© SANS Institute 2024

SEC670 | RED TEAMING TOOLS: DEVELOPING WINDOWS IMPLANTS, SHELLCODE, COMMAND AND CONTROL

670.4

Persistence: Die Another Day

SANS | GIAC CERTIFICATIONS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

© SANS Institute 2024

© 2023 Jonathan Reiter. All rights reserved to Jonathan Reiter and/or SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE (DEFINED BELOW) ASSOCIATED WITH THE SANS INSTITUTE COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND THE ESCAL INSTITUTE OF ADVANCED TECHNOLOGIES, INC. /DBA SANS INSTITUTE (“SANS INSTITUTE”) FOR THE COURSEWARE. BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware means all printed materials, including course books and lab workbooks, slides or notes, as well as any digital or other media, audio and video recordings, virtual machines, software, technology, or data sets distributed by SANS Institute to User for use in the SANS Institute course associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. USER FURTHER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree to the terms of this CLA, User should not access the Courseware. User may return the Courseware to SANS Institute for a refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form to any person or entity without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS Institute may suspend and/or terminate User’s access to and require immediate return of any Courseware in connection with any (i) material breaches or material violation of this CLA or general terms and conditions of use agreed to by User; (ii) technical or security issues or problems caused by User that materially impact the business operations of SANS Institute or other SANS Institute customers, or (iii) requests by law enforcement or government agencies.

SANS Institute acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There’s an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP® and PMBOK® are registered trademarks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

REMnux® is a registered trademark of Zeltser Security Crop. Used with permission.

VMware Workstation Pro®, VMWare Workstation Player®, VMWare Fusion®, and VMware Fusion Pro® are registered trademarks of VMware, Inc. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

Courseware licensed to User under this Agreement may be subject to export laws and regulations of the United States of America and other jurisdictions. User warrants he or she is not listed (i) on any sanction programs list maintained by the U.S. Office of Foreign Assets Control within the U.S. Treasury Department (“OFAC”), or (ii) denied party list maintained by the U.S. Bureau of Industry and Security within the U.S. Department of Commerce (“BIS”). User agrees to not allow access to any Courseware to any person or entity in a U.S. embargoed country or in violation of a U.S. export control law or regulations. User agrees to cooperate with SANS Institute as necessary for SANS Institute to comply with export requirements and recordkeeping required by OFAC, BIS or other governmental agency.

All reference links are operational in the browser-based delivery of the electronic workbook.

© SANS Institute 2024

SEC670.4

Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control

SANS

Persistence: Die Another Day

© 2024 Jonathan Reiter | All Rights Reserved | Version J01_03

Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control: 670.4
Welcome to Section 4 of SEC670. In this section, we will discuss a number of ways that you could persist on target.

© SANS Institute 2024

Table of Contents (1)

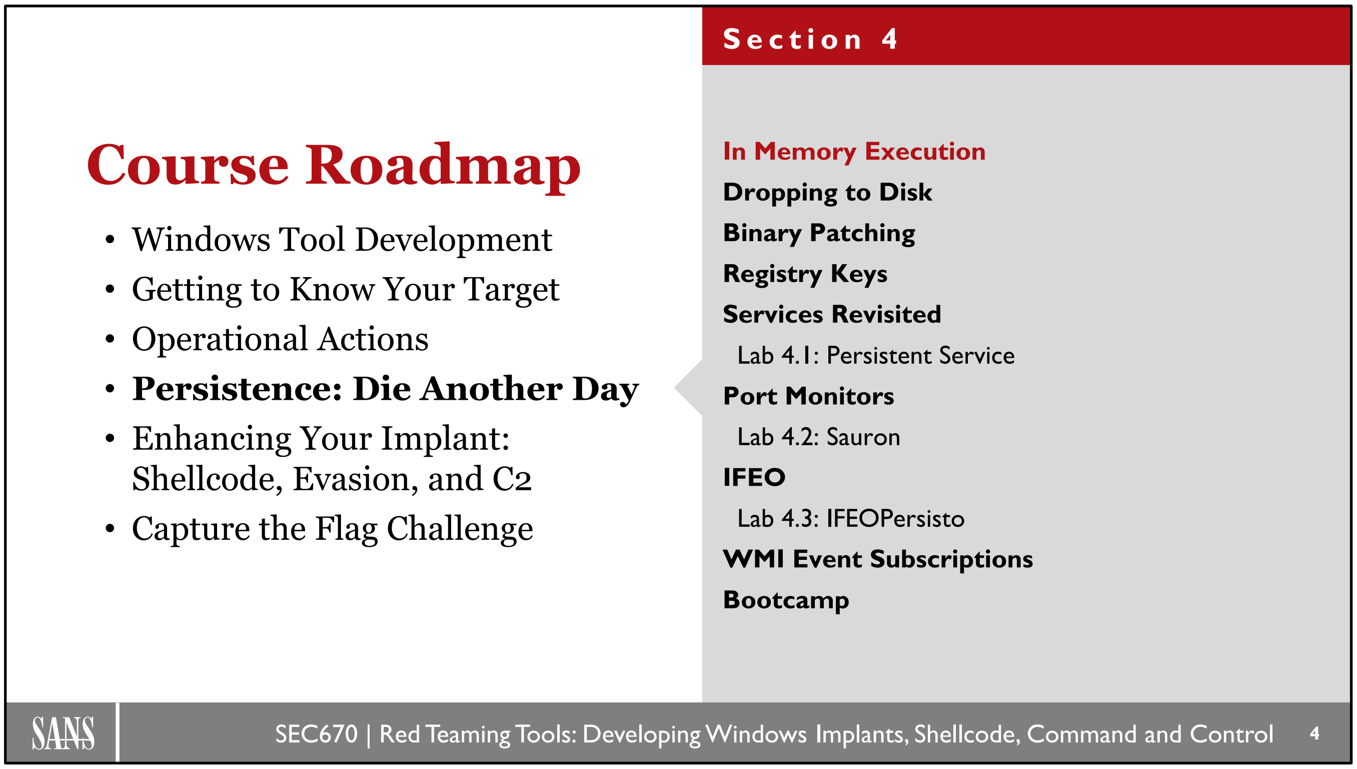
This page intentionally left blank.

© SANS Institute 2024

Table of Contents (2)

This page intentionally left blank.

© SANS Institute 2024



Course Roadmap

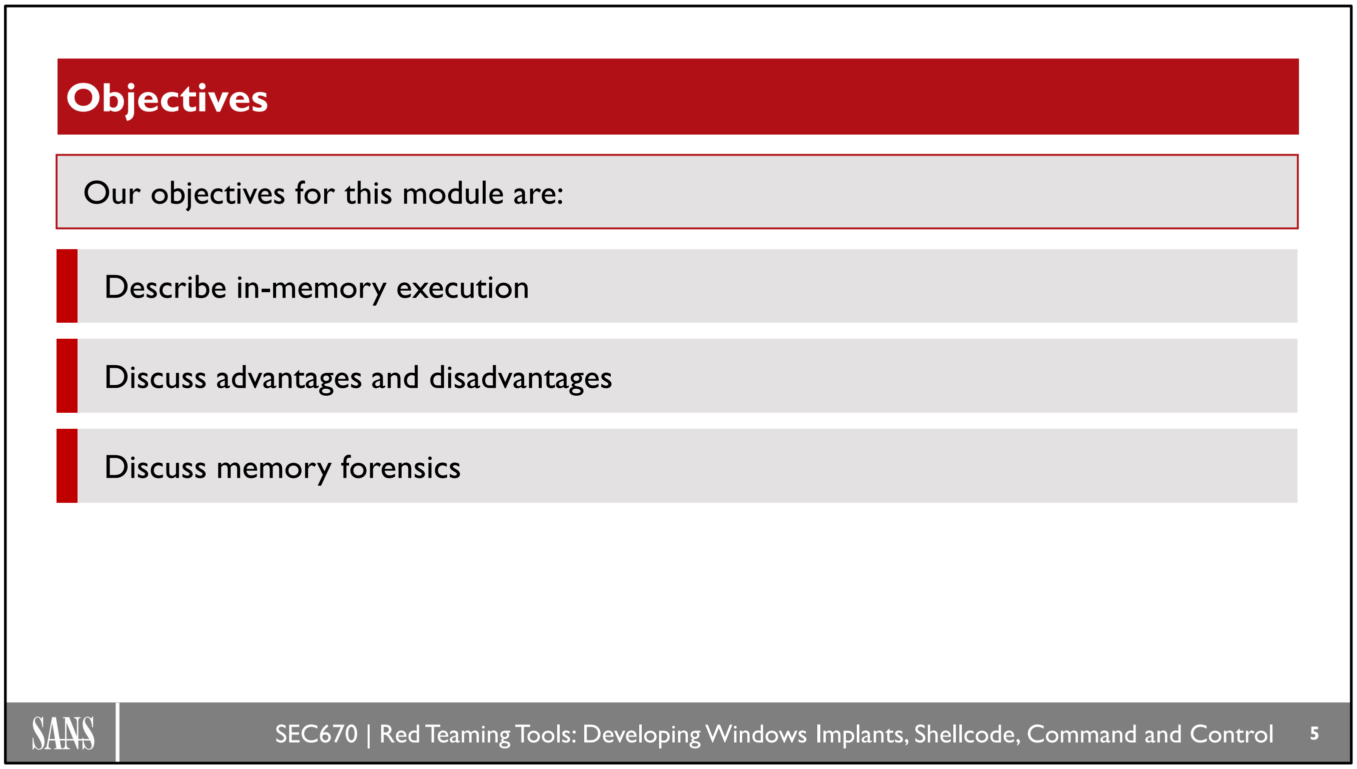
- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

In Memory Execution
In this module, we will discuss what in memory execution is and how it can benefit us. We will also look at some disadvantages with only living in memory. Hint: it is volatile.

© 2024 Jonathan Reiter

© SANS Institute 2024



Objectives

Our objectives for this module are:

- Describe in-memory execution
- Discuss advantages and disadvantages
- Discuss memory forensics

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 5

Objectives
The objectives for this module are to describe what in-memory execution is, discuss some advantages of it, as well as a few disadvantages, before wrapping it all up with a brief note on memory forensics.

© SANS Institute 2024

In Memory Execution

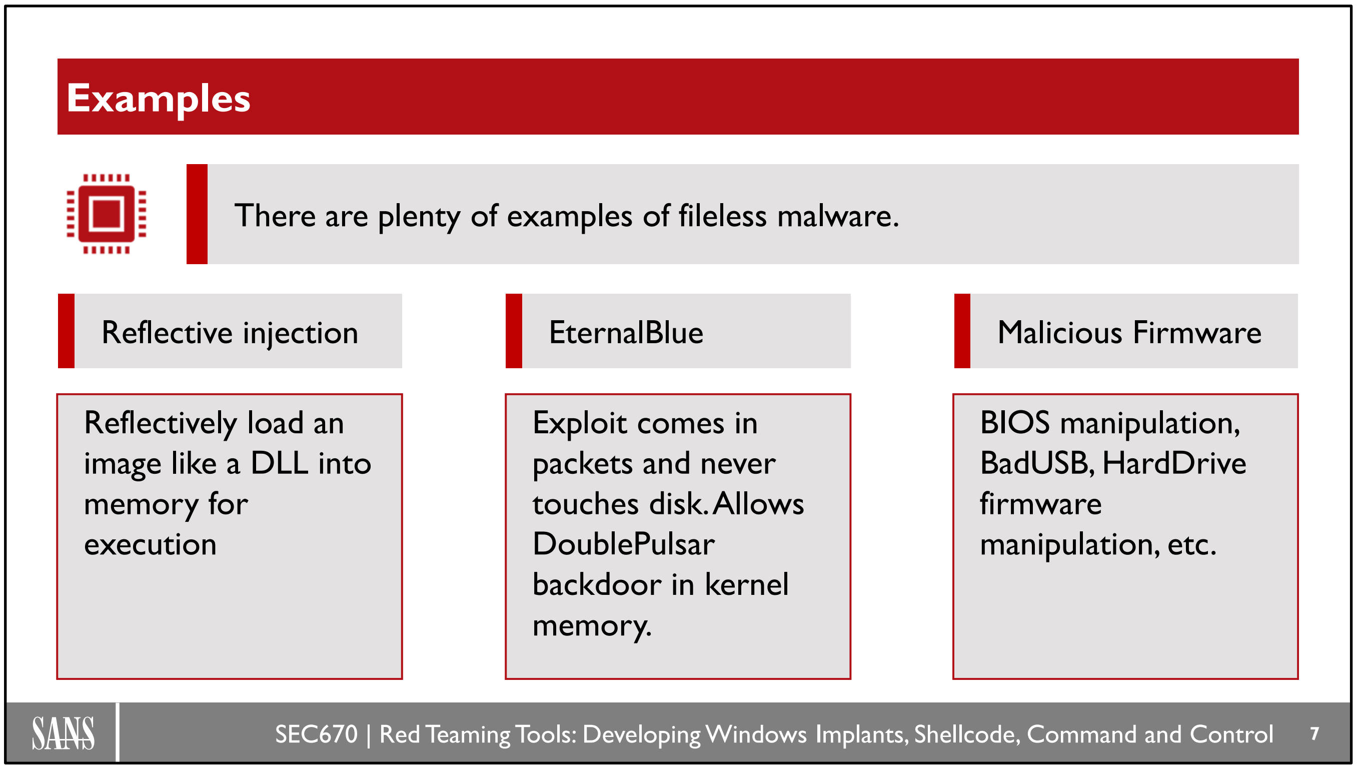
Code execution not backed by a file on disk

In memory means exactly that—code is running in memory. For our purposes, however, we are going to define it as having an image in memory that is not backed by a file residing on the target's disk. Perhaps you have heard of fileless malware?

In Memory Execution
We could say that in-memory execution is when an executable image is mapped into memory and then a main thread is created to kick off the instructions located at the address of entry point, but we are not going to say that. Instead, we are going to say that in-memory execution is the execution of code, like shellcode or an executable image, executing in memory that is not backed by anything on the disk. Effectively, this is fileless malware. Fileless malware is malware that only exists in memory without ever leaving a single file on disk. Fileless code execution is very beneficial for the attacker. It is also more difficult to detect, which is another reason why fileless malware has been and is so popular. There have been many vulnerabilities in the past that attackers have been able to take advantage of for fileless execution. A prime situation for fileless code execution comes in the form of specially crafted packets that hit the NIC first and foremost. Kernel drivers, such as tcpip.sys, http.sys, netio.sys and others, are responsible for making sense of the data the NIC is holding in its buffers. That process presents an opportunity for kernel-level execution. Let us look a few examples.

© 2024 Jonathan Reiter

© SANS Institute 2024



Examples

There are plenty of examples of fileless malware.

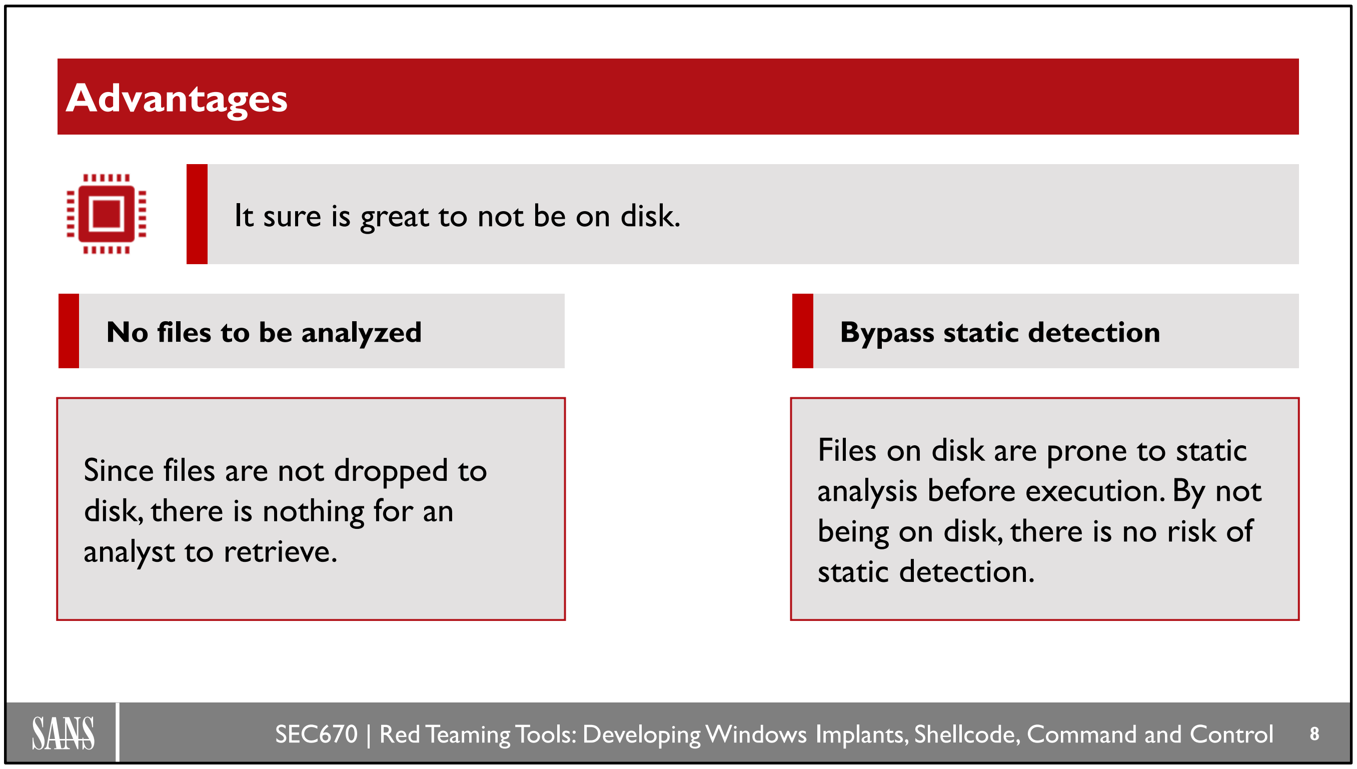
Reflective injection	EternalBlue	Malicious Firmware
Reflectively load an image like a DLL into memory for execution	Exploit comes in packets and never touches disk. Allows DoublePulsar backdoor in kernel memory.	BIOS manipulation, BadUSB, HardDrive firmware manipulation, etc.

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 7

Examples
When it comes to examples of fileless malware and methods to achieve fileless execution, there is no shortage. Perhaps one of the more famous examples is WannaCry with its usage of EternalBlue. If you recall, EternalBlue came in the form of specially crafted network packets taking advantage of a 2017 SMB vulnerability. The nice thing about packets is they do not touch disk at all unless someone is saving them off using a packet capture tool like Wireshark. The backdoor that followed up with the exploit was dubbed DoublePulsar, which resided in the kernel. Another example of fileless malware or implants could come in the form of manipulating the BIOS. Not many EDR/AV solutions are built to check the BIOS. Same goes for performing a malicious firmware update to the controller of a hard drive. Going that low is incredibly stealthy and can provide amazing persistence and hiding capabilities. Red Balloon Security offers a unique hard drive challenge for its applicants, and it is quite the challenge! Also, you can reflectively inject an executable image into another process to achieve fileless execution. We will take a deeper look at Reflective DLL Injection during Section 5.

References:
https://awakesecurity.com/glossary/fileless-malware/
https://gbhackers.com/fileless-malware-wmi-eternalblue/

© SANS Institute 2024



Advantages

It sure is great to not be on disk.

No files to be analyzed

Since files are not dropped to disk, there is nothing for an analyst to retrieve.

