is part of the service installation shown on the previous slide.

Other interesting keywords in the rule include:

- privilege::debug
- sekurlsa::logonPasswords
- p::d (alias for privilege::debug)
- s::l (alias for sekurlsa::logonPasswords)

Note that these keywords are, of course, not exhaustive! Please refer to the public SIGMA repository by Florian Roth for additional details: https://github.com/Neo23x0/sigma



Summarizing Prevention / Detection

In summary, there's a few strategies we can use to prevent LSASS credential dumping from being successful:

- Adversaries will require administrative privileges in order to access the lsass memory, so we should focus on limiting administrative privileges
- When we have a system with Windows 10 (and Virtualization Based Security), we can deploy Credential Guard, which effectively protects credentials by relying on the Hyper-V hypervisor
- When we have a system with Windows 8, we can define LSASS as a protected process. Note that this can be bypassed by installing the Mimikatz service (which can, however, be detected)

Detection-wise, there's quite a few options as well:

- We can look for typical Mimikatz commands and command-line arguments in Sysmon Process Creation events (event ID 1)
- We can look for any forms of weird interaction with lsass; key things to look for:
  - Unsigned images being loaded in lsass (event ID 7)
  - Remote threads being created in lsass (event ID 8)
  - Processes accessing lsass (event ID 10)
- Finally, we can also look for the Mimikatz service being installed (Windows event ID 7045)

Given the above, many adversaries are moving to credential dumping strategies where they don't interact with lsass!

Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.

EXERCISE: STEALING CREDENTIALS FROM LSASS

Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

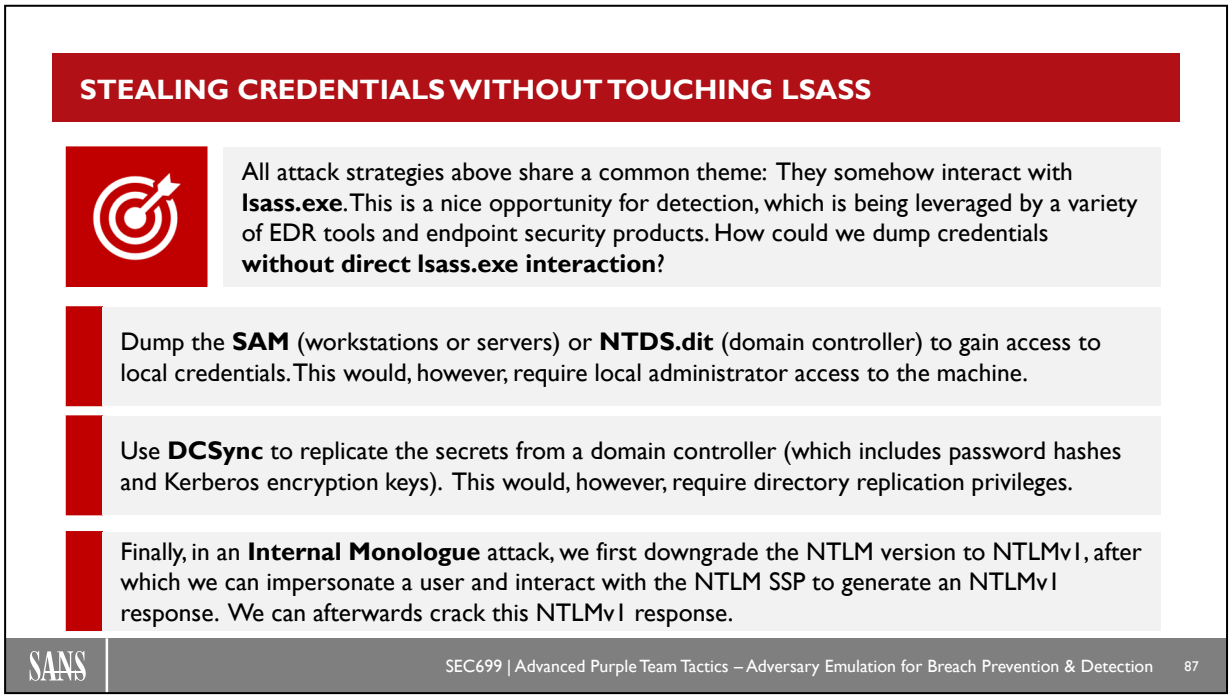
Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

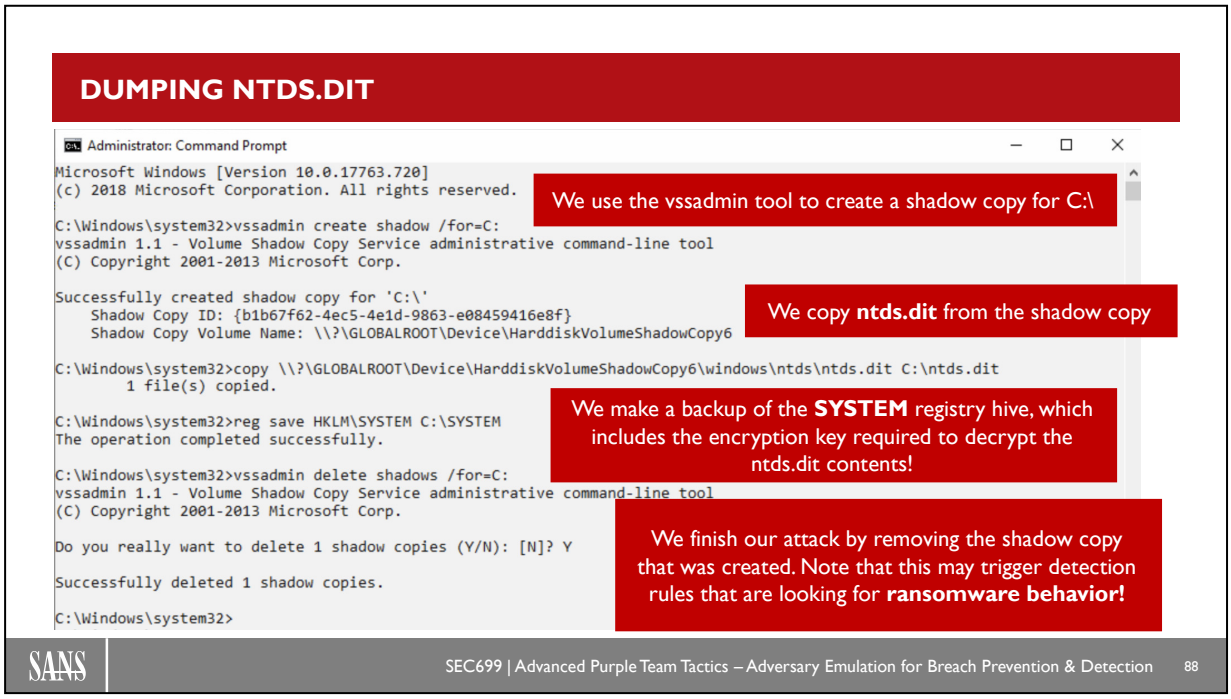
This page intentionally left blank.



Stealing Credentials without Touching LSASS
All attack strategies above share a common theme: They somehow interact with lsass.exe. This is a nice opportunity for detection, which is being leveraged by a variety of EDR tools and endpoint security products. How could we dump credentials without direct lsass.exe interaction?

There's many alternative ways that can be used to obtain credentials (using for example keyloggers). There's three main techniques we'll zoom in on, however:

- Dump the SAM (workstations or servers) or NTDS.dit (domain controller) to gain access to local credentials. This would, however, require local administrator access to the machine.
- Use DCSync to replicate the secrets from a domain controller (which includes password hashes and Kerberos encryption keys). This would, however, require directory replication privileges. More specifically, the below rights are required:
  - DS-Replication-Get-Changes - 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2
  - DS-Replication-Get-Changes-All - 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2
  - DS-Replication-Get-Changes-In-Filtered-Set - 89e95b76-444d-4c62-991a-0facbeda640c
- Finally, in an Internal Monologue attack, we first downgrade the NTLM version to NTLMv1, after which we can impersonate a user and interact with the NTLM SSP to generate an NTLMv1 response. We can afterwards crack this NTLMv1 response.



Dumping NTDS.DIT

We cannot just copy / paste ntds.dit, as the file is locked by the Operating System. Even when local administrator credentials are obtained, we still need to find a way around the "lock" that was put in place by the OS.

We can, of course, attempt to recover the file from a backup of the system, but could we extract the files from a live system?

Yes, we can! We can leverage the volume shadow copy service for this:

1. We first create a shadow copy for C:\

   vssadmin create shadow /for=C:

2. We copy the ndts.dit from the shadow copy we just created:

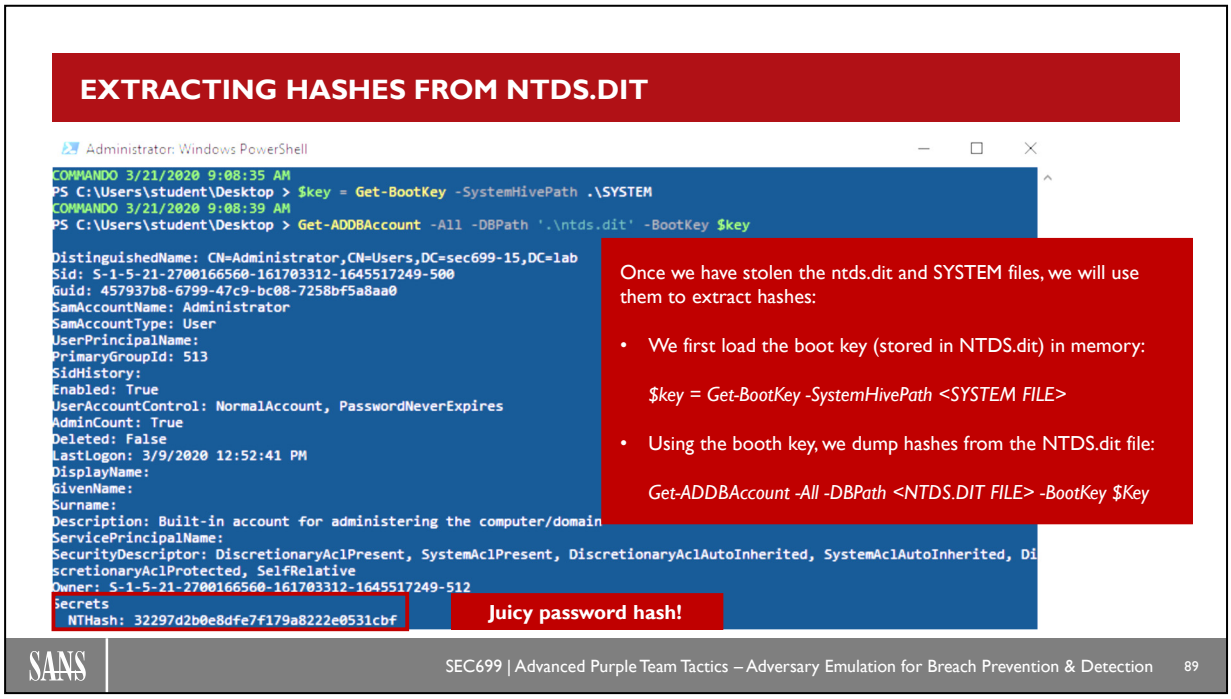
   copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy{X}\windows\ntds\ntds.dit C:\ntds.dit

3. We make a backup of the SYSTEM registry hive, which includes the encryption key required to decrypt the ntds.dit contents!

   reg save HKLM\SYSTEM C:\SYSTEM

4. We delete the shadow copy we created (optional):

   vssadmin delete shadows /for=C:



Extracting Hashes from NTDS.DIT
Once we have copied the NTDS.DIT and SYSTEM files, we can proceed to extract hashes by using the Get-ADDBAccount PowerShell command.

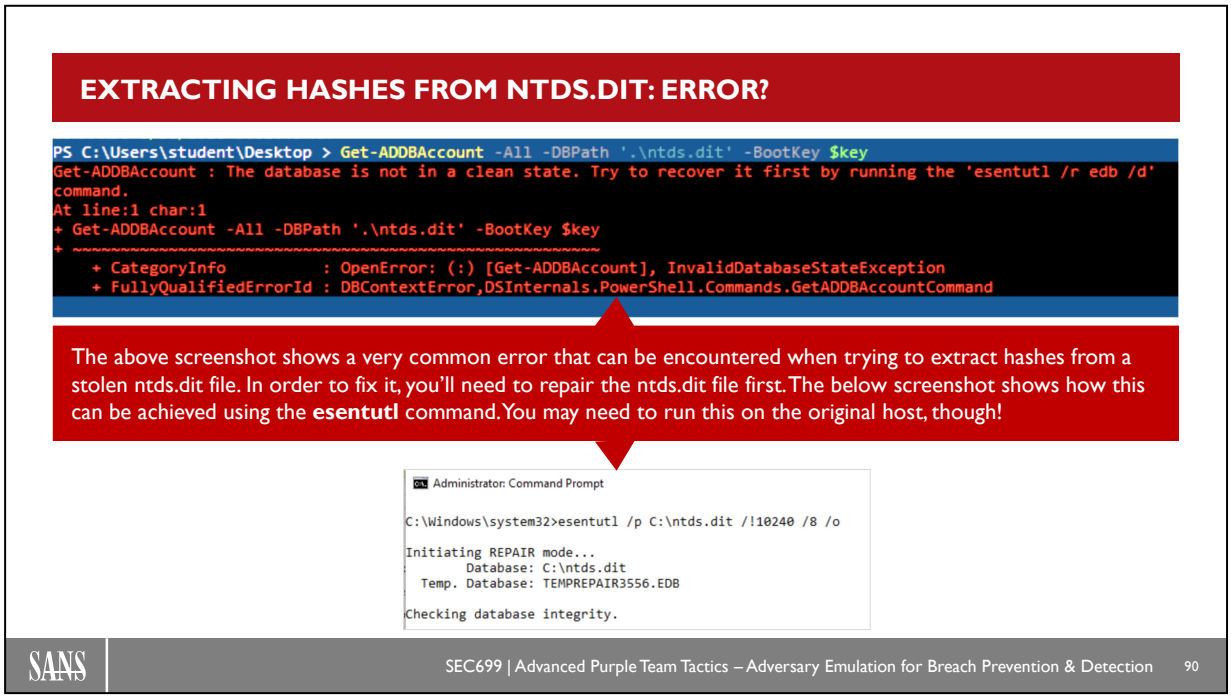
This command is available from the DSInternals framework, which can be found at https://github.com/MichaelGrafnetter/DSInternals/. It comes pre-installed on a CommandoVM virtual machine, though!

In order to successfully extract hashes (and other interesting information), we'll need to run the following commands:

- We first load the boot key (stored in NTDS.dit) in memory:

  $key = Get-BootKey -SystemHivePath <SYSTEM FILE>
- Using the booth key, we dump hashes from the NTDS.dit file:

  Get-ADDBAccount -All -DBPath <NTDS.DIT FILE> -BootKey $Key



Extracting Hashes from NTDS.DIT: Error?

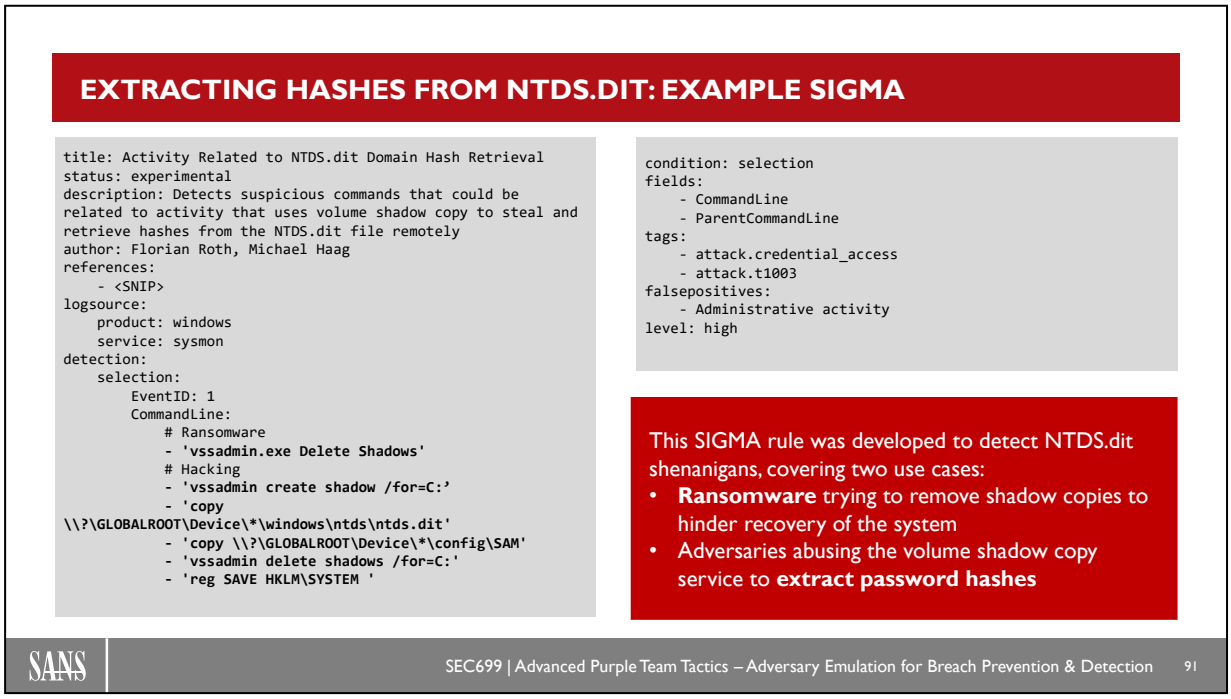
The above screenshot on the slide shows a very common error that can be encountered when trying to extract hashes from a stolen ntds.dit file:

"The database is not in a clean state. Try to recover it first by running the 'esentutl /r edb /d' command."

In order to fix it, you'll need to repair the ntds.dit file first. The below screenshot shows how this can be achieved using the esentutl command:

esentutl /p <NTDS FILE> /!10240 /8 /o

You may need to run this on the original host, though!



Extracting Hashes from NTDS.DIT: Example SIGMA

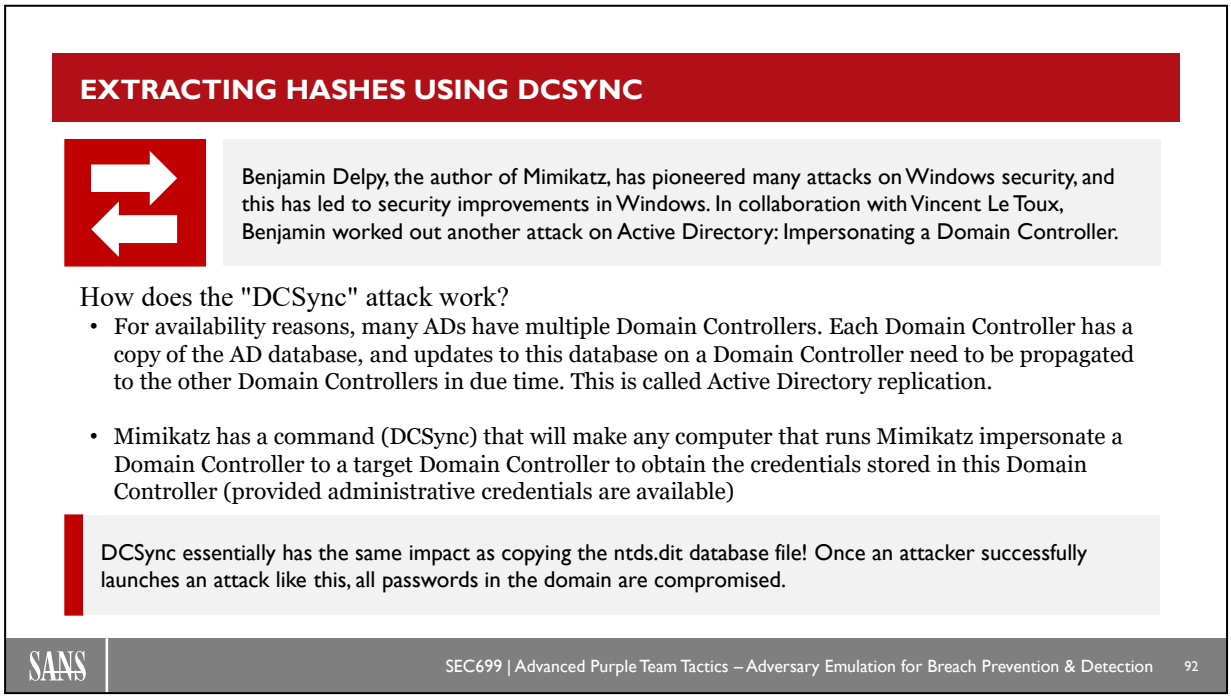
This SIGMA rule was developed to detect NTDS.dit shenanigans, covering two use cases:

- Ransomware trying to remove shadow copies to hinder recovery of the system
- Adversaries abusing the volume shadow copy service to extract password hashes

Strings / commands that are being alerted on include:

- vssadmin.exe Delete Shadows
- vssadmin create shadow /for=C:
- copy \\?\GLOBALROOT\Device*\windows\ntds\ntds.dit
- copy \\?\GLOBALROOT\Device*\config\SAM
- vssadmin delete shadows /for=C:
- reg SAVE HKLM\SYSTEM

Note that these keywords are, of course, not exhaustive! Please refer to the public SIGMA repository by Florian Roth for additional details: https://github.com/Neo23x0/sigma



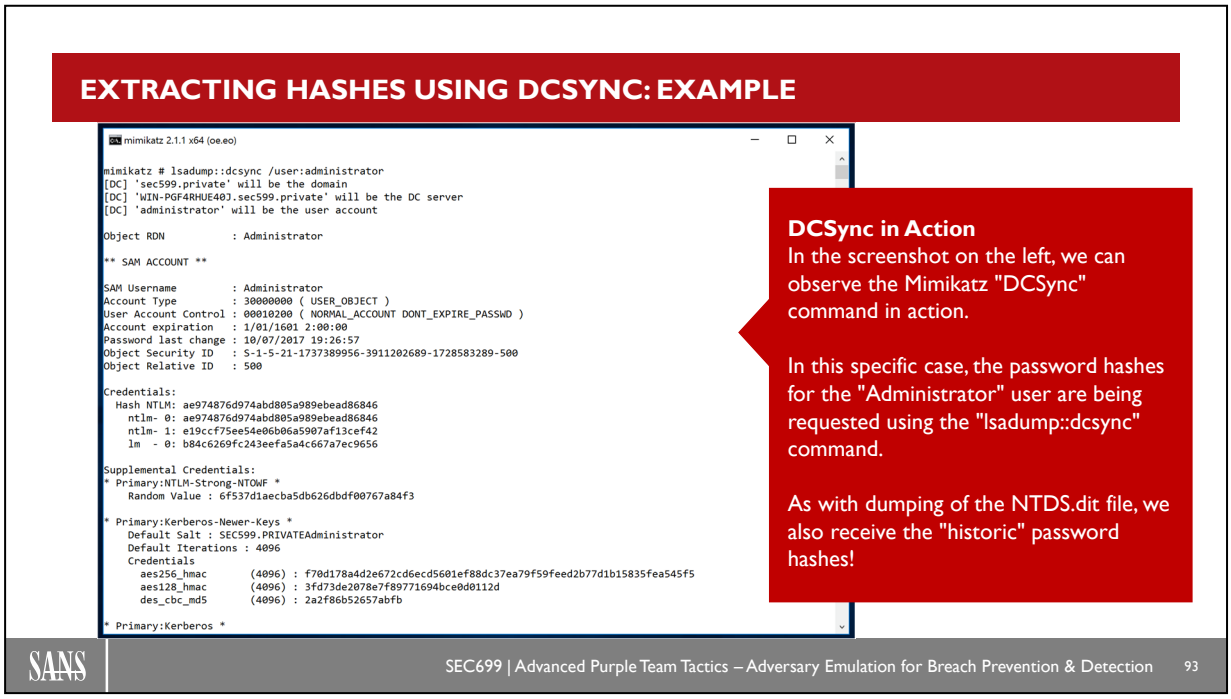
Extracting Hashes Using DCSync
Benjamin Delpy, the author of Mimikatz, has pioneered many attacks on Windows security, and this has led to security improvements in Windows. In collaboration with Vincent Le Toux, Benjamin worked out another attack on Active Directory: Impersonating a Domain Controller.

For availability reasons, administrators deploy more than one Domain Controller in an Active Directory infrastructure. Each Domain Controller has a copy of the Active Directory database, and updates to this database on a Domain Controller (for example, the creation of a new user) need to be propagated to the other Domain Controllers in due time. This is called Active Directory replication: A set of methods and protocols to synchronize the database of Active Directory Domain Controllers.

Vincent and Benjamin have worked out these methods and protocols for use by Mimikatz: Mimikatz has a command (DCSync) that will make any computer that runs Mimikatz impersonate a Domain Controller to a target Domain Controller to obtain the credentials stored in this Domain Controller.

Of course, normal users cannot access this information. One needs domain replication rights to participate in data replication.

DCSync can dump the hashes of all users or of a selected user.



Extracting Hashes Using DCSync: Example
This example shows the DCSync command. By issuing the "kerberos::dcsync /user:administrator" command, we send a request to a Domain Controller for an administrator's credentials. The command "kerberos::dcsync" would list the credentials of all users.

When this command is issued without extra options, Mimikatz selects the domain and the Domain Controller automatically extracts the credentials from this Domain Controller via replication using the Directory Replication Service Remote (MS-DRSR) protocol.

This is a very powerful attack: Once an attacker has obtained domain admin credentials, he/she can use DCSync to connect to a Domain Controller and extract the credentials of the krbtgt account. This data can then be used to create a Golden Ticket, and then it is game over: The only recourse you have is to change the krbtgt account password. This password never expires and is never changed, unless it is done manually. If you discover that the krbtgt NTLM hash has been compromised, you will have to change the password.

It is possible to detect and prevent a DCSync attack. MS-DRSR network traffic should only occur between Domain Controllers. If you detect MS-DRSR network traffic between a Domain Controller and a workstation, you know a DCSync attack took place.

If you segment your Domain Controllers in a dedicated network segment, with advanced firewalls as chokepoints between the other network segments, you can block MS-DRSR traffic outside the Domain Controller network segment.

EXTRACTING HASHES USING DCSYNC: EXAMPLE SIGMA

```
title: Mimikatz DC Sync
id: 611eab06-a145-4dfa-a295-3ccc5c20f59a
description: Detects Mimikatz DC sync security events
status: experimental
date: 2018/06/03
modified: 2019/10/08
author: Benjamin Delpy, Florian Roth
references:
    -
https://twitter.com/gentilkiwi/status/1003236624925413376
    -
https://gist.github.com/gentilkiwi/dcc132457408cf11ad2061340d
cb53c2
tags:
    - attack.credential_access
    - attack.s0002
    - attack.t1003
logsource:
    product: windows
    service: security
detection:
    selection:
        EventID: 4662
        Properties:
            - '*Replicating Directory Changes All*'
            - '*1131f6ad-9c07-11d1-f79f-00c04fc2dcd2*'
    filter1:
        SubjectDomainName: 'Window Manager'
    filter2:
        SubjectUserName:
            - 'NT AUTHORITY*'
            - '*$'
    condition: selection and not filter1 and not filter2
falsepositives:
    - Valid DC Sync that is not covered by the filters;
please report
level: high
```

This SIGMA rule was developed to detect DCSync abuses. It looks for:

- Event ID 4662: An operation was performed on an object (this is rather noisy)
- Properties: Replicating Directory Changes All (or its GUID equivalent 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2)

Extracting Hashes Using DCSync: Example SIGMA

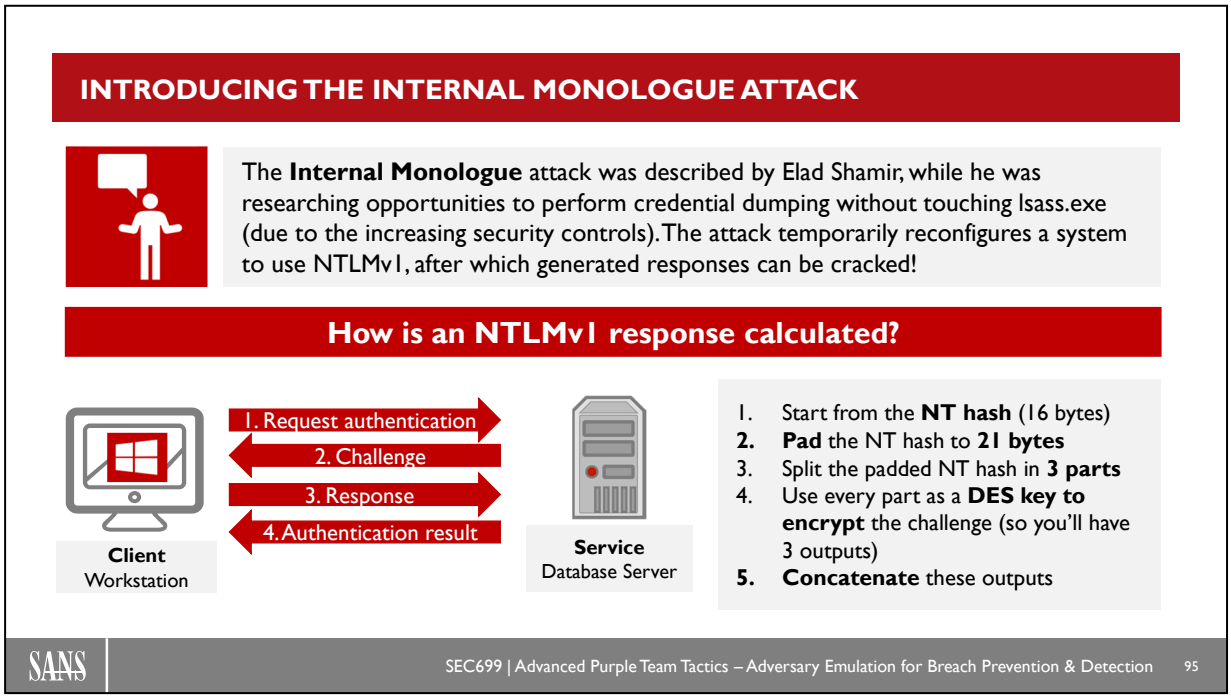
This SIGMA rule was developed to detect DCSYNC abuses. It looks for:

- Event ID 4662: An operation was performed on an object (this is rather noisy)
- Properties: Replicating Directory Changes All (or its GUID equivalent 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2)

The rule could be further fine-tuned to include the other replication privileges:

- DS-Replication-Get-Changes - 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2
- DS-Replication-Get-Changes-In-Filtered-Set - 89e95b76-444d-4c62-991a-0facbeda640c

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma



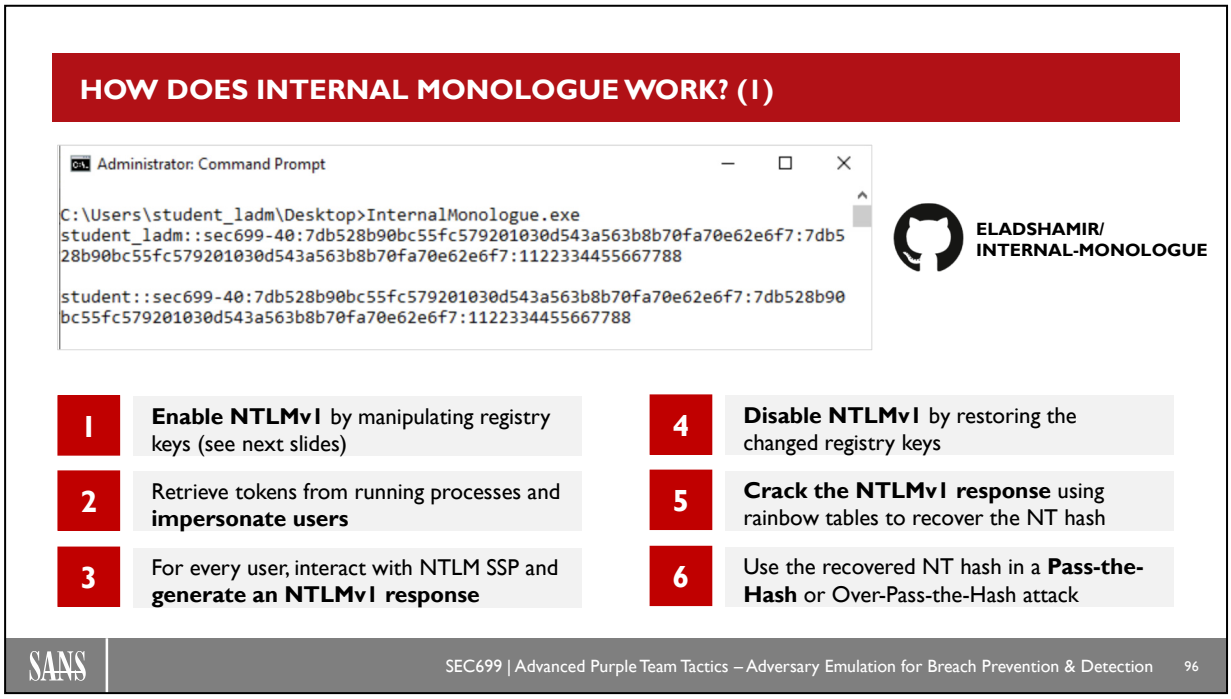
Introducing the Internal Monologue Attack

The Internal Monologue attack was described by Elad Shamir, while he was researching opportunities to perform credential dumping without touching lsass.exe (due to the increasing security controls). The attack temporarily reconfigures a system to use NTLMv1, after which generated responses can be cracked!