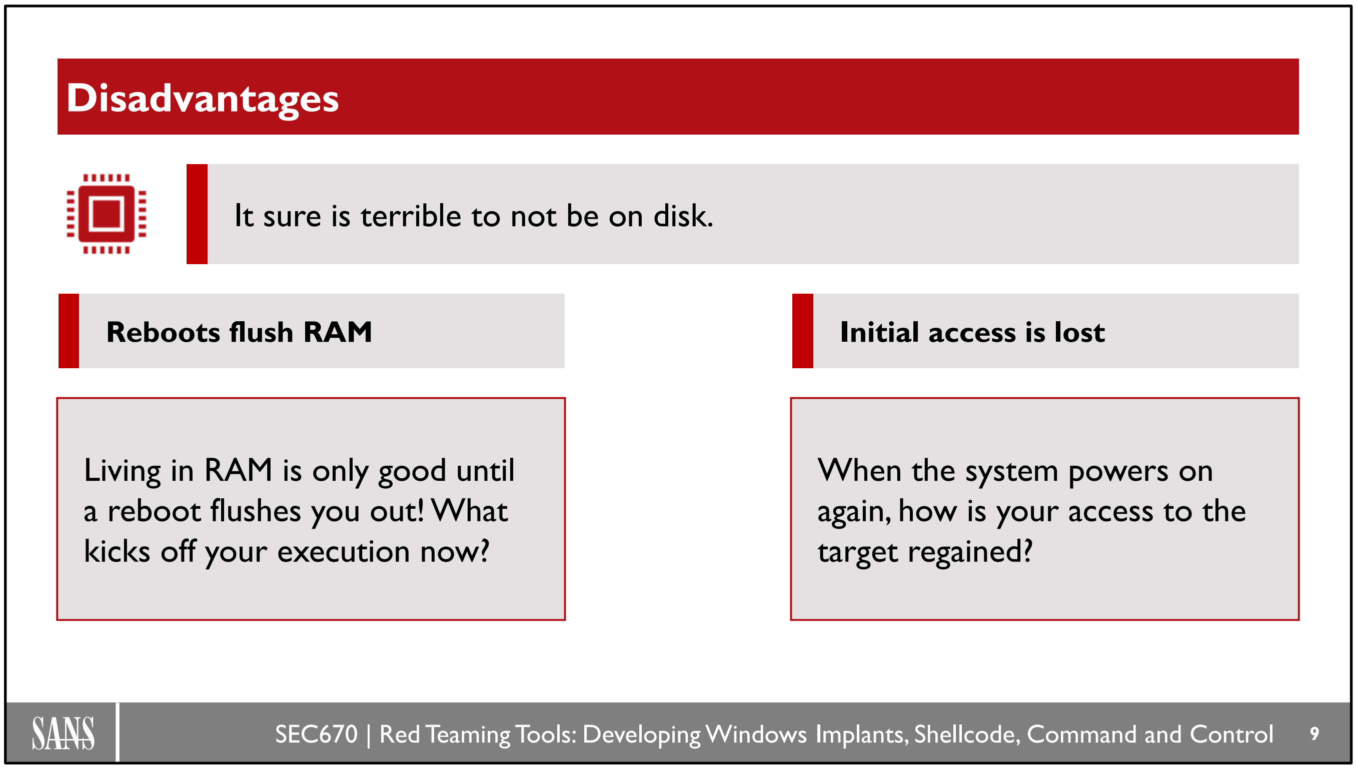
Bypass static detection

Files on disk are prone to static analysis before execution. By not being on disk, there is no risk of static detection.

Advantages
The biggest advantage for fileless malware is the fact that nothing is dropped to disk. The fact that certain programs, system files, tools, etc. cannot be blocked by IT staff without hindering support keeps enabling fileless attacks. Have you seen an organization that has completely blocked and/or removed PowerShell? Even without PowerShell you can still execute PowerShell; PowerShell without PowerShell. A forensic analyst or a reverse engineer would have nothing to do without files or without a memory dump. Another advantage is that by not having anything on disk, you will not be the victim of static analysis by some AV/EDR solution. There is nothing to hash, no bytes on disk to scan, nothing. The fact that fileless techniques are so powerful has forced EDRs and AV solutions to step up their game with behavior detection, memory scanning, and leveraging machine learning for their next-gen products. Even still, there is no single product out there that will always catch everything 100% of the time. It would be too resource intensive for products to constantly scan all memory regions of every process all the time.

© 2024 Jonathan Reiter

© SANS Institute 2024



Disadvantages

It sure is terrible to not be on disk.

Reboots flush RAM

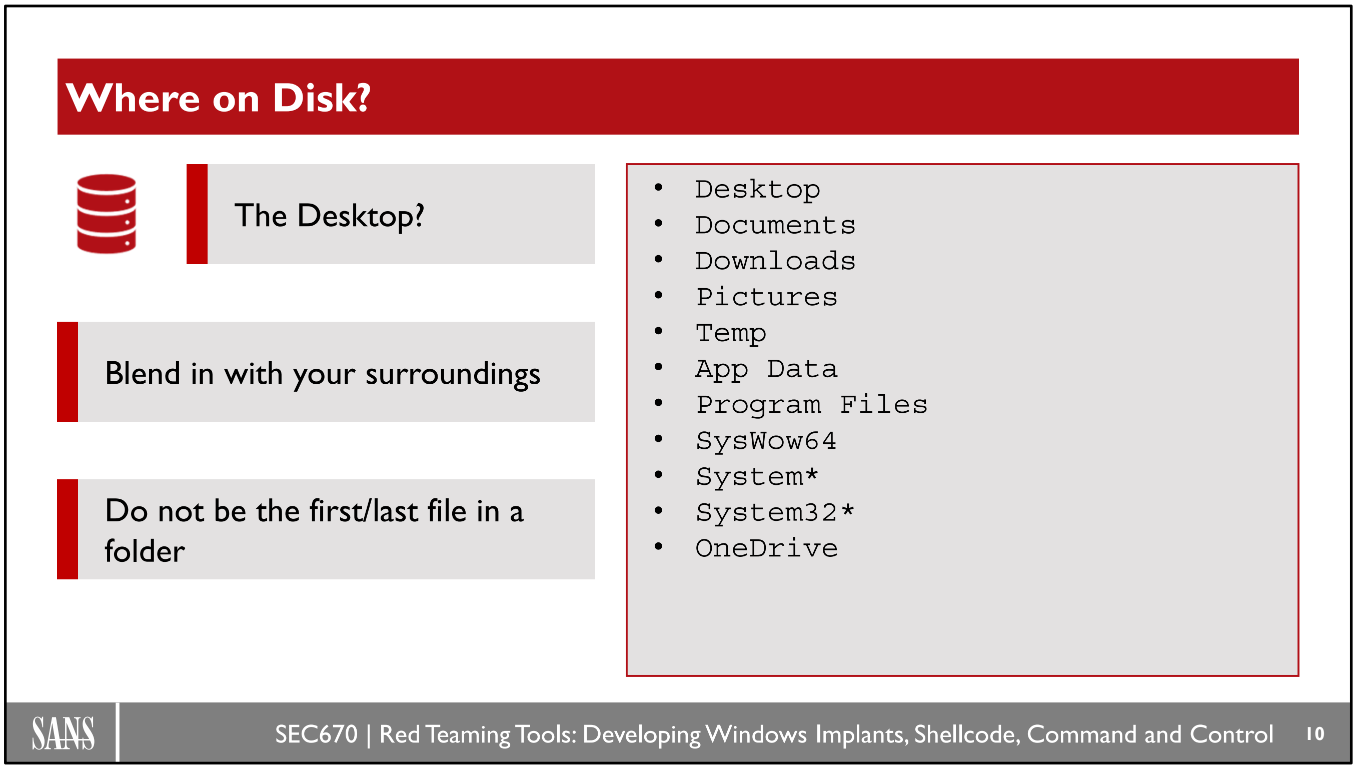
Living in RAM is only good until a reboot flushes you out! What kicks off your execution now?

Initial access is lost

When the system powers on again, how is your access to the target regained?

Disadvantages
How do you persist if you are only living in memory? Spoiler alert: you really cannot. This is arguably the biggest disadvantage when it comes to fileless malware. Machines are rebooted from time to time for various reasons: installing updates, power failure, hardware upgrades, scheduled downtime, etc. If you are someone who really enjoys bragging about system uptime, then fileless malware thanks you. Bottom line, though—a system will be rebooted at some point, so you must be prepared for this event. If you are not prepared, then your initial access is gone in a cloud of smoke. How do you get back on the target? Do you attempt to throw your exploit again to get you back on the system? The risks of getting caught go up much more significantly by throwing another exploit at the target. If the target is of no real importance to you, then you just move on and chalk it up as a temporary setback. However, if that system is of high value, then you need to maintain access to it, and that could mean dropping something to disk. It is often said that having two methods of persistence is just one method. Having just one method of persistence is really not having any at all.

© SANS Institute 2024



Where on Disk?

The Desktop?

Blend in with your surroundings

Do not be the first/last file in a folder

- `Desktop`
- `Documents`
- `Downloads`
- `Pictures`
- `Temp`
- `App Data`
- `Program Files`
- `SysWow64`
- `System*`
- `System32*`
- `OneDrive`

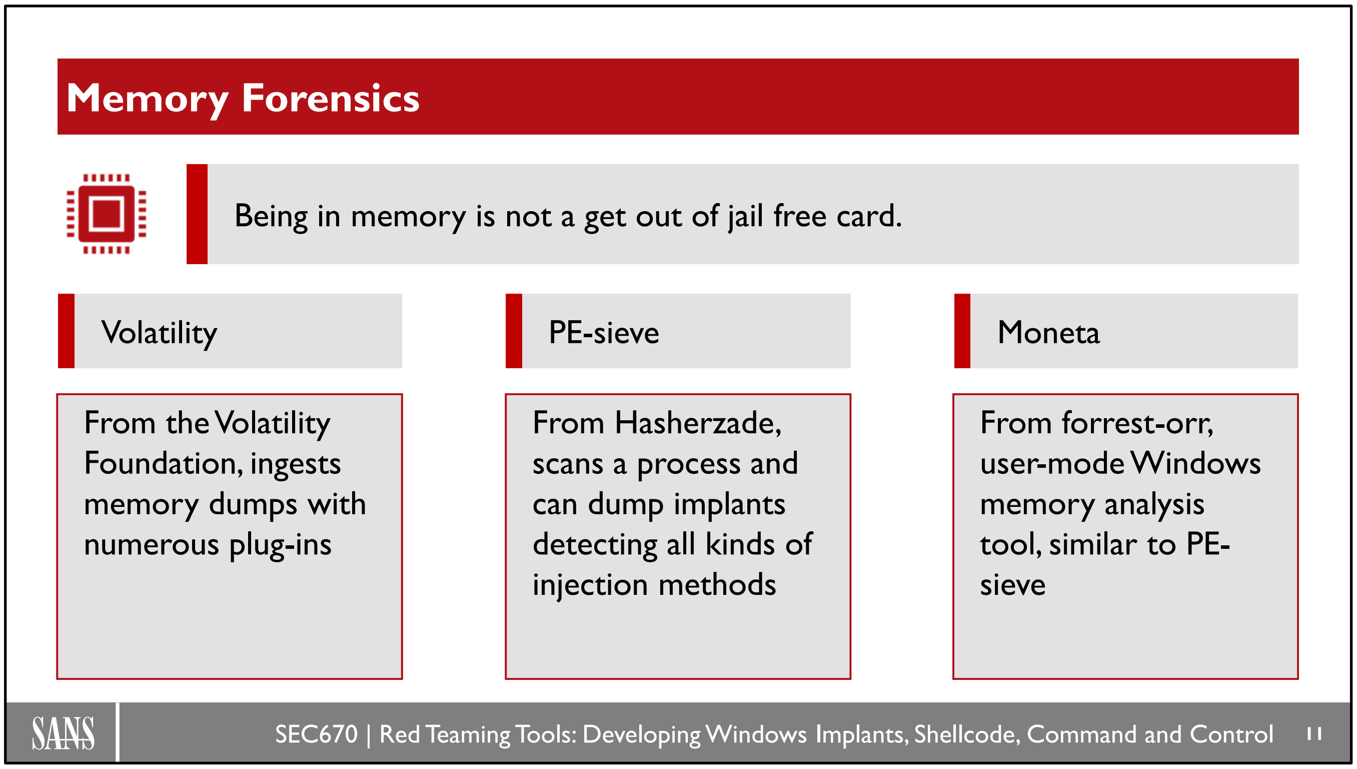
SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 10

Where on Disk?
Imagine this scenario: you are running only in memory at the moment and your implant is monitoring for system shutdown events or changes to the power status, like if a laptop is no longer being charged. The system notifies you that a shutdown is pending, and you must maintain your access to this system because there is still intel on it that you have yet to collect. You make the decision to drop to disk so when the system powers back on, your file is triggered, and your access is renewed. The question you need to ask yourself is: Where do I drop to disk? This is a vulnerable moment for your implant, but it is also a necessity, so we must choose a smart location. The location could be entirely dependent on the system's environment, like what third party applications are present, what folders do you have write access to, and if there is cloud storage attached that you can drop into?

There are some general tactics and criteria you could use when looking for a possible location. Find a folder with a decent number of files in it. This would allow you to hide in plain sight somewhere past the halfway point in the listing. You do not necessarily want to be first, nor do you want to be last. You should avoid locations that are frequently browsed by the user like the Desktop, Documents, or Downloads locations. There is one caveat to the Desktop: some users like to store everything on the Desktop. They can be so cluttered that you can hardly see the user's background wallpaper. If that is the case, you could consider that as a possible location.

© 2024 Jonathan Reiter

© SANS Institute 2024



Memory Forensics

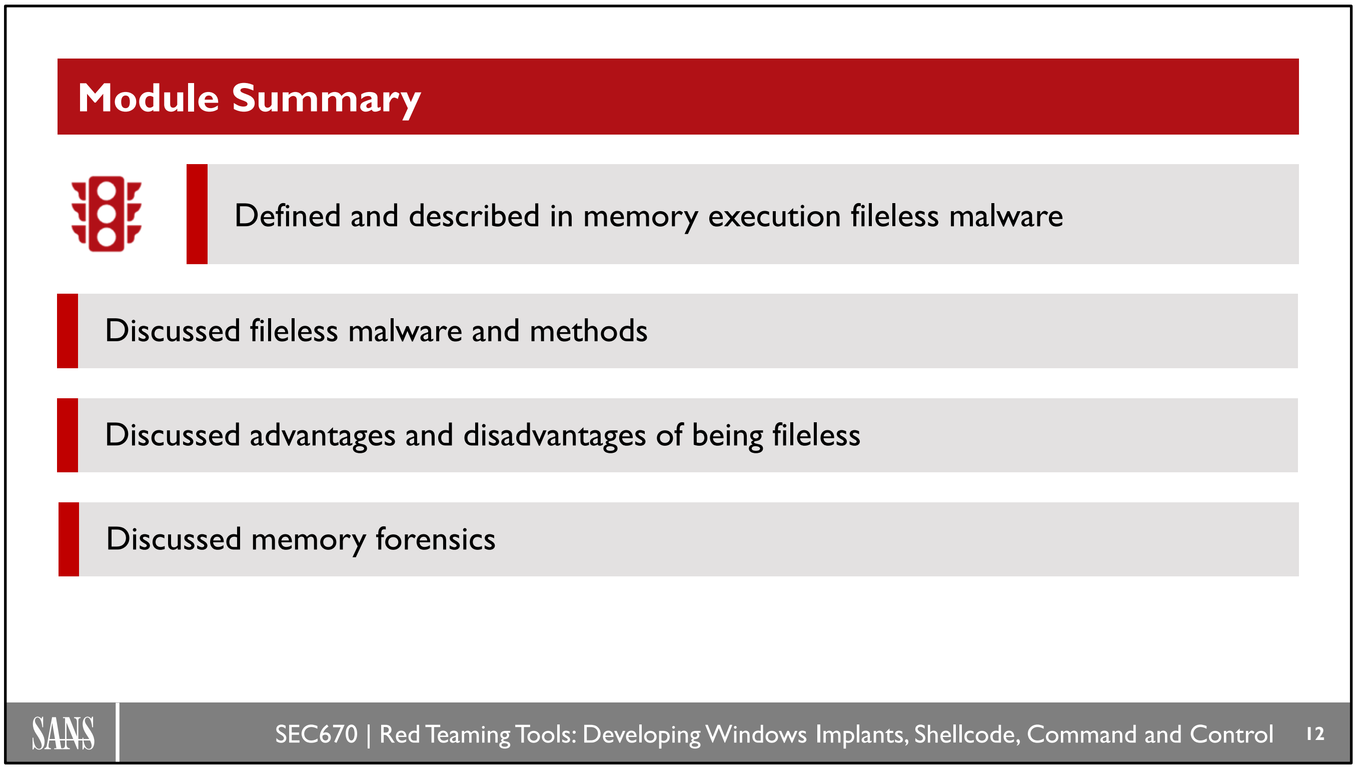
Being in memory is not a get out of jail free card.

Volatility	PE-sieve	Moneta
From the Volatility Foundation, ingests memory dumps with numerous plug-ins	From Hasherzade, scans a process and can dump implants detecting all kinds of injection methods	From forrest-orr, user-mode Windows memory analysis tool, similar to PE-sieve

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 11

Memory Forensics
Almost everything you do on a Windows system can be logged by something. Even if a certain Windows tool does not catch your activity, highly motivated memory analysts with time on their hands will. There are several tools that can run live on a system, like PE-sieve and Moneta. They are both amazing tools and hiding from them has been seen as a challenge by researchers and developers alike. Only recently was Moneta bypassed by the awesome work of the MDSecLabs crew with their C2 Framework Nighthawk. Both PE-sieve and Moneta can scan processes and report suspicious findings, if any. Volatility is a Python framework geared at finding practically anything that was still resident in memory at the time of capture. They also host plug-in contests and some truly amazing plug-ins come from that. There are many more tools out there that offer similar capabilities to the ones noted on the slide. Bottom line, it is only a matter of time before you get caught. One great advantage here is that security products that are performing memory scanning simply cannot scan all regions of memory of every process every second of the day. It must be targeted and done in such a way so that it does not cause a performance hit for the user. There is nothing users hate more than a slow system. If a security product is causing high levels of degradation on system performance, odds are the user will remove it or ask that it be removed.

© SANS Institute 2024



Module Summary

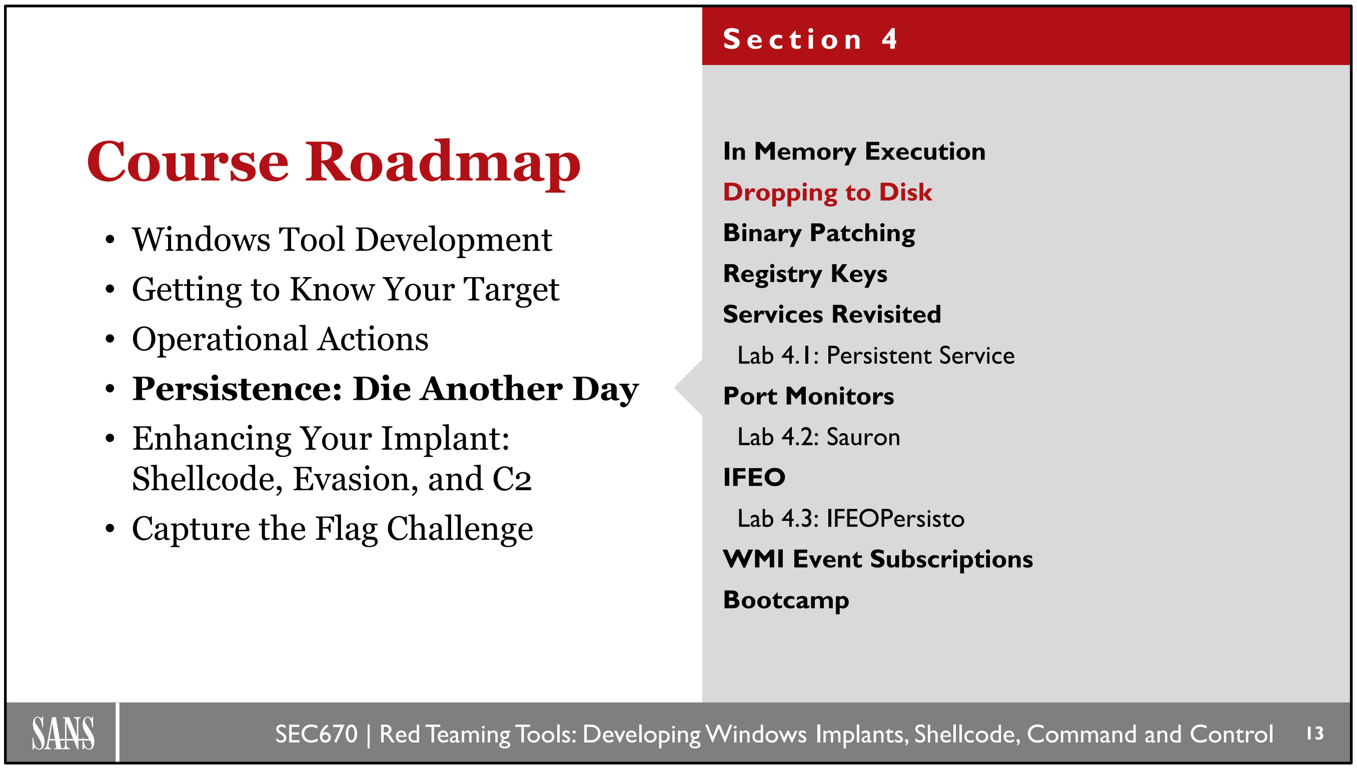
- Defined and described in memory execution fileless malware
- Discussed fileless malware and methods
- Discussed advantages and disadvantages of being fileless
- Discussed memory forensics

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 12

Module Summary
In this module we covered what in memory execution is as it relates to implant development. We also discussed fileless malware and some popular examples, like WannaCry and EternalBlue SMB vulnerability. We also took a brief look at some advantages and disadvantages of only being in RAM. Are you ready for a shutdown/reboot?

© 2024 Jonathan Reiter

© SANS Institute 2024



Course Roadmap

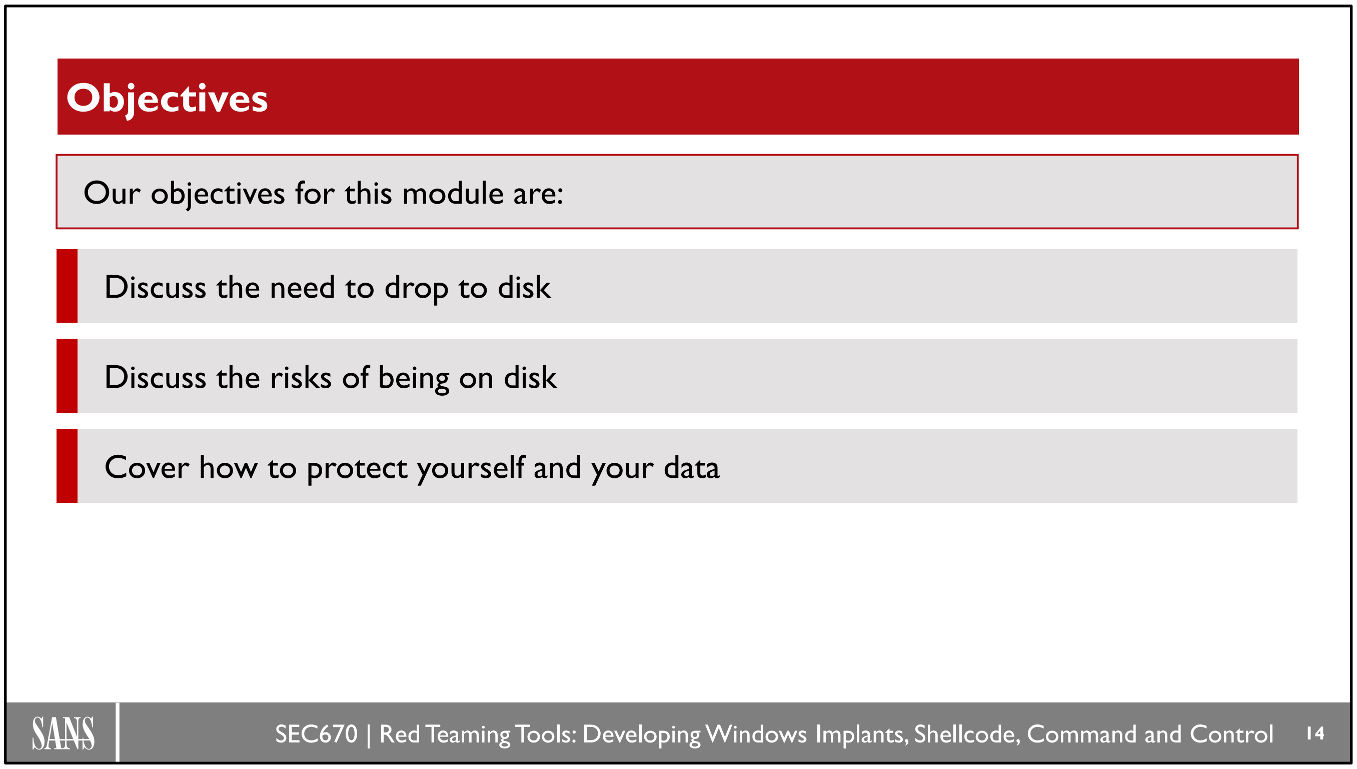
- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 13

In this module, we will discuss why you would need to drop something to disk, where to drop, cleaning up, and more.

© SANS Institute 2024



Objectives

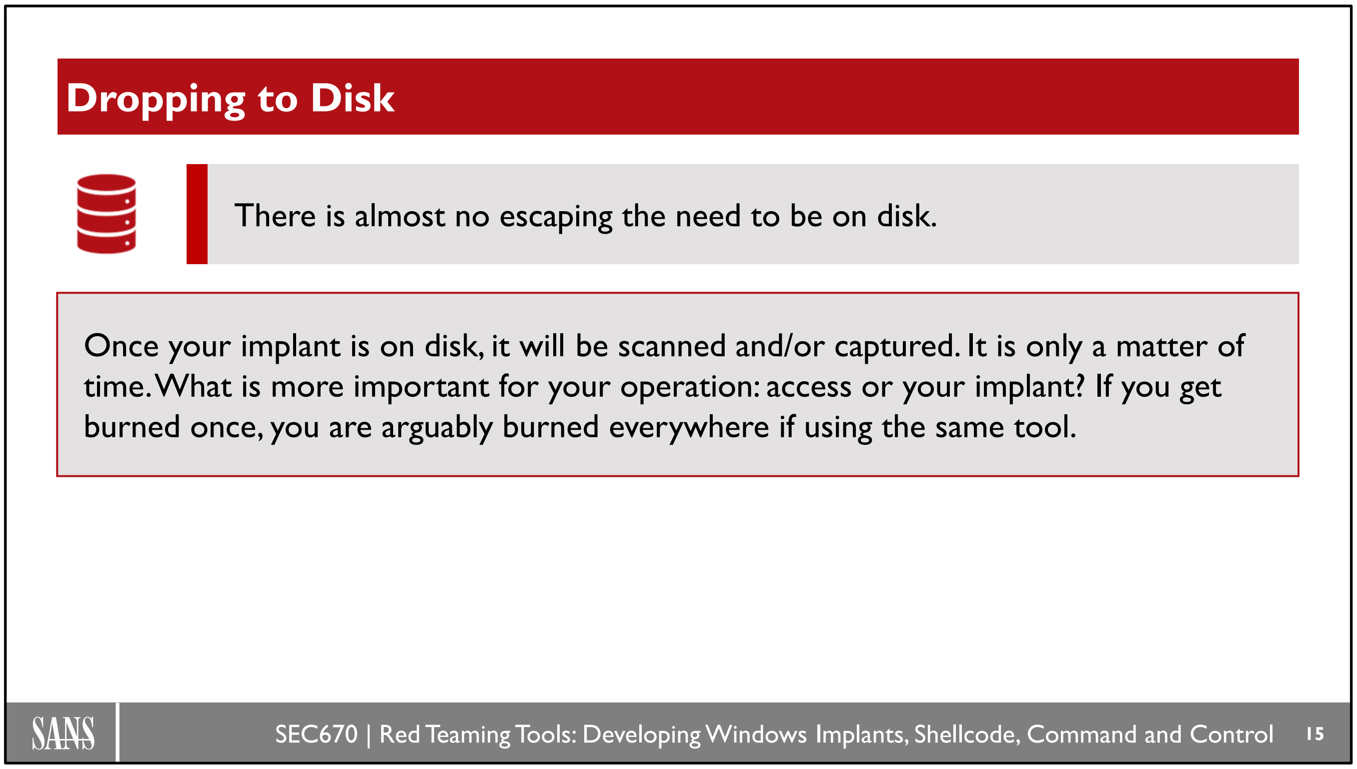
Our objectives for this module are:

- Discuss the need to drop to disk
- Discuss the risks of being on disk
- Cover how to protect yourself and your data

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 14

Objectives
The objectives for this module are to discuss the need to drop to disk and when to do it. Being on disk brings some risk and we will discuss those risks. If we must be on disk, then we should try to protect ourselves as best as possible if our files become discovered.

© SANS Institute 2024



Dropping to Disk

There is almost no escaping the need to be on disk.

Once your implant is on disk, it will be scanned and/or captured. It is only a matter of time. What is more important for your operation: access or your implant? If you get burned once, you are arguably burned everywhere if using the same tool.

Dropping to Disk
There will come a moment during the lifetime of your implant that something will have to touch disk. Dropping to disk is practically unavoidable these days when you are trying to survive across reboots. There are several risk factors that must be taken into consideration when dropping to disk, and the following is a non-exhaustive list of questions that should be answered before dropping to disk:

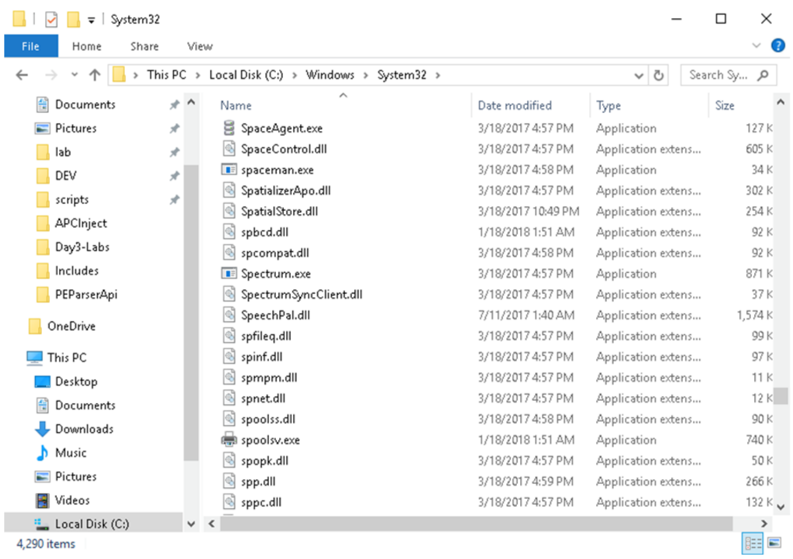
- Where should I drop to disk?
- Where do I have read/write permissions?
- What binary should I drop?
- How should I blend in with other items in the same folder?
- Is it acceptable to be scanned by AV solution?

The listing of considerations/questions should be answered with the data retrieved during the host survey phase. It would not be wise to blindly drop to some random folder and hope for the best. A location should be chosen strategically, one that would not indicate your presence to the user or sysadmin too quickly. If you are not elevated to at least a local Admin, you will not be able to drop your binary to some locations like System32, for example. System32 has many entries in it, and it would be easier to blend in there versus the current user's Downloads folder. Also, being on disk might mean that the installed AV solution might scan your binary. Is that okay? Perhaps the binary you drop only has basic features like your persistence/callback feature and that is it. This way, your entire capability is not discovered, and you do not have to spend months retooling your entire implant.

© SANS Institute 2024

Blending In

On this Windows 10 VM, there are over 4,200 items in the System32 folder. Plenty of options to blend in with files around you.

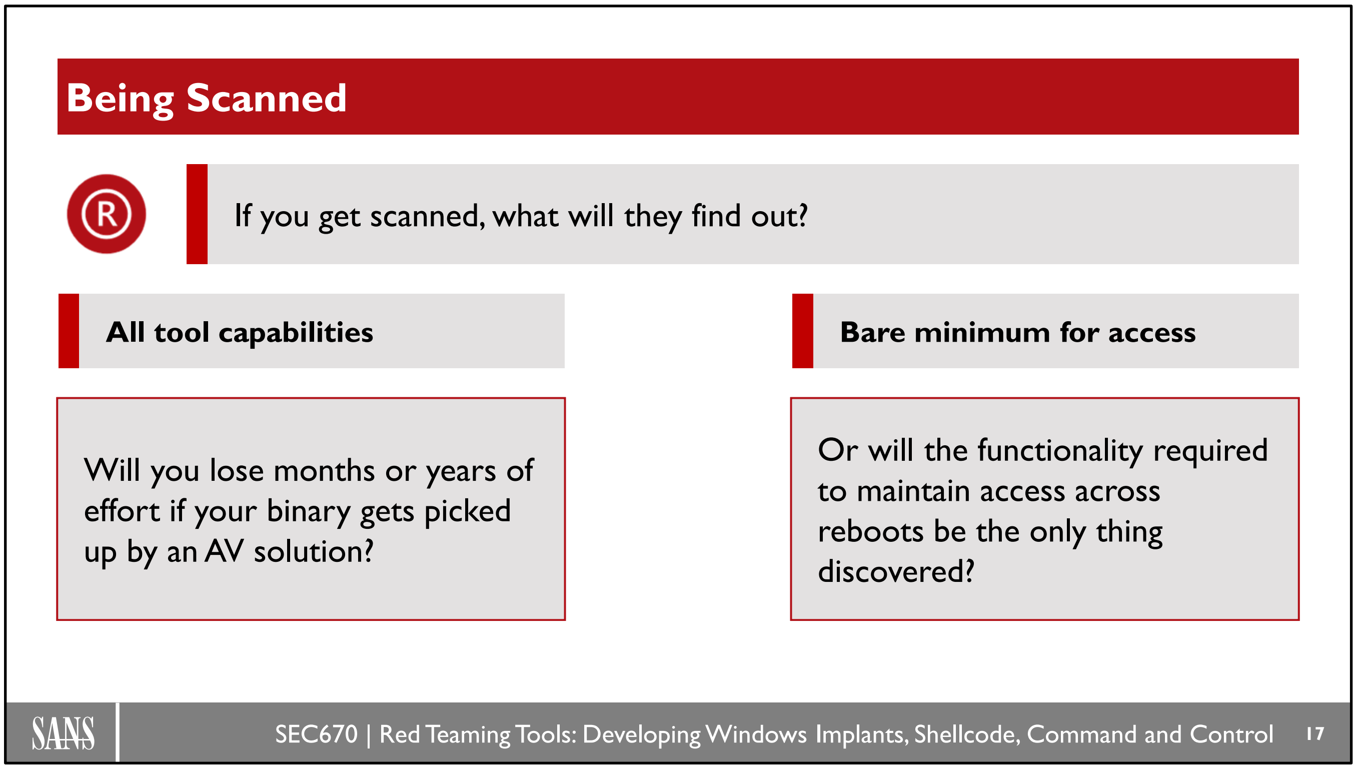


Name	Date modified	Type	Size
SpaceAgent.exe	3/18/2017 4:57 PM	Application	127 K
SpaceControl.dll	3/18/2017 4:57 PM	Application extens...	605 K
spaceman.exe	3/18/2017 4:58 PM	Application	34 K
SpatializerApo.dll	3/18/2017 4:57 PM	Application extens...	302 K
SpatialStore.dll	3/18/2017 10:49 PM	Application extens...	254 K
spbcd.dll	1/18/2018 1:51 AM	Application extens...	92 K
spcompat.dll	3/18/2017 4:58 PM	Application extens...	92 K
Spectrum.exe	3/18/2017 4:57 PM	Application	871 K
SpectrumSyncClient.dll	3/18/2017 4:57 PM	Application extens...	37 K
SpeechPal.dll	7/11/2017 1:40 AM	Application extens...	1,574 K
spfileq.dll	3/18/2017 4:57 PM	Application extens...	99 K
spinf.dll	3/18/2017 4:57 PM	Application extens...	97 K
spmpm.dll	3/18/2017 4:57 PM	Application extens...	11 K
spnet.dll	3/18/2017 4:57 PM	Application extens...	12 K
spoolss.dll	3/18/2017 4:58 PM	Application extens...	90 K
spoolsv.exe	1/18/2018 1:51 AM	Application	740 K
spopk.dll	3/18/2017 4:57 PM	Application extens...	50 K
spp.dll	3/18/2017 4:59 PM	Application extens...	266 K
sppc.dll	3/18/2017 4:57 PM	Application extens...	132 K

Blending In

When it comes to blending in, it can be somewhat easy with the proper level of permissions. As one possible example, a prime spot could be the System32 folder where there are well over 4,200 items to surround yourself. You would not necessarily want to be the first or the last entry in the folder but pick a spot that would require the user to scroll down for quite some time. Users are notorious for scrolling right past something unless they are specifically looking for it. Even then, most users still go past what they are looking for at least once or twice. Once your location has been determined, a filename would have to be chosen. The screenshot above shows several files that start with the letter "s." If there are more s-named files, then perhaps naming your binary something that starts with "s" might be a good idea. The screenshot also shows dates for the folder's files. The dates should be taken into consideration and your binary should reflect something close to them. If none of the files in the folder are beyond a certain year, like 2019, then you would not want to be the only binary reflecting the current year. Regardless of location, the bottom line is this, the more care you take with trying to blend in with your surroundings, the longer you might maintain a presence on that target.

© SANS Institute 2024



Being Scanned

If you get scanned, what will they find out?

All tool capabilities

Will you lose months or years of effort if your binary gets picked up by an AV solution?

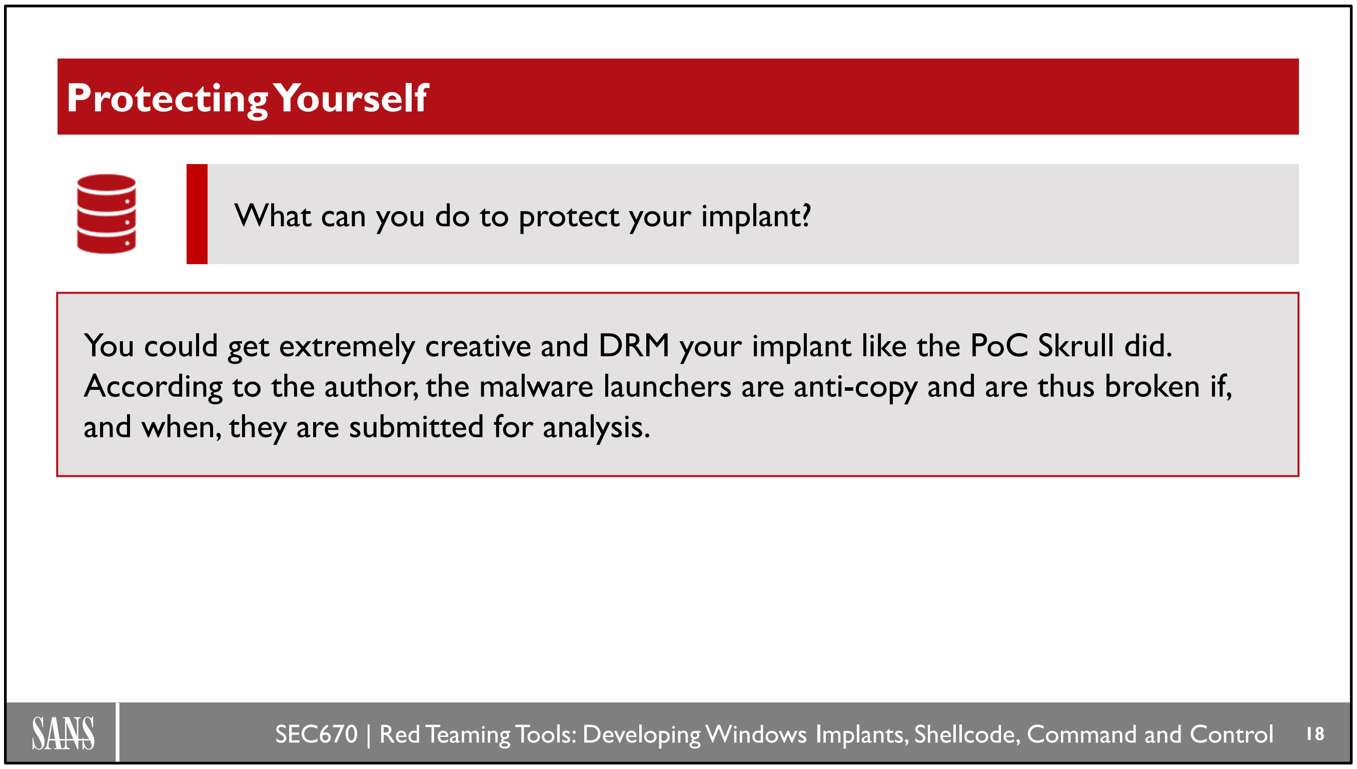
Bare minimum for access

Or will the functionality required to maintain access across reboots be the only thing discovered?