In order for us to understand how the attack works, we need to understand how an NTLMv1 response is built. We've seen the overall NTLMv1 / NTLMv2 authentication mechanism before (on Day 2). The interesting part is, however, when the client generates a response to a challenge:

1. The process start from the NT hash (16 bytes)
2. The NT hash is padded to exactly 21 bytes
3. The padded NT hash is split in 3 parts
4. Every part is used as a DES key to encrypt the challenge (so you'll have 3 outputs)
5. These outputs are concatenated, resulting in the response

Full information can be found here: https://github.com/eladshamir/Internal-Monologue

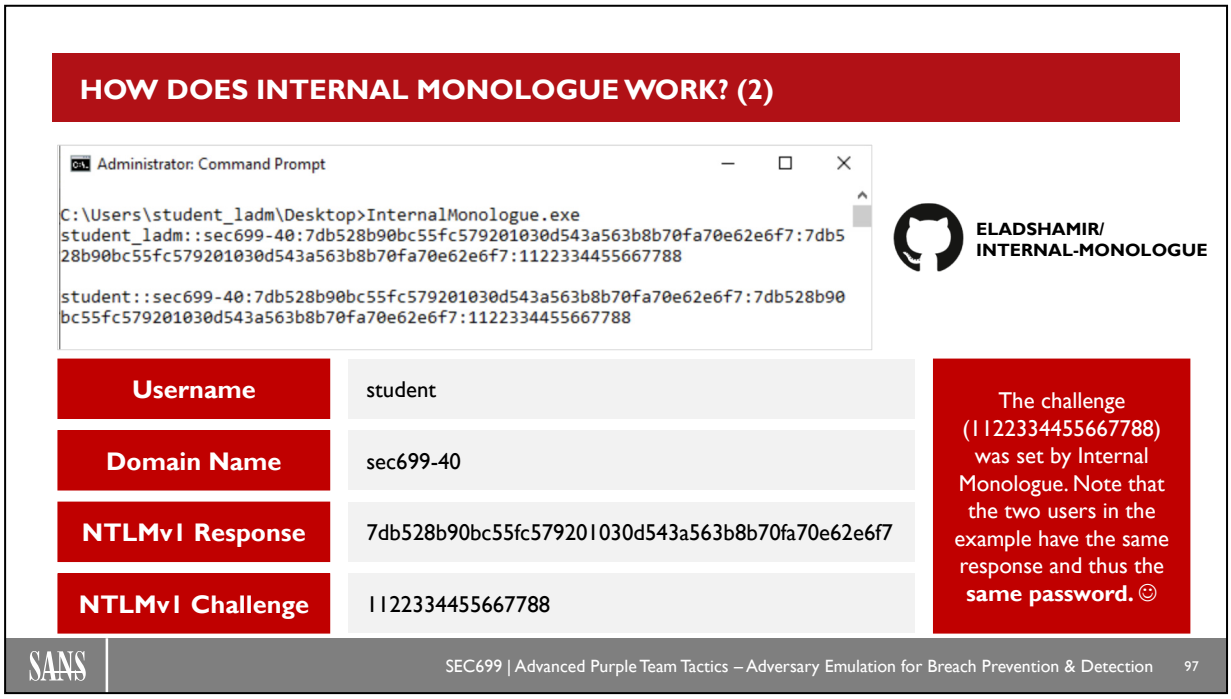


How Does Internal Monologue Work? (1)

Let's walk through the internal monologue attack step by step:

1. In a first step, the adversary manipulates the local system registry hive to enable NTLMv1, which is typically disabled on modern Windows systems;
2. The adversary will leverage token impersonation and retrieve tokens from running processes on the systems;
3. For every user it can impersonate, it will interact with the NTLM Security Support Provider and generate an NTLMv1 response;
4. Once the NTLMv1 response is obtained, the registry settings are reverted to their original values;
5. The obtained NTLMv1 responses can be cracked using a tool such as Hashcat. Furthermore, a rainbow table of NTLMv1 responses exists;
6. Finally, the recovered NT hash can be used in a Pass-the-Hash or Over-Pass-the-Hash attack.

Full information can be found here: https://github.com/eladshamir/Internal-Monologue

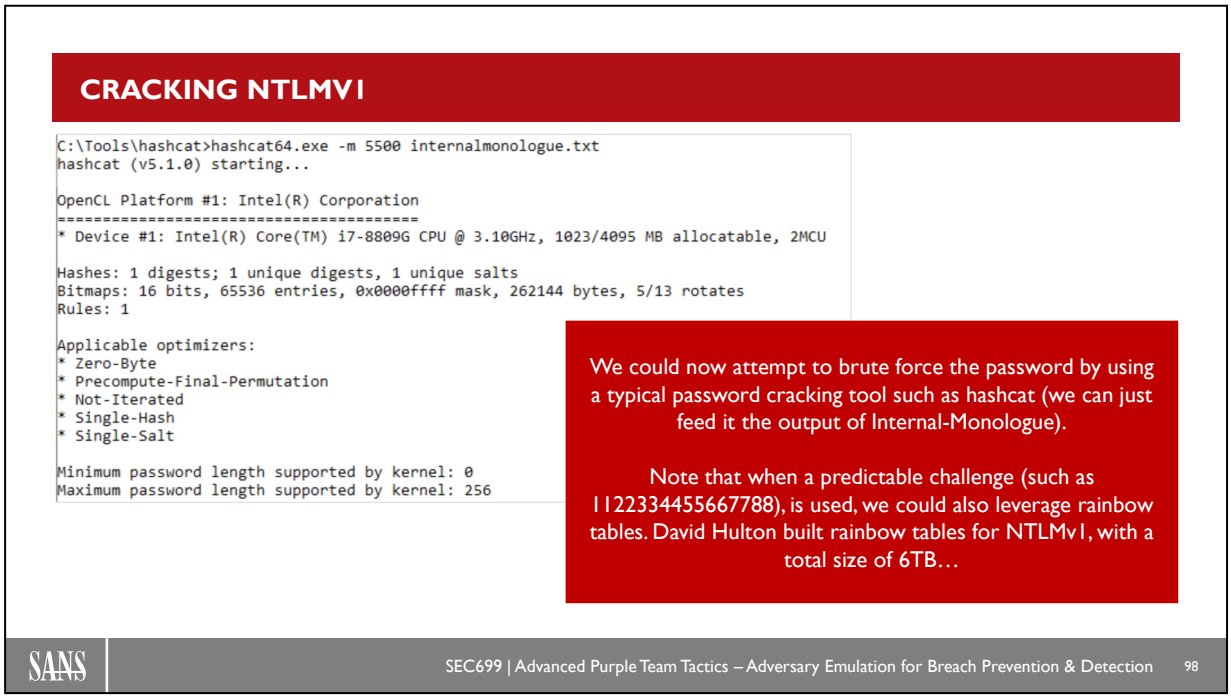


How Does Internal Monologue Work? (2)

Let's take a closer look at the output of the command. In the output of Internal-Monologue, we can distinguish the following information:

- Username: student
- Domain name: sec699-40
- NTLMv1 response: 7db528b90bc55fc579201030d543a563b8b70fa70e62e6f7
- NTLMv1 challenge: 1122334455667788 (set by Internal Monologue)

Note that the NTLMv1 response is the same for both of the users in the output. This reveals that their password is the same as well!



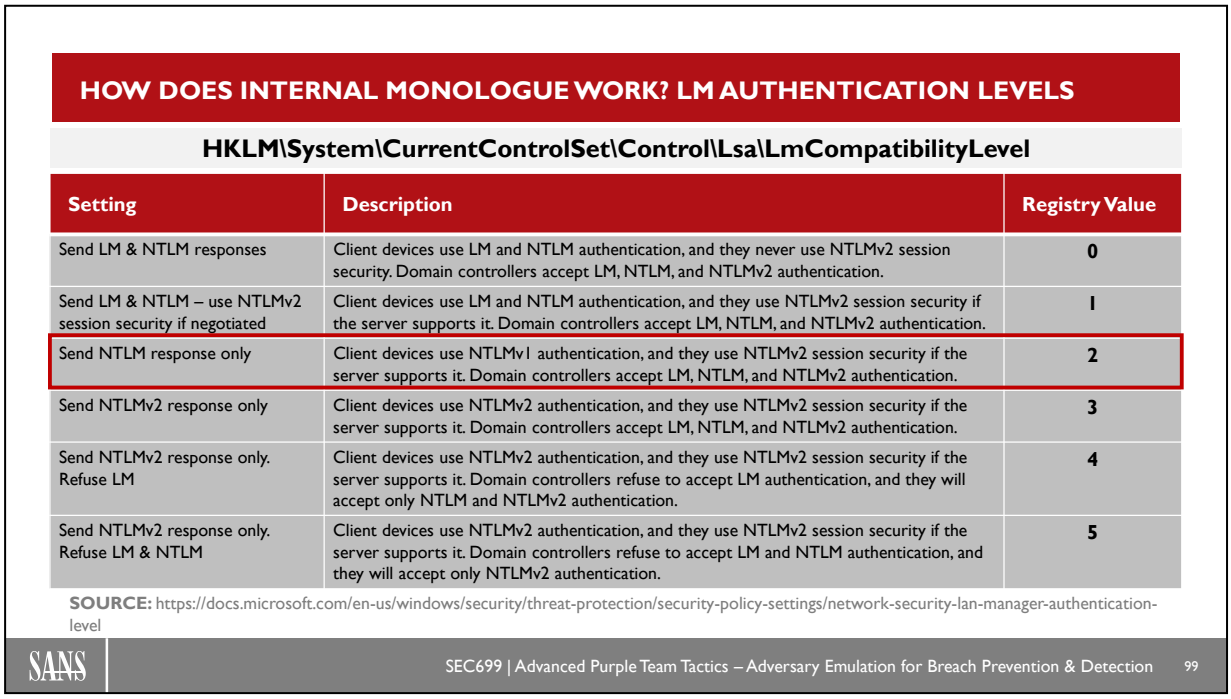
Cracking NTLMv1

We could now attempt to brute force the password by using a typical password cracking tool such as hashcat (we can just feed it the output of Internal-Monologue).

In hashcat, hash type 5500 stands for NTLMv1 (NetNTLMv1), so we can specify this using the "-m" flag.

Note that when a predictable challenge (such as 1122334455667788), is used, we could also leverage rainbow tables to facilitate cracking!

David Hulton (0x31337) built rainbow tables for NTLMv1, with a total size of 6TB.



HOW DOES INTERNAL MONOLOGUE WORK? LM AUTHENTICATION LEVELS

HKLM\System\CurrentControlSet\Control\Lsa\LmCompatibilityLevel

Setting	Description	Registry Value
Send LM & NTLM responses	Client devices use LM and NTLM authentication, and they never use NTLMv2 session security. Domain controllers accept LM, NTLM, and NTLMv2 authentication.	0
Send LM & NTLM – use NTLMv2 session security if negotiated	Client devices use LM and NTLM authentication, and they use NTLMv2 session security if the server supports it. Domain controllers accept LM, NTLM, and NTLMv2 authentication.	1
Send NTLM response only	Client devices use NTLMv1 authentication, and they use NTLMv2 session security if the server supports it. Domain controllers accept LM, NTLM, and NTLMv2 authentication.	2
Send NTLMv2 response only	Client devices use NTLMv2 authentication, and they use NTLMv2 session security if the server supports it. Domain controllers accept LM, NTLM, and NTLMv2 authentication.	3
Send NTLMv2 response only. Refuse LM	Client devices use NTLMv2 authentication, and they use NTLMv2 session security if the server supports it. Domain controllers refuse to accept LM authentication, and they will accept only NTLM and NTLMv2 authentication.	4
Send NTLMv2 response only. Refuse LM & NTLM	Client devices use NTLMv2 authentication, and they use NTLMv2 session security if the server supports it. Domain controllers refuse to accept LM and NTLM authentication, and they will accept only NTLMv2 authentication.	5

SOURCE: https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/network-security-lan-manager-authentication-level

How Does Internal Monologue Work? LM Authentication Levels

The main registry value that controls the NTLM (& LM) authentication level is the LmCompatibilityLevel; it can be found in the following location:

HKLM\System\CurrentControlSet\Control\Lsa\LmCompatibilityLevel

There's 6 possible settings:

Send LM & NTLM responses (registry value 0)

Client devices use LM and NTLM authentication, and they never use NTLMv2 session security. Domain controllers accept LM, NTLM, and NTLMv2 authentication.

Send LM & NTLM – use NTLMv2 session security if negotiated (registry value 1)

Client devices use LM and NTLM authentication, and they use NTLMv2 session security if the server supports it. Domain controllers accept LM, NTLM, and NTLMv2 authentication.

Send NTLM response only (registry value 2)

Client devices use NTLMv1 authentication, and they use NTLMv2 session security if the server supports it. Domain controllers accept LM, NTLM, and NTLMv2 authentication.

This is the setting that Internal Monologue will set!

Send NTLMv2 response only (registry value 3)

Client devices use NTLMv2 authentication, and they use NTLMv2 session security if the server supports it. Domain controllers accept LM, NTLM, and NTLMv2 authentication.

Send NTLMv2 response only. Refuse LM (registry value 4)

Client devices use NTLMv2 authentication, and they use NTLMv2 session security if the server supports it. Domain controllers refuse to accept LM authentication, and they will accept only NTLM and NTLMv2 authentication.

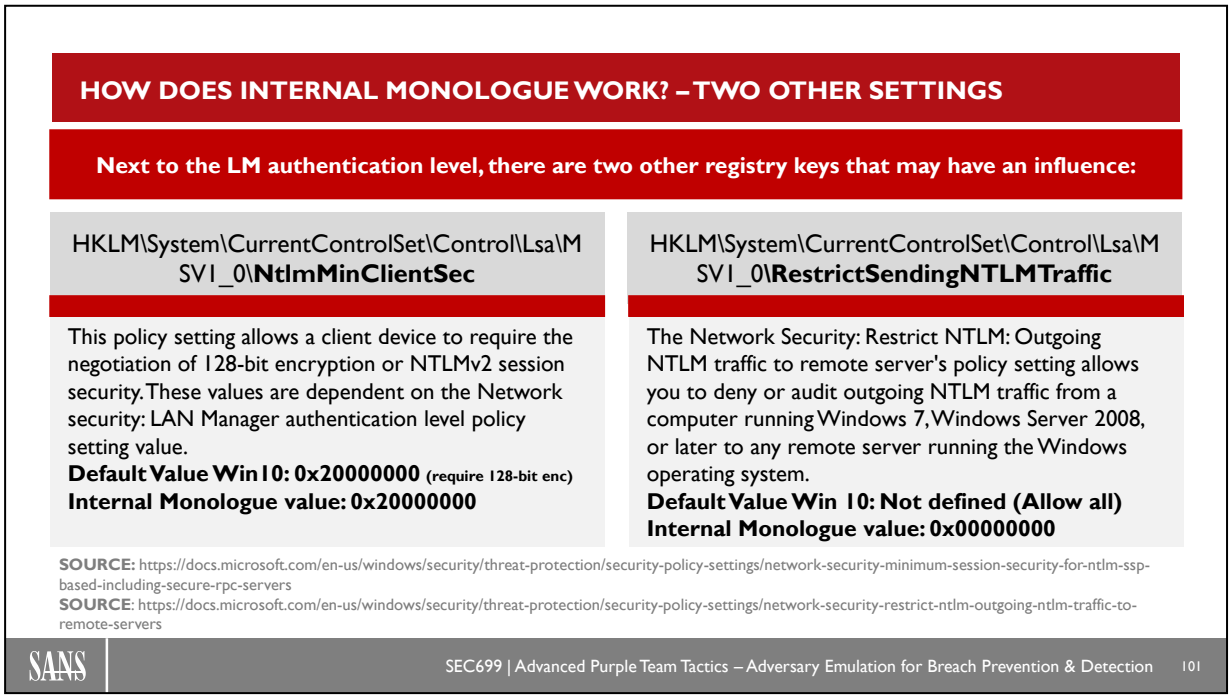
Send NTLMv2 response only. Refuse LM & NTLM (registry value 5)

Client devices use NTLMv2 authentication, and they use NTLMv2 session security if the server supports it. Domain controllers refuse to accept LM and NTLM authentication, and they will accept only NTLMv2 authentication.

It is fully documented in Microsoft's documentation:

https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/network-security-lan-manager-authentication-level

There's a few other registry keys that could affect the attack, so let's investigate…



How Does Internal Monologue Work? Two Other Settings
The most important value in the attack is, of course, the LM Authentication Level. When this is set to NTLMv1, though, there are still two other registry keys that may have an influence.

They are thus also manipulated by Internal-Monologue:

HKLM\System\CurrentControlSet\Control\Lsa\MSV1_0\NtlmMinClientSec

This policy setting allows a client device to require the negotiation of 128-bit encryption or NTLMv2 session security. These values are dependent on the Network security: LAN Manager authentication level policy setting value.

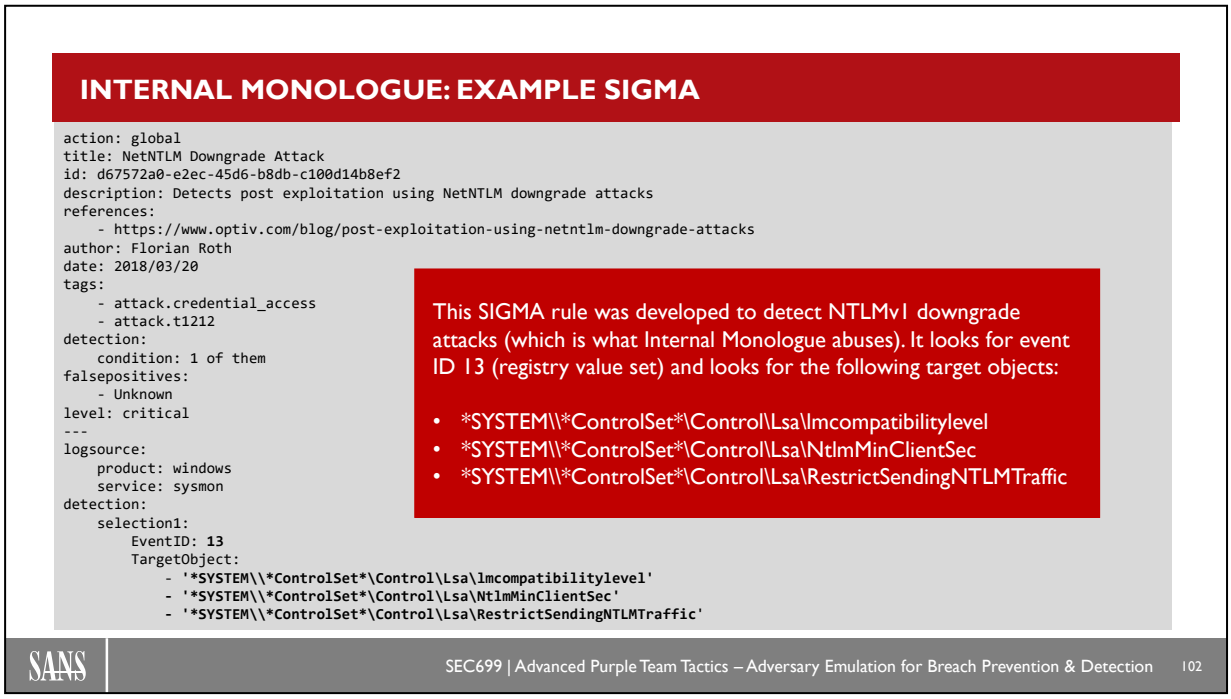
On a Windows 10 system, the default value of this key is 0x20000000, which means 128-bit encryption is enforced. As this does not affect Internal-Monologue, it is thus not manipulated when the default setting is set.

HKLM\System\CurrentControlSet\Control\Lsa\MSV1_0\RestrictSendingNTLMTraffic

The Network Security: Restrict NTLM: Outgoing NTLM traffic to remote server's policy setting allows you to deny or audit outgoing NTLM traffic from a computer running Windows 7, Windows Server 2008, or later to any remote server running the Windows operating system.

On a Windows 10 system, this registry key is by default not defined (which defaults to “allow all”). Internal-Monologue will explicitly set this key to 0x00000000 (allow all).

References:
https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/network-security-minimum-session-security-for-ntlm-ssp-based-including-secure-rpc-servers
https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/network-security-restrict-ntlm-outgoing-ntlm-traffic-to-remote-servers



Internal Monologue: Example Sigma

This SIGMA rule was developed to detect NTLMv1 downgrade attacks (which is what Internal Monologue abuses). It looks for event ID 13 (registry value set) and looks for the following target objects:

- *SYSTEM*ControlSet*\Control\Lsa\lmcompatibilitylevel
- *SYSTEM*ControlSet*\Control\Lsa\NtlmMinClientSec
- *SYSTEM*ControlSet*\Control\Lsa\RestrictSendingNTLMTraffic

This is in line with the registry keys we just reviewed when zooming in on the Internal Monologue attack!

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma

SUMMARIZING PREVENTION / DETECTION

Security Control	Implementation Ease?	Effectiveness?	Comment?
Limit administrative privileges	Easy	High	Most attacks require administrative access
Review accounts with replication privileges (e.g., using BloodHound)	Medium	High	Will have to whitelist expected accounts

Detection Logic	Logs required?	False positive ratio?	Comment?
Review registry manipulation for NTLM downgrade attacks	Registry interaction (Sysmon event ID 12, 13, 14)	Very Low	SIGMA rules exist
Detect execution of tools (vssadmin, reg save,…)	Process Creation (Sysmon event ID 1)	Very Low	SIGMA rules exist
Detect object access using replication privileges	Windows event ID 4662	Low	Event ID 4662 is noisy, but the replication privilege should be a reliable filter

The described attack approach relies on **administrative privileges** to the system. Detection is possible by looking for execution of the tools, registry manipulation (downgrade attacks), and object access using replication privileges.

Summarizing Prevention / Detection

In summary, there's a few strategies we can use to prevent LSASS credential dumping from being successful:

- Adversaries will require administrative privileges in order to execute the attack strategies described, so we should focus on limiting administrative privileges
- For DCSync attacks, we can review accounts that have replication privileges

Detection-wise, there's quite a few options as well:

- Review registry manipulation for NTLM downgrade attacks
  - Look for Sysmon event IDs 12, 13, 14 (registry manipulation)
  - Note that SIGMA rules already exist
- We can attempt to detect the different tools and their command-line arguments
  - Process Creation, Sysmon event ID 1
  - Several SIGMA rules already exist
- For DCSync, we can attempt to look for event ID 4662 (object access) and look for objects being accessed with replication privileges

Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.

EXERCISE: INTERNAL MONOLOGUE IN NTLMV1 DOWNGRADES

Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

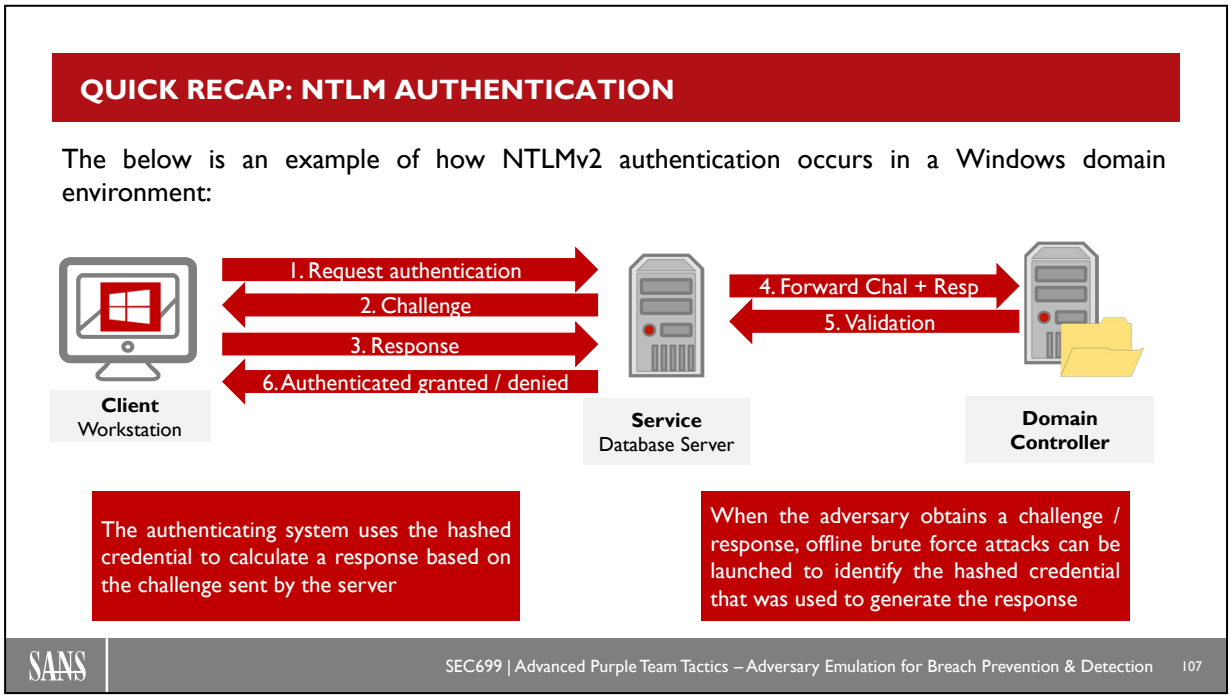
Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.



Quick Recap: NTLM Authentication
A very quick recap on NTLM authentication: When Kerberos authentication is not possible, Windows will fall back to NTLM authentication.

This can even happen between machines that are members of the same domain, but when all necessary conditions to use Kerberos are not in place. For example, Kerberos works with so-called service names. If we don't have a name, Kerberos cannot be used. This is the case when we access a share on a file server by using the IP address of the server instead of its server name.

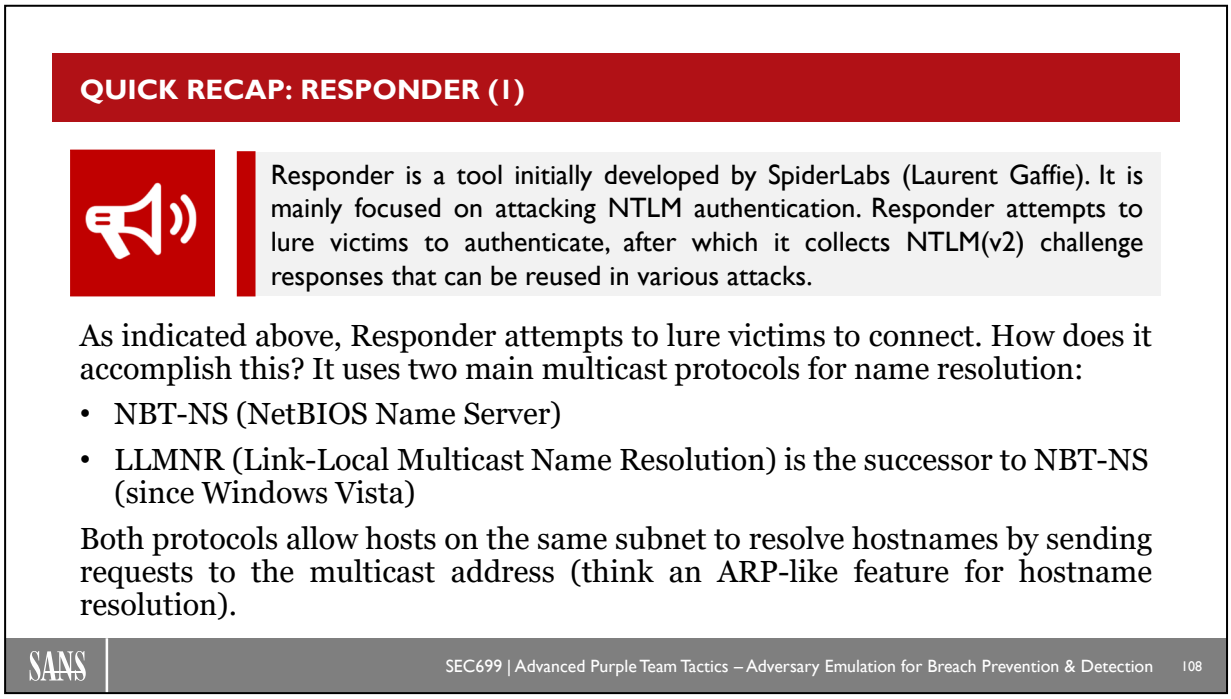
NTLM authentication is a 2-party authentication: The client and the server. It takes 3 steps:

1. Negotiate
2. Challenge
3. Response

First, the client sends a negotiate packet to the server to request the authentication. There are different parameters and versions for NTLM, and the client has to inform the server what it is capable of. This is done with a negotiate packet. Next, (step 2) the server sends a challenge packet to the client. This challenge includes a so-called "nonce". A nonce is a random number of 16 bytes. Finally (step 3), the client sends the response to the server: It calculates a response based on its password and the nonce and sends that to the server.

Using a nonce allows the 2 parties to perform authentication without having to send the password (cleartext or encrypted) over the network.

The server checks the credentials of the client by performing the same calculation as the client for the response, and if the response calculated by the server is the same as the response calculated by the client, then the client is authenticated to the server. The server needs the credentials of the client to perform the calculation; in an Active Directory environment, the server obtains the credentials from a domain controller.



Quick Recap: Responder (1)
So how do I get a victim system to connect to my machine? Enter "Responder"! Responder is a "penetration tester's best friend"! It's been around for a few years and has proven to be a highly effective tool.

Responder is a tool initially developed by SpiderLabs (Laurent Gaffie). It is mainly focused on attacking NTLM authentication. Responder attempts to lure victims to authenticate, after which it collects NTLM(v2) challenge responses that can be reused in various attacks.

So how does it reach its goal? Responder relies on two main protocols to lure victims to connect (and authenticate):

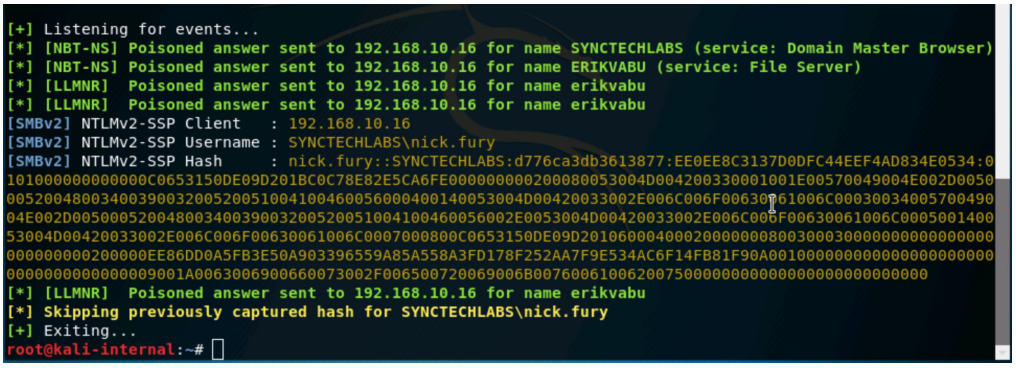
- NBT-NS (NetBIOS Name Server)
- LLMNR (Link-Local Multicast Name Resolution) is the successor to NBT-NS (since Windows Vista)

Both protocols allow hosts on the same subnet to resolve hostnames by sending requests to the multicast address (think an ARP-like feature for hostname resolution). What could possibly go wrong? You guessed it: Much like ARP spoofing, someone unexpected will respond to the name resolution requests. ☺

The latest version of Responder can be found here: https://github.com/lgandx/Responder.

QUICK RECAP: RESPONDER (2)

In the screenshot below, we can see Responder fulfilling its bidding: It's capturing an NTLM(v2) challenge response hash from a system in the network that tried to connect to a system called "erikvabu".



Quick Recap: Responder (2)

In the screenshot, we can see Responder fulfilling its bidding: It's capturing an NTLM(v2) challenge response hash from a system. We can deduce the following:

- The victim IP address is 192.168.10.16
- The victim username is SYNCTECHLABS\nick.fury
- Responder tricked the victim to connect by "poisoning" the resolution for the hostname "erikvabu".

QUICK RECAP: RESPONDER (3)

Responder attack success rates are dropping. First, NTLMv1 hashes, which are easier to brute force, are less prevalent. However, obtaining hashes using Responder is also more difficult than before. Companies are starting to disable those broadcast resolution protocols and even if they are enabled, often there will be end point defenses that prevent the connect-back after a spoofed reply.

The good news is that there are other, creative ways to obtain hashes. A selection includes:

- The use of office documents that try to load network resources
- SCF files on file shares (requires world-writable share or previous authenticated access)
- IPv6-based attacks

These first two techniques are also referred to as "Forced Authentication", which corresponds to MITRE ATT&CK Technique T1187.

Quick Recap: Responder (3)
Responder attack success rates are dropping. First, NTLMv1 hashes, which are easier to brute force, are less prevalent. However, obtaining hashes using Responder is also more difficult than before. Companies are starting to disable those broadcast resolution protocols and even if they are enabled, often there will be end point defenses that prevent the connect-back after a spoofed reply.

The good news is that there are other, creative ways to obtain hashes. A selection includes:

- The use of office document that try to load network resources
- SCF files on file shares (require world-writable share or previous authenticated access)
- IPv6-based attacks

These first two techniques are also referred to as "Forced Authentication", which corresponds to MITRE ATT&CK Technique T1187.

CAPTURING NTLM CHALLENGE / RESPONSES USING OFFICE

Microsoft has historically had some interesting issues with Single Sign On and "background authentication"…

In the Tweet to the left, we can see the developer of Responder reporting an interesting attack strategy:

- Open a new MS Word document
- Import a picture from a remote location (from an attacker-controlled system that has Responder listening)
- Mark the field as "not stored in the document"
- Whenever the document is opened, Word will attempt to fetch the image and, thus, perform background authentication to the remote system

Who needs VBA / Macro? ☺

SOURCE: https://twitter.com/pythonresponder/status/1161338972955697152

Capturing NTLM Challenge / Responses Using Office

Microsoft has historically had some interesting issues with Single Sign On and "background authentication". These attacks typically include the loading of a remote resource (e.g., an image, a script,…) from an attacker-controlled machine. As Microsoft's software attempts to fetch the resource, it performs SSO and thus sends credentials!

@PythonResponder wrote an interesting tweet in August 2019, where he described a very basic mechanism to have Microsoft Office Word fetch a remote image, thereby performing NTLM SSO. The attack chain works as follows:

1. Open a new MS Word document
2. Import a picture from a remote location (from an attacker-controlled system that has Responder listening)
3. Mark the field as "not stored in the document"
4. Whenever the document is opened, Word will attempt to fetch the image and thus perform background authentication to the remote system

CAPTURING NTLM CHALLENGE / RESPONSES USING OFFICE: STEP 1

```
python ./responder.py -I ALL -rPvF
```

LGANDX/
RESPONDER

SOURCE: https://twitter.com/pythonresponder/status/1161338972955697152

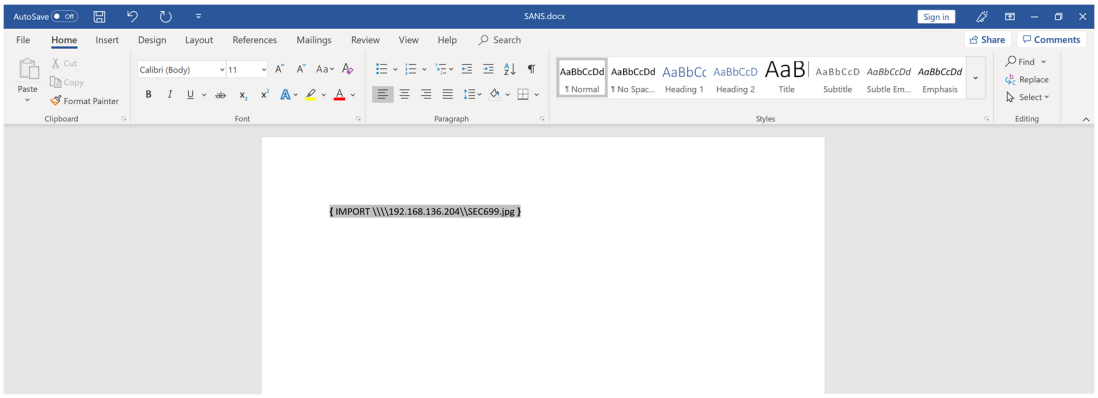
Capturing NTLM Challenge / Responses Using Office: Step 1
Let's quickly run through this attack chain. As a first step, run Responder to capture hashes!
In the example on the slide, we are using the following command-line syntax:

python ./responder.py –I ALL –rPvF

In short, this will launch Responder on all interfaces.

CAPTURING NTLM CHALLENGE / RESPONSES USING OFFICE: STEP 2

- Use "`Ctrl + F9`" and "`IMPORT`" from a network share.



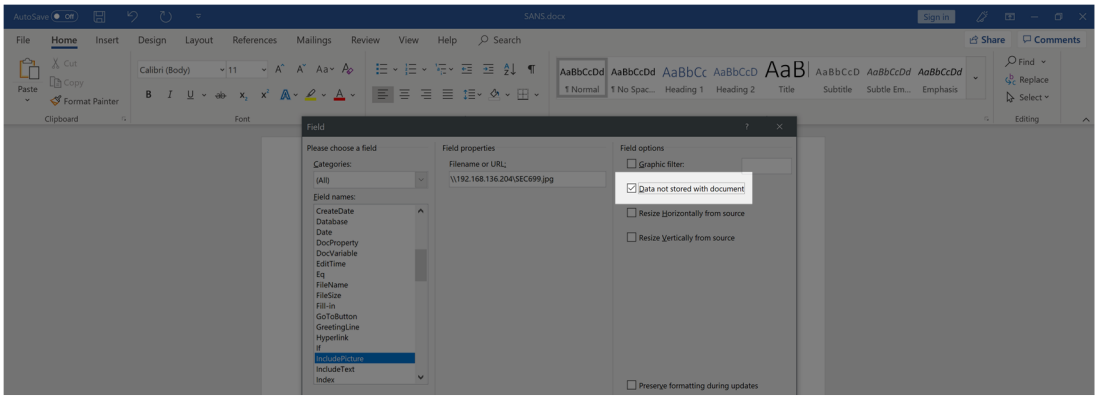
SOURCE: https://twitter.com/pythonresponder/status/1161338972955697152

Capturing NTLM Challenge / Responses Using Office: Step 2

In this specific attack scenario, we will not even rely on the multi-cast name resolution, as we will immediately force the Office application to load remote resources from the system that is running Responder: In the second step of the attack, we will now create an office document that attempts to fetch a remote resource. In the example on the slide, we are trying to load "SEC699.jpg" from a remote system 192.168.136.204.

CAPTURING NTLM CHALLENGE / RESPONSES USING OFFICE: STEP 3

From "**`Right Click > Edit Field…`**" enable "**`Data not stored with document`**" and save.



SOURCE: https://twitter.com/pythonresponder/status/1161338972955697152

Capturing NTLM Challenge / Responses Using Office: Step 3

Finally, the newly created field we made needs to be configured to not store the remote resource with the document. If we forget to enable this box, the image will be stored with the Word document and will thus not be fetched when the document is opened!

CAPTURING NTLM CHALLENGE / RESPONSES USING OFFICE: STEP 4

Once opened, the Word document will attempt to authenticate to the faked network share.

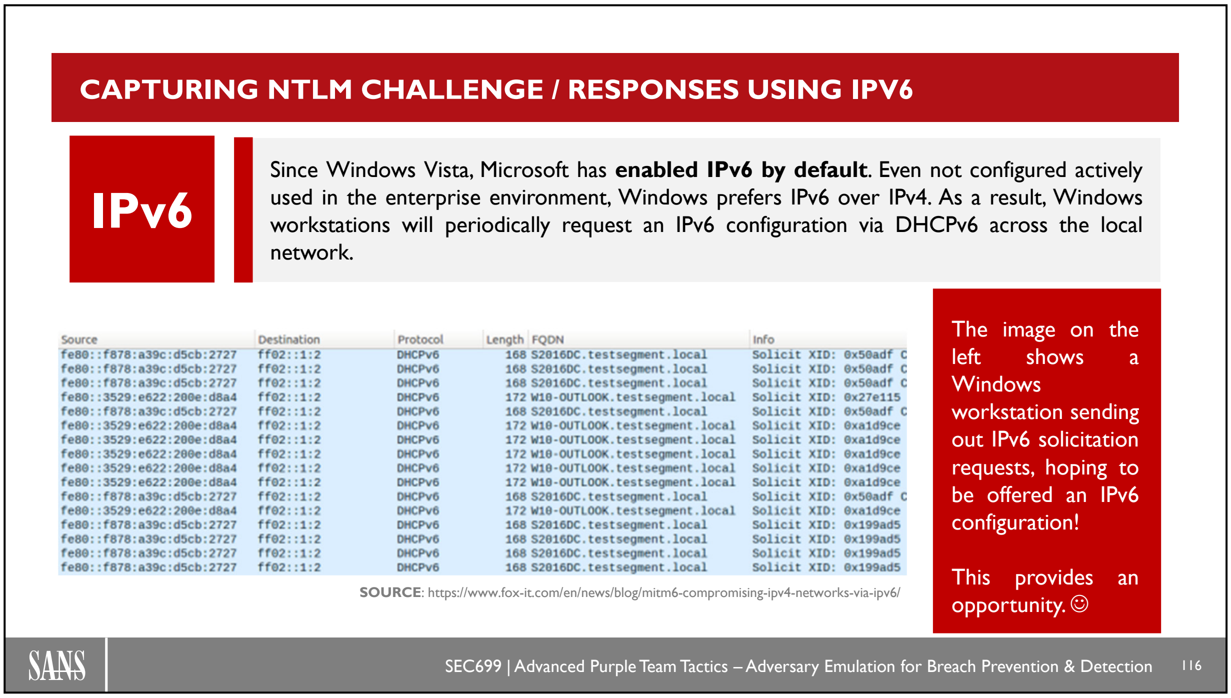
SOURCE: https://twitter.com/pythonresponder/status/1161338972955697152

Capturing NTLM Challenge / Responses Using Office: Step 4
Finally, in the above screenshot, we can see incoming NTLMv2 hashes when the Office document is opened and Word attempts to open the remote resource!

Should you be wondering what the NTLMv2 format looks like, please find below how the NTLMv2 response is calculated:

SC = 8-byte server challenge, random
CC = 8-byte client challenge, random
CC* = (X, time, CC2, domain name)
v2-Hash = HMAC-MD5(NT-Hash, username, domain name)
LMv2 = HMAC-MD5(v2-Hash, SC, CC)
NTv2 = HMAC-MD5(v2-Hash, SC, CC*)
response = LMv2 | CC | NTv2 | CC*

An excellent comparison between LM, NTLM, NTLMv1, and NTLMv2 can be found here:
https://medium.com/@petergombos/lm-ntlm-net-ntlmv2-oh-my-a9b235c58ed4



Capturing NTLM Challenge / Responses Using IPV6

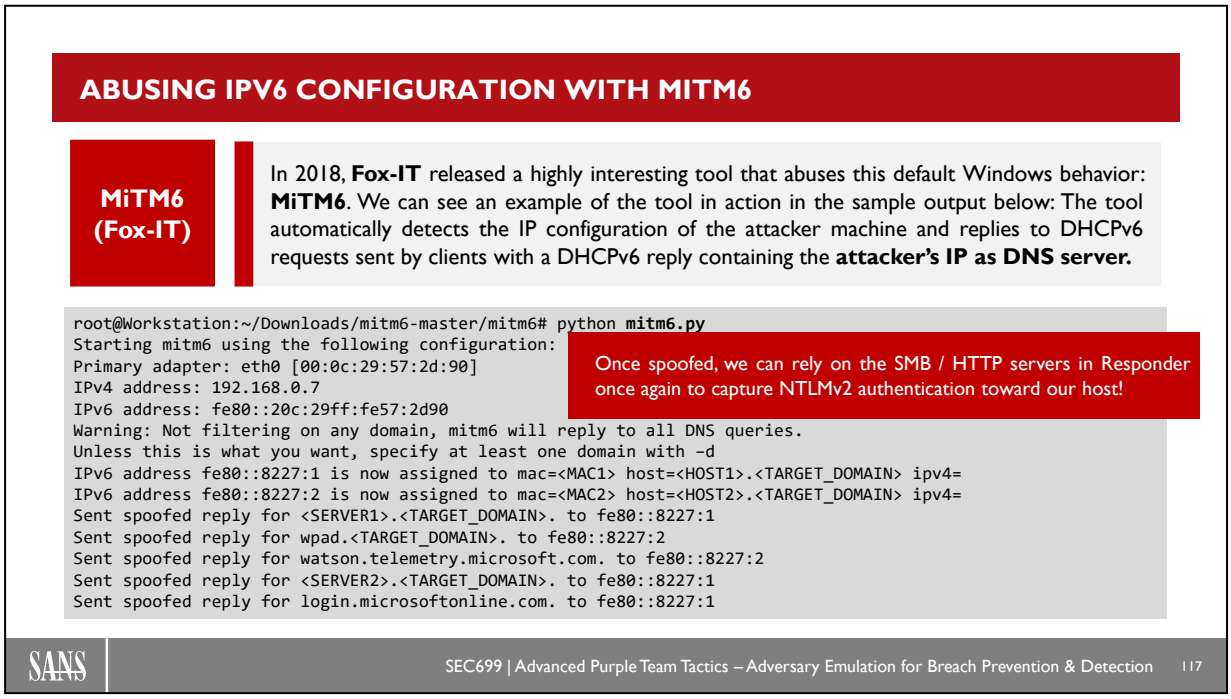
Since Windows Vista, Microsoft has enabled IPv6 by default. Even not configured actively used in the enterprise environment, Windows prefers IPv6 over IPv4. As a result, Windows workstations will periodically request an IPv6 configuration via DHCPv6 across the local network.

The image on the slide shows a Windows workstation sending out IPv6 solicitation requests, hoping to be offered an IPv6 configuration! Note that in an IPv6 network that was properly configured, these addresses are auto assigned by the hosts themselves and are thus not configured by a central DHCP server.

This provides an opportunity:

By listening and replying to the above DHCPv6 requests, an adversary could assign target systems an IPv6 address within the link-local range.

Such an IPv6 address configuration would include an IPv6 DNS server address, handily provided by the attacker!



Abusing IPv6 Configuration with MiTM6
To easily abuse this default behavior of Windows when it comes to IPv6, we can make use of MiTM6. MiTM6 was released in 2018 by Fox-IT.

This tool will listen on the local network and reply to DHCPv6 messages. It will provide a victim with a link-local IPv6 address and configure the attacker's IP as the victim's DNS server. Once in this position, the victim will send DNS requests to the attacker, which get picked up by MiTM6. Depending on the config, it can reply to all requests or a selection of specific domains. As a result, the victim's traffic to those domains will be sent to/via the attacker.

The attached snippet shows a sample output of MiTM6 in an actual corporate network. The tool automatically detects the IP configuration of the attacker machine and replies to DHCPv6 requests sent by clients in the network with a DHCPv6 reply containing the attacker's IP as DNS server. This allows the sending of spoofed replies to victims who want to resolve hostnames.

In this case, MiTM6 was run without parameters, thereby targeting every system and domain in the network. Several filtering options are available to select which hosts you want to attack and spoof. First there are the --host-whitelist and --host-blacklist options (or -hw and -hb for short), which take a (partial) domain as argument. Incoming DHCPv6 requests will be filtered against this list. The property checked is the DHCPv6 FQND option, in which the client provides its hostname. The same applies for DNS requests, for this the --domain option (or -d) is available, where you can supply which domain(s) you want to spoof. Blacklisting is also possible.

You can find MiTM6 here: https://github.com/fox-it/mitm6

IPV6, NTLMRELAYX, AND LDAP(S) (1)

Once we have spoofed a victim using mitm6, we can now attempt to obtain an NTLMv2 hash to crack. Alternatively, we could also consider setting up a relay, using for example **ntlmrelayx.py**. Next to the classic SMB relay, we could also leverage an LDAP relay!

In the screenshot to the right, we are leveraging our ntlmrelay to create a computer account.

By default, any user in Active Directory can create up to 10 computer accounts.

A computer account will provide a credential for further enumeration!

```
dirkjan@ubuntu:~/impacket-py3$ ntlmrelayx.py -t ldaps://icorp-dc.internal.corp --add-computer
Impacket v0.9.19-dev - Copyright 2019 SecureAuth Corporation

[*] Servers started, waiting for connections
[*] Setting up HTTP Server
[*] HTTPD: Received connection from 192.168.111.73, attacking target ldaps://icorp-dc.internal.corp
[*] HTTPD: Client requested path: /
[*] HTTPD: Received connection from 192.168.111.73, attacking target ldaps://icorp-dc.internal.corp
[*] HTTPD: Client requested path: /
[*] HTTPD: Client requested path: /
[*] Authenticating against ldaps://icorp-dc.internal.corp as ICORP\ICORP-W10$ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large domains
[*] Attempting to create computer in: CN=Computers,DC=internal,DC=corp
[*] Adding new computer with username: GTIJUZEC$ and password: H-(oB>w59"h4Zy} result: OK
```

SOURCE: https://dirkjanm.io/worst-of-both-worlds-ntlm-relaying-and-kerberos-delegation/

We relay to LDAP over TLS (instead of plaintext LDAP) because creating accounts is not allowed over an unencrypted connection!

IPv6, Ntlmrelayx, and LDAP(S) (1)
Once we have spoofed a victim using MiTM6, we can now attempt to obtain an NTLMv2 hash to crack. Alternatively, we could also consider setting up a relay, using for example ntlmrelayx.py.
Next to the classic SMB relay, we could also leverage an LDAP relay!

In the screenshot to the right, we are leveraging our ntlmrelay to create a computer account. By default, any user in Active Directory can create up to 10 computer accounts. A computer account will provide a credential for further enumeration! A minor, yet important, detail: We relay to LDAP over TLS (instead of plaintext LDAP) because creating accounts is not allowed over an unencrypted connection!

A full attack chain is described by Dirk-Jan Mollema here:
https://dirkjanm.io/worst-of-both-worlds-ntlm-relaying-and-kerberos-delegation/

IPV6, NTLMRELAYX, AND LDAP(S) (2)

Through this **relay attack**, we were able to create a new computer account on the domain, for which we know the password as well. We have thus obtained a foothold in the domain and we can use this account for reconnaissance (e.g., query domain info or run BloodHound) or other attacks, such as Kerberoasting!

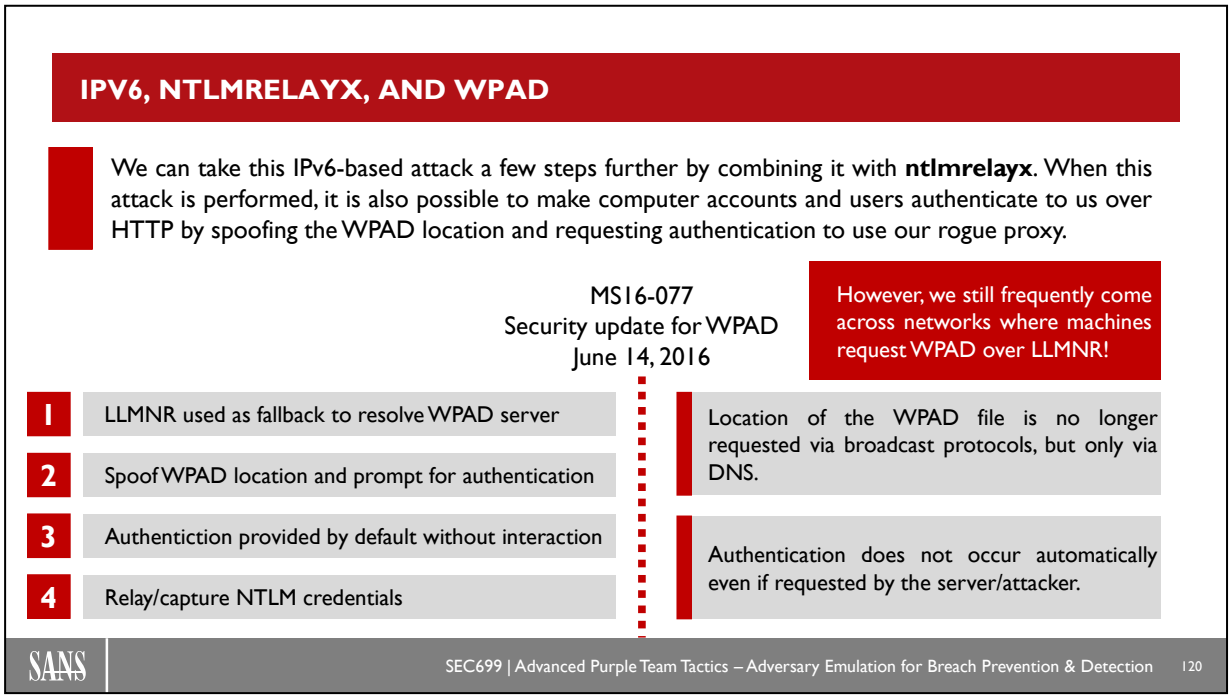
The example below shows BloodHound execution using the newly created computer account.

```
dirkjan@ubuntu:~/BloodHound.py$ python bloodhound.py -d internal.corp -u GTIJUZEC\$ -p 'H-(oB>w59"h4Zy}'
INFO: Found AD domain: internal.corp
INFO: Connecting to LDAP server: ICORP-DC.internal.corp
INFO: Found 1 domains
INFO: Found 1 domains in the forest
INFO: Found 5 computers
INFO: Connecting to LDAP server: ICORP-DC.internal.corp
INFO: Found 29 users
INFO: Found 68 groups
```

SOURCE: https://dirkjanm.io/worst-of-both-worlds-ntlm-relaying-and-kerberos-delegation/

IPv6, Ntlmrelayx, and LDAP(S) (2)
Through this relay attack, we were able to create a new computer account on the domain, for which we know the password as well. We have thus obtained a foothold in the domain and we can use this account for reconnaissance (e.g., query domain info or run BloodHound) or other attacks, such as Kerberoasting!

The example shows BloodHound execution using the newly created computer account.



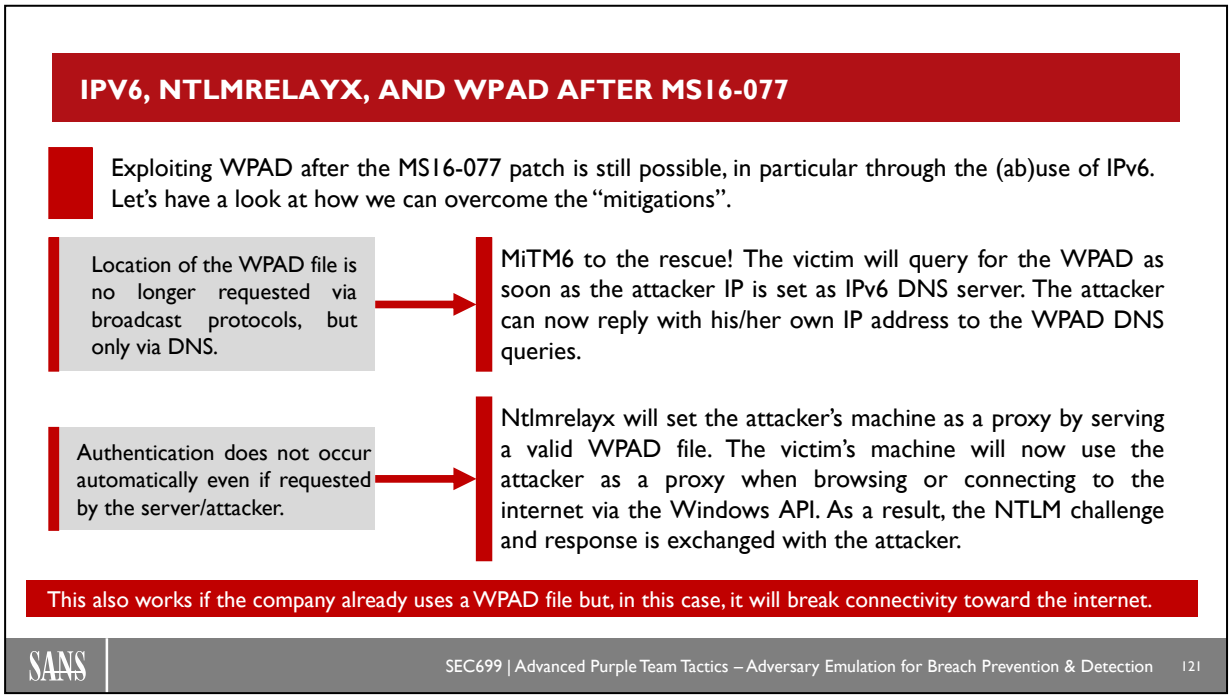
IPv6, Ntlmrelayx, and WPAD
We can take this IPv6-based attack a few steps further by combining it with ntlmrelayx.

Ntlmrelayx is another tool in the Impacket arsenal and performs NTLM relay attacks, setting an SMB and HTTP Server and relaying credentials to many different protocols (SMB, HTTP, MSSQL, LDAP, IMAP, POP3, etc.). By combining MiTM6 and ntlmrelayx, it is possible to make computer accounts and users authenticate to us over HTTP by spoofing the WPAD location and requesting authentication to use our rogue proxy. WPAD spoofing is another of ntlmrelayx' functionalities.

WPAD's legitimate use case is to automatically detect a network proxy used for connecting to the internet in corporate environments. Historically, the address of the server providing the wpad.dat file (which provides this information) would be resolved using DNS. If there was no DNS entry available, the address would be resolved via fallback broadcast protocols such as Link-Local Multicast Name Resolution (LLMNR). This could be abused by listening on the local network, for example using Responder, and replying to these broadcast requests. As a result, the victim would believe the WPAD to be located on the attacker's server and try to access it, only to be prompted for authentication by the attacker. This would be provided by default, without user interaction, resulting in an NTLM handshake with the attacker machine. The resulting authentication could then be relayed to another server or the Net-NTLM hash could be retrieved for offline cracking.

This attack was mitigated in 2016 when Microsoft published security bulletin MS16-077, adding two limitations:

- Broadcast protocols, such as LLMNR, are no longer used to retrieve the WPAD location. Instead, only DNS is used.
- Authentication no longer happens automatically, even when requested by the server/attacker.



IPv6, Ntlmrelayx, and WPAD after MS16-077
Exploiting WPAD after the MS16-077 patch is still possible, in particular through the (ab)use of IPv6. Let's have a look at how we can overcome the "mitigations".

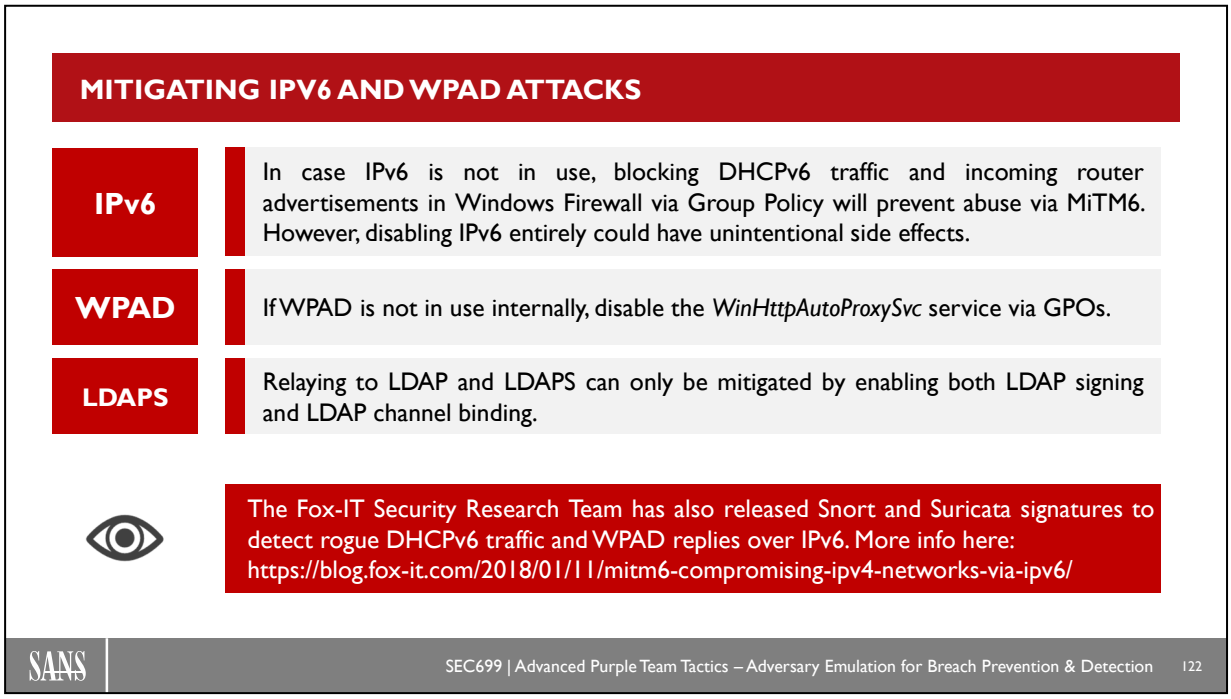
The first restriction, which limits WPAD requests to DNS and avoids broadcast protocols, can be bypassed with MiTM6. The victim will query for the WPAD as soon as the attacker IP is set as IPv6 DNS server. The attacker can now reply with his/her own IP address to the WPAD DNS queries. This method still works in case the company already uses a WPAD file. However, performing the attack will break internet connectivity.

The restriction, which prevents credentials from being provided by default, requires a workaround. After positioning ourself as the DNS server, we can provide a WPAD file to the victim when they request one. In that file, we will specify our attacking machine as a proxy. The victim's machine will now use the attacker host as a proxy when browsing or connecting to the internet via the Windows API. This works in Edge, Internet Explorer, Firefox, and Chrome, since they all respect the WPAD system settings by default.

When the victim connects to our host, i.e., the proxy, we would see the CONNECT HTTP verb being used or a full URI after the GET verb. In response, we can send an HTTP 407 Proxy Authentication required to entice the victim into providing authentication. IE, Edge, and Chrome (which makes use of IE's settings as well) will authenticate to the proxy as a result. In Firefox, automatic authentication setting can be configured, but it is enabled by default.

Windows will now provide the NTLM challenge/response to the attacker, who can relay it to different services or obtain the hash for offline cracking. With this relaying attack, the attacker can authenticate as the victim on services, access information on websites and shares, and if the victim has enough privileges, the attacker can even execute code on computers or even take over the entire Windows Domain.

More information can be found in this interesting blog post:
https://blog.fox-it.com/2018/01/11/mitm6-compromising-ipv4-networks-via-ipv6/



Mitigating IPv6 and WPAD Attacks

The combined IPv6 attack chain described in the previous slides can be mitigated at multiple stages:

Breaking MiTM6 attack strategy

MiTM6 abuses Windows' preference of IPv6 over IPv4, even in case IPv6 is not actively in use on the coprorate network. In case IPv6 is not in use, blocking DHCPv6 traffic and incoming router advertisements in Windows Firewall via Group Policy will prevent abuse via MiTM6. However, disabling IPv6 entirely could have unintentional side effects. Setting the following predefined rules to Block instead of Allow prevents the attack from working:

- (Inbound) Core Networking - Dynamic Host Configuration Protocol for IPv6(DHCPV6-In)
- (Inbound) Core Networking - Router Advertisement (ICMPv6-In)
- (Outbound) Core Networking - Dynamic Host Configuration Protocol for IPv6(DHCPV6-Out)

Mitigating WPAD abuse

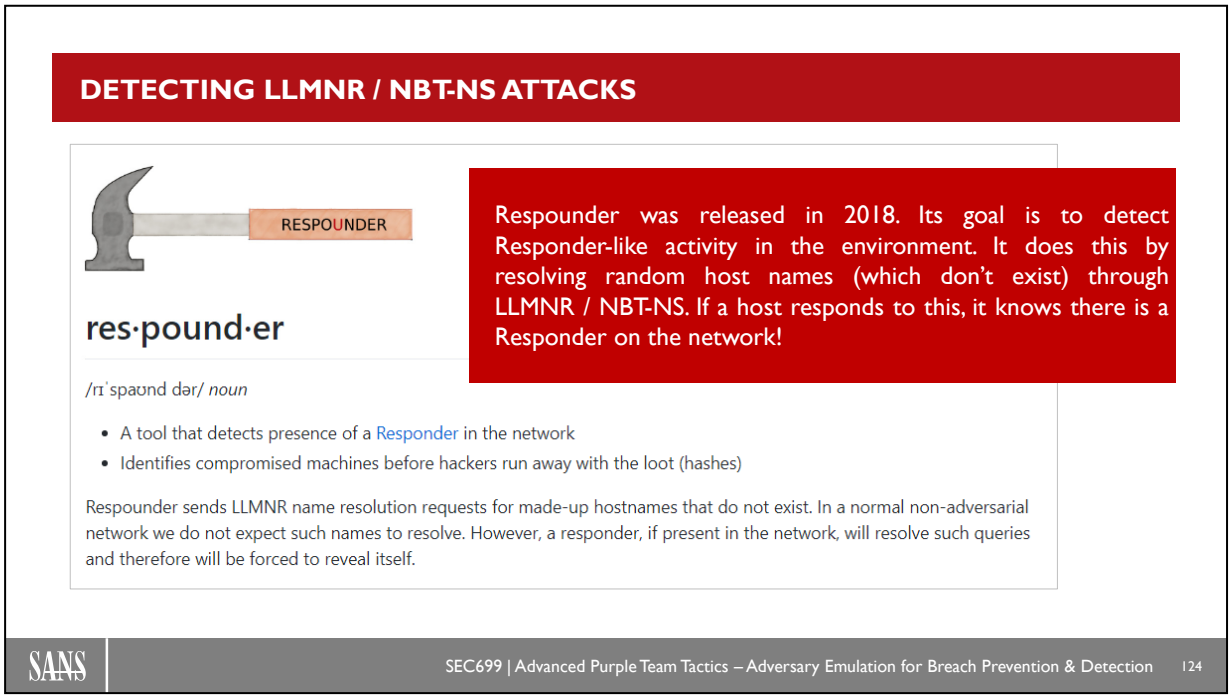
If WPAD is not in use internally, disable the *WinHttpAutoProxySvc* service via GPOs.

If your company uses a proxy configuration file internally (PAC file) it is recommended to explicitly configure the PAC URL instead of relying on WPAD to detect it automatically.

Mitigating LDAP relay

Relaying to LDAP and LDAPS can only be mitigated by enabling both LDAP signing and LDAP channel binding.

Channel binding is a technique where the transport layer and the application layer are “bound” together. It is binding the TLS tunnel and the LDAP application layer together so that the LDAP messages cannot be intercepted and inserted into a different tunnel or TLS channel. Since NTLM relaying involves intercepting the communications and forwarding it somewhere else, this requires starting different TLS tunnels. While TLS already protects against message tampering, and it already provides encryption, the LDAP application layer can provide an integrity mechanism (signing) that defends against tampering. Neither TLS nor LDAP had a way to protect against credential relaying, until the introduction of channel binding.