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 17

Being Scanned
Another risk you must consider is should there be some AV solution installed like Defender, will that deter you from dropping to disk? If not, are you okay with it possibly being scanned? Some AV solutions require the submission to be sent to its cloud-based analysis engine if it has not seen it before. Most Windows system binaries and files are common, and their hashes/signatures are known. As such, cloud AV solutions do not need to upload them for analysis. The one-off binary, though, like yours, hopefully has never been seen or even scanned before. All of that could change when the solution detects that a new file has been created on the file system. What will the vendor find out if your implant is scanned? Will all your trade secrets be swept up and sent to the cloud or perhaps only the required features to maintain your access to the target? I cannot answer these questions for you as they are questions that can be ever changing even between operations.

© SANS Institute 2024



Protecting Yourself

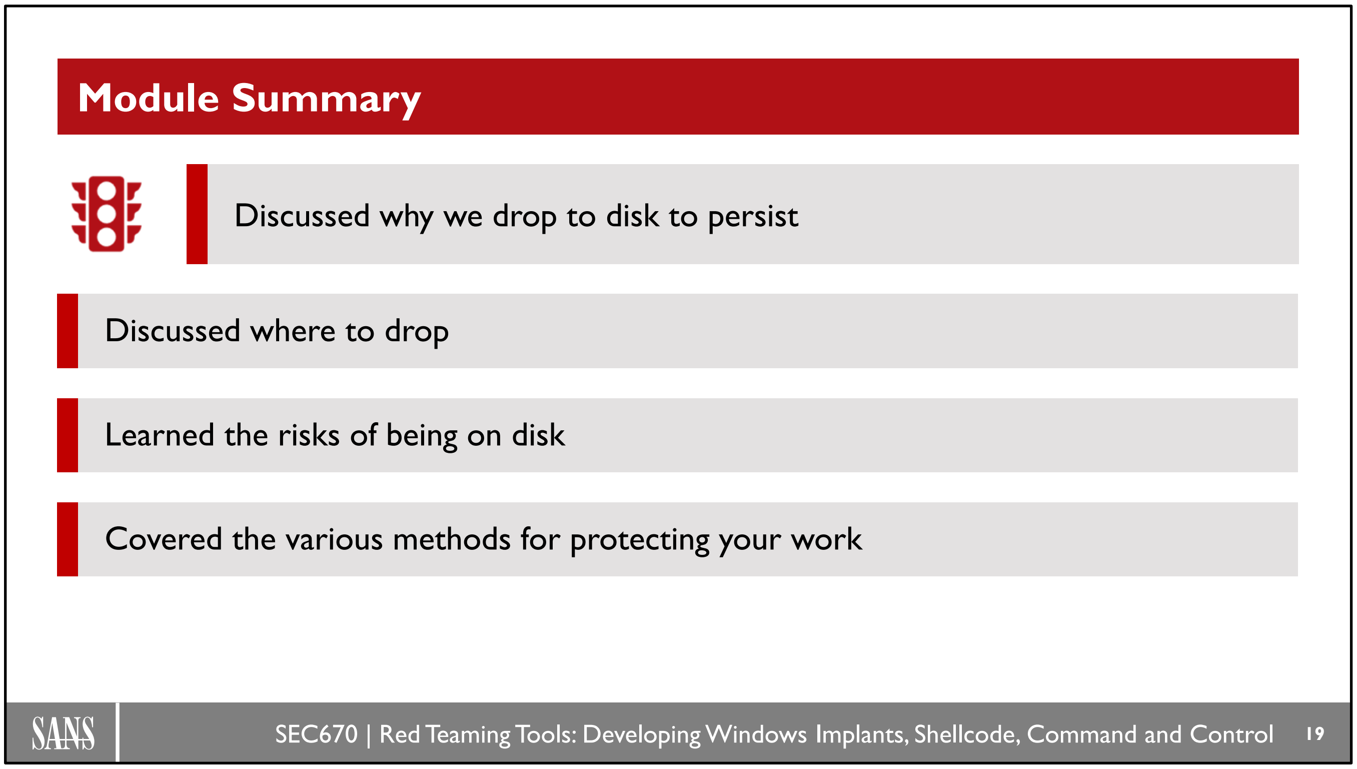
What can you do to protect your implant?

You could get extremely creative and DRM your implant like the PoC Skrull did. According to the author, the malware launchers are anti-copy and are thus broken if, and when, they are submitted for analysis.

Protecting Yourself
There are several public techniques and tools that are out there today that aid your efforts to protect yourself. There are commercial tools like packers and encryptors that do a tremendous job annoying reverse engineers. One such tool is Themida, which is made by the company Oreans. Themida is labeled as Advanced Windows software protection system and does so by protecting each code block, as well as adding an overall level of protection. You can read more on their website when you have a moment. Another interesting technique and perhaps the most interesting one that has recently surfaced, as of this writing, is Skrull. According to the author, Sheng-Hao Ma, "Skrull is a malware DRM, that prevents Automatic Sample Submission by AV/EDR and Signature Scanning from Kernel. It generates launchers that can run malware on the victim using the Process Ghosting technique. Also, launchers are totally anti-copy and naturally broken when got submitted." Seems very interesting and clever. The source code for Skrull is hosted on GitHub for your viewing pleasure.

References:
https://www.oreans.com/themida.php
https://github.com/aaaddress1/Skrull

© SANS Institute 2024

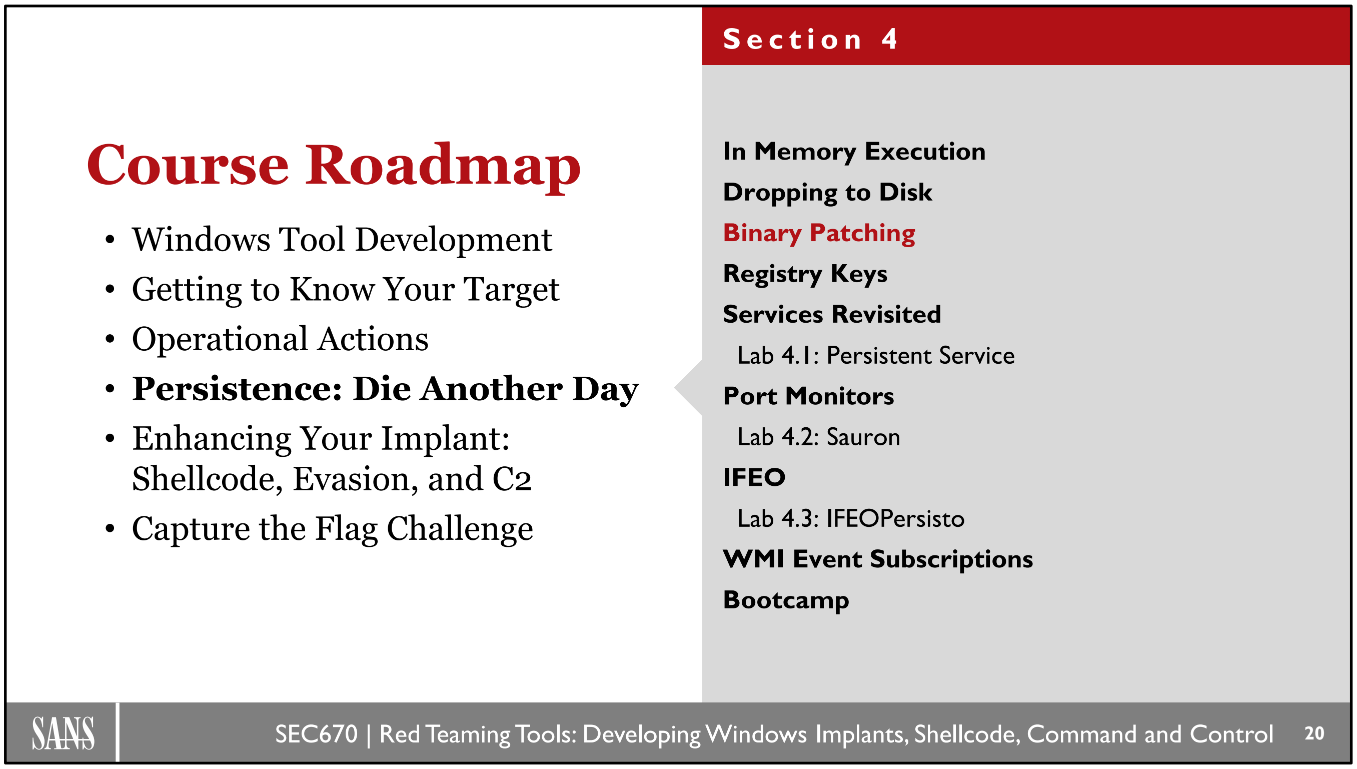


Module Summary

- Discussed why we drop to disk to persist
- Discussed where to drop
- Learned the risks of being on disk
- Covered the various methods for protecting your work

Module Summary
In this module, we discussed why we would even drop to disk, where to drop, the risk(s) of being on disk, and how to protect what we have on disk.

© SANS Institute 2024



Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

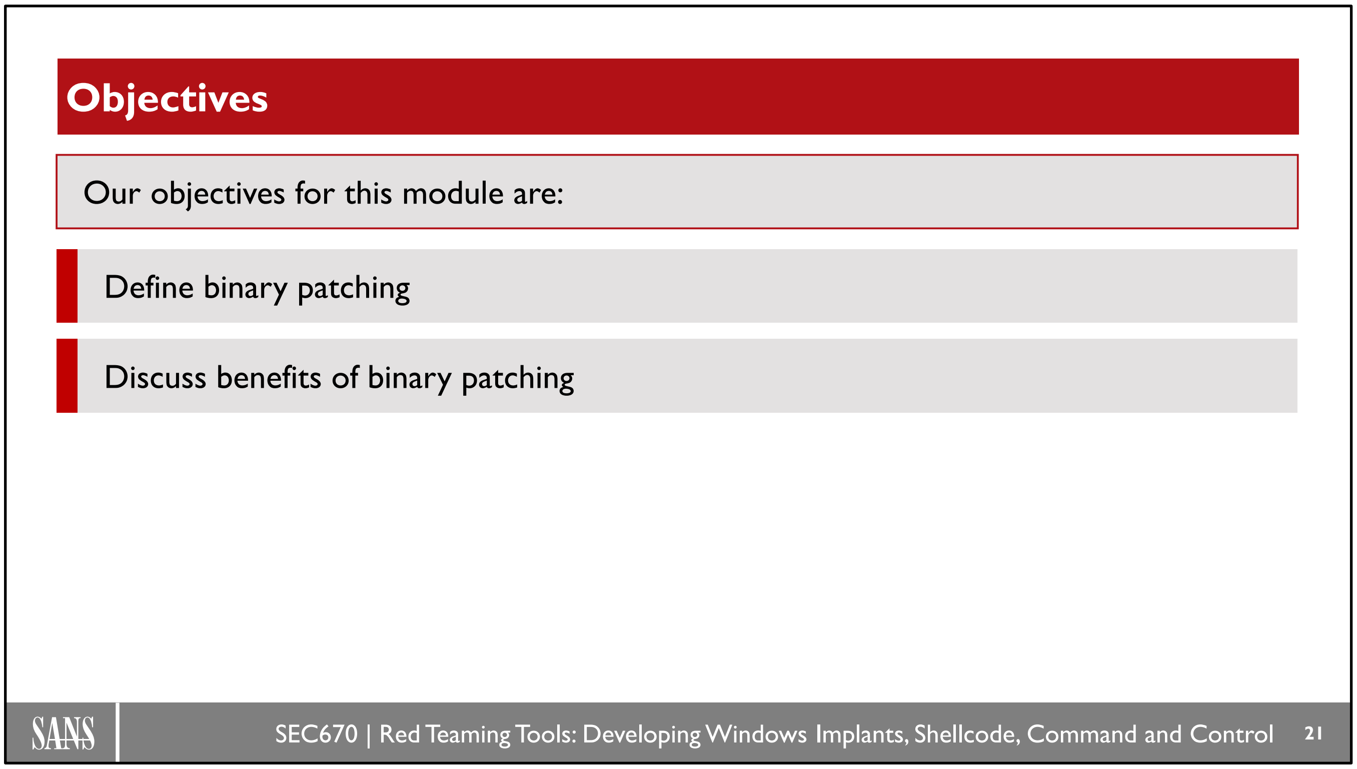
Section 4

In Memory Execution
Dropping to Disk
Binary Patching
Registry Keys
Services Revisited
  Lab 4.1: Persistent Service
Port Monitors
  Lab 4.2: Sauron
IFEO
  Lab 4.3: IFEOPersisto
WMI Event Subscriptions
Bootcamp

SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 20

In this module, we will discuss what binary patching is and how we can leverage it for persistence on target.

© SANS Institute 2024



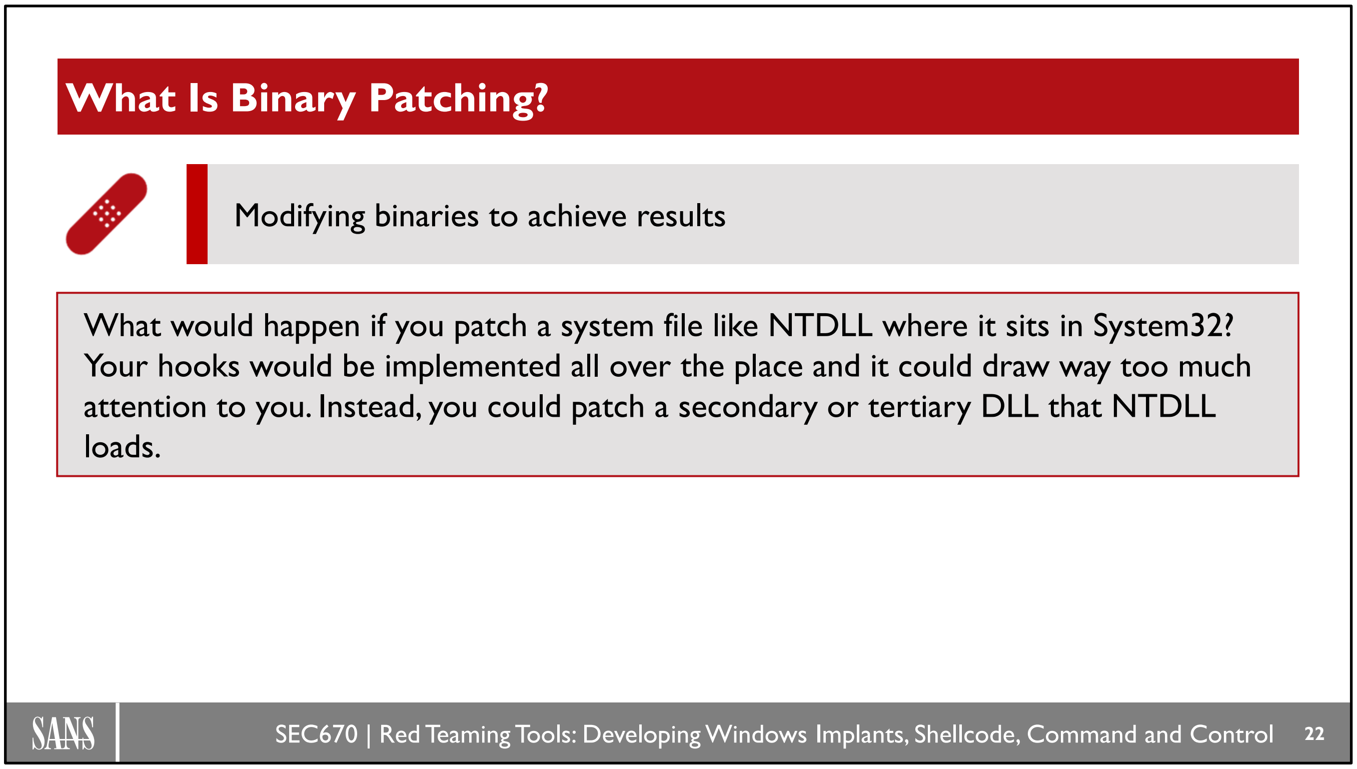
Objectives

Our objectives for this module are:

- Define binary patching
- Discuss benefits of binary patching

Objectives
The objectives for this module are to define what binary patching is and discuss some of the benefits that binary patching has to offer.

© SANS Institute 2024



What Is Binary Patching?

Modifying binaries to achieve results

What would happen if you patch a system file like NTDLL where it sits in System32? Your hooks would be implemented all over the place and it could draw way too much attention to you. Instead, you could patch a secondary or tertiary DLL that NTDLL loads.

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 22

What Is Binary Patching?
Binary patching is often referred to as modifying a binary as it resides on disk or in memory with the intention of changing how it executes. In memory patching is often done by AV/EDR solutions to change how functions of interest behave. What they implement is function hooking, which will be covered in detail in Section 5. Regardless, they are still modifying bytes of a file in memory. Patching files on disk can be challenging and could have damaging effects. Take for example an attempt to patch ntdll.dll: the file resides in the System32 folder and as such, you must have at least local Admin privileges. Because ntdll.dll is so heavily utilized, you could cause the system to become unstable and crash. Another common tactic is to patch the AV/EDR solution itself. Identifying a weakness in a solution can yield incredible results for not only persistence but also for protecting your toolset.

© 2024 Jonathan Reiter

© SANS Institute 2024

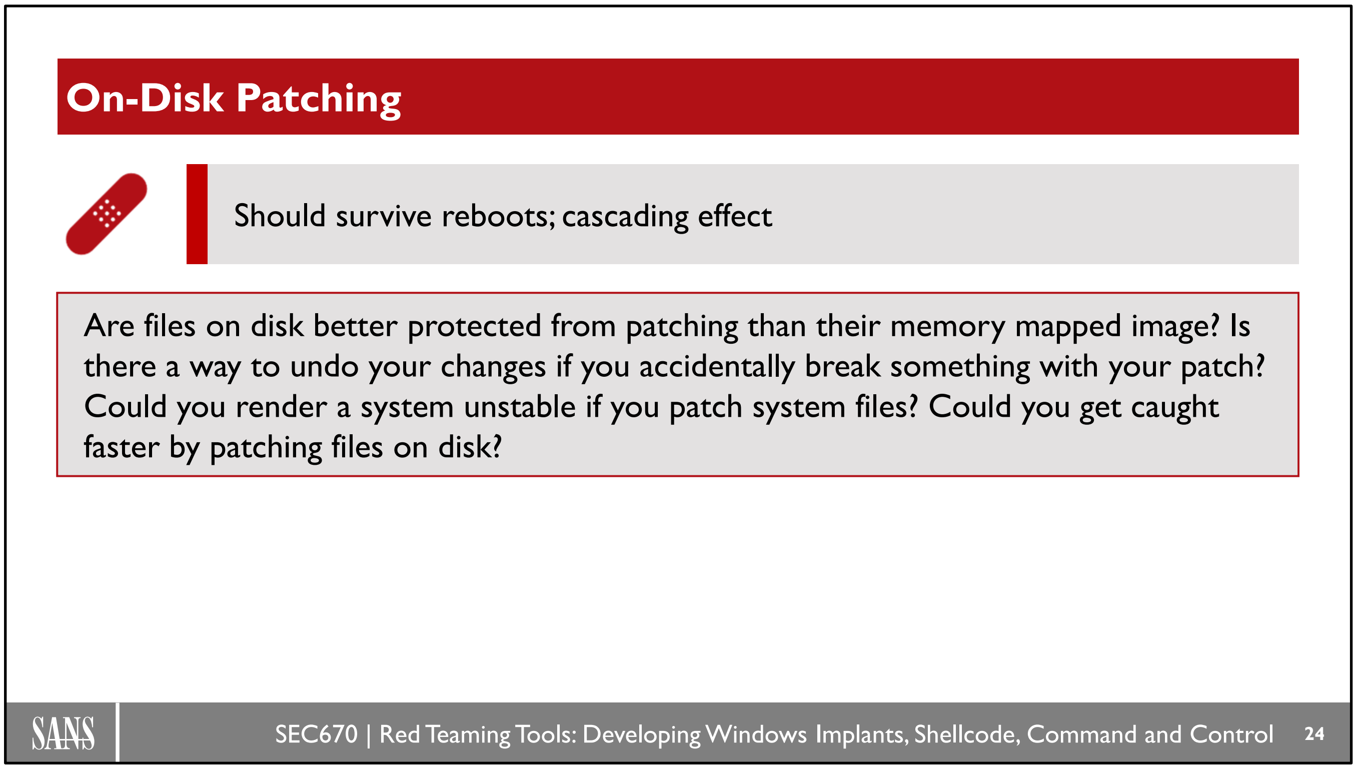
In-Memory Patching

The patch is not permanent; will not survive reboots

Once you are injected into a process, how do you find what it is you are trying to patch? How about walking the Import Address Table? How about enumerating processes and obtaining handles to a process of concern? These are valid questions that could be answered either during your op or beforehand.

In-Memory Patching
Patching an image as it sits in memory is often referred to as in-memory patching. This is relatively safe since no changes are made to the binary on disk, so if the computer were to ever reboot, the changes would be flushed out of memory as if nothing ever happened. The big question you might have is, how do we find images in memory if we were just injected into a process? This is where process enumeration would come into play. If there is a process that you know about ahead of time based on gathered intel, then you can look for that process specifically. Otherwise, enumerate all processes and pick one from that newly collected information. Once you have found a process you want to patch, you need to obtain a process handle to it. If you recall, the handle is really the base address of the executable's image in memory, so if you were to read the first three bytes of memory you should find the famous MZ (\x4d\x5a) signature followed by a NOP, \x90. When conducting memory scanning and looking for the start of an image, it is always best to be as specific as possible so looking for \x4d\x5a\x90\x00 would possibly lower false positives. Now that you found the start of the PE header, it is simply a matter of parsing the rest of the PE headers to get to what you want. Often, you would be looking to patch out calls to certain functions like ***AmsiScanBuffer*** or ***AmsiScanString*** APIs from the Amsi.dll if you are trying to bypass AMSI. You could also decide to patch specific functions to change how they behave. You will see later how we can change how ***AmsiScanBuffer*** behaves.

© SANS Institute 2024



On-Disk Patching

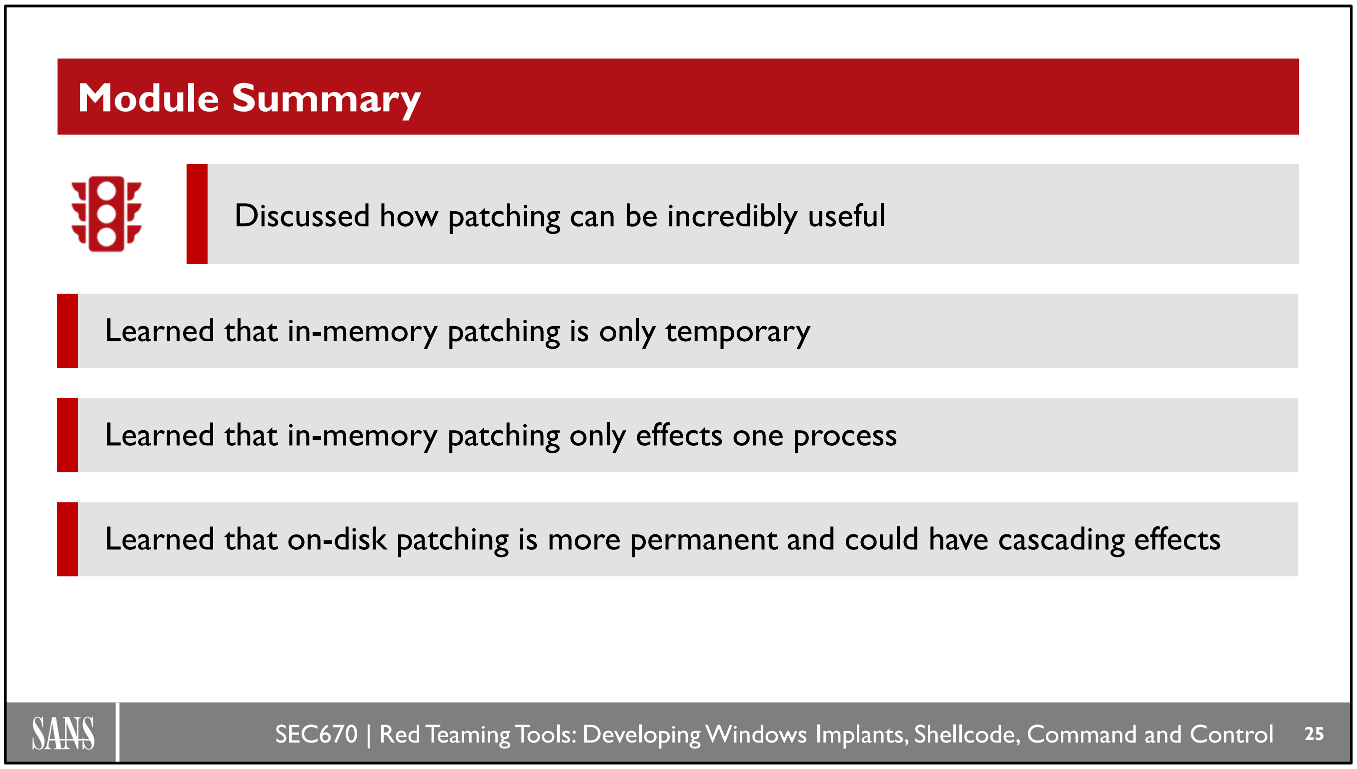
Should survive reboots; cascading effect

Are files on disk better protected from patching than their memory mapped image? Is there a way to undo your changes if you accidentally break something with your patch? Could you render a system unstable if you patch system files? Could you get caught faster by patching files on disk?

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 24

On-Disk Patching
There might come a time when you would have to patch the file as it resides on the file system. If you have the proper permissions, you might be able to patch the binaries that offer signature scanning for the AV that might be installed on the system. Perhaps you found a way to get your tool added to some whitelist and you implement a patch to make sure you remain on the whitelist. System files might be able to be patched too, with the right permissions, of course. Take for an example patching Ntdll.dll. How many processes rely on Ntdll.dll? Practically all of them because the system loader is implemented in Ntdll.dll and it is the first module that is present in the process. Whatever patch you do on disk, each process is going to be affected by that change at some point in time, like after a reboot. This could make a system unstable and would require rigorous testing ahead of time. My recommendation is to avoid patching critical system DLLs unless there is a dire need that requires it. Until that time, stick to patching third party binaries.

© SANS Institute 2024

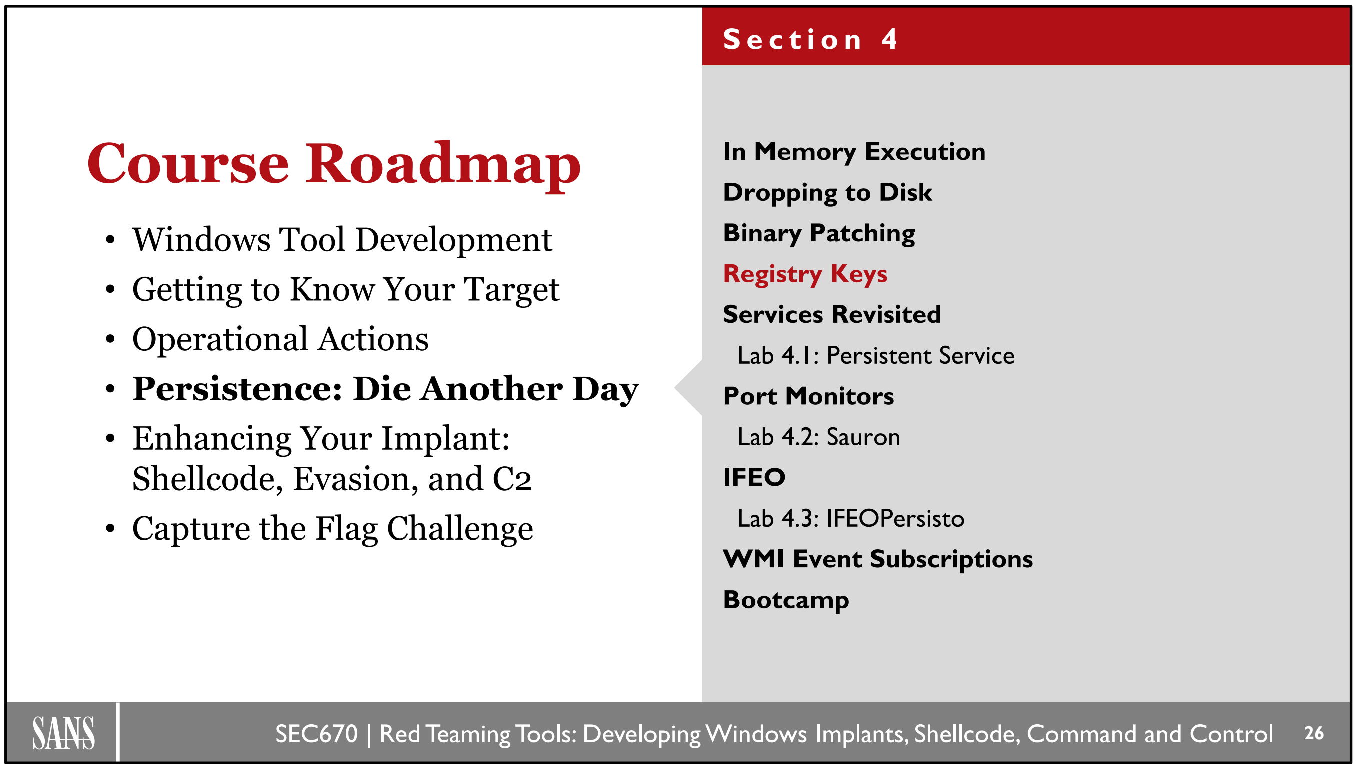


Module Summary

- Discussed how patching can be incredibly useful
- Learned that in-memory patching is only temporary
- Learned that in-memory patching only effects one process
- Learned that on-disk patching is more permanent and could have cascading effects

Module Summary
In this module, we discussed two types of patching: in-memory and on-disk. Both have certain tradeoffs and which one you choose would be dependent on what your end goal is for your operation.

© SANS Institute 2024



Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

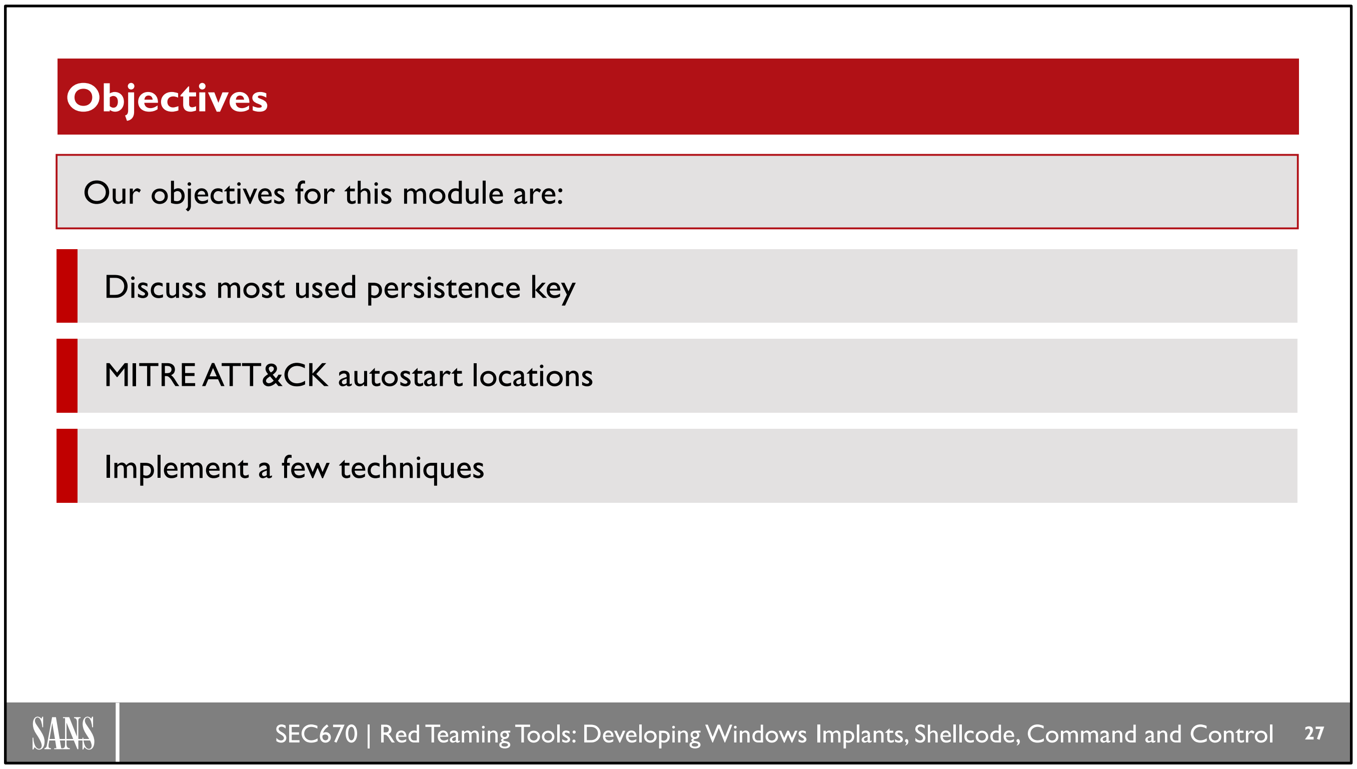
Section 4

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 26

Registry Keys
In this module, we will discuss how to persist using the Windows Registry.

© 2024 Jonathan Reiter

© SANS Institute 2024



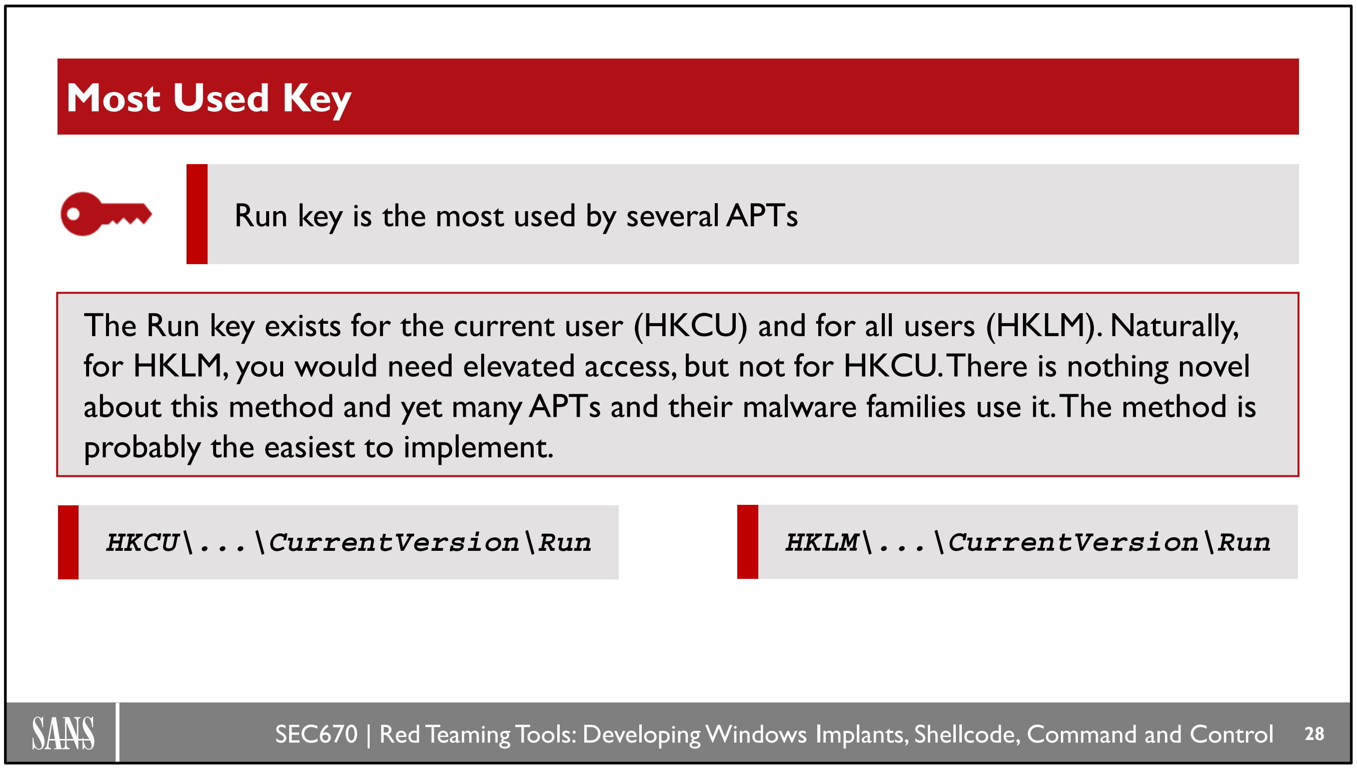
Objectives

Our objectives for this module are:

- Discuss most used persistence key
- MITRE ATT&CK autostart locations
- Implement a few techniques

Objectives
The objectives for this module are to discuss what the most used key was for persistence. We will also explore several possible keys that can be leveraged for autostart entries. Lastly, we will reinforce the topics with source code review and a lab.