Detecting LLMNR / NBT-NS Attacks
Respounder was released in 2018. Its goal is to detect Responder-like activity in the environment. It does this by resolving random host names (which don't exist) through LLMNR / NBT-NS. If a host responds to this, it knows there is a Responder on the network! It can be found here: https://github.com/codeexpress/respounder

An alternative tool is Responder-Guard, which takes a similar approach; it can be found here: https://github.com/CredDefense/CredDefense/blob/master/scripts/ResponderGuard.ps1

SUMMARIZING PREVENTION / DETECTION

Security Control	Implementation Ease?	Effectiveness?	Comment?
Disable LLMNR / NBT-NS	Medium	Medium	Would only mitigate Responder-style attacks
Disable NTLMv2	Hard	High	Will require extensive testing
Block Outbound SMB on the Perimeter	Easy	High	Would not stop internal credential stealing attacks
Enable SMB Signing	Hard	Medium	Would stop relay attacks, but not hash stealing
Disable IPv6	Medium	Medium	Would only stop IPv6-based attacks
Disable Auto Proxy Service	Easy	Medium	Would only stop WPAD attacks, but not hash stealing
Enable LDAP Signing & Channel Binding	Medium	Medium	Would only stop LDAP-based attacks, but not hash stealing

Detection Logic	Logs required?	False positive ratio?	Comment?
Respounder (or similar tools)	None	Very Low	Only detects tools that abuse LLMNR / NBT-NS
Review Network Traffic	Network traffic	Very Low	Detect rogue DHCPv6 traffic and WPAD replies over IPv6

Summarizing Prevention / Detection
In order to stop the attack techniques explained, we can consider any of the below security controls:

- Disable NTLMv2
- Block Outbound SMB on the perimeter
- Enable SMB Signing
- Disable IPv6
- Disable Auto Proxy service
- Enable LDAP signing and channel binding

Detection-wise, most of the detection can be done by reviewing network traffic for anomalies.
For LLMNR / NBT-NS style attacks, we can consider using a tool such as Respounder!

Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.

EXERCISE: CREATIVE NTLMV2 CHALLENGE-RESPONSE STEALING

Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

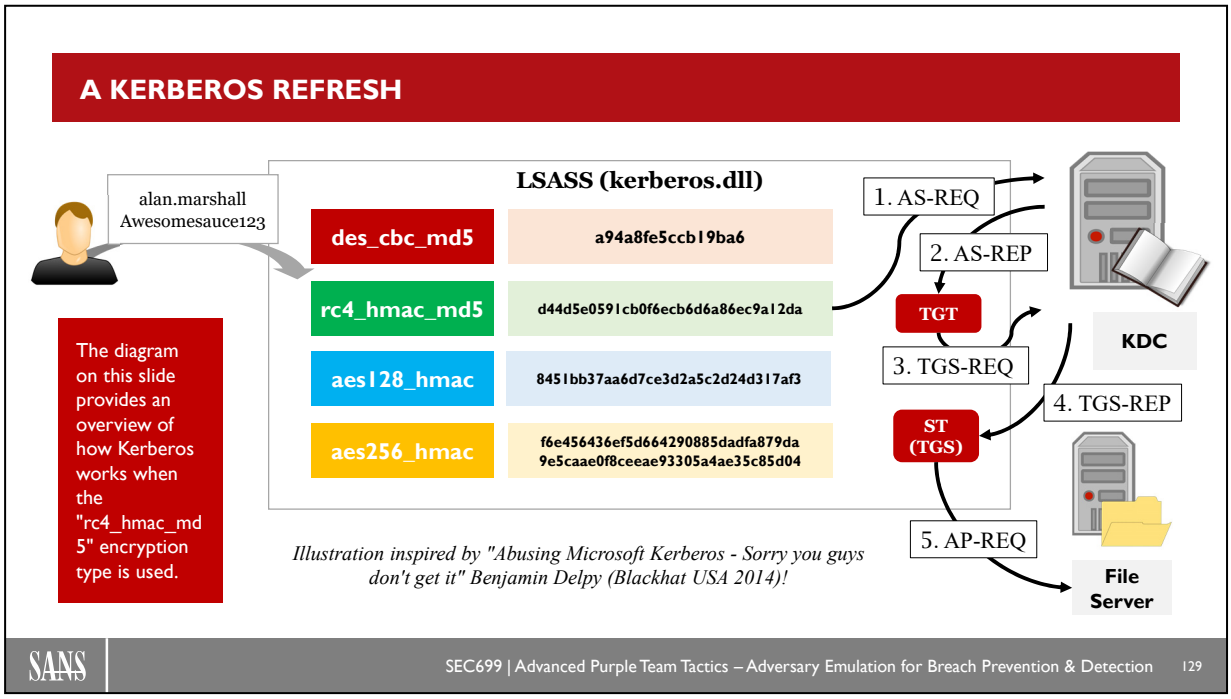
Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.



A Kerberos Refresh

We already briefly discussed how Kerberos works, but let's look at it a bit more… The diagram on the slide above illustrates how Kerberos works using the rc4_hmac_md5 encryption type. At first, the user of course provides his / her password, after which a number of keys are stored in LSASS. Tickets are requested in the following way:

1. AS-REQ – As an initial step, the client performs pre-authentication by sending an AS-REQ to the KDC (in this case, the AS component of the KDC (Key Distribution Center)). This AS-REQ includes an encrypted version of the timestamp (encrypted using the Kerberos encryption key of the client account), which is validated by the KDC;
2. AS-REP – If authentication succeeds (timestamp can be decrypted), the KDC sends two items to the client:
    - The Ticket Granting Ticket (TGT), which includes the Client / TGS session key and is encrypted using the KDC long-term key (in case of rc4_hmac_md5, the krbtgt NT password hash). This KDC long-term key is also used to sign the PAC (which is also part of the TGT and includes information on who the user is and what groups he is a member of);
    - The Client / TGS session key, encrypted using the client long-term key (in case of rc4_hmac_md5, the user password hash);
3. TGS-REQ – Thirdly, the user wants to authenticate to a certain service and thus sends the following to the KDC (in this case, the TGS component of the KDC):
    - An authenticator message (encrypted using the Client / TGS session key)
    - The encrypted TGT and a ticket request (referencing a certain Service Principal Name, SPN).

4. TGS-REP – The KDC will validate whether the service exists and create a service ticket (ST) that is sent back. It's important to note that the KDC does not do any validation of privileges (it leaves that to the target service). The service ticket has two parts:
   A client portion, which is encrypted using the Client / TGS session key (so this can be decrypted by the client);
   A server portion, which is encrypted using the target long-term key (in case of rc4_hmac_md5, the password hash of the target service). This server portion also includes the PAC of the user. It is the service ticket that the user will subsequently submit to the service he or she is trying to access.
5. Finally, the client can now use the server portion of the ticket to try to access the desired service (e.g., a file server). The target server will try to decrypt the server portion of the service ticket using the target long-term key (in case of rc4_hmac_md5, the password hash of the target service). It will use the PAC information included in the Service Ticket to validate the privileges of the user.

Please note that the illustration used in this slide was based on an amazing presentation by Benjamin Delpy at Blackhat 2014: "Abusing Microsoft Kerberos – Sorry you guys don't get it."

A KERBEROS REFRESH: ENCRYPTION TYPES

TGT ST

So, what encryption type does Kerberos use? By default, the system will use the highest method of encryption that is supported by the client! For Microsoft-based systems, as of Windows Vista, all Microsoft machines support AES encryption types!

Encryption	Key Length	OS Supported?	Notes
AES256-CTS-HMAC-SHA1-96	256-bit	As of Windows 7 and Server 2008 R2	Uses 4096 rounds of PBKDF2 Uses domain & username as salt
AES128-CTS-HMAC-SHA1-96	128-bit	As of Windows Vista, Windows Server 2008	Uses 4096 rounds of PBKDF2 Uses domain & username as salt
RC4-HMAC	128-bit	As of Windows 2000	RC4 key = NT hash of account password
DES-CBC-MD5	56-bit	As of Windows 2000, disabled as of Windows 7 / Server 2008 R2	Not used as default
DES-CBC-CRC	56-bit	As of Windows 2000, disabled as of Windows 7 / Server 2008 R2	Not used as default

All Kerberos encryption keys are based on the password of the associated account!

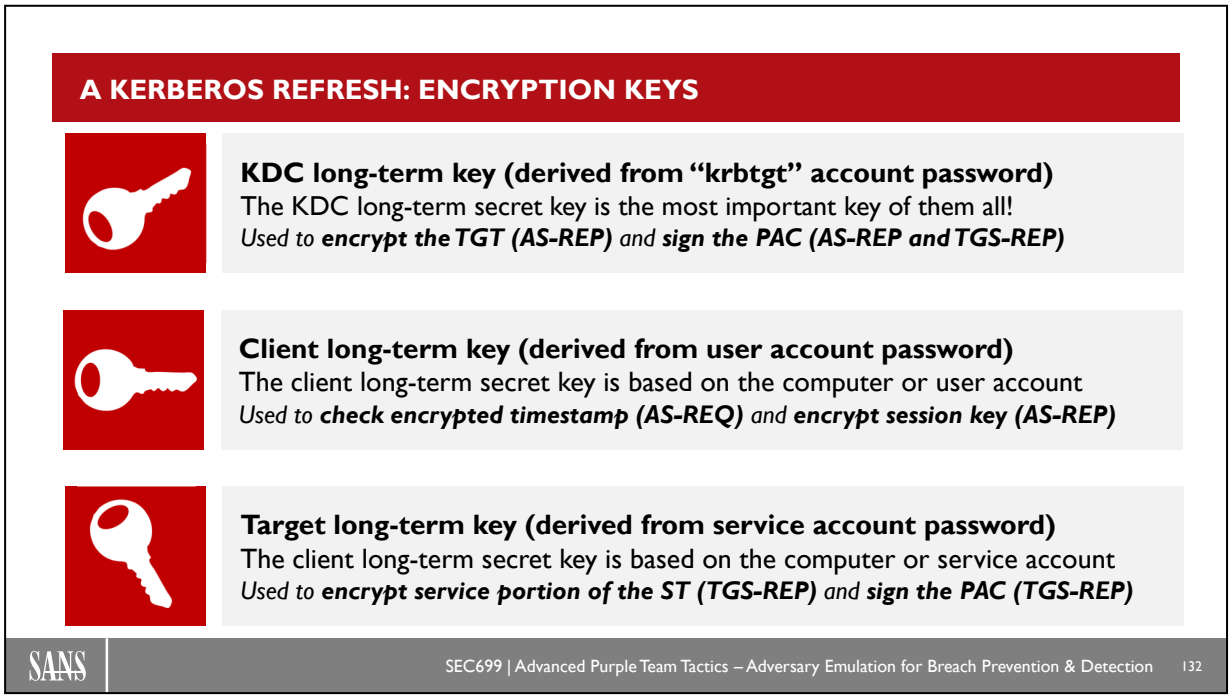
A Kerberos Refresh: Encryption Types

So, what encryption type does Kerberos use? By default, the system will use the highest method of encryption that is supported by the client! For Microsoft-based systems, as of Windows Vista, all Microsoft machines support AES encryption types! The table above shows the different encryption keys used by various versions of Microsoft Windows. The main encryption types that are relevant to us are AES and RC4.

A quick overview:

- AES-based encryption types: Available as of Windows Vista (128-bit) and Windows 7 (256-bit). In this mode, the Kerberos keys are obtained by using 4096 rounds of PBKDF2 (and the domain and username are used as a salt).
- RC4 encryption: Available as of Windows 2000. In this mode, the Kerberos keys are the NT hashes of the account password.
- DES-encryption types: These were all disabled as of Windows 7 and are thus less relevant to us.

It's important to note that all Kerberos encryption keys are based on the password of the associated account!

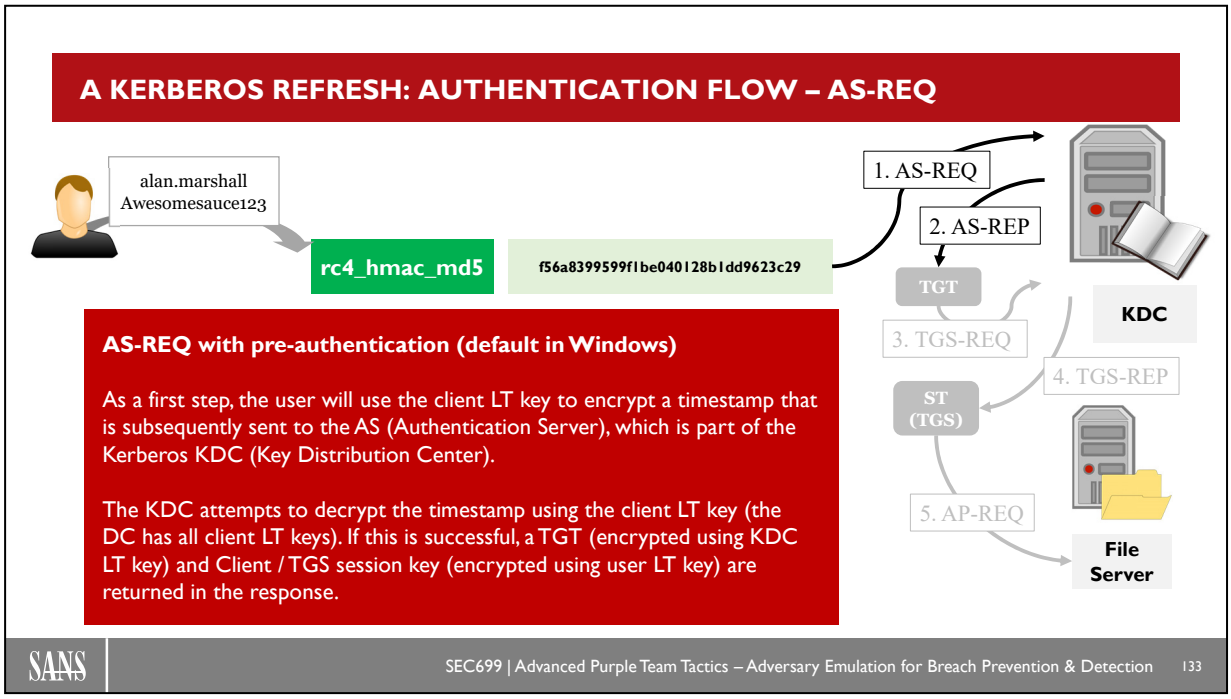


A Kerberos Refresh: Encryption Keys

There are three main security keys are used throughout the protocol:

- The KDC long-term secret key, often called "domain key", which is derived from the krbtgt service account. It is used in Kerberos to encrypt the TGT and sign the PAC.
- The client long-term secret key, which is based on the password of the client account. It is used in Kerberos to check the encrypted timestamp (AS-REQ) and encrypt the session key.
- The target long-term secret key, which is based on the password of the target service. It is used in Kerberos to encrypt the TGS and sign the PAC.

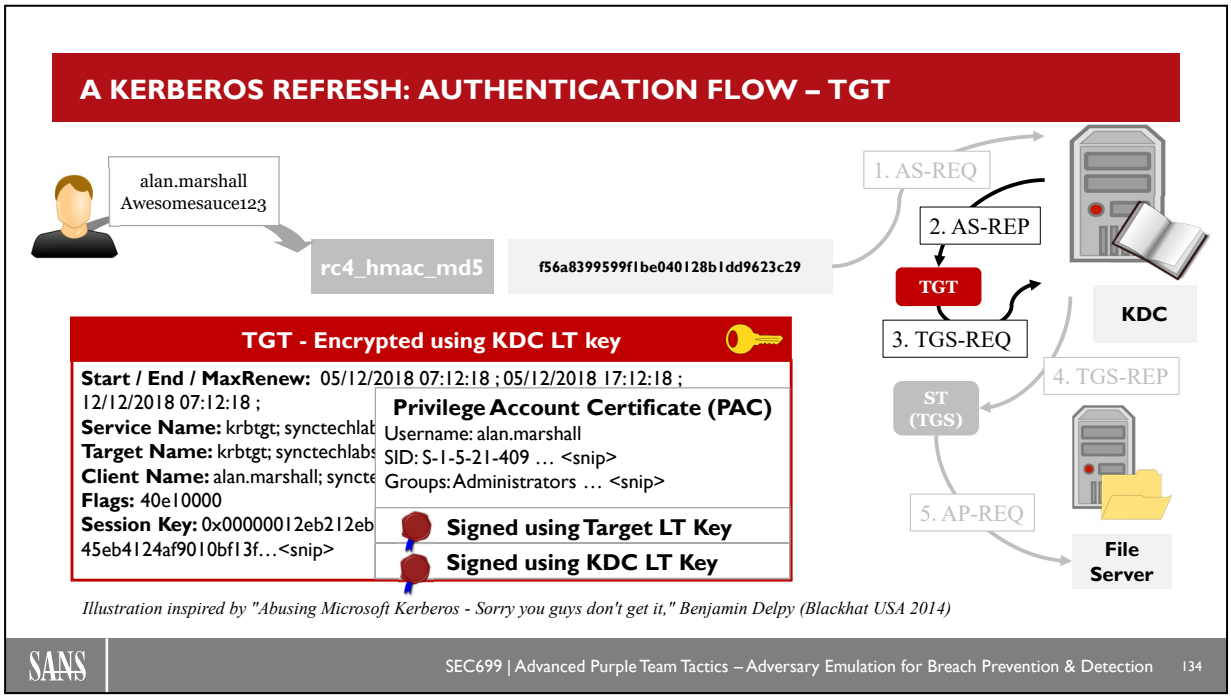
It should be clear that compromise of any key represents a security risk. There is, however, one key that is more important than the others: If the KDC long-term secret key is compromised, adversaries can recreate TGTs and sign PACs, which would allow them to obtain all privileges within the domain.



A Kerberos Refresh: Authentication Flow – AQ-REQ
Let's have a look at the initial step here AS-REQ!

1. AS-REQ – As an initial step, the client performs pre-authentication by sending an AS-REQ to the KDC (in this case, the AS component of the KDC). This AS-REQ includes an encrypted version of the timestamp (encrypted using the password hash of the client account), which is validated by the KDC. This "encrypted timestamp" is referred to as "pre-authentication" and is enabled (required) by default in Microsoft Kerberos environments.
2. AS-REP – If pre-authentication succeeds (timestamp can be decrypted), the KDC sends two items to the client:
   - The Ticket Granting Ticket (TGT), which includes the Client / TGS session key and is encrypted using the KDC long-term key (in case of rc4_hmac_md5, the krbtgt NT password hash). This KDC long-term key is also used to sign the PAC (which is also part of the TGT and includes information on who the user is and what groups he is a member of);
   - The Client / TGS session key, encrypted using the client long-term key (in case of rc4_hmac_md5, the user password hash);

It's interesting to note that when pre-authentication would not be enabled, the KDC will provide an AS-REP response without validating the encrypted timestamp. While this is not a "critical" issue that would compromise security of the Kerberos protocol (both response messages would still be encrypted with a key unavailable to the adversary), it would allow adversaries to launch brute-force attacks against the part of the response that is encrypted using the client long-term key (in case of rc4_hmac_md5, the user password hash).



A Kerberos Refresh: Authentication Flow – TGT

The TGT is the first ticket received by the client. What does it look like? The TGT includes the following information:

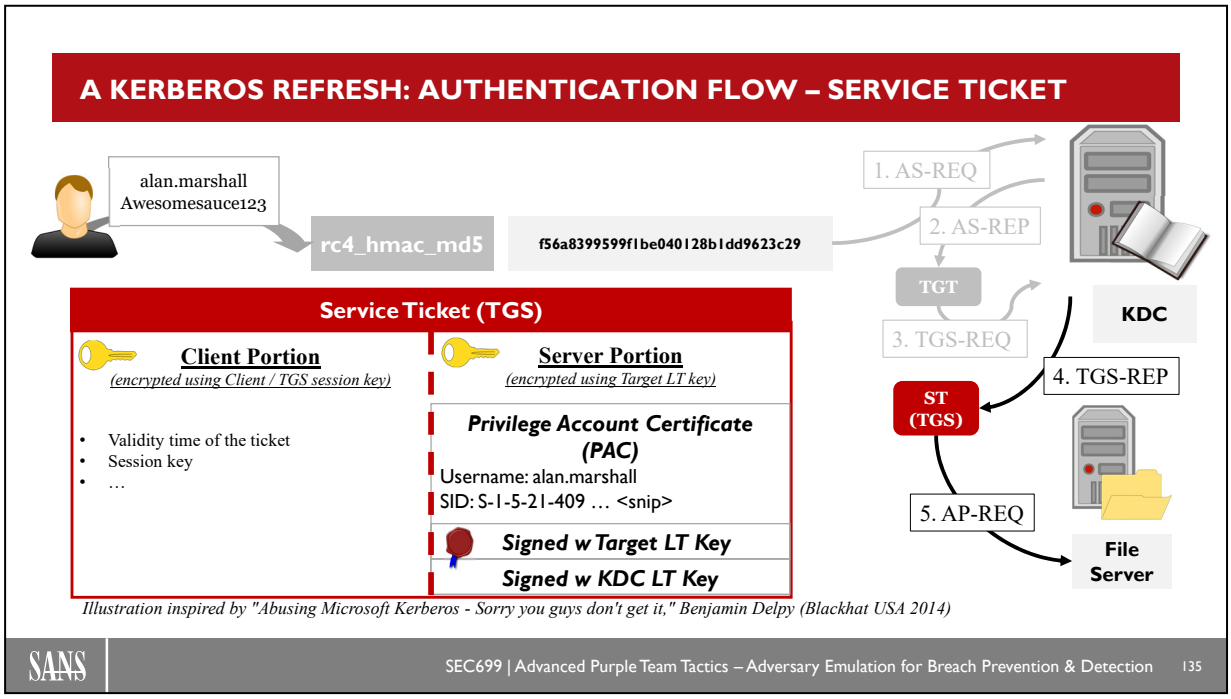
- Name: The user's name the ticket is associated with
- Start time and End time: Marks the validity period of the ticket. By default, in Windows AD environments, this would be 10 hours
- Authorization-data: Authorization data details the user's privileges and access rights. In Windows, the authorization data takes the form of a Privilege Attribute Certificate (PAC) object.
- The Client / TGS session key that can be used for future communications between the client and the TGS

It should be clear that the PAC is extremely sensitive and should under no circumstance be tampered with. In order to protect its integrity, it is signed with two keys:

- It is signed using the Target Long Term Key. Because this is a TGT, the target is the krbtgt account (so in rc4_hmac_md5, this would be the krbtgt NT hash)
- It is also signed using the KDC LT Key. As indicated many times before, in rc4_hmac_md5, this would be the krbtgt NT hash

Finally, to prevent the entire TGT from being tampered with, it is encrypted using the KDC long-term key (which is the krbtgt NT hash when rc4_hmac_md5 is used). The TGT details cannot be read by the client, but that's fine, as it only needs to send it back to the KDC for future validation (e.g., when a Service Ticket is requested). In the TGS-REQ, the user wants to authenticate to a certain service and thus sends the following to the KDC (in this case, the TGS component of the KDC):

- An authenticator message (encrypted using the Client / TGS session key)
- The encrypted TGT and a ticket request (referencing a certain Service Principal Name, SPN)

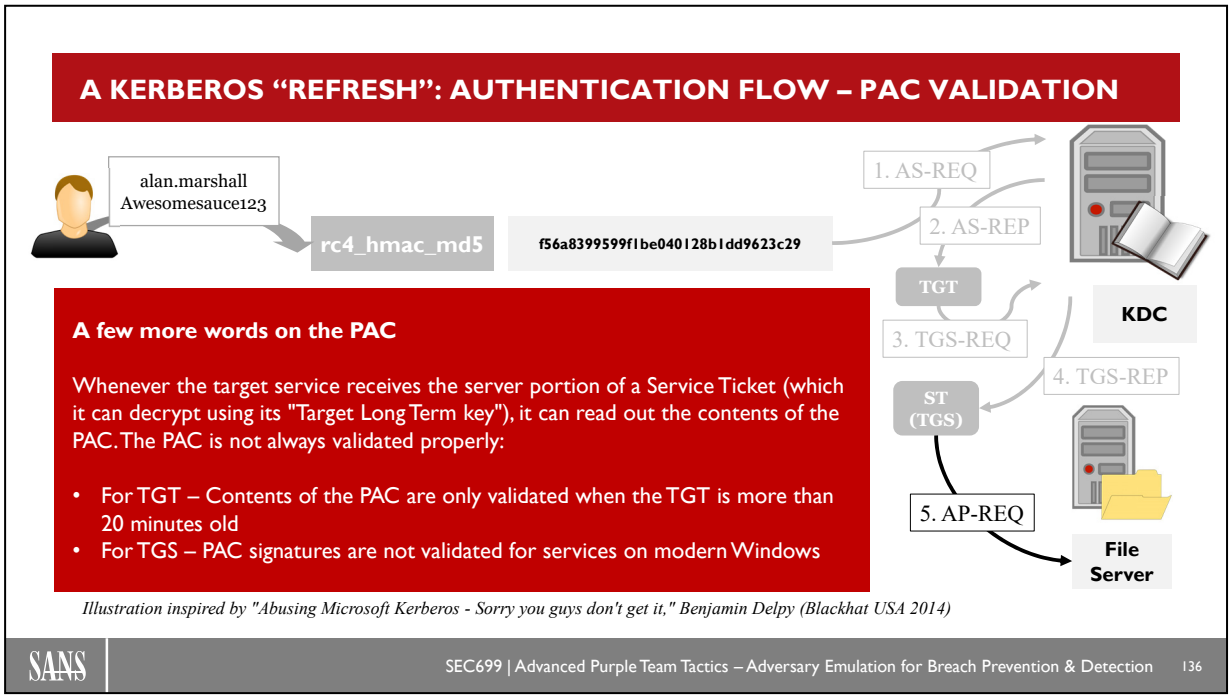


A Kerberos Refresh: Authentication Flow – Service Ticket
If the KDC receives a TGS-REQ with a valid TGT, it will send back a response, TGS-REP. The KDC will validate whether the service to which the client requests access exists and subsequently creates a service ticket (ST) that is sent back. It's important to note that the KDC does not do any validation of privileges (the target service can do that by itself, by reviewing the PAC that is included in the service ticket; see below).

The service ticket has two parts:

- A client portion, which is encrypted using the Client / TGS session key (so this can be decrypted by the client);
- A server portion, which is encrypted using the target long-term key (in case of rc4_hmac_md5, the password hash of the target service). This server portion also includes the PAC of the user. It is this service ticket that the user will subsequently submit to the service he or she is trying to access.

Illustration inspired by "Abusing Microsoft Kerberos - Sorry you guys don't get it," Benjamin Delpy (Blackhat USA 2014).



A Kerberos "Refresh": Authentication Flow – PAC Validation

Whenever the target service receives the server portion of a Service Ticket (which it can decrypt using its "Target Long Term key"), it can read out the contents of the PAC. The target service will validate the signature that was created using the Target Long Term key, but will not always validate the signature that was created using the KDC Long Term Key!

This is from Microsoft's MS-KILE specification:

"Kerberos V5 does not provide account revocation checking for TGS requests, which allows TGT renewals and service tickets to be issued as long as the TGT is valid even if the account has been revoked. KILE provides a check account policy (section 3.3.5.7.1) that limits the exposure to a shorter time. KILE KDCs in the account domain are required to check accounts when the TGT is older than 20 minutes. This limits the period that a client can get a ticket with a revoked account while limiting the performance cost for AD queries."

What does this practically mean for us?

- For a TGT: If the TGT is more than 20 minutes old, the PAC contents are validated. This thus opens a 20-minute window where the contents are not validated. This means that, if you have the KDC long-term hash (NTLM hash of the krbtgt account when using rc4_hmac_md5), you can even create tickets with bogus account information (e.g., user "Invisible" as a member of the "Domain Admins" group), which will be valid for 20 minutes. Within these 20 minutes, you could request multiple Service Tickets that are valid for 10 hours.
- For a Service Ticket: Modern Windows systems (as of Windows Vista) do not validate the PAC by default for services. Windows still validates the PAC for processes that are not running as services. PAC validation can be enabled when the application server is not running in the context of local system, network service, or local service.

We will discuss some possible attack vectors soon!

WELL-KNOWN KERBEROS ATTACKS (1)

Attack	What?	Prerequisite	Prevention?	Detection?
Kerberoast	Request service tickets for accounts with an SPN. Brute-force "server portion" of service ticket to find NT hash of service account.	Valid domain user	Disable Kerberos RC4 (could break legacy systems) Strong passwords for service accounts (Managed Service Accounts)	One user requesting many service tickets (Windows event id 4769) Usually leverages RC4
ASReqroast	Sniff AS-REQ requests, which include a timestamp encrypted using NT hash of user. Brute-force timestamp.	Access to network traffic (e.g. MiTM)	Disable Kerberos RC4 (could break legacy systems) Ensure strong passwords are used Add users to "protected users" groups	Usually leverages RC4, so review unexpected RC4 activity (Kerberos events on the DC, ATA)
ASReproast	Request AS-REP responses when Kerberos pre-authentication is disabled. The AS-REP will be encrypted using the NT hash of the user, which can be brute-forced.	Valid domain user	Disable Kerberos RC4 (could break legacy systems) Ensure strong passwords are used. Enforce Kerberos pre-authentication (default)	Usually leverages RC4, so review unexpected RC4 activity (Kerberos events on the DC, ATA)

Well-Known Kerberos Attacks (1)

So, what are some of the known attack strategies against Kerberos?

Kerberoast

In Kerberoasting, a valid domain user account is used to request RC4 service tickets for accounts with an SPN (Service Principal Name). The "server portion" of the service ticket is subsequently brute-forced by:

- Guessing a possible password
- Generating the Kerberos key for said password
- Decrypting the service ticket with the generated Kerberos key
- Repeating if decryption was not successful

Kerberoasting can be defeated by implementing one of the following controls:

- Disabling Kerberos RC4 (could break legacy systems). Note that Kerberoasting also works with the AES encryption type, but it's slower.
- Configuring strong passwords for service accounts (e.g., using Managed Service Accounts)

Kerberoasting can be detected by:

- Reviewing Kerberos activity and looking for one user requesting many service tickets in a short timeframe (especially RC4 service tickets)

ASReqroast

In ASReqroast, an adversary sniffs AS-REQ RC4 requests, which include a timestamp encrypted using the NT hash of user, the timestamp is subsequently brute-forced. This would, however, require the adversary to have access to network traffic (e.g., when doing a MiTM attack).

ASReqroast can be defeated by implementing one of the following controls:

- Disable Kerberos RC4 (could break legacy systems). Note that ASReqroast also works with the AES encryption type, but it's slower.
- Ensure all accounts use a strong password.
- Add users to "protected users" groups.

ASReqroast can be detected by:

- Review unexpected RC4 activity (Kerberos events on the DC, ATA)

ASReproast

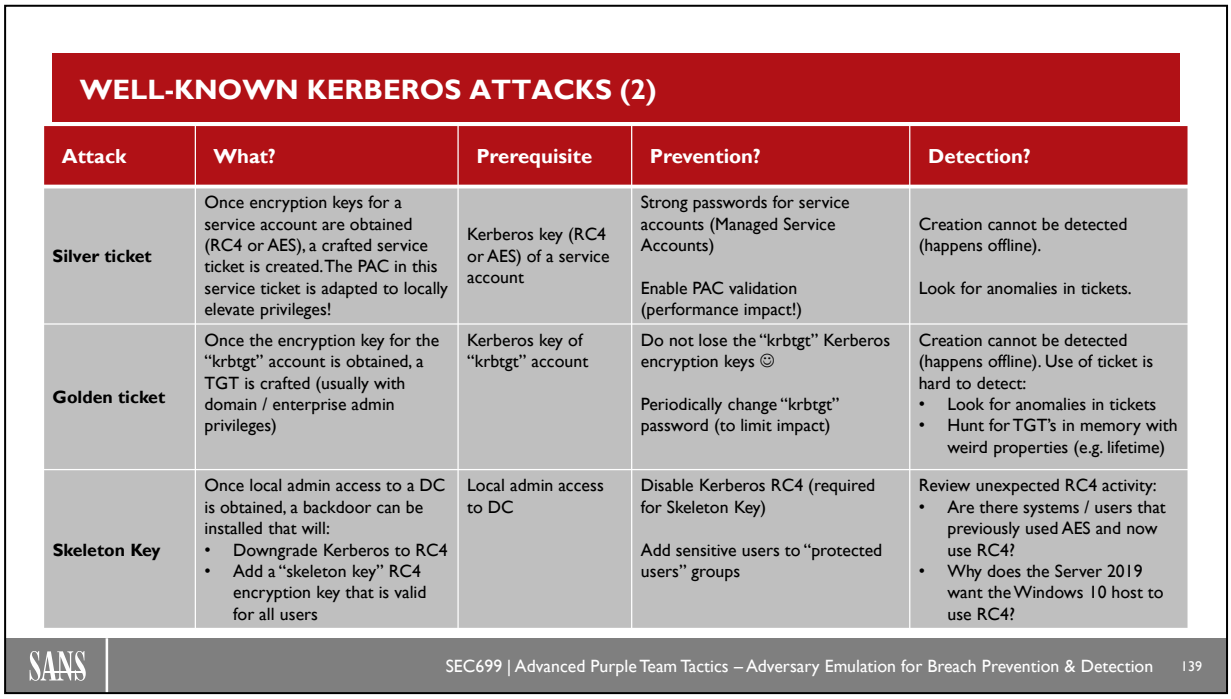
Similar to ASReqroast, but in ASReproast, the adversary sniffs AS-REP RC4 responses when Kerberos pre-authentication is disabled (pre-authentication is enabled by default in Microsoft). The AS-REP will be encrypted using the NT hash of the user, which can be brute-forced. As with ASReqroast, this will require the attacker to have network access to sniff the requests.

ASReproast can be defeated by implementing one of the following controls:

- Disable Kerberos RC4 (could break legacy systems). Note that ASReproast also works with the AES encryption type, but it's slower.
- Ensure all accounts use a strong password.
- Enable Kerberos pre-authentication for all accounts (default setting)

ASReproast can be detected by:

- Review unexpected RC4 activity (Kerberos events on the DC, ATA)



Well-Known Kerberos Attacks (2)

So, what are some of the known attack strategies against Kerberos?

Silver ticket

Once encryption keys for a service account are obtained (RC4 or AES), a crafted service ticket can be created. We call such a service ticket a "Silver ticket". The PAC in this Silver Ticket is adapted to locally elevate privileges!

Silver tickets can be defeated by implementing one of the following controls:

- Strong passwords for service accounts (Managed Service Accounts)
- Enabling PAC validation (which will have a performance impact)

Silver ticket detection is not straightforward. Creation happens offline and can thus not be observed. Use of crafted tickets could be observed, but this would rely on anomalies in the crafted tickets (e.g., Mimikatz used to leave a watermark in crafted tickets).

Golden ticket

Once the encryption key for the "krbtgt" account is obtained, a TGT can be crafted by the adversary (usually with domain / enterprise admin privileges).

Golden tickets can be defeated by implementing one of the following controls:

- Not losing the "krbtgt" Kerberos encryption keys ☺
- Periodically changing the "krbtgt" service password to invalidate previously crafted golden tickets

Golden ticket detection is not straightforward. Creation happens offline and can thus not be observed. Use of crafted tickets could be observed, but this would rely on anomalies in the crafted tickets (e.g., Mimikatz used to leave a watermark in crafted tickets).

Skeleton key

Once local admin access to a DC is obtained, a backdoor can be installed that will:

- Downgrade Kerberos encryption type to RC4
- Add a “skeleton key” RC4 encryption key that is valid for all users (this is effective as AES uses a salt and would thus have a different encryption keys for different users with the same password)

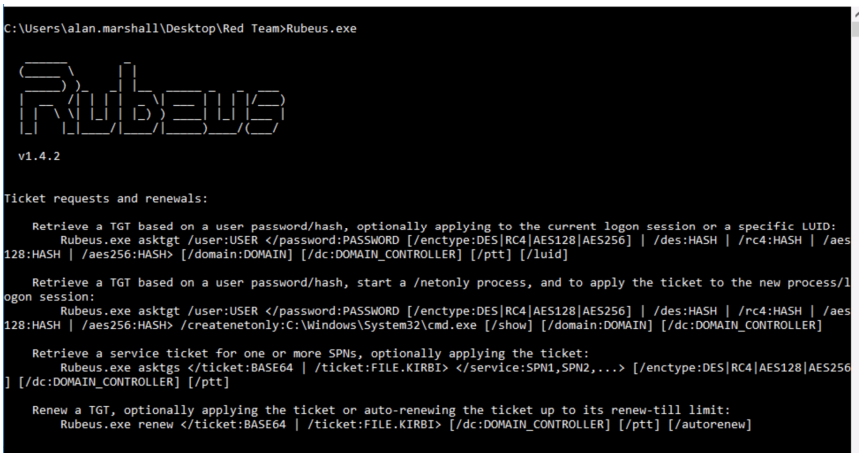
The skeleton key can be defeated by:

- Disabling the Kerberos RC4 encryption type
- Adding sensitive users to the “protected users” group

The skeleton key can be detected by reviewing unexpected RC4 activity:

- Are there systems / users that previously used AES and now use RC4?
- Why does the Server 2019 want the Windows 10 host to use RC4?

KERBEROS TOOLS: RUBEUS



SOURCE: https://github.com/ghostpack/rubeus

Rubeus is a C# toolset for raw Kerberos interaction developed by @harmj0y (Will Schroeder).

It is heavily adapted from Benjamin Delpy's Kekeo project (CC BY-NC-SA 4.0 license) and Vincent Le Toux's MakeMeEnterpriseAdmin project (GPL v3.0 license).

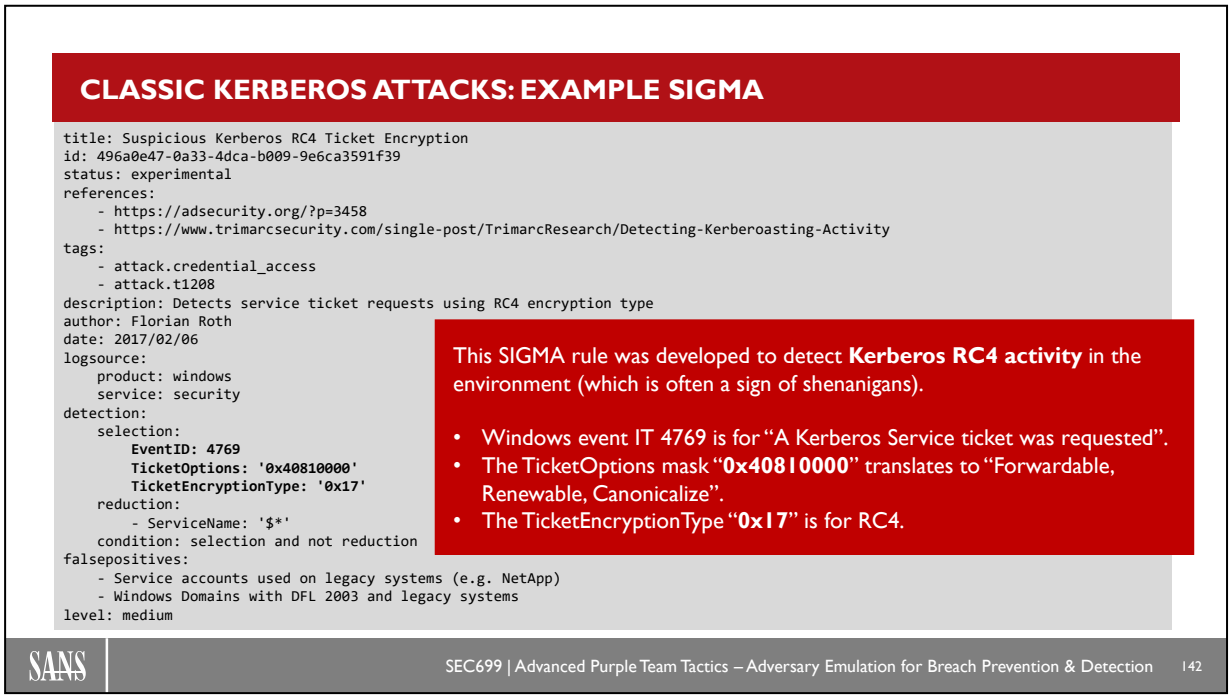
Kerberos Tools: Rubeus

Rubeus is a C# toolset for raw Kerberos interaction developed by @harmj0y (Will Schroeder). It is heavily adapted from Kekeo. Kekeo (CC BY-NC-SA 4.0 license) was developed by Benjamin Delpy (Mimikatz author) to manipulate / interact with Kerberos in C. It relies on a commercial library ("OSS ASN.1/C") to deal with Kerberos ASN.1 structure. It can be used to perform / test a variety of Kerberos attacks!

Among others, Rubeus can perform the following actions:

- General ticket extraction and harvesting
- Kerberoasting
- ASReproasting
- Constrained delegation attacks
- Calculate encryption keys from a user password

We will leverage several of the Rubeus features in our upcoming lab. The full documentation can be found at https://github.com/GhostPack/Rubeus.

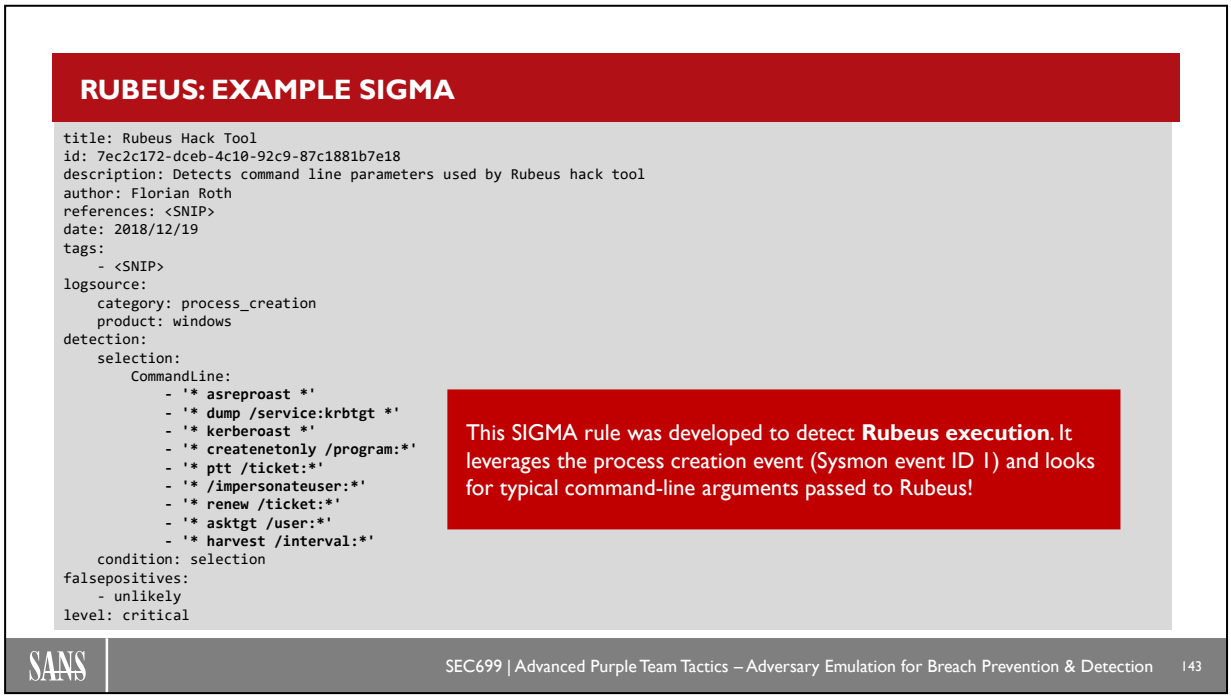


Classic Kerberos Attacks: Example SIGMA

This SIGMA rule was developed to detect RC4 shenanigans in the environment, which are often part of Kerberos attacks. It looks for event ID 4769 (A Kerberos Service Ticket was requested) and includes the following additional filters:

- The TicketOptions mask "0x40810000" translates to "Forwardable, Renewable, Canonicalize".
- The TicketEncryptionType "0x17" is for RC4.

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma



Rubeus: Example SIGMA

This SIGMA rule was developed to detect Rubeus execution. It leverages the process creation event (Sysmon event ID 1) and looks for typical command line arguments passed to Rubeus!

It looks for the following command-line arguments:

- * asreproast *
- *dump /service:krbtgt *
- * kerberoast *
- * createnetonly /program:*
- * ptt /ticket:*
- * /impersonateuser:*
- * renew /ticket:*
- * asktgt /user:*
- * harvest /interval:*

Please refer to the public SIGMA repository by Florian Roth for additional details:
https://github.com/Neo23x0/sigma

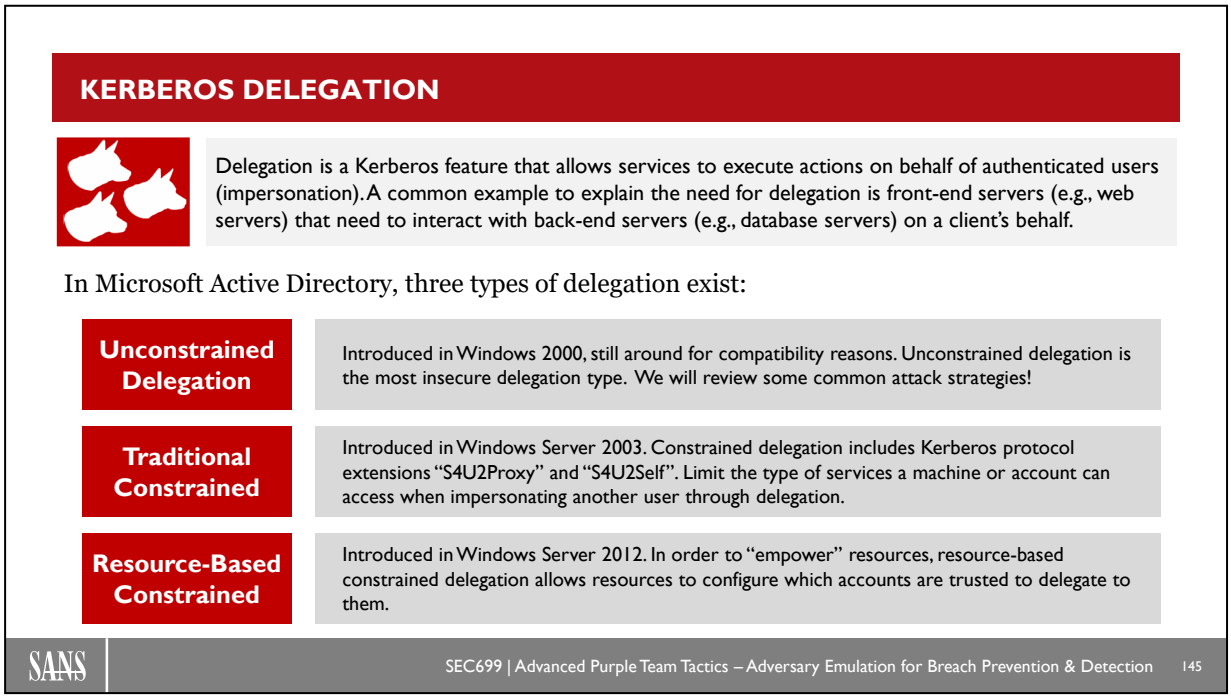
Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.



Kerberos Delegation

Delegation is a feature in Kerberos that allows services to execute actions on behalf of authenticated users ("impersonate). A common example to explain the need for delegation is front-end servers (e.g., web servers) that need to interact with back-end servers (e.g., database servers) on a client's behalf.

In Microsoft Active Directory, three types of delegation exist:

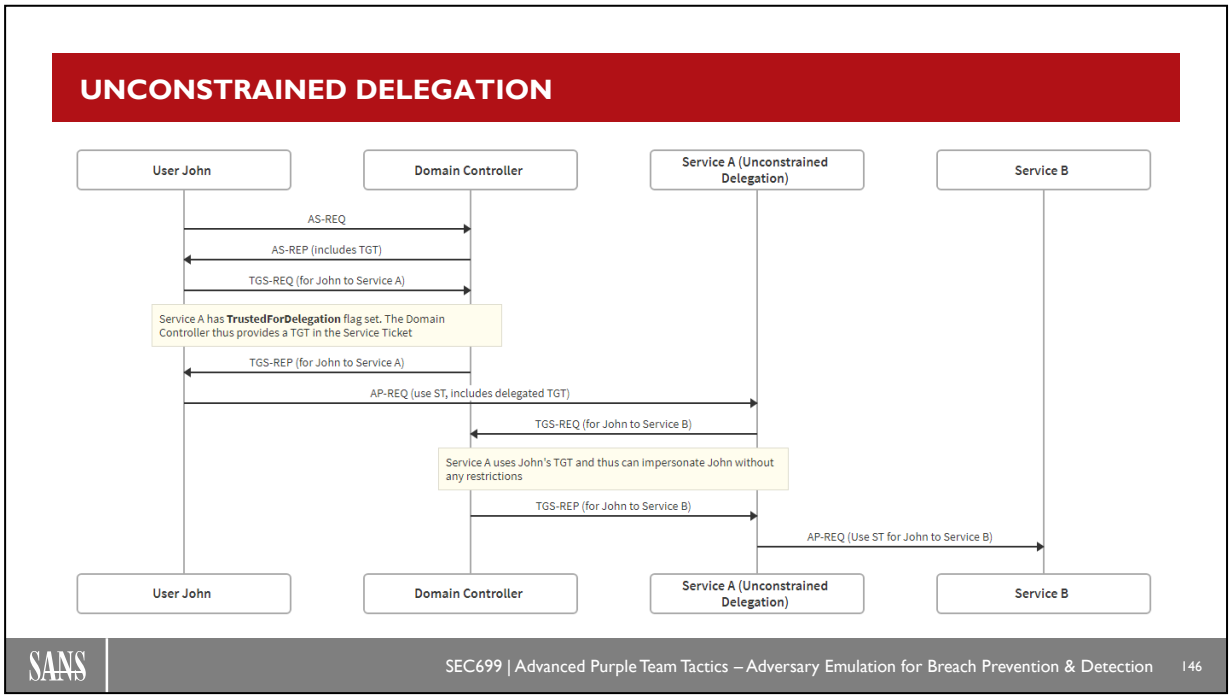
- Unconstrained delegation. Introduced in Windows 2000, still around for compatibility reasons. Unconstrained delegation is the most insecure delegation type. We will review some common attack strategies!
- Traditional constrained delegation: Introduced in Windows Server 2003. Constrained delegation includes Kerberos protocol extensions "S4U2Proxy" and "S4U2Self". Limit the type of services a machine or account can access when impersonating another user through delegation.
- Resource-based constrained delegation: Introduced in Windows Server 2012. In order to "empower" resources, resource-based constrained delegation allows resources to configure which accounts are trusted to delegate to them.

Delegation is a highly interesting concept, which is well-described in the following resources:

http://www.harmj0y.net/blog/activedirectory/s4u2pwnage/

https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html

https://shenaniganslabs.io/2019/08/08/Workshop-Slides.html



Unconstrained Delegation

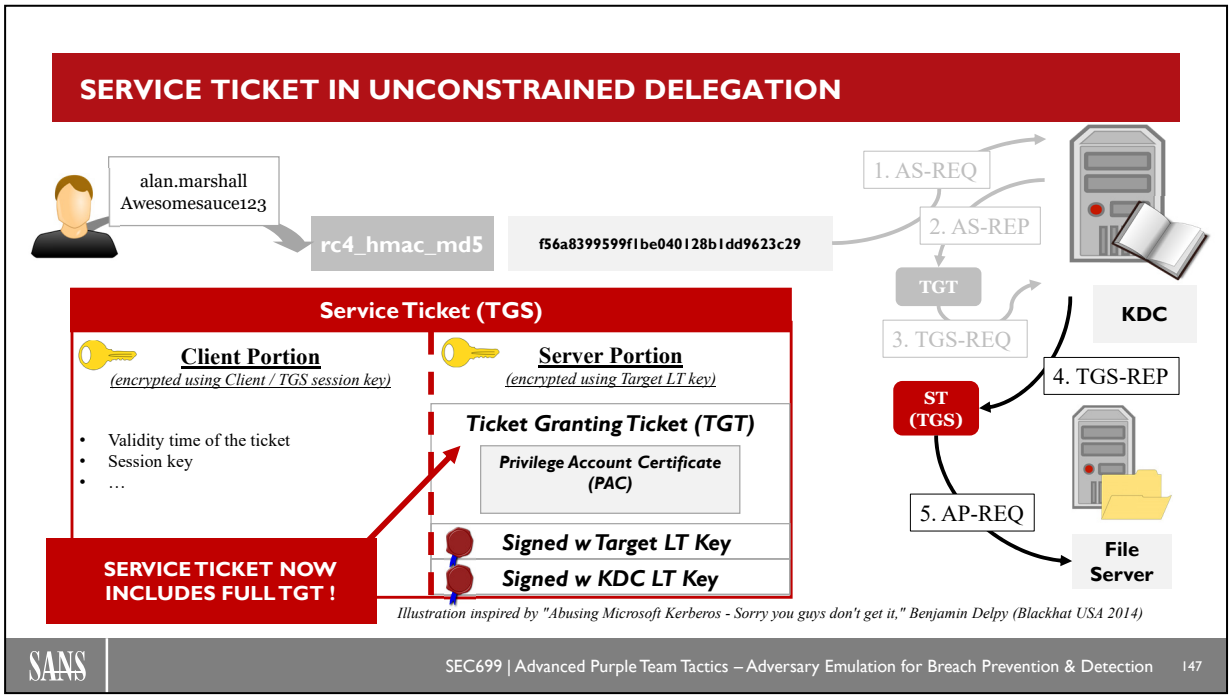
When Kerberos unconstrained delegation is used, the overall Kerberos steps are similar to the normal authentication flow; please find a detailed scheme on the slide. This was extracted from the official Microsoft documentation on the topic:

1. The user requests a TGT with pre-authentication to the KDC
2. If authentication is successful, the user receives a TGT from the KDC
3. The user now requests a "forwarded TGT", which can be used for delegation
4. A "forwarded" TGT is returned if the service has the TrustedForDelegation flag set
5. The forwarded TGT is provided to the service that will perform the impersonation (in this example, service A). This forwarded TGT is included in the service ticket which will be sent to service A. Service A will thus be able to extract the user TGT! Using the client's TGT, service A can now freely request additional service tickets to perform actions on the client's behalf.

We call this "Unconstrained Delegation"! Given the security implications, unconstrained delegation is not enabled by default and has to be explicitly enabled.

Microsoft's documentation on the topic can be found on https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-sfu/1fb9caca-449f-4183-8f7a-1a5fc7e7290a.

For author / instructor reference, the swimlanes diagram in the slide can be generated using the following URL: https://swimlanes.io/#lZLRToMwFIbveYpzpdCIiYtXxJiA20zMkiGwqJdknElj1872sM23t90MbGMx2R2Q//z9+h0KTgIjeFLSkC65xAqGKPCzJK4k+Pn9bJCPJmO4AveYZtP3j8Dzhmpps26KtBICNTyEkKNe8zlCHEHxnIfZ6PVgPoSF0tAYG31RtbQdnlSEUVEjtCHcEkrjDubGZasb2NSosauGlUaDkgxwujb2nAJIgcbvBg1B2QYLPv9CAn88zd7ibBgnk1FwTLB763p918FtOTBW6MYQVoWKG6rHSndCGANjay0gY+0sY8HtOSXhY19JCv4xhaVvQ5eK3W2jb/bIa6rV9ucisf5d0MktZQX+INg1208nev9T76B6nhMrypH1r7nhQsC6FLwqCR0cWT444LPgpRBqY39QeyxfrlAbJV341GMCPmNLU5nwb4BUtd8gkrIbXHAU1VnXl24sCbwO8GA4iSBO93uaGWzlFcEv

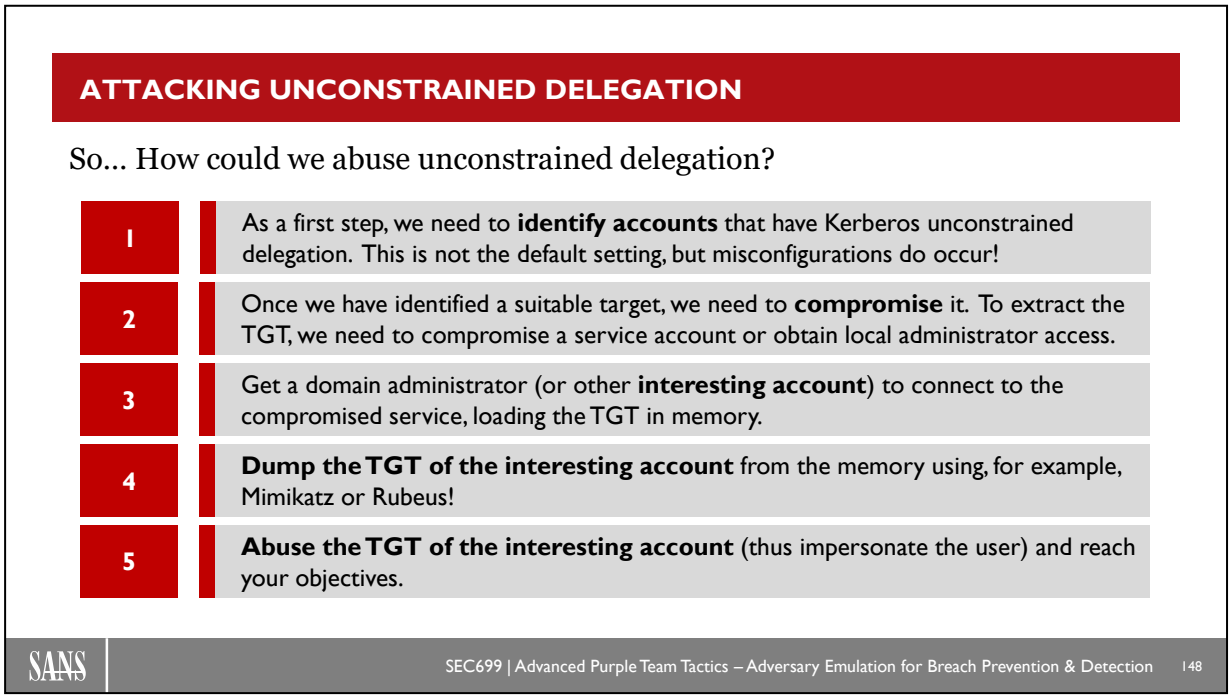


Service Ticket in Unconstrained Delegation
When a client requests a service ticket for a service that is configured with unconstrained delegation, the KDC will do something very interesting:

In a normal situation (without unconstrained delegation), the client would send its TGT, after which the KDC would extract the Privilege Account Certificate (PAC) and copy it in the server portion of the Service Ticket. When unconstrained delegation is used, however, the entire TGT is copied in the server portion of the Service Ticket. Remember that the Server Portion of the Service Ticket is encrypted using the target service Kerberos key.

The TGT can thus afterwards be extracted by the target service. When the target service extracts the TGT, the TGT can now be used to impersonate the client, without any restrictions!

Here are some excellent references that further explain how unconstrained delegation works:
https://adsecurity.org/?p=1667
https://www.cyberark.com/resources/threat-research-blog/weakness-within-kerberos-delegation



Attacking Unconstrained Delegation
Now that we know what a service ticket looks like when Kerberos unconstrained delegation is in use, let's see how we can abuse it.

There's a few steps we'd need to take:

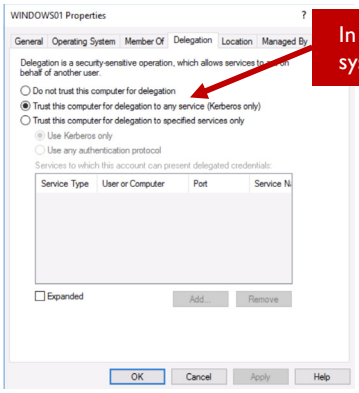
1. As a first step, we need to identify accounts that have Kerberos unconstrained delegation. This is not the default setting, but misconfigurations do occur! Standard domain users have the required privileges to enumerate this type of information.
2. Once we have identified a suitable target, we need to compromise it. To extract the TGT, we need to compromise either a service account or obtain local administrator access to a machine with unconstrained delegation set.
3. Get a domain administrator (or other interesting account) to connect to the compromised service, loading the TGT in memory. Spoiler alert: There are some interesting ways to achieve this. We will leverage the "printer bug" for this, which will be further explained in due course.
4. Once this interesting account has connected, we will dump the TGT of the interesting account from memory. Several tools exist that can facilitate this; some of the more well-known ones include Mimikatz and Rubeus!
5. Finally, you want to abuse the obtained TGT of the interesting account. This would mean we successfully impersonate the user and thus reach our objectives.

We will walk through these steps one by one in the next slides!

ATTACKING UNCONSTRAINED DELEGATION: STEP 1

1

As a first step, we need to **identify computers** that have Kerberos unconstrained delegation. This is not the default setting, but misconfigurations do occur!



In the screenshot, WINDOWS01 has been configured as a system that can perform "unconstrained delegation".

The below two PowerShell command lines can be used on a system with the ActiveDirectory PowerShell module installed. They will enumerate any computer or user with "Unconstrained Delegation" ("TRUSTED_FOR_DELEGATION") set :

```
get-adcomputer -Filter {(TrustedForDelegation –eq "True")} -Properties
TrustedForDelegation,ServicePrincipalName,Description
```

```
get-aduser -Filter {(TrustedForDelegation –eq "True")} -Properties
TrustedForDelegation,ServicePrincipalName,Description
```

SANS SEC699 | Advanced Purple Team Tactics – Adversary Emulation for Breach Prevention & Detection 149

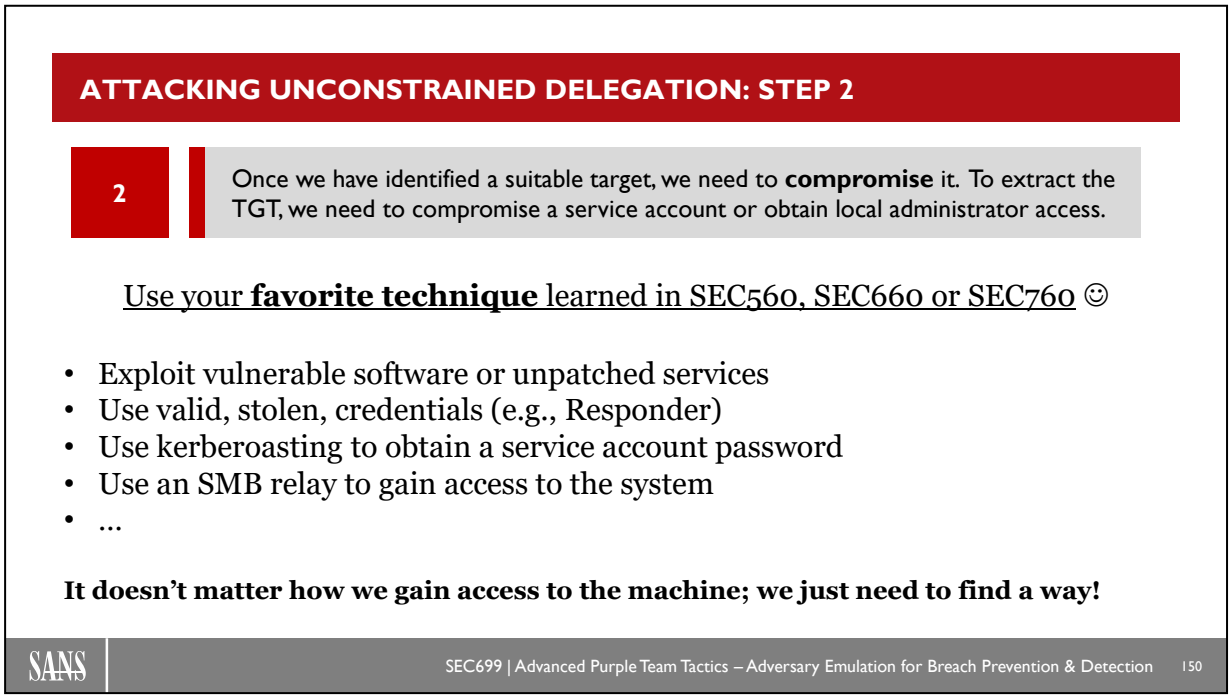
Attacking Unconstrained Delegation: Step 1

Let's zoom in on some of the details of this attack. As a first step, we need to identify computers that have Kerberos unconstrained delegation. This is not the default setting, but misconfigurations do occur! The screenshot in the slide shows the computer properties of a system in the Active Directory (WINDOWS01) that is configured for unconstrained delegation ("Trust this computer for delegation to any service (Kerberos only")).

Adversaries typically don't have GUI-style access to check these settings, but this information can also be enumerated from the command line. The below two PowerShell command lines can be used on a system with the ActiveDirectory PowerShell module installed. They will enumerate any computer or user with "Unconstrained Delegation" ("TRUSTED_FOR_DELEGATION") set :

get-adcomputer -Filter {(TrustedForDelegation –eq "True")} -Properties TrustedForDelegation,ServicePrincipalName,Description

get-aduser -Filter {(TrustedForDelegation –eq "True")} -Properties TrustedForDelegation,ServicePrincipalName,Description



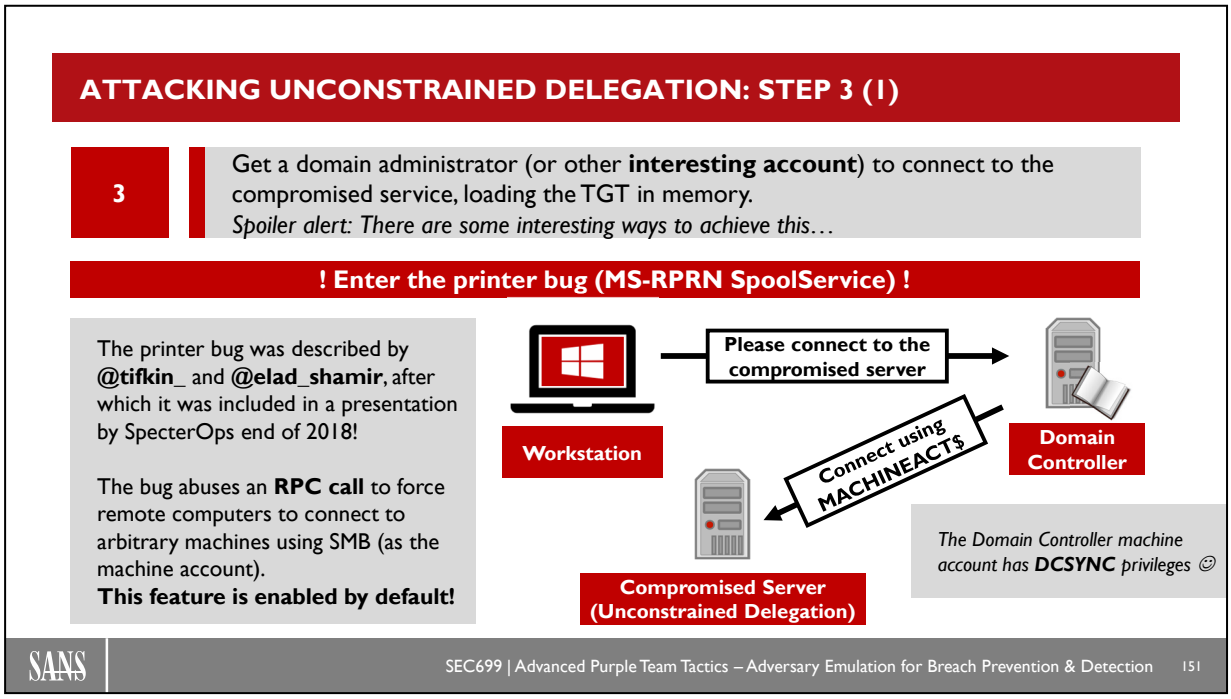
Attacking Unconstrained Delegation: Step 2

Once we have identified a suitable target server, we need to compromise it. To extract the TGT, we need a service account (which has the Kerberos encryption key) or a local administrative account on a server where the service is active (as this will also allow access to the Kerberos encryption key used by the service).