© SANS Institute 2024



Most Used Key

Run key is the most used by several APTs

The Run key exists for the current user (HKCU) and for all users (HKLM). Naturally, for HKLM, you would need elevated access, but not for HKCU. There is nothing novel about this method and yet many APTs and their malware families use it. The method is probably the easiest to implement.

```
HKCU\...\CurrentVersion\Run
```

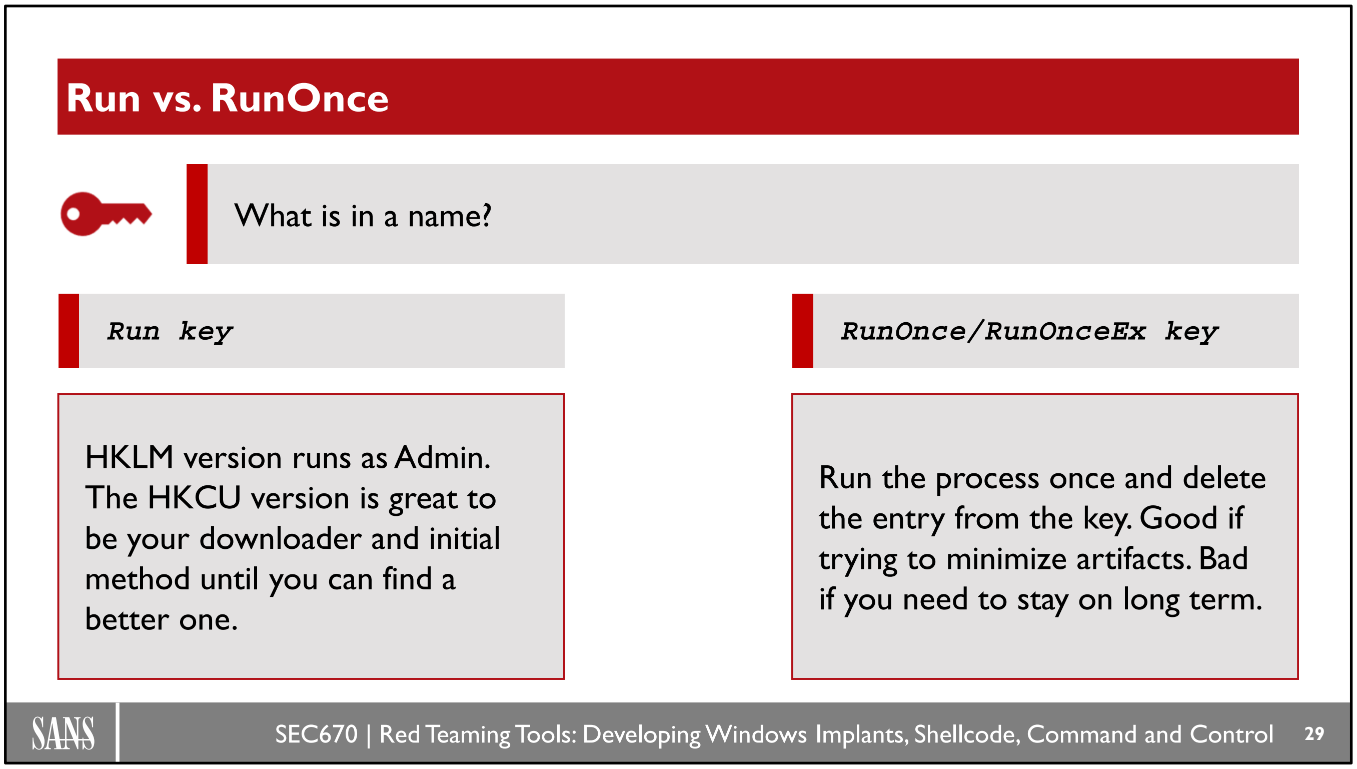
```
HKLM\...\CurrentVersion\Run
```

Most Used Key
If you were to study all of the publicly available malware samples that have a persistence mechanism, you might find that perhaps the most used method is the Registry. More specifically, the Registry Key chosen for persistence is the Run key. Red Canary, Cynet, and a few other companies that specialize in malware analysis have noted that the Run key is most commonly used by malware authors. The interpretation is that they do not get caught right away, so why waste valuable time and effort trying to discover a novel method? APT28 (SOFACY), APT30, DarkComet, Emotet, FIN7, Gazer, Lazarus Group and more have all used this key as one of their persistence methods.

One could operate based on the concept of ”one is none, and two is one” where having only one method of persistence as your only lifeline is really not having any persistence at all. If you have two methods, then should one of the methods get burned or simply fail to trigger, you still maintain your foothold.

References:
https://attack.mitre.org/techniques/T1547/001/
https://www.cynet.com/attack-techniques-hands-on/the-art-of-persistence/

© SANS Institute 2024



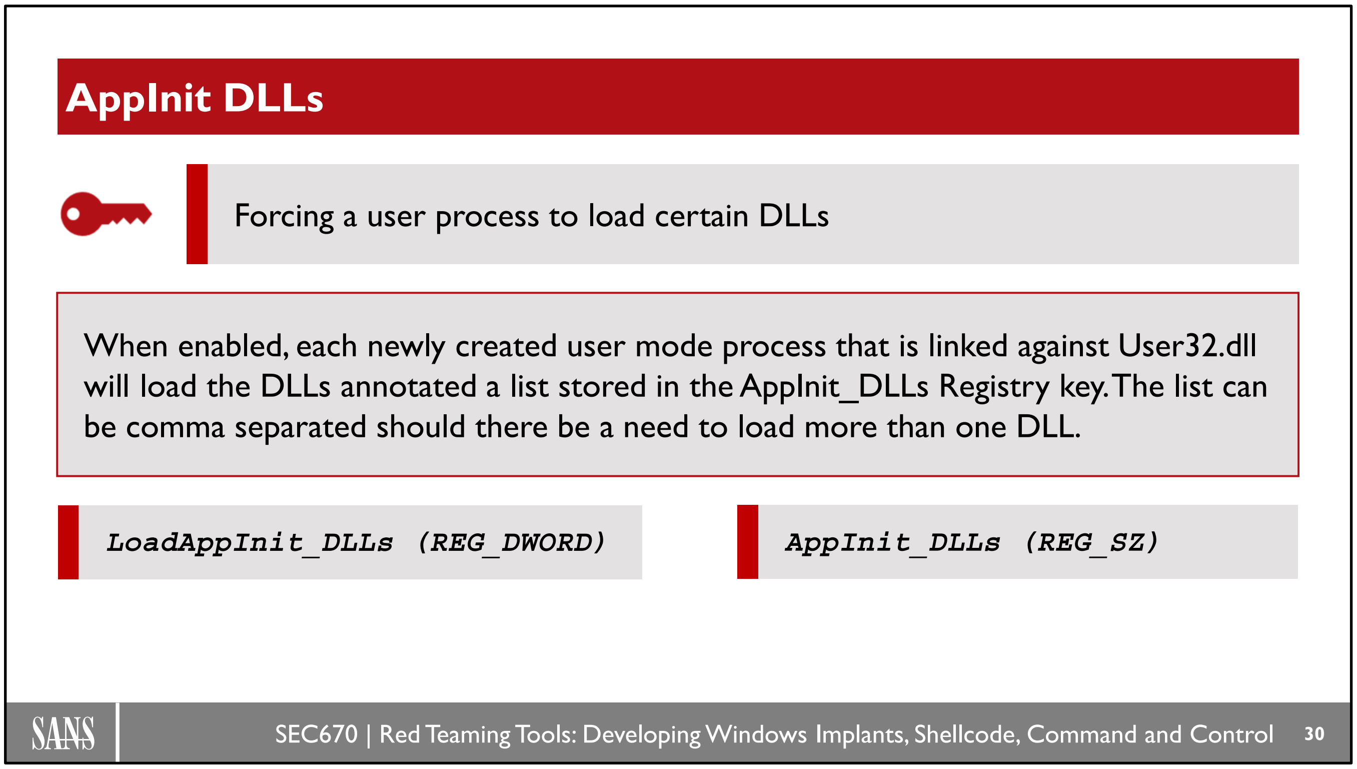
Run vs. RunOnce

What is in a name?

`Run key`	`RunOnce/RunOnceEx key`
HKLM version runs as Admin. The HKCU version is great to be your downloader and initial method until you can find a better one.	Run the process once and delete the entry from the key. Good if trying to minimize artifacts. Bad if you need to stay on long term.

Run vs. RunOnce
There are several Registry keys that a person could use for persistence, but for this slide we will just focus on two of them. The keys under HKLM require Admin privileges and without them we would be forced to use a key under HKCU, which could expose our presence on the system quickly. The HKCU versions would require the user to log on to the workstation, whereas the HKLM versions kick off when the system boots up regardless of anyone logging on to the workstation. Sticking with the most used key among APTs, we have Run and RunOnce. The two keys do offer a means to accomplish your persistence, but they are definitely not equal. Despite the Run key being the most used method of persistence, the RunOnce key gets used as well, but is perhaps not always fully understood by everyone. The RunOnce key will do exactly what is sounds like it will do: it will run your application just once and be done. The difference is that after the one-time execution, your persistence mechanism has gone up in a cloud of smoke because your RunOnce entry will be deleted. Of course, it is not hard to make another entry after it has been deleted, just annoying is all.

© SANS Institute 2024



AppInit DLLs

Forcing a user process to load certain DLLs

When enabled, each newly created user mode process that is linked against User32.dll will load the DLLs annotated a list stored in the AppInit_DLLs Registry key. The list can be comma separated should there be a need to load more than one DLL.

`LoadAppInit_DLLs (REG_DWORD)`

`AppInit_DLLs (REG_SZ)`

AppInit DLLs
Windows provides users the flexibility to allow a pre-determined list of customized DLLs to be loaded into practically each user process. The catch is that the process must be linked against the User32.dll, and thankfully most, if not all GUI applications will be linked against it. The AppInit_DLLs Registry key is the key that dictates if the feature should be enabled or disabled for a system. The key is located under the HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows key for x64-bit processes and under HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\windows key for 32-bit processes. Under this key, a few items and values are looked for by User32.dll. User32.dll imports a function called ***LoadAppInitDlls***, which is implemented in Kernel32.dll. The module will perform a check to see if the LoadAppInit_DLLs key is set (1) or cleared (0). If it is set, then it is enabled system wide, and the list of DLLs specified in the AppInit_DLLs key will be loaded into the processes. Since this is under the HKLM hive, we would of course need to have Administrator privileges.

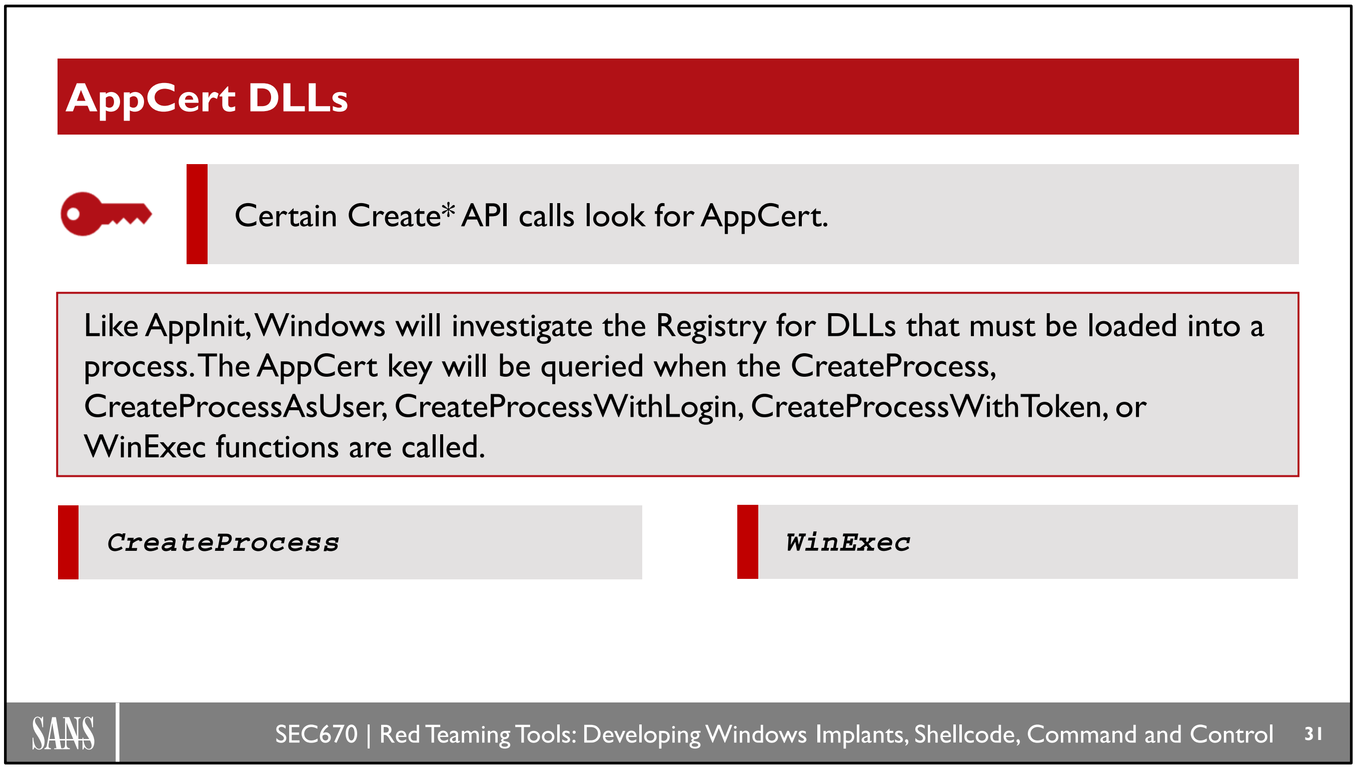
After our privilege escalation is done, we can create our list of DLLs with just one entry; the absolute path to our malicious DLL. After we do that, we enable the loading of AppInit DLLs by setting the LoadAppInit_DLLs key. A question that you may have had is what kind of DLLs can we use for this. Practically any DLL can be used with this technique so as long as it does not create a new process via the ***CreateProcess*** API. The resulting effect would cause an infinite loop of sorts because each time your DLL is loaded, it creates a new process, which will load your DLL, which creates a new process. One method to solve the infinite loop issue is to register a global namespace mutex that could be checked, and if it already exists, you know you have been triggered before so no need to execute again. The action of having our DLL creating a new process might not always be desired, so perhaps you could create some sort of check to see what process should be the one to load your malicious DLL.

This technique has been used by APTs such as APT39, CherryPicker, and T9000.

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

© 2024 Jonathan Reiter

© SANS Institute 2024



AppCert DLLs

Certain Create* API calls look for AppCert.

Like AppInit, Windows will investigate the Registry for DLLs that must be loaded into a process. The AppCert key will be queried when the CreateProcess, CreateProcessAsUser, CreateProcessWithLogin, CreateProcessWithToken, or WinExec functions are called.

`CreateProcess`

`WinExec`

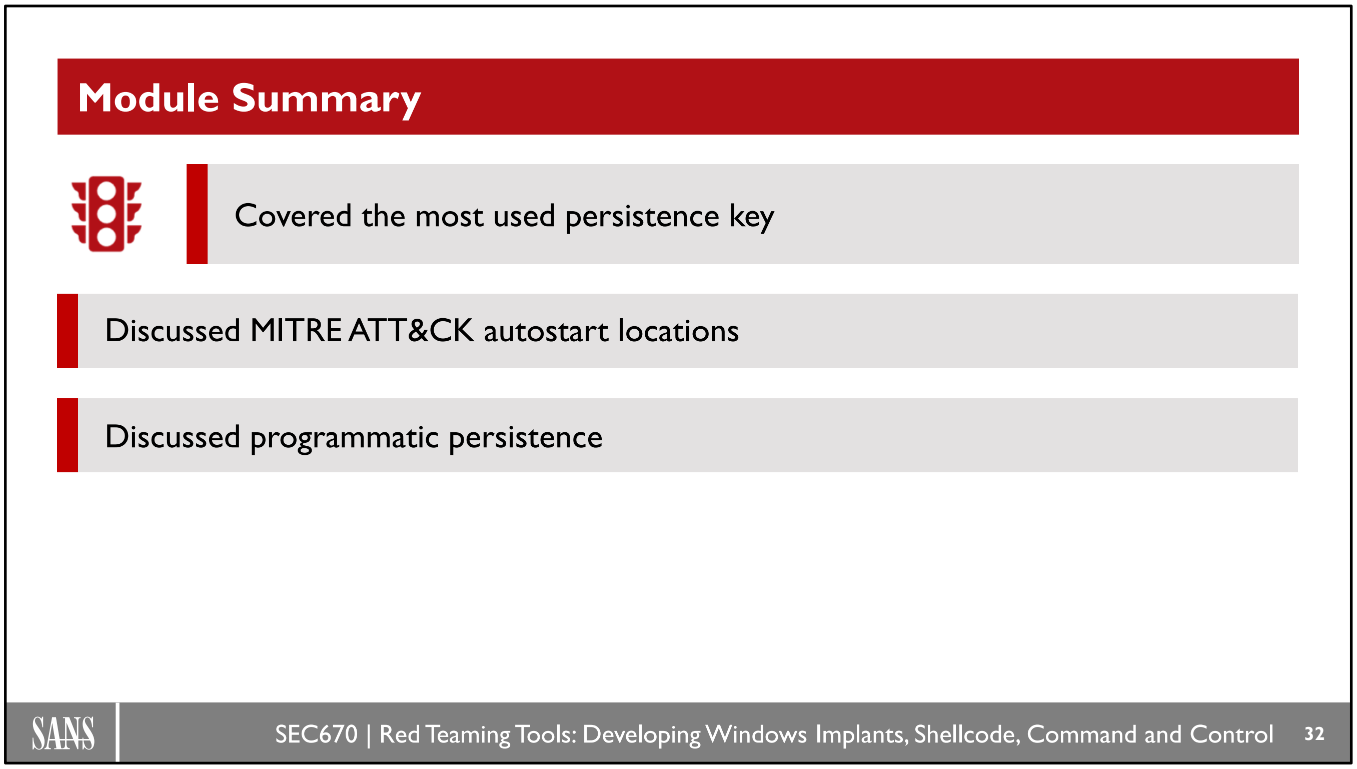
SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 31

AppCert DLLs
Almost as a compliment to AppInit, there is AppCert, which is extremely similar in the sense that a Registry key will be queried to find a list of DLLs a process must load. The AppCert key would be under the HKLM\SYSTEM\CurrentControlSet\Current\Session Manager\AppCertDlls key, which means Administrator privileges are required. The key is queried whenever a process makes its first call to one of the following APIs: ***CreateProcess***, ***CreateProcessAsUser***, ***CreateProcessWithLogin***, ***CreateProcessWithToken***, or ***WinExec***. The major difference with AppCert is the format of the DLL being used. The DLL must export a specific function called ***CreateProcessNotify*** that will be called when the system is loading your DLL. One major drawback to this method is that the target computer must be rebooted before any of the changes you made are implemented. As such, AppCert is not stealthy and is easily detected by tools such as AutoRuns from SysInternals. We would never force a reboot live on a target even more so when a user is actively logged on to it. The best thing could be to wait for a corporate reboot policy to kick in and reboot the target for you.

This technique has been used by APTs such as Honeybee and PUNCHBUGGY.

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

© SANS Institute 2024



Module Summary

Covered the most used persistence key

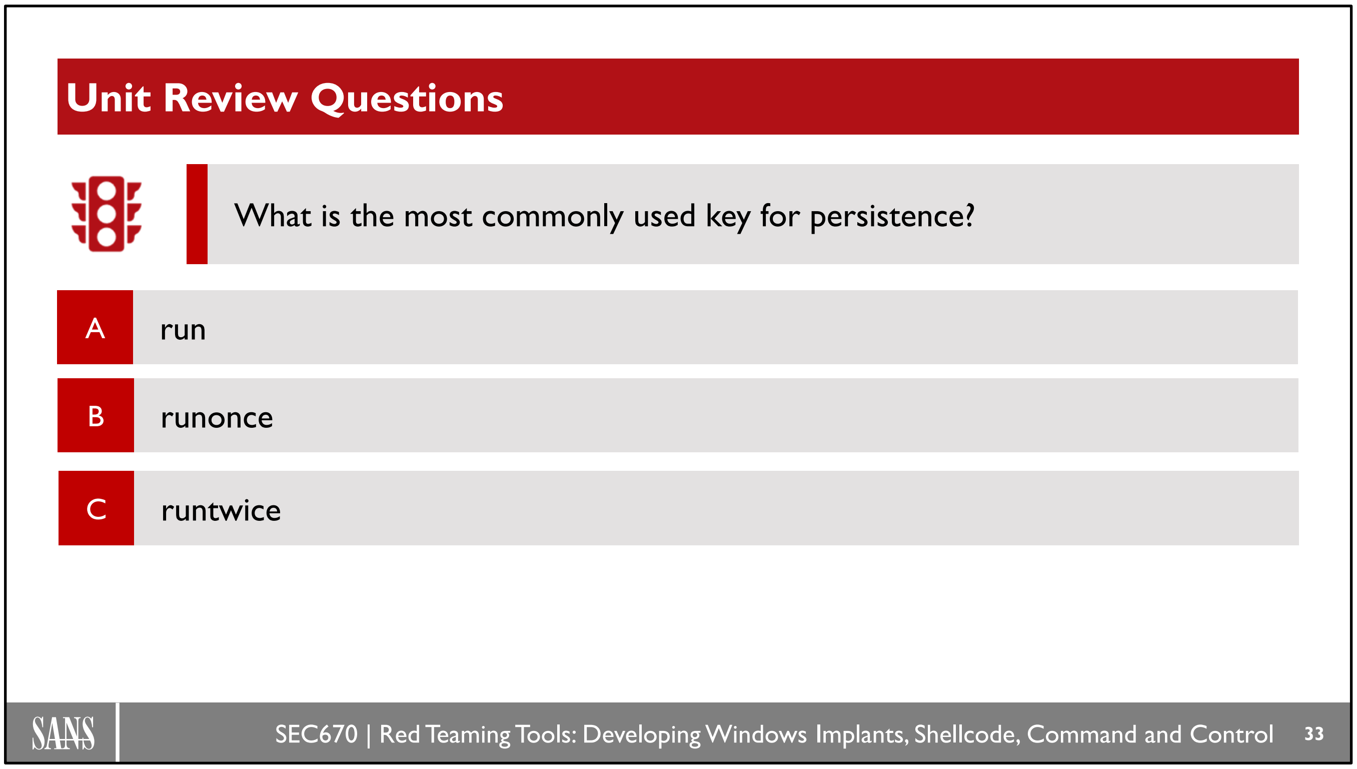
Discussed MITRE ATT&CK autostart locations

Discussed programmatic persistence

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 32

Module Summary
In this module, we covered current trends for registry persistence and what the most used key was. We also discussed the differences between the HKLM and the HKCU versions of keys like the Run key. We also hit on a few other keys, like AppInit and AppCert, along with the differences between them. All of these methods should be easily detected using tools like AutoRuns from SysInternals.

© SANS Institute 2024



Unit Review Questions

What is the most commonly used key for persistence?

A	run
B	runonce
C	runtwice

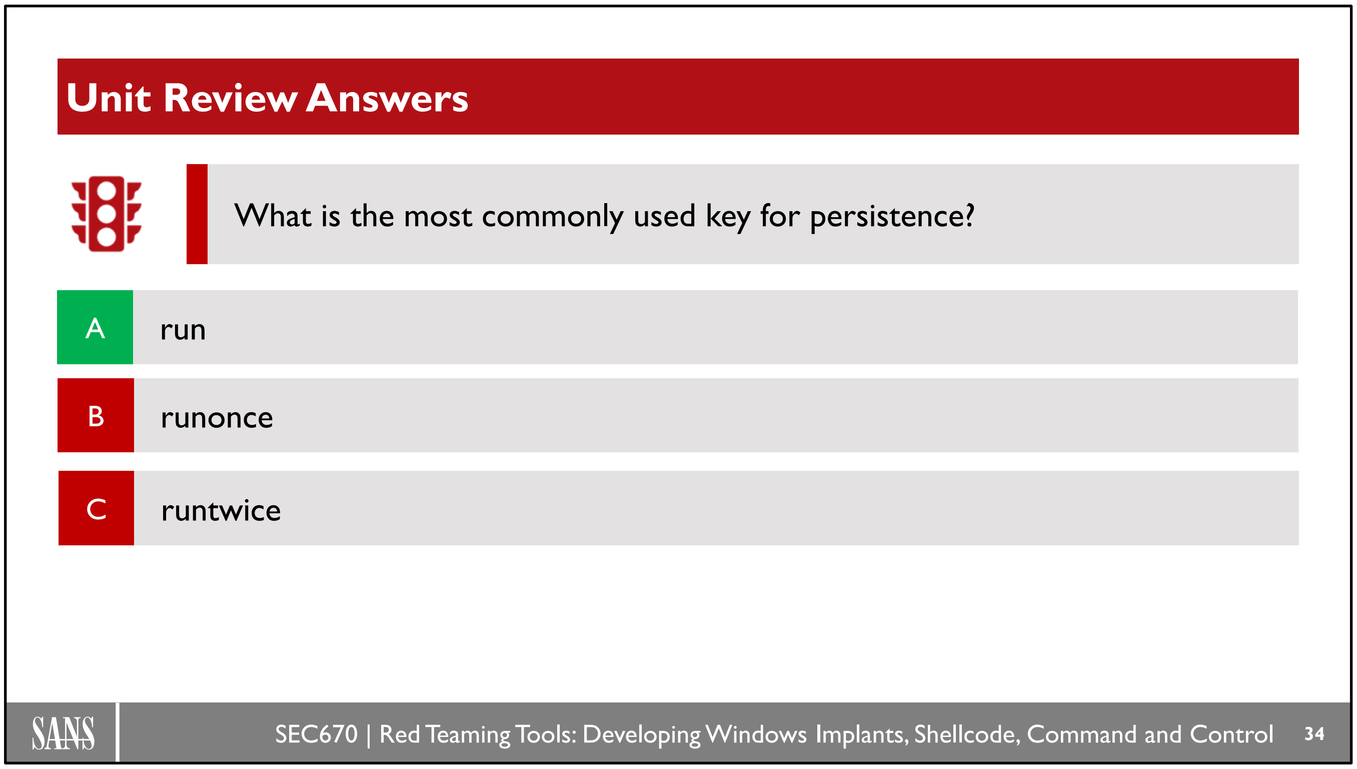
Unit Review Questions
Q: What is the most commonly used key for persistence?

A: run

B: runonce

C: runtwice

© SANS Institute 2024



Unit Review Answers

What is the most commonly used key for persistence?

A run

B runonce

C runtwice

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 34

Unit Review Answers
Q: What is the most commonly used key for persistence?

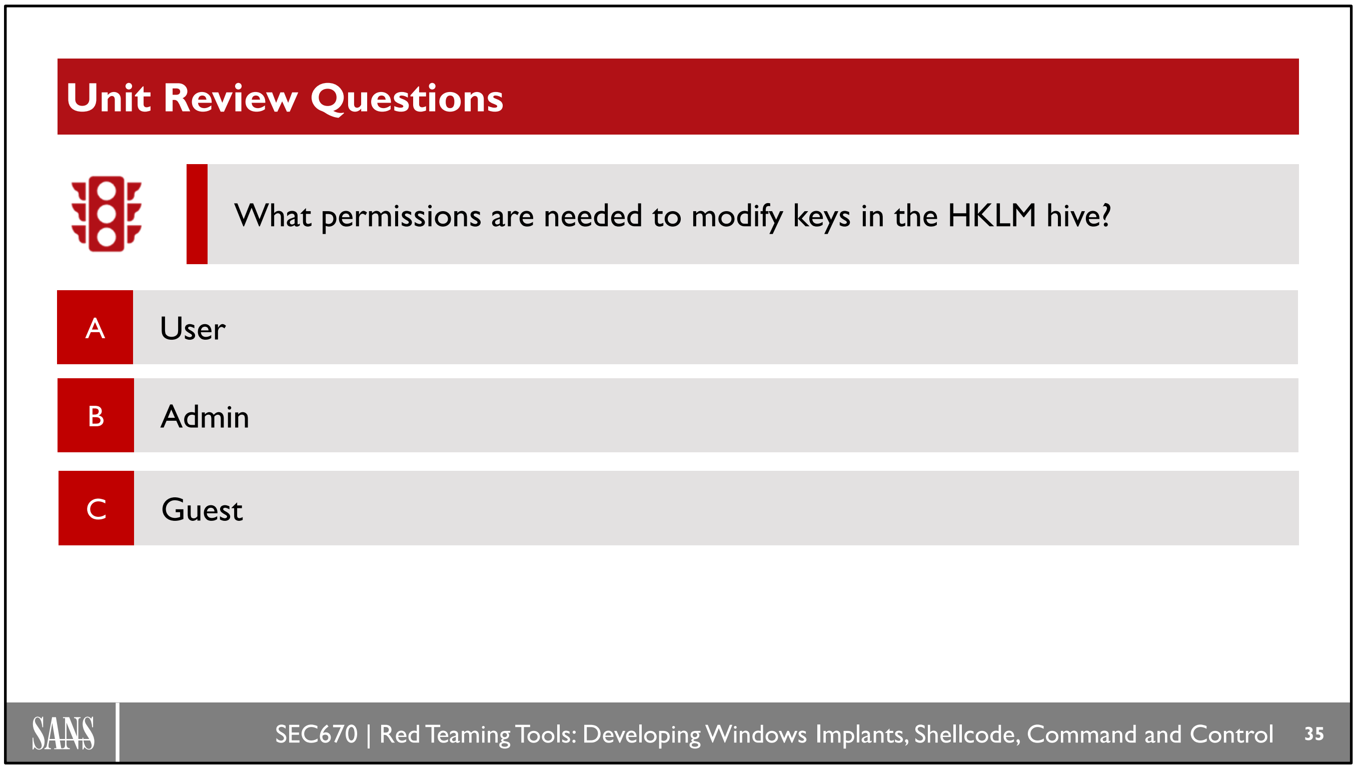
A: run

B: runonce

C: runtwice

© 2024 Jonathan Reiter

© SANS Institute 2024



Unit Review Questions

What permissions are needed to modify keys in the HKLM hive?

A User

B Admin

C Guest

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 35

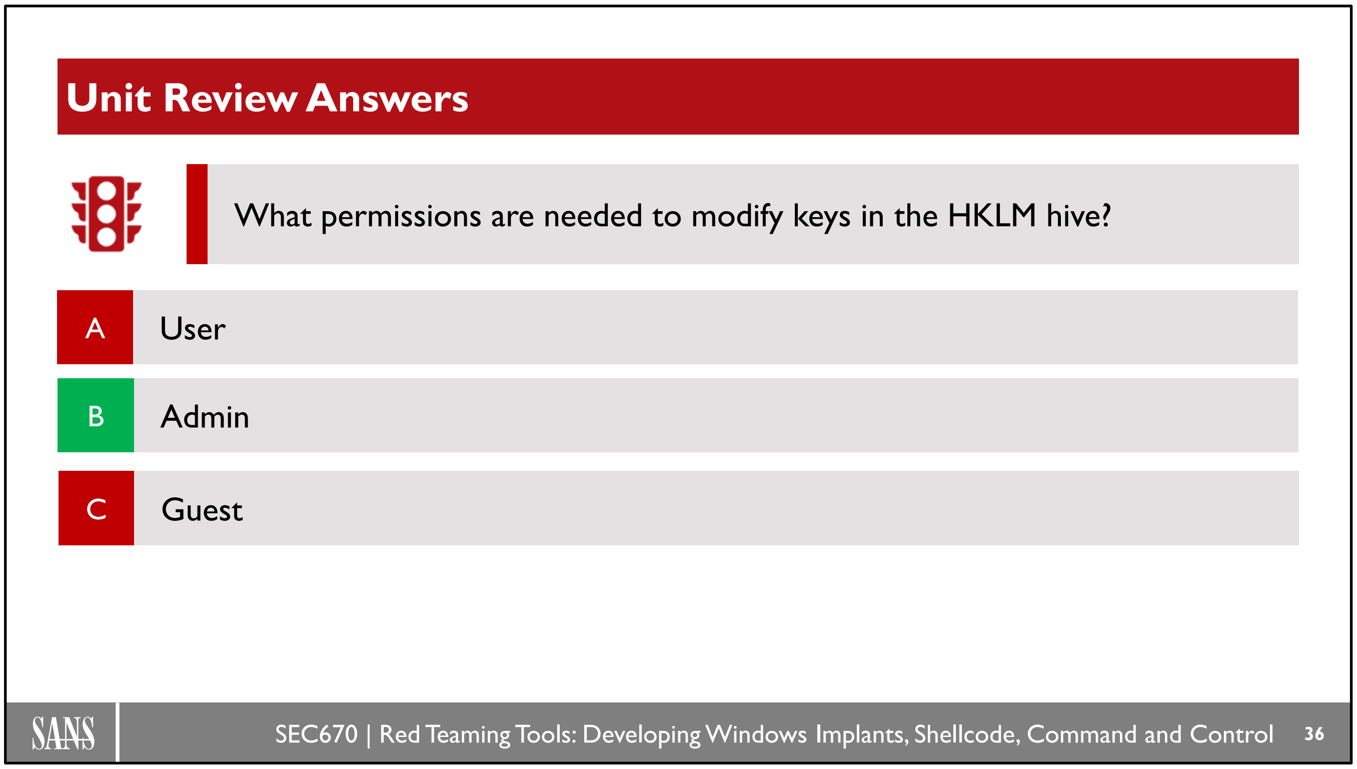
Unit Review Questions
Q: What permissions are needed to modify keys in the HKLM hive?

A: User

B: Admin

C: Guest

© SANS Institute 2024



Unit Review Answers

What permissions are needed to modify keys in the HKLM hive?

A	User
B	Admin
C	Guest

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 36

Unit Review Answers
Q: What permissions are needed to modify keys in the HKLM hive?

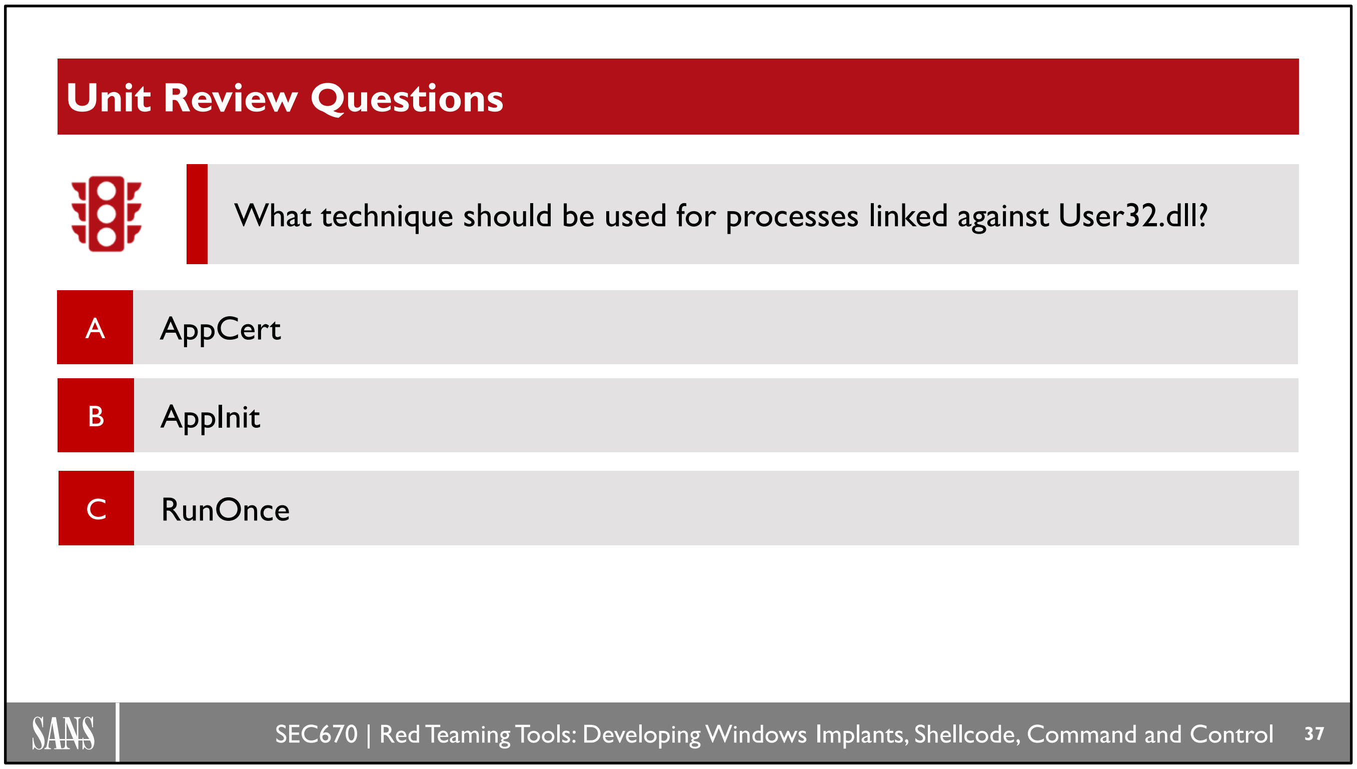
A: User

B: Admin

C: Guest

© 2024 Jonathan Reiter

© SANS Institute 2024



Unit Review Questions

What technique should be used for processes linked against User32.dll?

A AppCert

B AppInit

C RunOnce

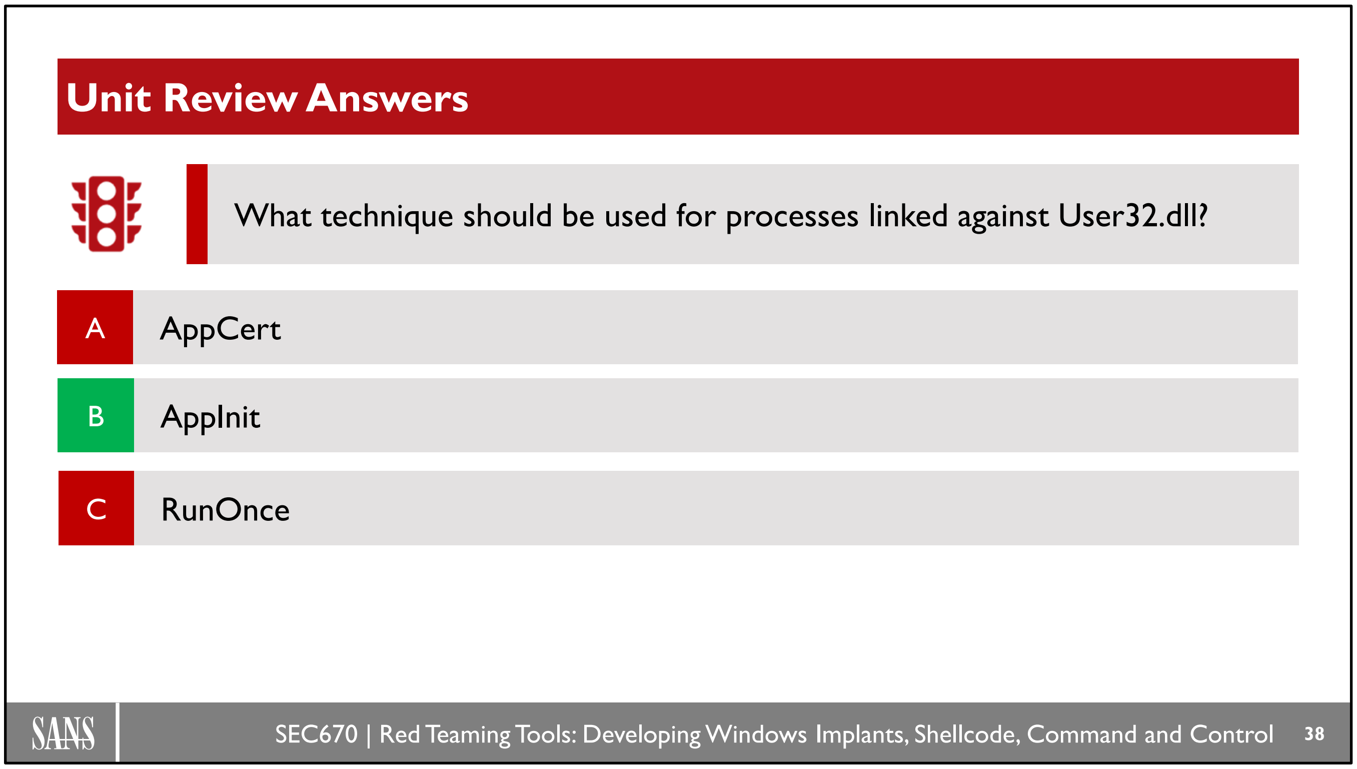
Unit Review Questions
Q: What technique should be used for processes linked against User32.dll?