How could we do this? We suggest to use your favorite technique learned in SEC560, SEC660 or SEC760! Examples include:

- Exploit vulnerable software or unpatched services
- Use valid, stolen, credentials (e.g., Responder)
- Use kerberoasting to obtain a service account password
- Use an SMB relay to gain access to the system
- …

It doesn't matter how we get access to the machine; we just need to find a way!



Attacking Unconstrained Delegation: Step 3 (1)

Once we have obtained access to a compromised system with unconstrained delegation, the next step is to trick someone into connecting to the system (or service)! In an adversary's wildest dreams, this would be a domain administrator (or an account with similar privileges). Once this victim connects, we can extract the TGT from memory!

But do we really need social engineering / trickery? It turns out we don't…. Enter the "printer bug".

The printer bug was described by @tifkin_ and @elad_shamir, after which it was included in a presentation by SpecterOps end of 2018! The Print System Remote Protocol (MS-RPRN) has two highly interesting methods that allow us to trick a remote system to set up an SMB connection by providing the remote system with a hostname/IP address, after which it will connect back for the purpose of sending notifications. These methods are:

- RpcRemoteFindFirstPrinterChangeNotification
- RpcRemoteFindFirstPrinterChangeNotificationEX

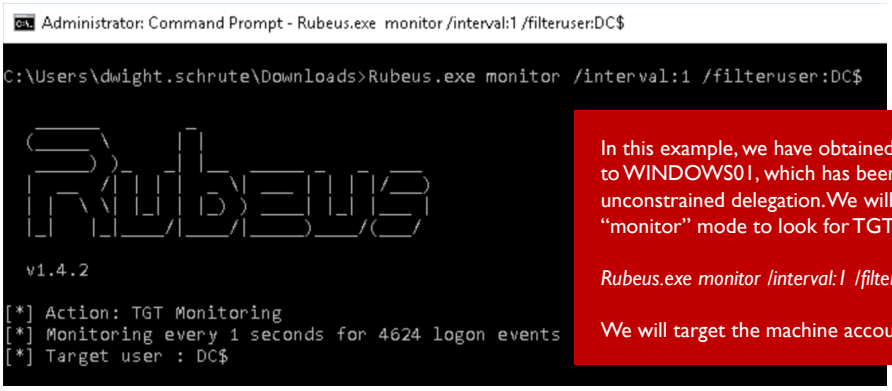
This "return" connection uses a named pipe over SMB; also note that the Printer Spooler service runs with LOCAL SYSTEM privileges.

Imagine abusing the printer bug against a domain controller: The domain controller will connect back using the MACHINEACCT$ to a system of our choosing. We could thus have the domain controller connect back to a compromised system that has unconstrained delegation... Note that the DC machine account has DCSYNC privileges, so this would effectively compromise the entire domain!

ATTACKING UNCONSTRAINED DELEGATION: STEP 3 (2)

3

Get a domain administrator (or other **interesting account**) to connect to the compromised service, loading the TGT in memory.
Spoiler alert: There are some interesting ways to achieve this…



In this example, we have obtained local administrator access to WINDOWS01, which has been configured for unconstrained delegation. We will now launch Rubeus in "monitor" mode to look for TGTs; the syntax we use is:

Rubeus.exe monitor /interval:1 /filteruser:DC$

We will target the machine account of the DC (DC$).

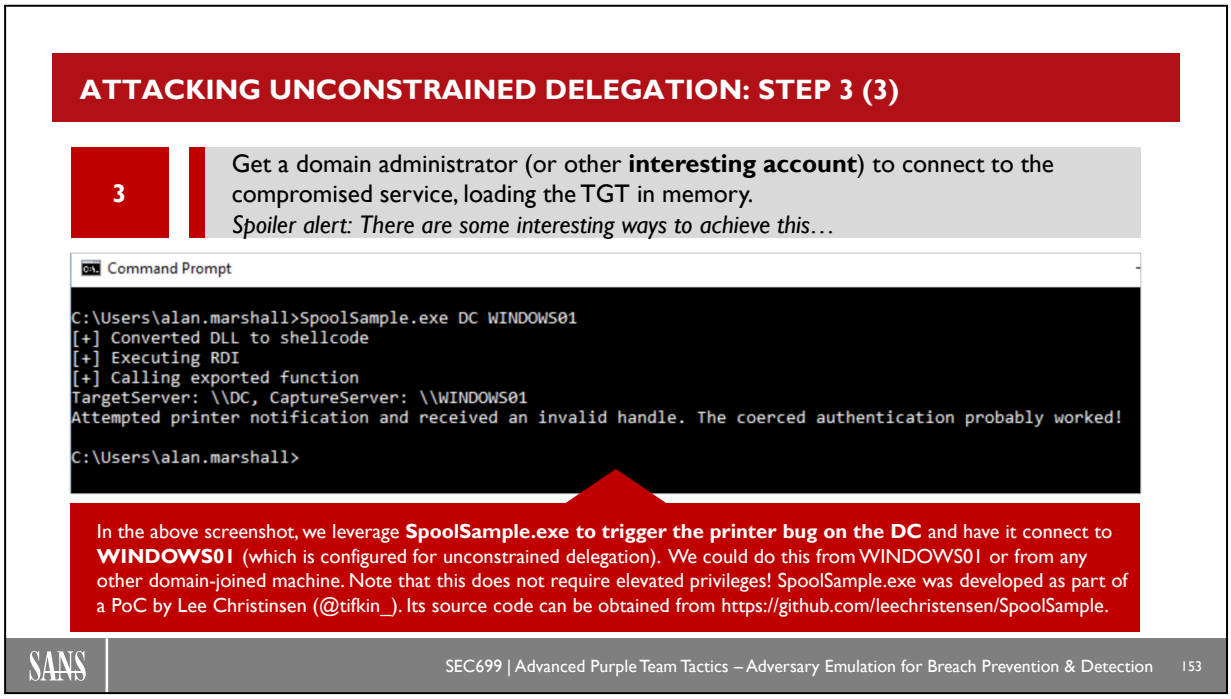
Attacking Unconstrained Delegation: Step 3 (2)

Let's look at a practical example! In the screenshot on the slide, we are running Rubeus in order to monitor the current machine for TGTs. In this example, we have obtained local administrator access to WINDOWS01, which has been configured for unconstrained delegation. We will now launch Rubeus in "monitor" mode to look for TGTs, the syntax we use is:

Rubeus.exe monitor /interval:1 /filteruser:DC$

A few words on the syntax of the command that is used:

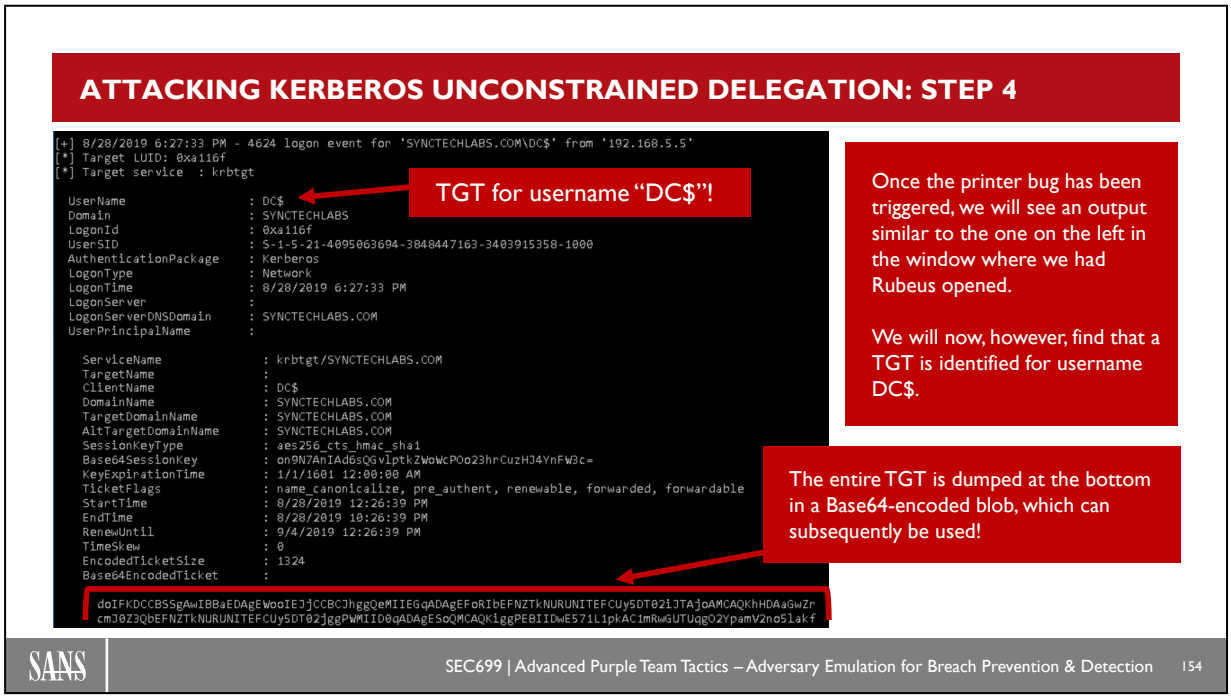
- monitor – we want to use the Rubeus "monitor" feature which will review the system for TGTs present
- /interval:1 – we want to refresh every second
- /filteruser:DC$ – we want to filter for the "DC$" machine account, which is the target we will attack!



Attacking Unconstrained Delegation: Step 3 (3)

Now that Rubeus is looking for a TGT for the DC$ computer account, let's force the domain controller (DC) to connect using its machine account (DC$). We can do this by triggering the printer bug. Many tools exist that can leverage / trigger the printer bug, one of them being SpoolSample.exe.

In the above screenshot, we leverage SpoolSample.exe to trigger the printer bug on the DC and have it connect to WINDOWS01 (which is configured for unconstrained delegation and where Rubeus is running). We could do this from WINDOWS01 or from any other machine with a domain account. Note that this does not require elevated privileges! SpoolSample.exe was developed as part of a PoC by Lee Christensen (@tifkin_). Its source code can be obtained from https://github.com/leechristensen/SpoolSample.



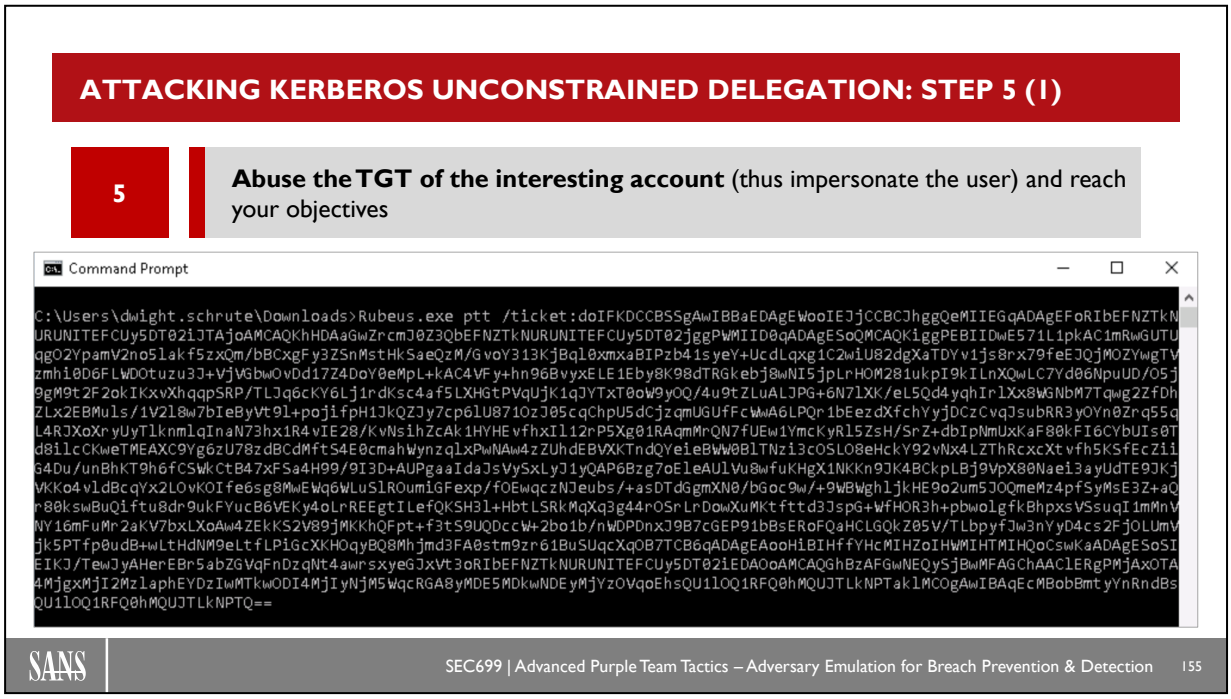
Attacking Kerberos Unconstrained Delegation: Step 4

Once the printer bug has been triggered, we will see an output similar to the one on the left in the window where we had Rubeus opened in monitor mode. We will now, however, find that a TGT is identified for username DC$. We can confirm that it's a TGT, as the ServiceName is "krbtgt".

Excellent, it appears our attack was successful!

The Base64 encoded ticket is printed at the very end of the output. We can either copy / paste this ticket and feed it into Rubeus (see next slide), or we can choose to write it to a file: It can be written to a file by using the following command (where we replace <BASE64BLOB> with the Base64-encoded ticket (one long string):

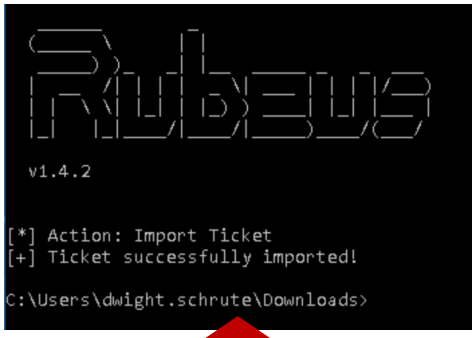
[IO.File]::WriteAllBytes("ticket.kirbi", [Convert]::FromBase64String("**<BASE64BLOB>**"))



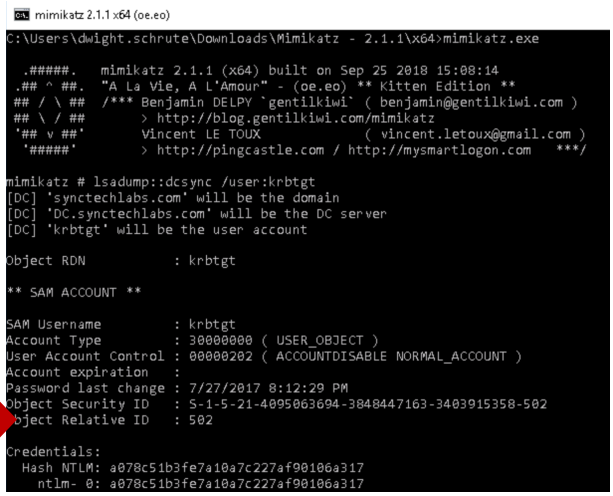
Attacking Kerberos Unconstrained Delegation: Step 5 (1)

Finally, let's abuse the TGT of the interesting account to impersonate the user. In the screenshot above, we see Rubeus being used to load in ticket in memory using the "ptt" feature. We are specifying the entire ticket as a Base64-encoded blob in the command!

ATTACKING KERBEROS UNCONSTRAINED DELEGATION: STEP 5 (2)



Once the ticket is loaded in memory, we launch Mimikatz and use dcsync to dump "krbtgt" account secrets (NT hash, Kerberos keys,…). As we all know: **krbtgt compromised = domain compromised**!



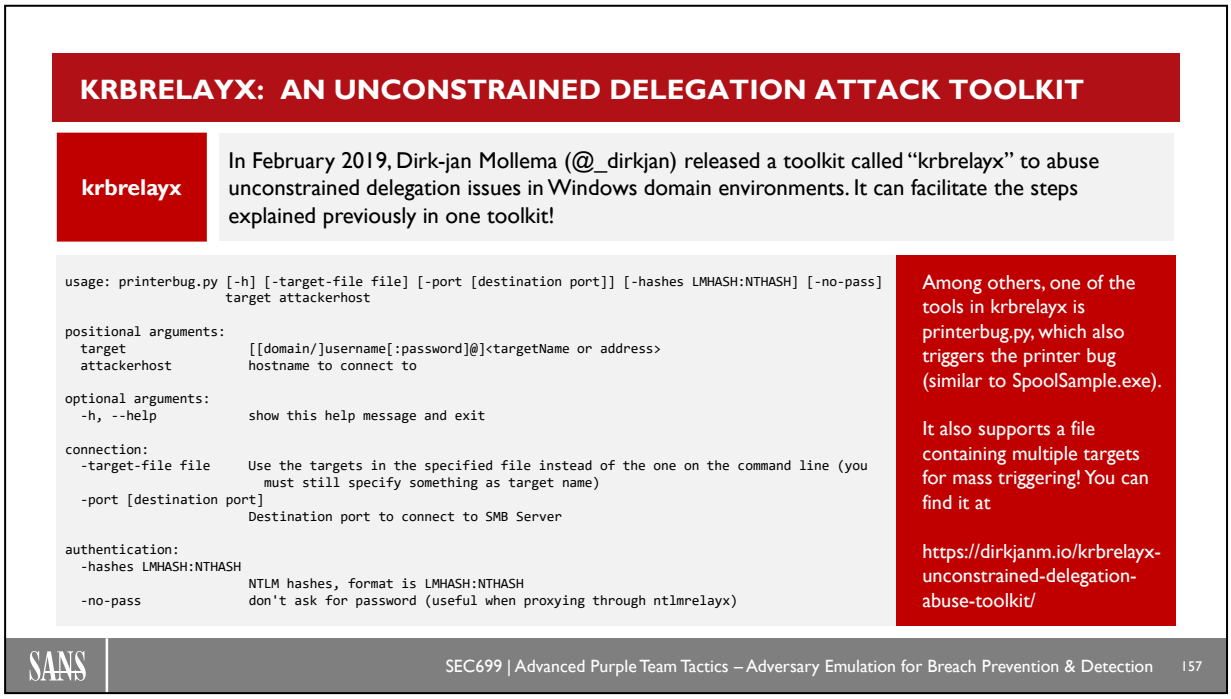
Attacking Kerberos Unconstrained Delegation: Step 5 (2)

Once the TGT for the domain controller machine account is loaded in memory (see Rubeus screenshot), we can now abuse it.

Note that the domain controller machine account has "dcsync" (replication) privileges. We can thus use it to compromise the entire domain. In the screenshot to the right, we launch Mimikatz (in a normal, non-elevated prompt) and launch dcsync to retrieve the secrets of the krbtgt account:

Lsadump::dcsync /user:krbtgt

Once we compromise the krbtgt account, we have compromised the entire domain and can start crafting TGTs at will (e.g., golden tickets)!



Krbrelayx: An Unconstrained Delegation Attack Toolkit

In February 2019, Dirk-jan Mollema (@_dirkjan) released a toolkit called "krbrelayx" to abuse unconstrained delegation issues in Windows domain environments. While we previously used a variety of different tools (Rubeus, Mimikatz,…), krbrelayx combines everything in one nice toolkit! Furthermore, Dirk-jan added additional tools for further exploration and exploitation, but we won't discuss these for now.

Among others, one of the tools in krbrelayx is printerbug.py, which also triggers the printer bug (similar to SpoolSample.exe). It also supports a file containing multiple targets for mass triggering! You can find the tool and a short manual at https://dirkjanm.io/krbrelayx-unconstrained-delegation-abuse-toolkit/.

Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.

EXERCISE: UNCONSTRAINED DELEGATION ATTACKS

Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

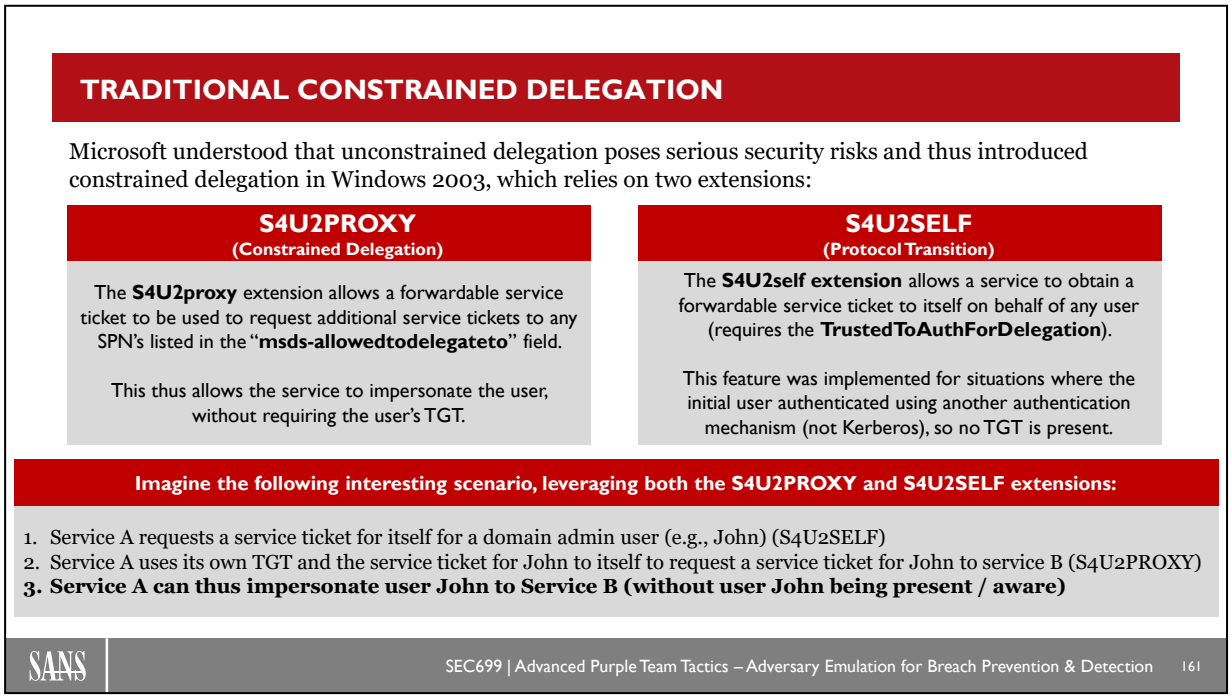
Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.



Traditional Constrained Delegation

Microsoft understood that unconstrained delegation poses serious security risks and thus introduced constrained delegation in Windows 2003. You can find the official Microsoft documentation on the following page:

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-sfu/1fb9caca-449f-4183-8f7a-1a5fc7e7290a

Traditional constrained delegation relies on two extensions:

S4U2PROXY (Constrained Delegation)

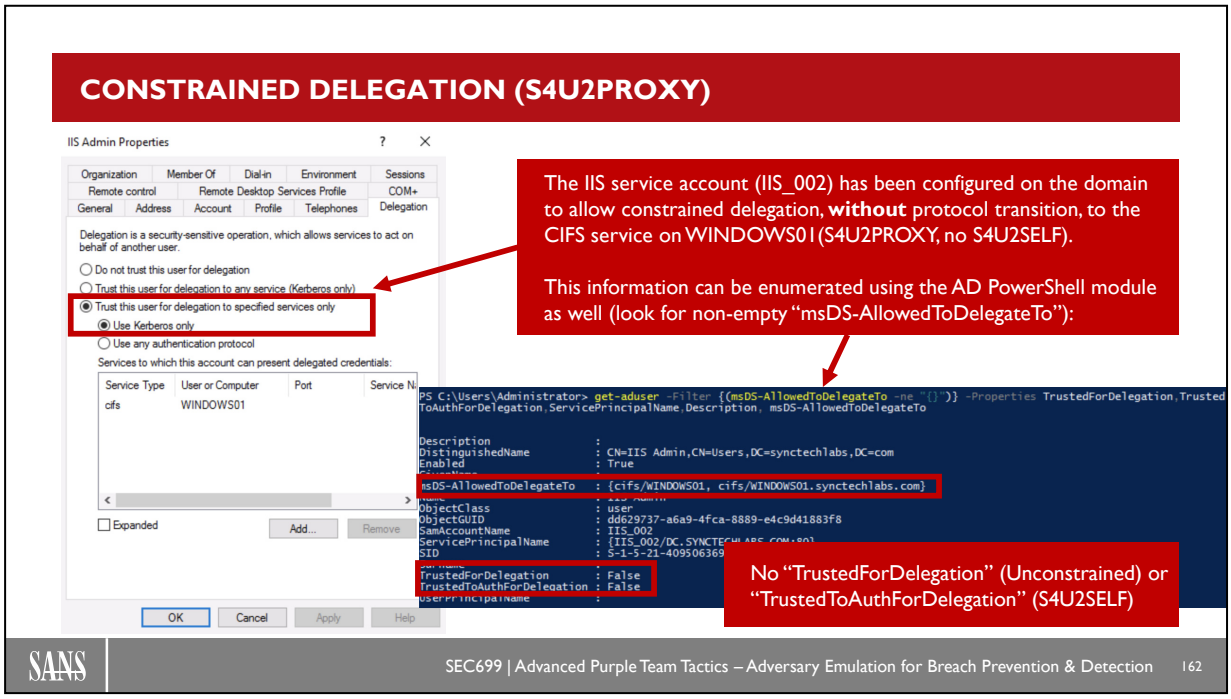
The S4U2proxy extension allows a forwardable service ticket to be used to request additional service tickets to any SPNs listed in the "msds-allowedtodelegateto" field. This thus allows the service to impersonate the user, without requiring the user's TGT.

S4U2SELF (Protocol Transition)

The S4U2self extension allows a service to obtain a forwardable service ticket to itself on behalf of a user (requires the TrustedToAuthForDelegation). This feature was implemented for situations where the initial user authenticated using another authentication mechanism (not Kerberos), so no TGT is present.

How could these two be used in a joint attack scenario? Imagine the following:

1. Service A requests a service ticket for itself for user John (S4U2SELF)
2. Service A uses its own TGT and the service ticket for John to itself to request a service ticket for John to Service B (S4U2PROXY)
3. Service A can now impersonate user John to Service B (without user John being present / aware)



Constrained Delegation (S4U2PROXY)

In the example configuration on the slide, the IIS service account (IIS_002) has been configured on the domain to allow constrained delegation, without protocol transition, to the CIFS service on WINDOWS01(S4U2PROXY, no S4U2SELF). On the left-hand side, we've taken a screenshot of the GUI-based configuration in AD, where we can clearly see that the user has been trusted for delegation to specified services only (hence "constrained delegation"). We also note that this delegation is for Kerberos only, thus there is no "protocol transition".

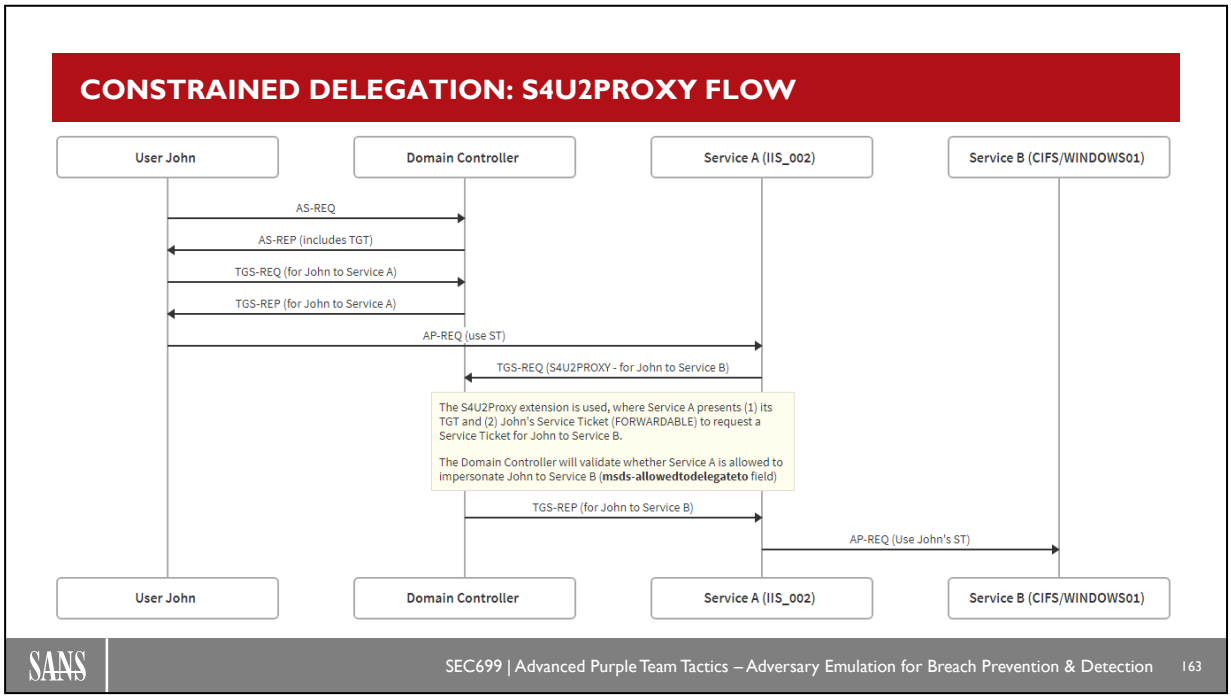
This information can be enumerated using the AD PowerShell module as well (look for non-empty "msDS-AllowedToDelegateTo"):

- For users, we can run the following PowerShell cmdlet (requires the PowerShell ActiveDirectory module):

  get-aduser -Filter {(msDS-AllowedToDelegateTo -ne "{}")} -Properties TrustedForDelegation,TrustedToAuthForDelegation,ServicePrincipalName, Description, msDS-AllowedToDelegateTo

- For computers, we can run the following PowerShell cmdlet (requires the PowerShell ActiveDirectory module):

  get-adcomputer -Filter {(msDS-AllowedToDelegateTo -ne "{}")} -Properties TrustedForDelegation,TrustedToAuthForDelegation,ServicePrincipalName, Description, msDS-AllowedToDelegateTo



Constrained Delegation: S4U2Proxy Flow

Let's look a bit closer at S4U2Proxy. As we said before, the S4U2proxy extension allows a forwardable service ticket (a service ticket is by default forwardable) to be used to request additional service tickets to any SPNs listed in the "msds-allowedtodelegateto" field. This thus allows the service to impersonate the user, without requiring the user's TGT.

How does this work in a bit more detail?

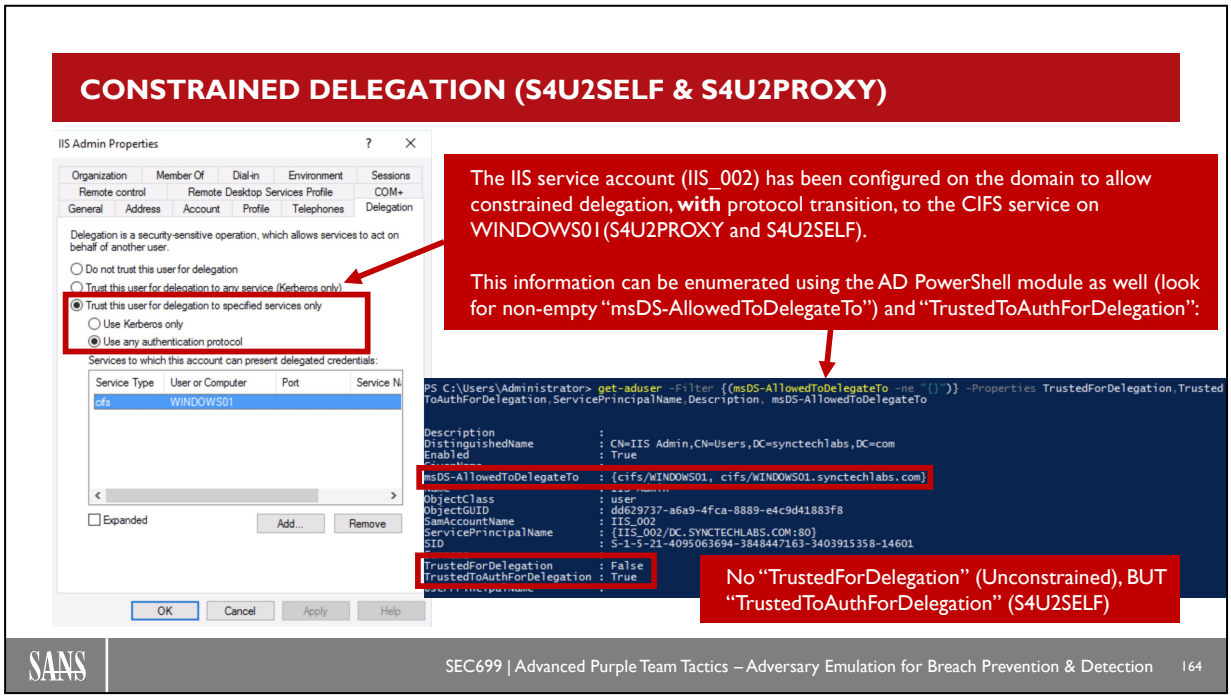
1. User John authenticates to the domain controller by sending an encrypted timestamp (AS-REQ)
2. John receives a TGT from the domain controller (AS-REP)
3. John requests a service ticket for service A (TGS-REQ)
4. John receives a service ticket for service A (TGS-REP)
5. John uses the service ticket to access service A (AP-REQ)

Up until this point, everything is normal, but here's what happens next:

6. Service A would impersonate user John toward Service B. In this traditional constrained delegation setup, the S4U2Proxy extension is used, where Service A presents (1) its TGT and (2) John's Service Ticket (FORWARDABLE) to request a Service Ticket for John for Service B.
7. The Domain Controller will validate whether Service A is allowed to impersonate John to Service B (by checking the msds-allowedtodelegateto field). More specifically, the KDC checks if the requested service (service B) is listed in the msds-allowedtodelegateto field of the requesting user (Service A).

For author / instructor reference, the swimlanes diagram in the slide can be generated using the following URL:

https://swimlanes.io/#hZJRT8IwFIXf9yvum+3iFIhPPJgMBmTGyGQj6JNZ6EUaS4ttAf33tkCWKUNet3Pu+XrvsdwK7EJfSWN1ySUySFDge2m5kkDyu2knm4xfXmkQBFODGh7UUkJ0D4laObk3Wq2EQN2FOI8mg+fg5I+XV96jLAPC5VxsGBooRgW9OLwY7acDWaijzirIUW/5HCGmF1MP/uys/1d+9R1ImuZvrVaHOu7sALAxCLkjbhBdYq/WCRE0cfRoIJXFbrF0EV6r1dc34JdFafw9uAGXzq5ht0SNNcy1RoPSGiBtCtzudwqlZEAclE+5MpW64PMPtECG48ksniRx73FAPYTGzw0aC+VfZSPpTRB4ytO177gQsC0FZ6VFD2odK9RY3SNKIdTOVc0N5Ks1aqOkF5+EAAnDlWEmOhqsYodyolVhCAuOgp05feMJ/y1B7+xJazz9dJjfztKnZDzLW+1aK1x/qkUX9Ac=



Constrained Delegation (S4U2SELF & S4U2PROXY)
In the example configuration on the slide, the IIS service account (IIS_002) has now been configured on the domain to allow constrained delegation, with protocol transition, to the CIFS service on WINDOWS01(S4U2PROXY and S4U2SELF). On the left-hand side, we've taken a screenshot of the GUI-based configuration in AD, where we can clearly see that the user has been trusted for delegation to specified services only (hence "constrained delegation"). We also note that this delegation is for any protocol, so there is "protocol transition".

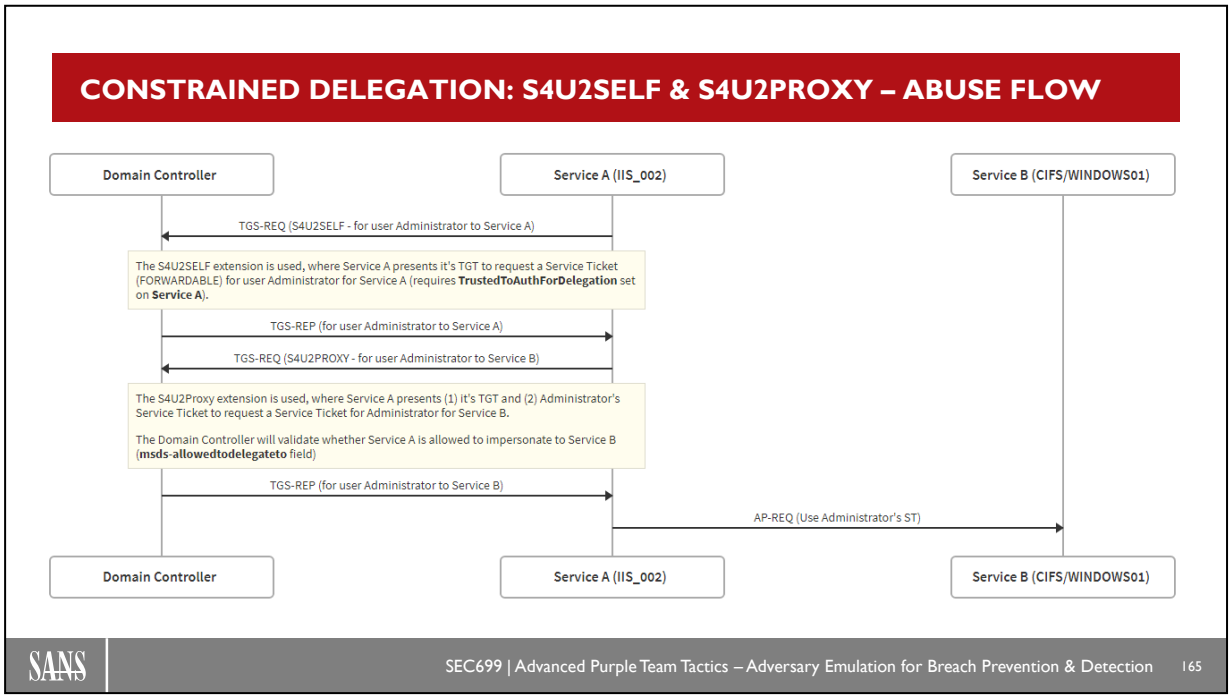
This information can be enumerated using the AD PowerShell module as well (look for non-empty "msDS-AllowedToDelegateTo" and "TrustedToAuthForDelegation"):

- For users, we can run the following PowerShell cmdlet (requires the PowerShell ActiveDirectory module):

  get-aduser -Filter {(msDS-AllowedToDelegateTo -ne "{}")} -Properties TrustedForDelegation,TrustedToAuthForDelegation,ServicePrincipalName, Description, msDS-AllowedToDelegateTo

- For computers, we can run the following PowerShell cmdlet (requires the PowerShell ActiveDirectory module):

  get-adcomputer -Filter {(msDS-AllowedToDelegateTo -ne "{}")} -Properties TrustedForDelegation,TrustedToAuthForDelegation,ServicePrincipalName, Description, msDS-AllowedToDelegateTo



Constrained Delegation: S4U2SELF & S4U2PROXY – Abuse Flow
Let's look a bit closer at the combination of S4U2SELF and S4U2Proxy. Let's imagine for this example that Service A (IIS_002) has "TrustedToAuthForDelegation" set. This implies that Service A (IIS_002) can request service tickets to itself for any user in the domain. Furthermore, in our example, Service A is allowed to delegate to Service B (CIFS/WINDOWS01) using the "msds-allowedtodelegateto" field.

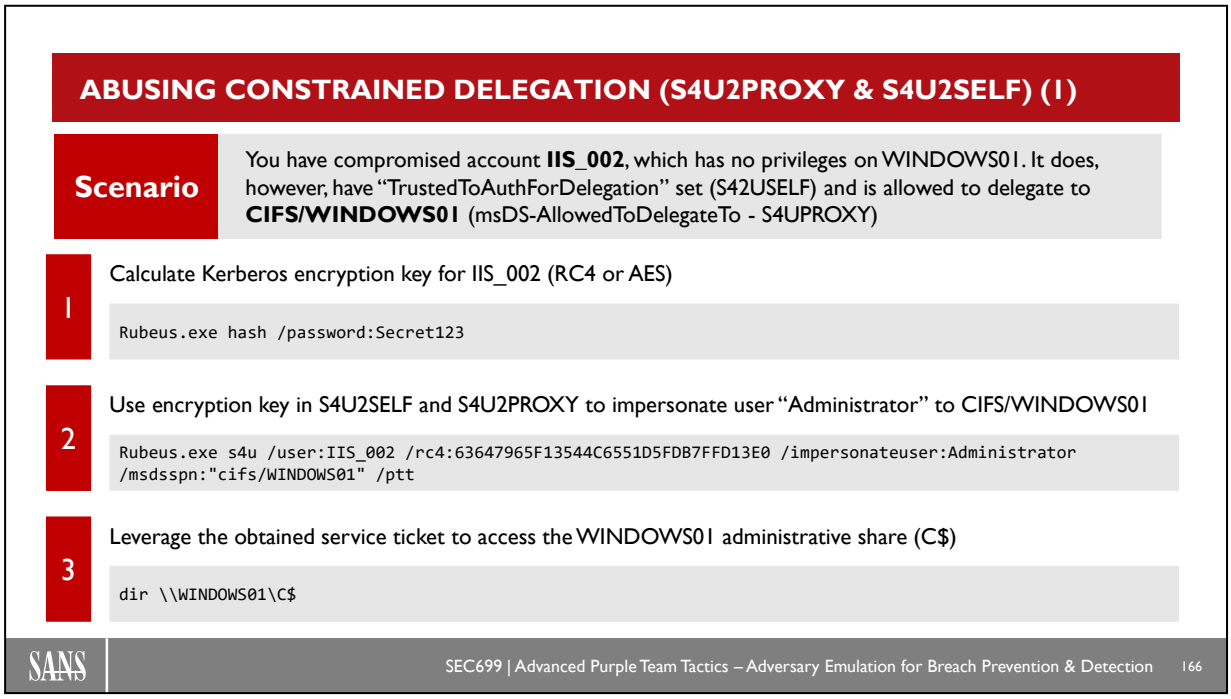
How could this be abused?

1. Service A (IIS_002) uses S4U2SELF to request a service ticket for user Administrator to itself
2. As Service A has the "TrustedToAuthForDelegation" flag set, this is allowed, and the DC provides a service ticket for user Administrator to Service A
3. Service A now uses S4U2PROXY to request a service ticket for user Administrator to Service B (CIFS/WINDOWS01)
4. As Service A is allowed to impersonate to Service B, this is allowed, and the DC provides a service ticket for user Administrator to Service B
5. Service A uses the service ticket for user Administrator to Service B and accesses it!

Note that during this entire interaction, user Administrator was not even involved, yet Service A was able to access Service B!

For author / instructor reference, the swimlanes diagram in the slide can be generated using the following URL:

https://swimlanes.io/#rZNPT8JAEMXvfIo5abuxCsQTMSblnyExgm0NejINO8rGZVd3B9Bv76waqIoEjbdmO/Pe2/drC0UaW9CxxpMrlUEJXdR4X5KyBqL8+KqZ9877sAfhcZQNr2/iWq1rZzwbtshZrdHBSQI5uoWaIKQQDQb5bb3ejFtQnOVJ1rusKCVwZx3MPS+lcqaMCsbER2TXEuxRM5awVUwRVqv4TGh8CKZ8UJAHsJyiw4r1o0OPhjwo2vfsXgRZh09z9ATlarBQkwckiPrDbJxm3bR93ot/yhWOK3cLYopdQIjCzT2hLGw6p2nfunVzQoBnfU4qxGpXiPhwU3fJ6bbuRhDtVtgfobwx3YlKO/7MZOTs88uvoESNeA2mNBKiZvzZj999YbSNX8j8M6w2tx2yfi9mqbSGRamVLAlDXOLEUEnMVym1tkv+HdhfzR7ReWvCcLUPiISYeemTj1my8v0LQLL8Bdwp1HIjmH8gzjA2SFSVOV5n0M+PxoOL7nCc1xtskI7ewV95/N57Eb8C



Abusing Constrained Delegation (S4U2SELF & S4U2PROXY) (1)

So how could we abuse constrained delegation? We can all agree that constrained delegation is a much safer option than unconstrained delegation. It's not waterproof, though, and misconfigurations can still exist! Let's imagine the following scenario: You have compromised account IIS_002, which has no privileges on WINDOWS01. It does, however, have "TrustedToAuthForDelegation" set (S42USELF) and is allowed to delegate to CIFS/WINDOWS01 (msDS-AllowedToDelegateTo - S4UPROXY). How could we compromise WINDOWS01?

- Step 1: Calculate Kerberos encryption key for IIS_002 (RC4 or AES). This can be achieved by using Rubeus with the following command line (we assume the password for IIS_002 is "Secret123"):

  Rubeus.exe hash /password:Secret123

- Step 2: Use encryption key in S4U2SELF and S4U2PROXY to impersonate user "Administrator" to CIFS/WINDOWS01

  Rubeus.exe s4u /user:IIS_002 /rc4:63647965F13544C6551D5FDB7FFD13E0 /impersonateuser:Administrator /msdsspn:"cifs/WINDOWS01" /ptt

- Step 3: Leverage the obtained service ticket to access the WINDOWS01 administrative share (C$)

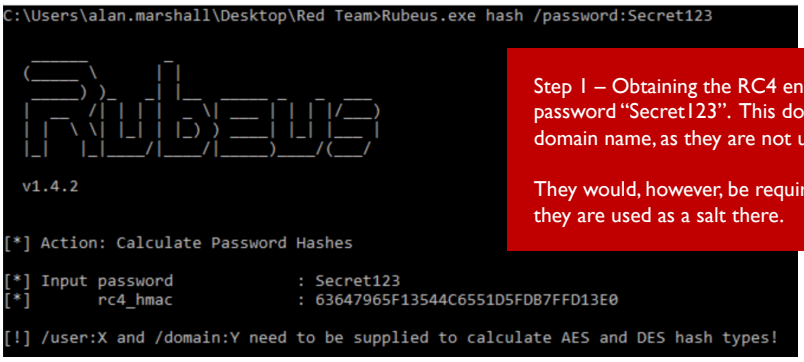
  dir \\WINDOWS01\C$

Let's take a look at these steps in a bit more detail!

ABUSING CONSTRAINED DELEGATION (S4U2PROXY & S4U2SELF) (2)

1 Calculate Kerberos encryption key for IIS_002 (RC4 or AES)

```
Rubeus.exe hash /password:Secret123
```



Step 1 – Obtaining the RC4 encryption key (i.e., NT hash) for password "Secret123". This does not require the username or domain name, as they are not used in this algorithm.

They would, however, be required when generating an AES key, as they are used as a salt there.

Abusing Constrained Delegation (S4U2PROXY & S42USELF) (2)
In the screenshot above, we can see Rubeus being used to obtain the RC4 encryption key (NT hash) for the "Secret123" password. Note that this command does not require the username or domain name, as they are not used in this algorithm. They would, however, be required when generating an AES key, as they are used as a salt there.

ABUSING CONSTRAINED DELEGATION (S4U2PROXY & S4U2SELF) (3)

2 Use encryption key in S4U2SELF and S4U2PROXY to impersonate user "Administrator" to CIFS/WINDOWS01

```
Rubeus.exe s4u /user:IIS_002 /rc4:63647965F13544C6551D5FDB7FFD13E0 /impersonateuser:Administrator
/msdsspn:"cifs/WINDOWS01" /ptt
```

Step 2 – This one-line Rubeus command will perform both S42USELF (to request a service ticket for user Administrator to itself) and S4U2PROXY (to request a service ticket for user Administrator to CIFS/WINDOWS01)

Abusing Constrained Delegation (S4U2PROXY & S42USELF) (3)
Once we have obtained the RC4 Kerberos encryption key for the IIS_002 user, we will now use it in Rubeus. In the screenshot above, we can see the one-line Rubeus command will perform both S42USELF (to request a service ticket for user Administrator to itself) and S4U2PROXY (to request a service ticket for user Administrator to CIFS/WINDOWS01).

Note that the output of this command includes the service ticket and is thus rather long (we did not fully print it in the screenshot).

ABUSING CONSTRAINED DELEGATION (S4U2PROXY & S4U2SELF) (4)

3 Leverage the obtained service ticket to access the WINDOWS01 administrative share (C$)

```
dir \\WINDOWS01\C$
```

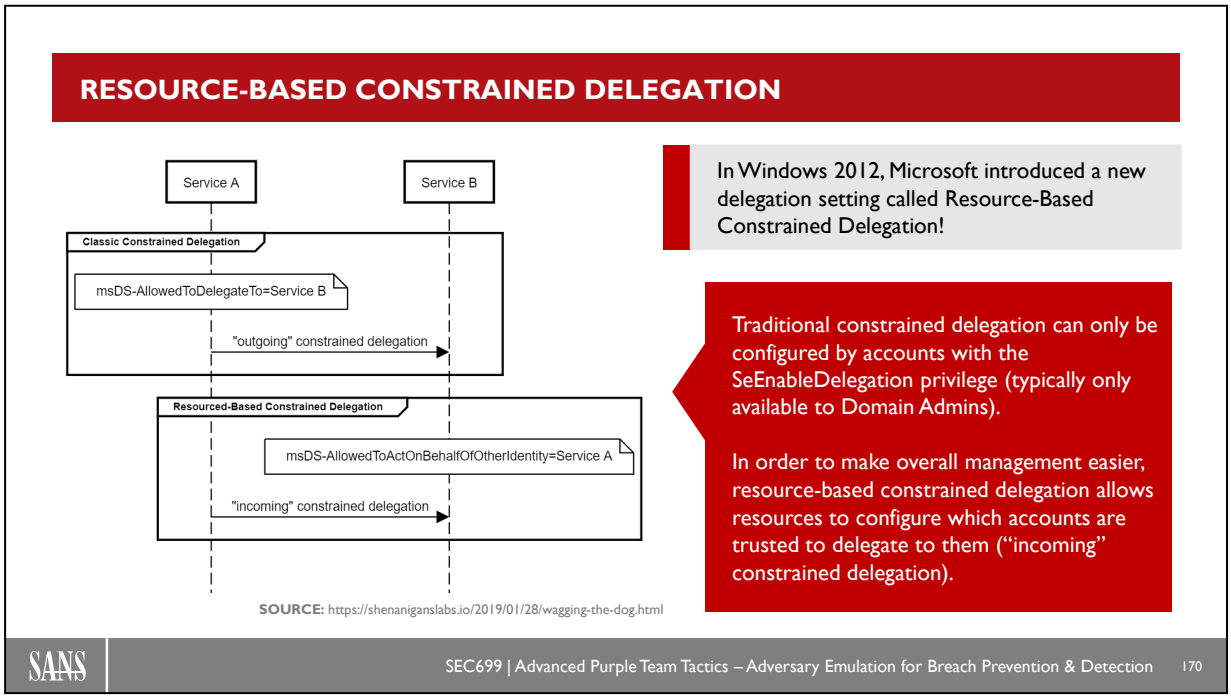
```
C:\Users\alan.marshall\Desktop\Red Team>dir \\WINDOWS01\c$
 Volume in drive \\WINDOWS01\c$ has no label.
 Volume Serial Number is 507D-1839

 Directory of \\WINDOWS01\c$

04/11/2018  11:38 PM    <DIR>          PerfLogs
12/27/2018  12:52 PM    <DIR>          Program Files
08/28/2019  12:27 PM    <DIR>          Program Files (x86)
02/03/2017  10:15 PM    <DIR>          tmp
01/30/2019  12:30 PM    <DIR>          Users
08/28/2019  12:55 PM    <DIR>          Windows
               0 File(s)              0 bytes
               6 Dir(s)  22,269,513,728 bytes free
```

Step 3 – We can now use the obtained service ticket to access any resource that is offered by the CIFS/WINDOWS01 services (e.g., the C$ administrative file share)

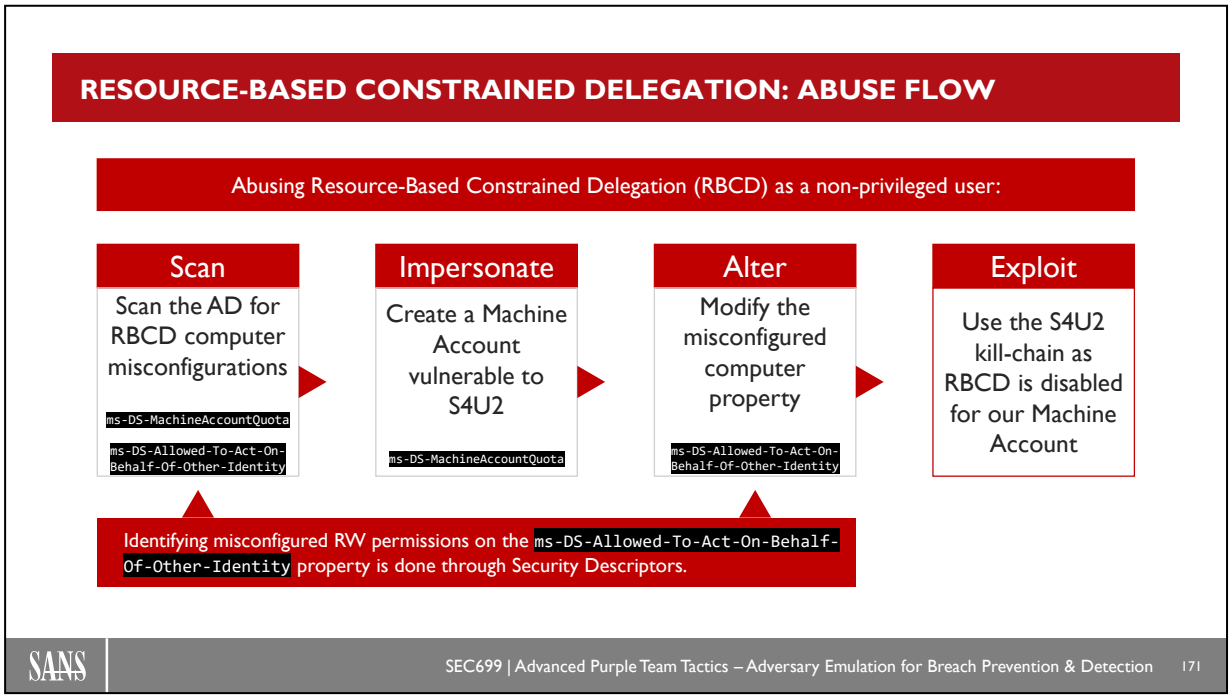
Abusing Constrained Delegation (S4U2PROXY & S42USELF) (4)
As a final step, we can now use the obtained service ticket to access any resource that is offered by the CIFS/WINDOWS01 services (e.g., the C$ administrative file share)!



Resource-based Constrained Delegation
In Windows 2012, Microsoft introduced a new delegation setting called Resource-Based Constrained Delegation! Traditional constrained delegation can only be configured by accounts with the SeEnableDelegation privilege (typically only available to Domain Admins). This can become quite resource-intensive to manage; domain administrators would have to set up all delegation settings for all accounts in the environment!

In order to make overall management easier, resource-based constrained delegation allows resources to configure which accounts are trusted to delegate to them ("incoming" constrained delegation).

A highly interesting read on resource-based constrained delegation (and abuse strategies) can be found here: https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html

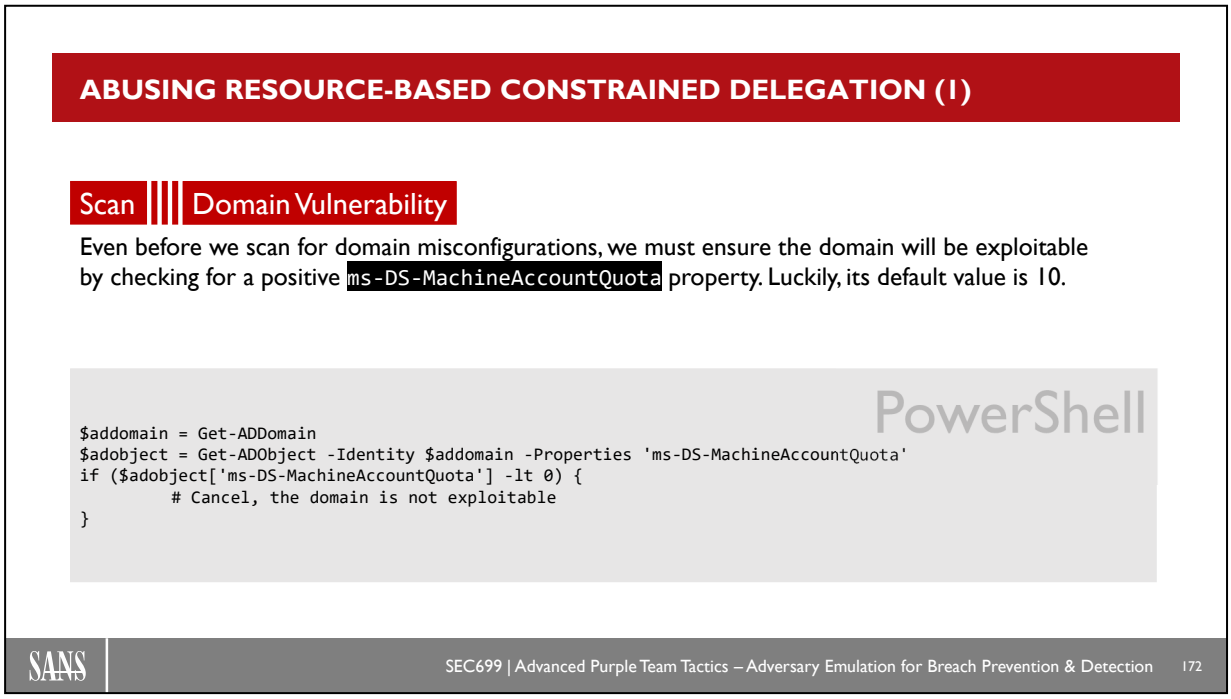


Resource-Based Constrained Delegation: Abuse Flow

Abusing RBCD as an unprivileged user is possible, although complex. The main objective of the abuse flow is to leverage Security Descriptors to render RBCD useless in order to fall back to the previous S4U2 abuses as follows:

1. Scan the AD to ensure the domain is exploitable and to identify vulnerable computers.
    1. To exploit the domain, the *"ms-DS-MachineAccountQuota"* property of the domain must be greater than zero. This property enables us to create machine accounts.
    2. For a computer to be vulnerable we must somehow be able to alter the *"ms-DS-Allowed-To-Act-On-Behalf-Of-Other-Identity"* property either through owner-hijacking or property writing.
2. Create a machine account vulnerable to the S4U2 abuse flow.
3. Alter the misconfigured computer's *"ms-DS-Allowed-To-Act-On-Behalf-Of-Other-Identity"* property to authorize our machine account for Resource-Based Constraint Delegation.
4. With RBCD out of our way, proceed with the S4U2 abuse flow.

Let's further examine the abuse flow…



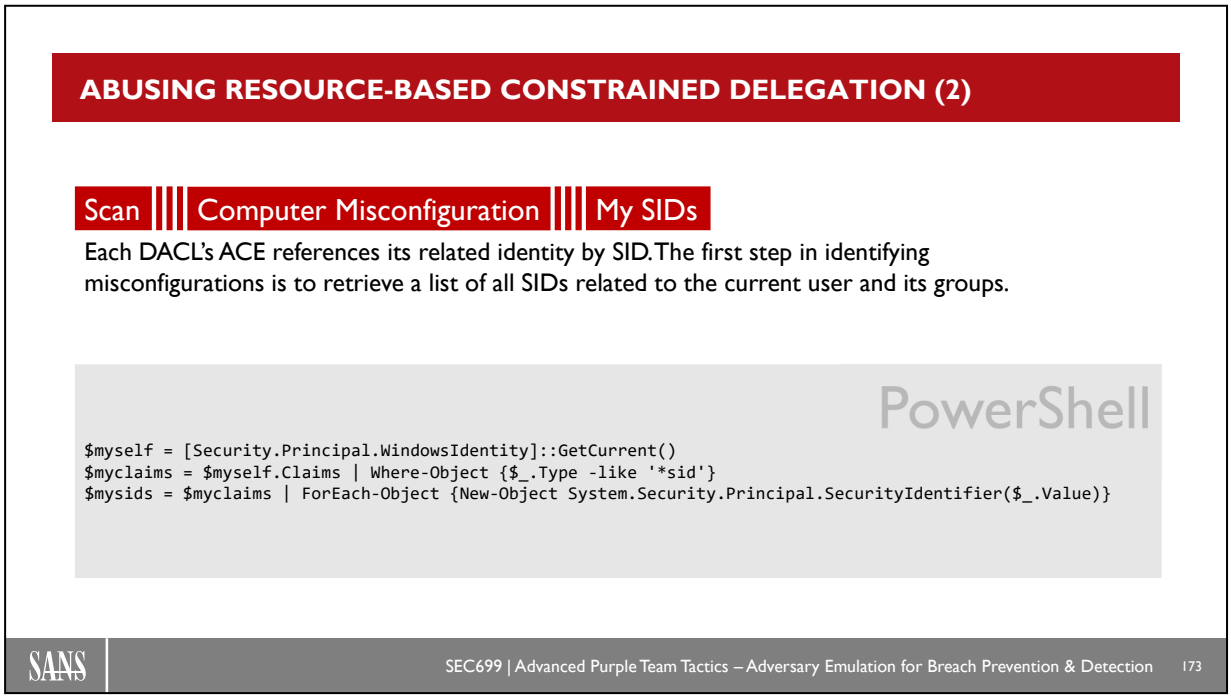
Abusing Resource-based Constrained Delegation (1)

The first stage of recon requires us to check the domain's *"ms-DS-MachineAccountQuota"* property to ensure we can later-on create a machine account.

AD-based recon through PowerShell is often eased by the use of dedicated tools such as PowerSploit's PowerView recon module. Although undeniably useful, relying on these scripts is not very stealth and is almost guaranteed to raise red flags. A more common and less malicious suite of tools that we will rely on are the Remote Server Administration Tools as these are provided by Microsoft.

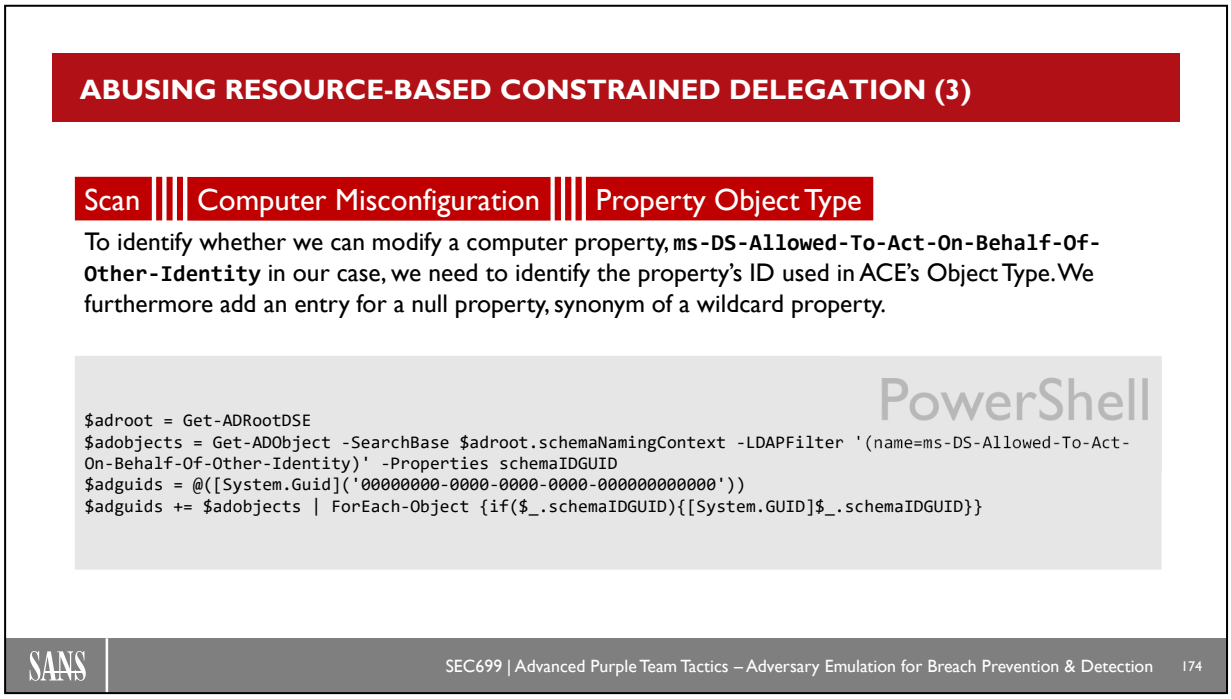
Getting the domain's *"ms-DS-MachineAccountQuota"* property is quite simple as we just need to retrieve the domain as an AD object. As showed in the above snippet, to avoid loading all properties, we will specify the sole property of interest.

A simple check to ensure the property is greater than zero will determine whether we can perform further recon or not.



Abusing Resource-based Constrained Delegation (2)
The next step in our recon process requires us to identify vulnerable computers. To do so, we must first of all identify all SIDs which we are part of (ourself, AD groups, built-in groups, …).

The easiest way to do this is by retrieving our Windows Identity. As each Windows Identity contains a list of claims, we can retrieve all claims related to SIDs and convert the SIDs from strings to the appropriate SecurityIdentifier object.



Abusing Resource-based Constrained Delegation (3)

Active Directory Securable Object's properties are referenced by a GUID. To identify our ability to write the desired "`ms-DS-Allowed-To-Act-On-Behalf-Of-Other-Identity`" property, we must retrieve its Active Directory GUID from the AD's schema. Due to the Security Descriptor's working, which will be covered in-depth tomorrow, we furthermore add the wildcard GUID as a valid property GUID.

ABUSING RESOURCE-BASED CONSTRAINED DELEGATION (4)

Scan | Computer Misconfiguration | ACL

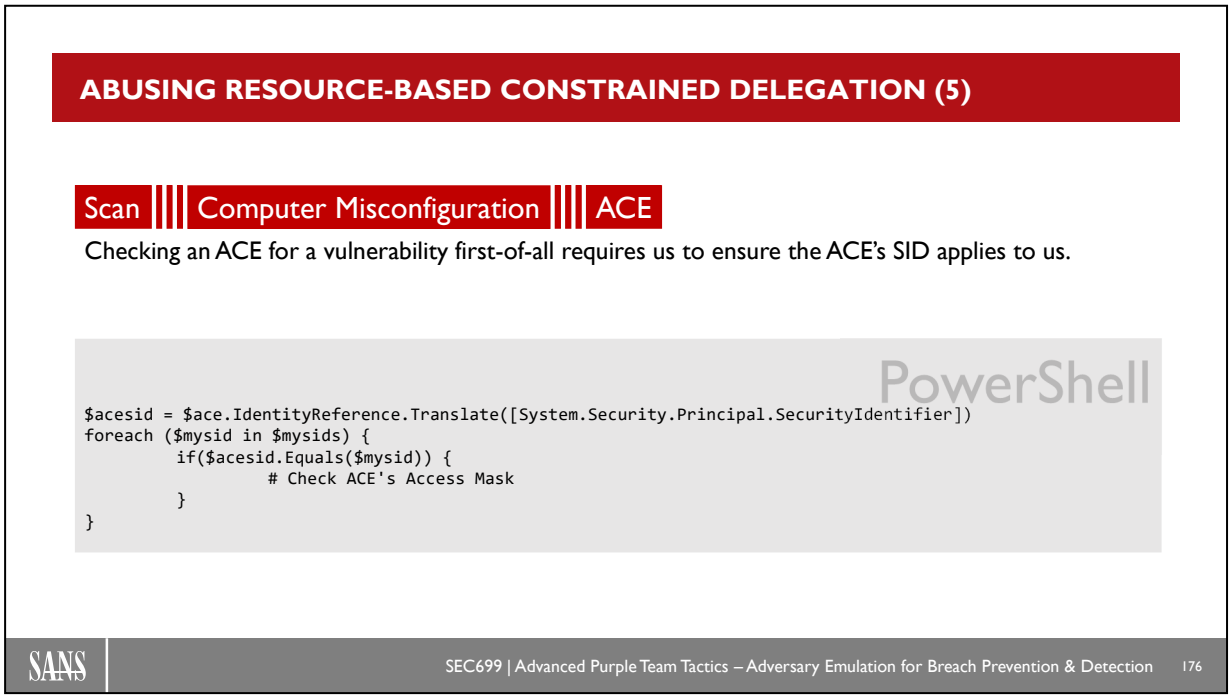
Identifying a misconfigured ACE requires us to enumerate them and perform checks one-by-one to identify either owner-hijacking possibilities or property-write permissions.

PowerShell

```
$adcomputers = Get-ADComputer -Filter * -Properties DistinguishedName
foreach ($adcomputer in $adcomputers){
         $acl = Get-Acl -Path "AD:$($adcomputer.DistinguishedName)"
         foreach($ace in $acl.Access){
                  # Check the ACE
         }
}
```

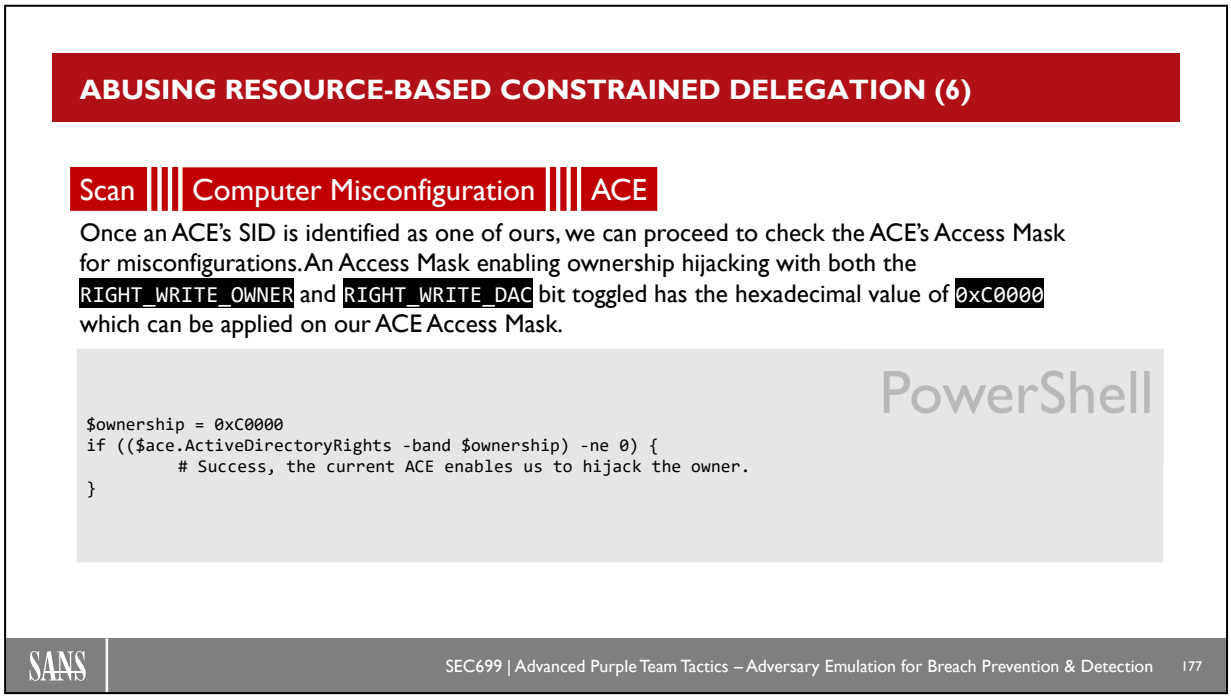

Abusing Resource-based Constrained Delegation (4)

Checking each computer for a misconfiguration requires us to retrieve all "Access Control Entries" related to the computers. An "Access Control Entry" is a permission relative to an object for a specific identity referenced by SID (hence our previous enumeration of valid SIDs). An object's "Access Control Entries" are grouped in a list called "Access Control List". The steps needed are hence the enumeration of all computers, for which we retrieve the ACL, for which we enumerate and check each ACE.



Abusing Resource-based Constrained Delegation (5)

The first check we have to perform for each ACE *(Access Control Entry)* is whether we are a subject of it. This is done by converting the ACE's SID to a valid SecurityIdentifier and checking whether it matches one of ours. In the situation where the ACE's SID matches one of ours, we can proceed to check the ACE's permissions (Access Mask) in a hope to identify a misconfiguration.

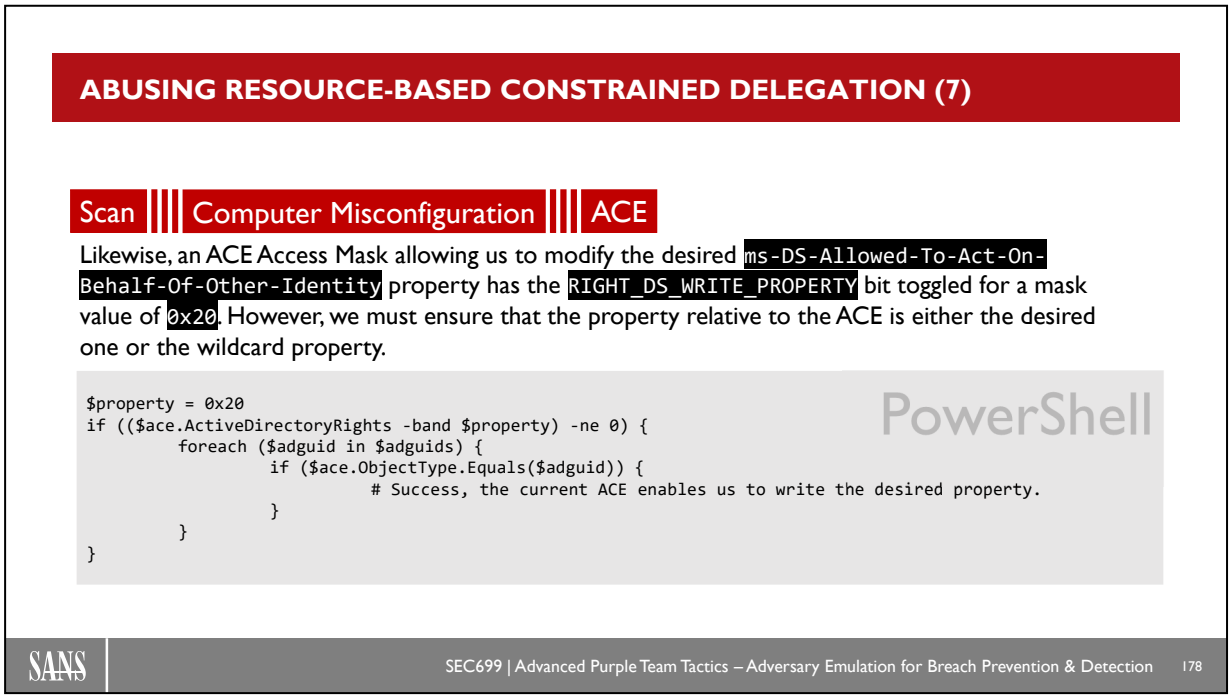


Abusing Resource-based Constrained Delegation (6)

A misconfiguration can occur in two situations:

1. Ownership Hijacking – The permissions allow us to grab the object's ownership and implicitly inherit of the object's full control.
2. Property Write-Rights – The permissions allow us to write one or all object properties.

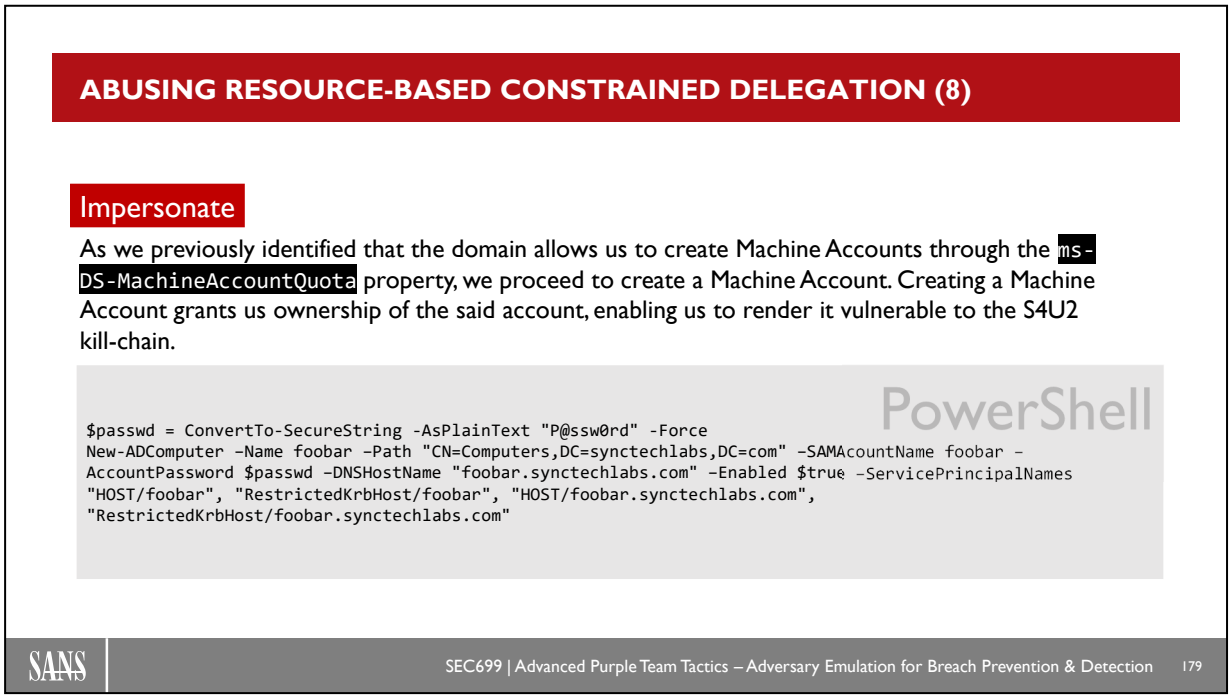
The permissions of an Access Control Entry are defined by the Access Mask, a 32-bit integer. The first misconfiguration, subject to ownership hijacking, can occur when either one of the *"RIGHT_WRITE_OWNER"* or *"RIGHT_WRITE_DAC"* right's bit is toggled. Combining these two bits into a mask of their own provides us with the value "0xC0000". Performing a binary and between both the Access Mask and this "ownership" mask provides us with a value which, if different from 0, indicates that we have the required rights to hijack the computer's ownership, granting us full control over the object in the process.



Abusing Resource-based Constrained Delegation (7)

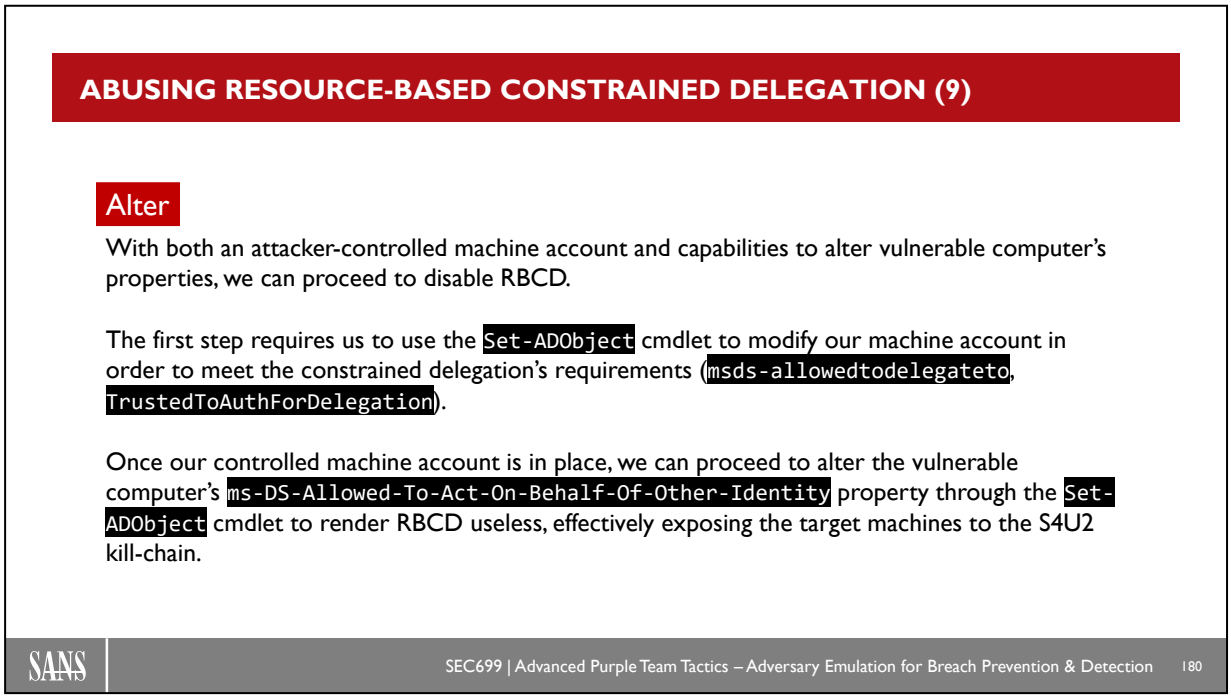
A second (more common) misconfiguration are too permissive rights allowing us to modify the target computer properties. The right we are interested in is the *"RIGHT_DS_WRITE_PROPERTY"* which, once toggled, gives us a mask value of "0x20". Like we did for the ownership hijacking mask, combining this property mask with the ACE's Access Mask outlines our permission to write a property.

In cases where we identify an access mask granting us write-rights on a property, we must further check whether the property itself is one of interest. This is done by checking whether the ACE's ObjectType (which references the subject property's GUID) matches either our desired property's GUID or the wildcard GUID.



Abusing Resource-based Constrained Delegation (8)
With the vulnerable computers being identified and our domain being confirmed as exploitable, we can proceed to create an attacker-controlled machine account. The role of this machine account will later be to request the Kerberos tickets used to impersonate anyone against the vulnerable computers through the S4U2 abuse flow.

Creating a machine account requires us to convert a password (needed later for Rubeus) to a secure string which will be given as argument for the computer creation as seen above.

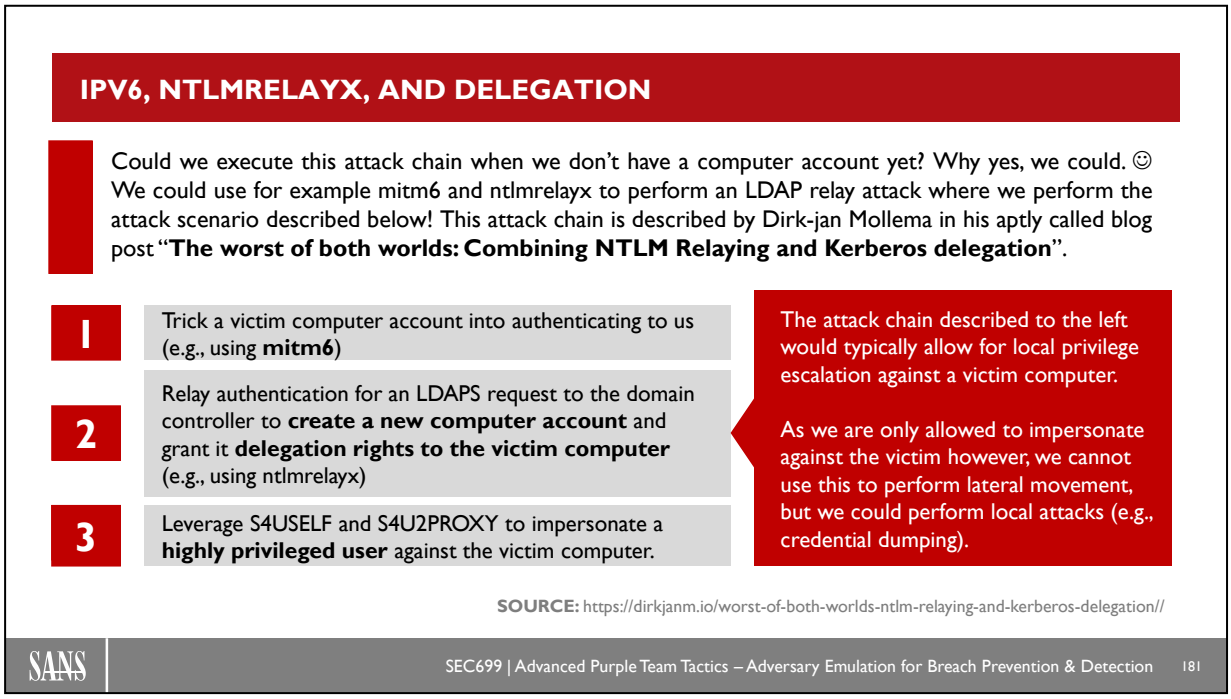


Abusing Resource-based Constrained Delegation (9)

With both an attacker-controlled machine account and capabilities to alter vulnerable computer's properties, we can proceed to disable RBCD.

The first step requires us to use the *"Set-ADObject"* cmdlet to modify our machine account in order to meet the constrained delegation's requirements (*"msds-allowedtodelegateto"*, *"TrustedToAuthForDelegation"*).

Once our controlled machine account is in place, we can proceed to alter the vulnerable computer's *"ms-DS-Allowed-To-Act-On-Behalf-Of-Other-Identity"* property through the *"Set-ADObject"* cmdlet to render RBCD useless, effectively exposing the target machines to the S4U2 kill-chain.



IPv6, Ntlmrelayx, and Delegation

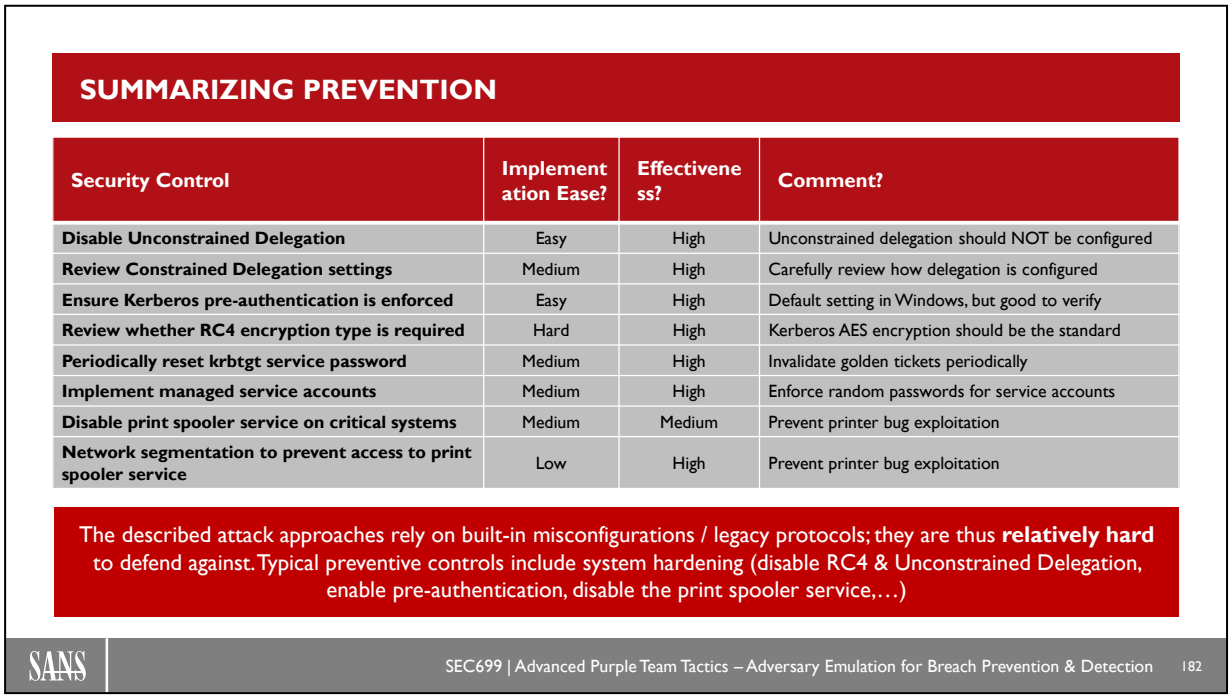
Could we execute this attack chain when we don't have a computer account yet? Why yes, we could! We could use for example mitm6 and ntlmrelayx to perform an LDAP relay attack where we perform the attack scenario described below! This attack chain is described by Dirk-jan Mollema in his aptly called blog post "The worst of both worlds: Combining NTLM Relaying and Kerberos delegation".

In order to run the attack, the following steps need to be taken:

1. Trick a victim computer account into authenticating to us (e.g., using mitm6)
2. Relay authentication for an LDAPS request to the domain controller to create a new computer account and grant it delegation rights (modify the *msDS-AllowedToActOnBehalfOfOtherIdentity* attribute) to the victim computer (e.g., using ntlmrelayx)
3. Leverage S4USELF and S4U2PROXY to impersonate a highly privileged user against the victim computer. We could do this, for example, by using Rubeus or getST.py (impacket).

The attack chain described to the left would typically allow for local privilege escalation against a victim computer. As we are only allowed to impersonate against the victim, however, we cannot use this to perform lateral movement, but we could perform local attacks (e.g., credential dumping).

The full blog post can be found here:
https://dirkjanm.io/worst-of-both-worlds-ntlm-relaying-and-kerberos-delegation

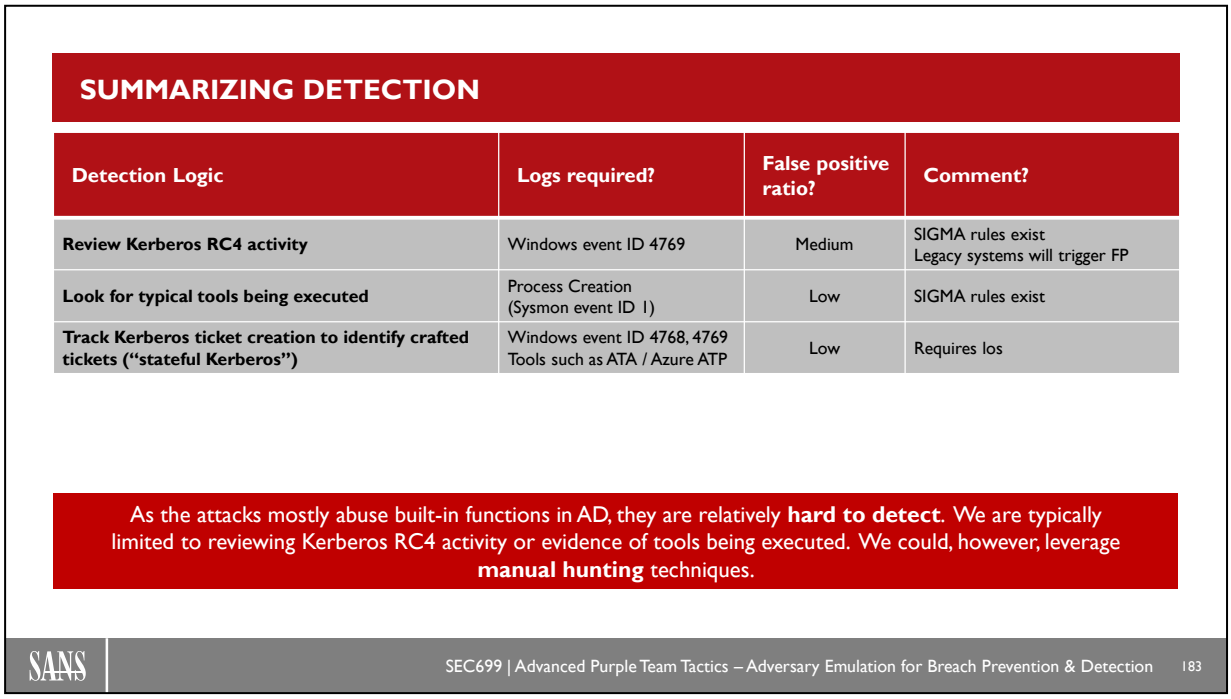


Summarizing Prevention

The described attack approaches rely on built-in misconfigurations / legacy protocols; they are thus relatively hard to defend against. Typical preventive controls include:

- Disable Unconstrained Delegation
- Review Constrained Delegation settings
- Ensure Kerberos pre-authentication is enforced
- Review whether RC4 encryption type is required
- Periodically reset krbtgt service password
- Implement managed service accounts
- Disable print spooler service on critical systems
- Network segmentation to prevent access to print spooler service

Note that several of these controls could impact legacy systems and should thus be carefully considered and tested.



Summarizing Detection

As the attacks mostly abuse built-in functions in AD, they are relatively hard to detect. We are typically limited to:

- Reviewing our environment for Kerberos RC4 activity (using Windows event ID 4769)
- Looking for proof of tool execution (using Process Creation / Sysmon event ID 1)
- Tracking Kerberos ticket creation and usage to create a "stateful" Kerberos (event ID 4768, 4769). This is effective against crafted tickets (e.g., silver or golden tickets).

Finally, it's worth performing manual hunting in the environment, which is described by Roberto Rodriguez (Cyb3rWarD0g). He wrote a highly interesting article that can be found here: https://posts.specterops.io/hunting-in-active-directory-unconstrained-delegation-forests-trusts-71f2b33688e1

Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.

EXERCISE: (RESOURCE-BASED) CONSTRAINED DELEGATION ATTACKS

Please refer to the workbook for further instructions on the exercise!

This page intentionally left blank.

Course Roadmap

- Introduction & Key Tools
- Initial Access
- **Lateral Movement**
- Persistence
- Azure AD & Emulation Plans
- Adversary Emulation Capstone

SEC699.3

Active Directory Enumeration
BloodHound Enumeration
Exercise: Analyzing BloodHound Attack Chains
Credential Dumping
LSASS Credential Stealing Techniques
Exercise: Stealing Credentials from LSASS
Stealing Credentials Without Touching LSASS
Exercise: Internal Monologue in NTLMv1 Downgrades
Stealing NTLMv2 Challenge-Response
Exercise: Creative NTLMv2 Challenge-Response Stealing
Kerberos Attacks
Kerberos Refresh
Unconstrained Delegation Attacks
Exercise: Unconstrained Delegation Attacks
(Resource-Based) Constrained Delegation
Exercise: (Resource-Based) Constrained Delegation
Conclusions

This page intentionally left blank.

CONCLUSIONS FOR THIS SECTION – DETECTION (1)

Detection Logic	Applicable Techniques	Logs required?	False positive ratio?
Look for execution of tools (SharpHound, Mimikatz, Rubeus, vssadmin, reg save,…)	T1087 – Account Discovery T1482 – Domain Trust Discovery T1069 – Permissions Groups Discovery T1003 – Credential Dumping T1558/003 – Kerberoasting T1550/003 – Pass-the-Ticket	Process Creation (Sysmon event ID 1)	Low
Look for client-side LDAP/LDAPS and SMB connectivity	T1087 – Account Discovery T1482 – Domain Trust Discovery T1069 – Permissions Groups Discovery	Network Connection (Sysmon event ID 3)	Low
Look for server-side AD enumeration		Windows event ID 5145	Low
Look for LSASS injection / interaction	T1003 – Credential Dumping	ImageLoaded (Sysmon event ID 7) CreateRemoteThread (Sysmon event ID 8) ProcessAccess (Sysmon event ID 10)	Average
Look for Mimikatz service	T1003 – Credential Dumping	Service Creation (Windows event 7045)	Low

This page intentionally left blank.

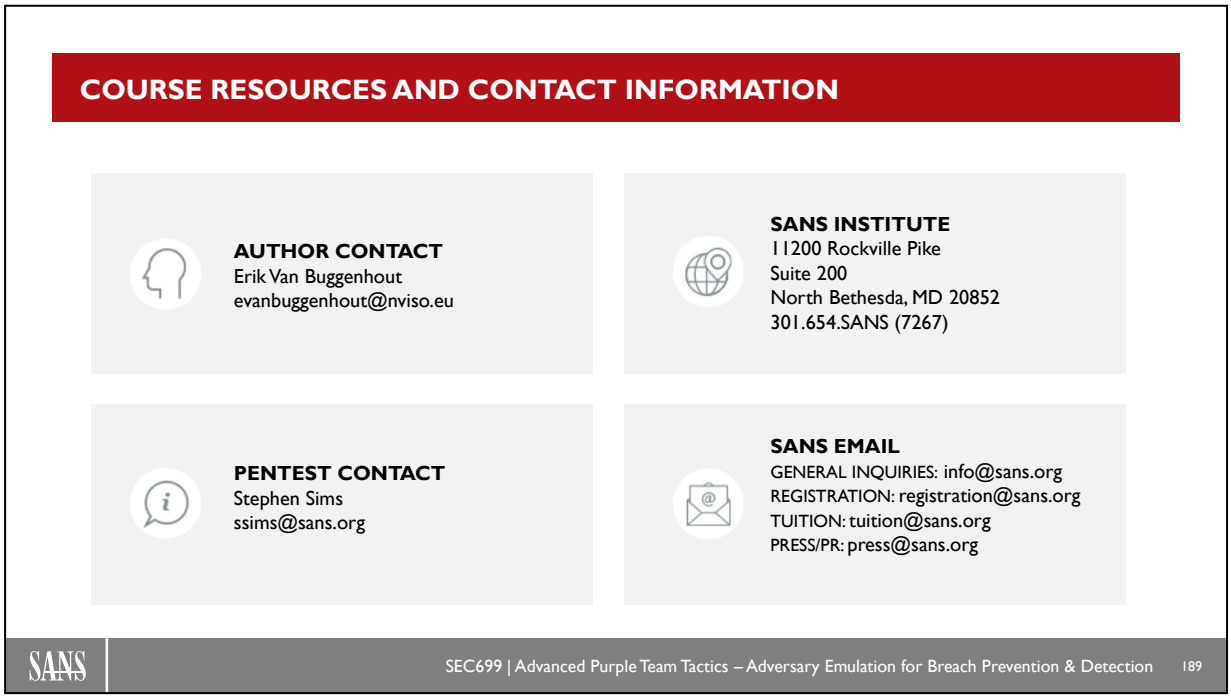
CONCLUSIONS FOR THIS SECTION – DETECTION (2)

Detection Logic	Applicable Techniques	Logs required?	False positive ratio?
Detect NTLM downgrade attacks	T1003 – Credential Dumping	Registry interaction (System event ID 12, 13, 14)	Very low
Detect object access using replication privileges	T1003 – Credential Dumping	Windows event ID 4662	Low
Review Kerberos RC4 activity	T1558/003 – Kerberoasting T1550/003 – Pass-the-Ticket	Windows event ID 4769	Average
Track Kerberos ticket creation to identify crafted tickets ("stateful Kerberos")	T1558/003 – Kerberoasting T1550/003 – Pass-the-Ticket	Windows event ID 4768, 4769 Tools such as ATA / Azure ATP	Low

Priority log sources for this section

- Sysmon Process Creation (Event ID 1)
- Sysmon lsass manipulation (Event ID 7, 8 & 10)
- Kerberos activity (Event ID 4768, 4769)
- Sysmon registry (Event ID 12, 13 & 14)

This page intentionally left blank.



This page intentionally left blank.