A: AppCert

B: AppInit

C: RunOnce

© SANS Institute 2024



Unit Review Answers

What technique should be used for processes linked against User32.dll?

A	AppCert
B	AppInit
C	RunOnce

Unit Review Answers
Q: What technique should be used for processes linked against User32.dll?

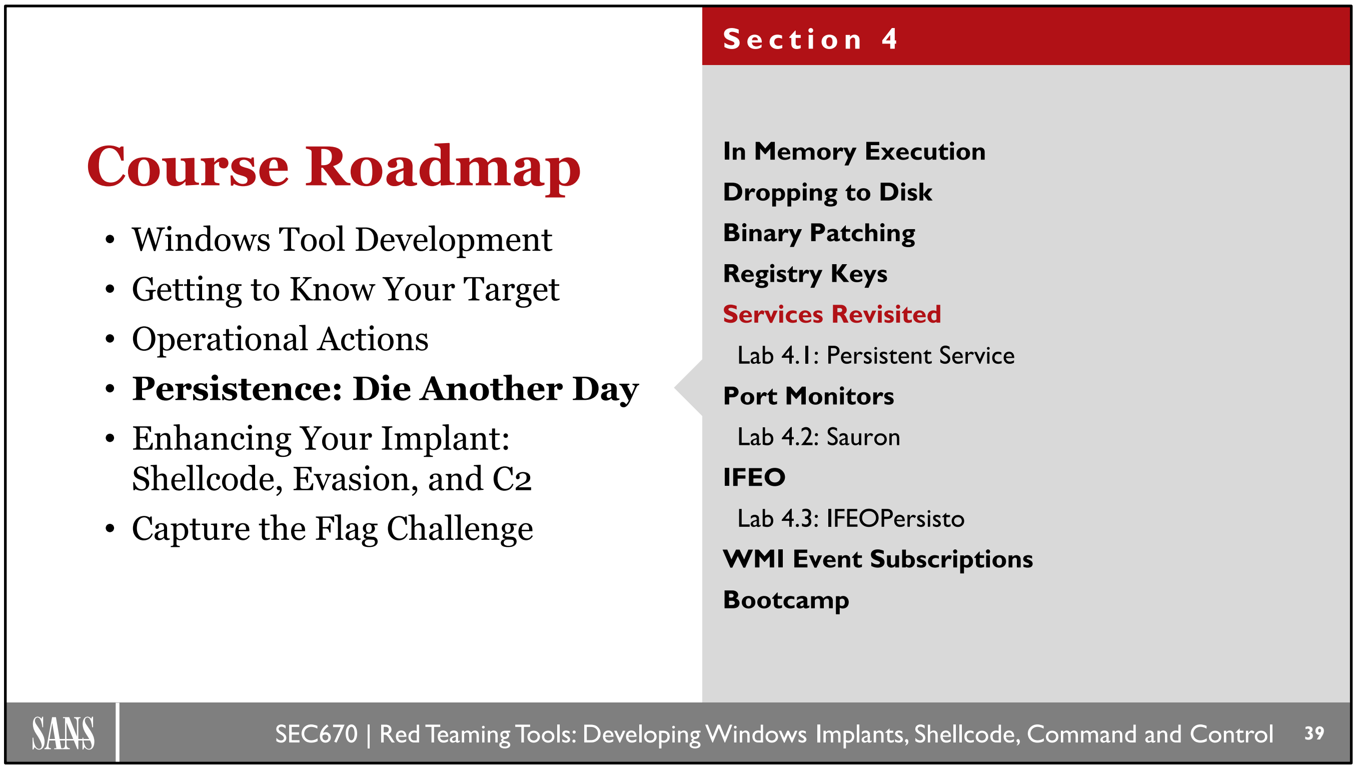
A: AppCert

B: AppInit

C: RunOnce

© 2024 Jonathan Reiter

© SANS Institute 2024



Course Roadmap

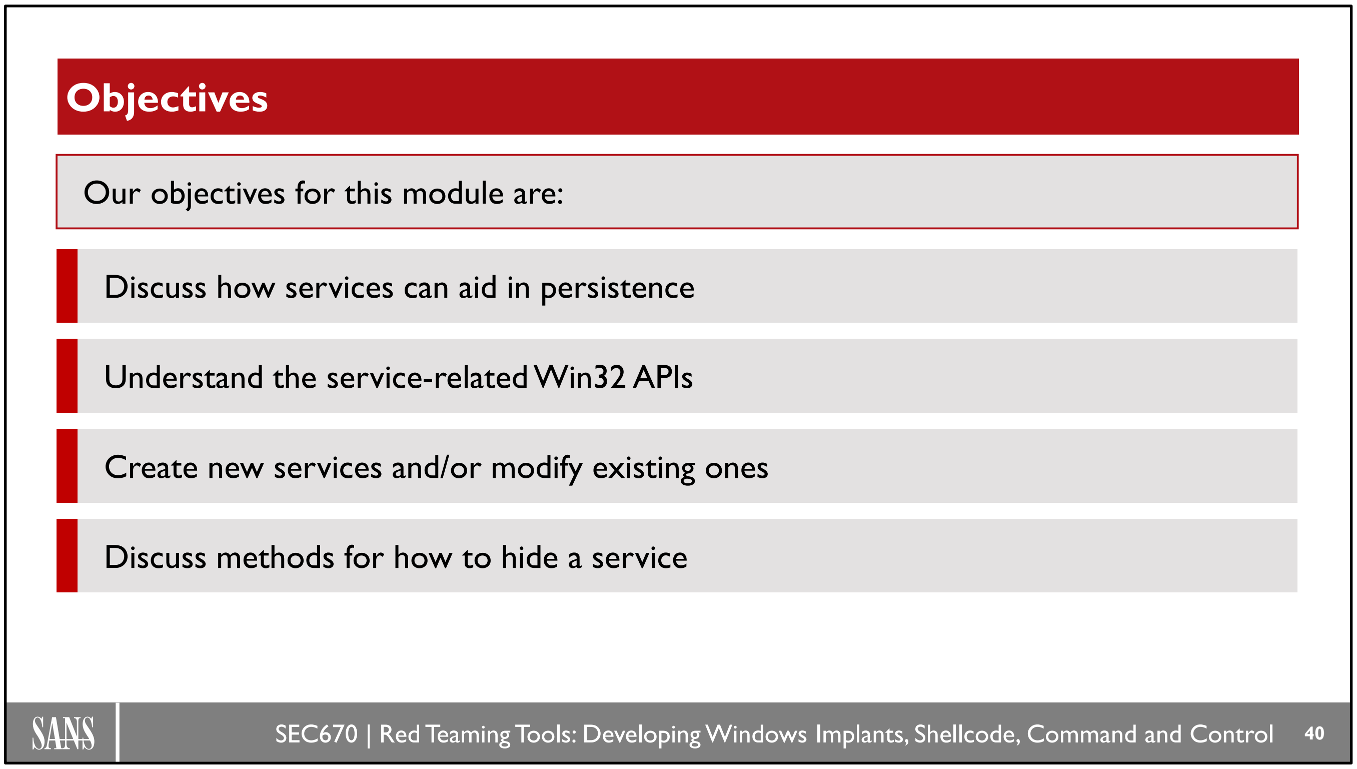
- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

Section 4

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 39

In this module, we will discuss how to persist using services. This includes services that we take advantage of or services that we create ourselves.

© SANS Institute 2024



Objectives

Our objectives for this module are:

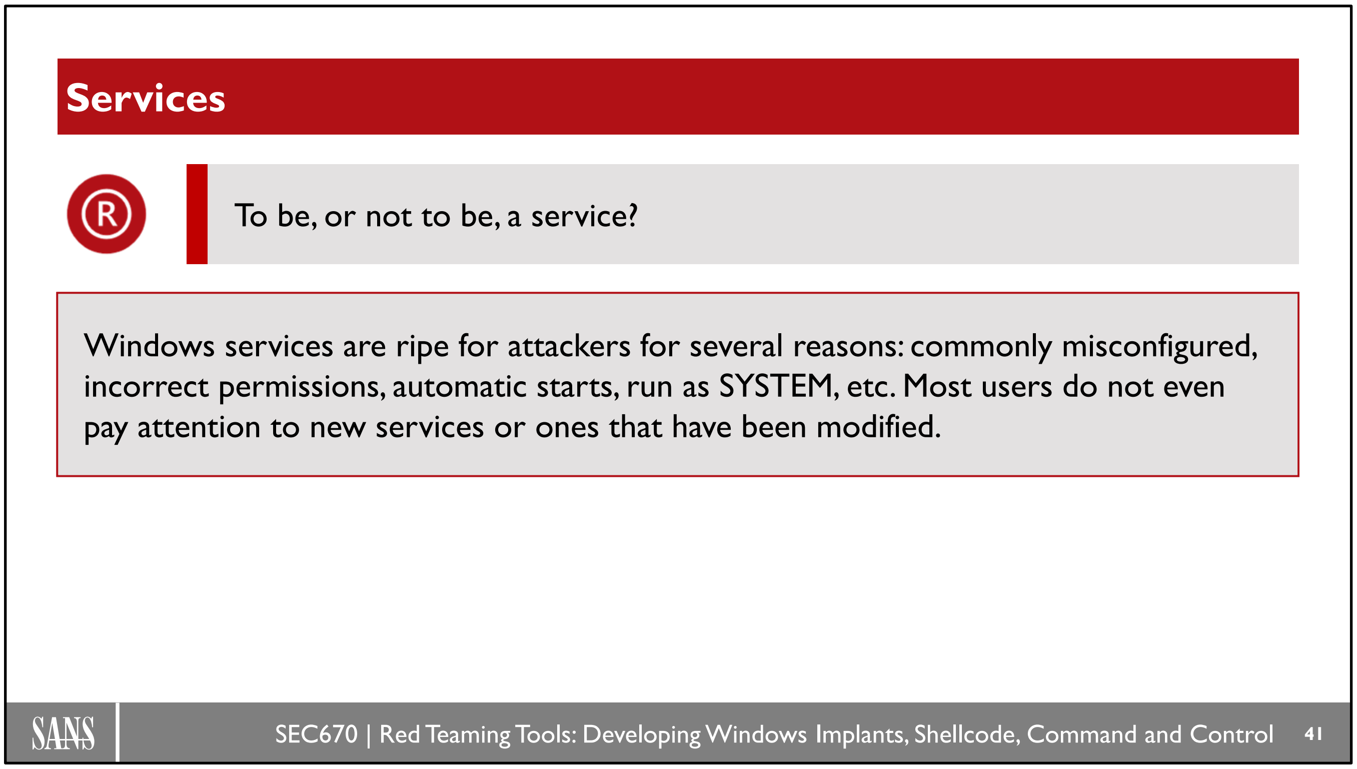
- Discuss how services can aid in persistence
- Understand the service-related Win32 APIs
- Create new services and/or modify existing ones
- Discuss methods for how to hide a service

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 40

Objectives

The objectives for this module are to revisit services, but this time around the focus is on persistence. We will spend some more time understanding how services are created by first understanding the Win32 APIs specific to Windows services. We will also explore how we could modify existing services to give us our persistence. Lastly, we will cover how services can be hidden from view and from tools that query services.

© SANS Institute 2024



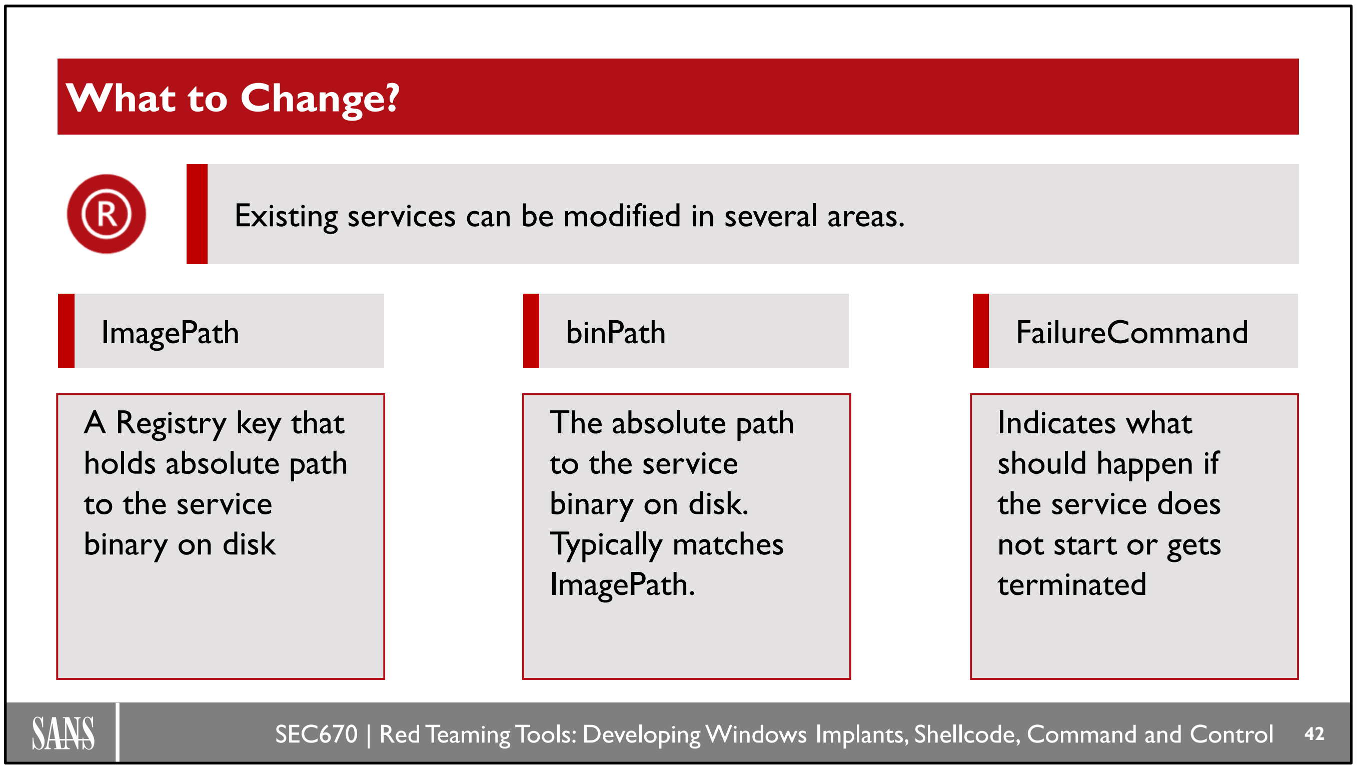
Services

To be, or not to be, a service?

Windows services are ripe for attackers for several reasons: commonly misconfigured, incorrect permissions, automatic starts, run as SYSTEM, etc. Most users do not even pay attention to new services or ones that have been modified.

Services
This is not the first time that Windows services have been mentioned, so they must be important, and they are. Services are not just important to the operating system, but they are also critical for red teamers. Earlier in the course we saw how they play a role for our enumeration and for privilege escalation. Now we can change our perspective to that of persistence. Red teamers, penetration testers, hackers, malware authors, etc. all look for services to use for their advantage. Services, especially third-party ones, might have been created with misconfigurations. A service's binary path, if it has spaces in it, must have quotes around it. The Registry key *ImagePath* can be targeted along with the service's Failure Command. We can either fire off our persistence binary when the target service starts, when it terminates improperly, or even if it fails to start properly. Of course, we can also create our own service for persistence instead of hijacking an existing one.

© SANS Institute 2024



What to Change?

Existing services can be modified in several areas.

ImagePath	binPath	FailureCommand
A Registry key that holds absolute path to the service binary on disk	The absolute path to the service binary on disk. Typically matches ImagePath.	Indicates what should happen if the service does not start or gets terminated

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 42

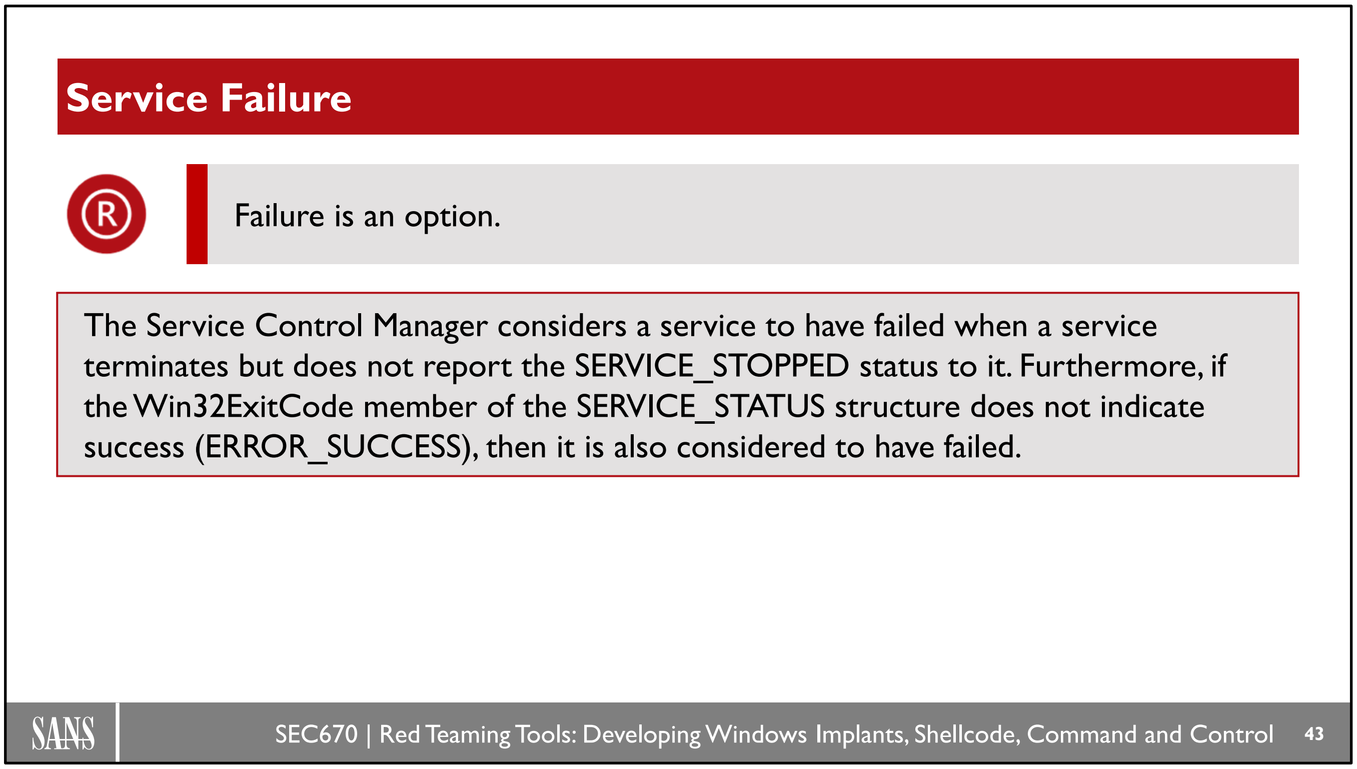
What to Change?
The *ImagePath* is a Registry key for the service. Most of the time, the value held is the path to the service executable on disk. The value, or arguments, in the key are passed to the service's main function—the service's entry point. When a program calls the ***CreateService*** API, the *lpBinaryPathName* argument indicates what the value will be for this key.

The *binPath* is the path of the service binary and the value typically will match that of the *ImagePath* Registry key. The system does not provide a default value for you, so this is a mandatory item. Many penetration testers or red teamers will forget to fulfill this requirement when making a service for their persistence only to see it fail within a few seconds. If we change the *binPath* to point to an executable that we dropped to disk, then we could achieve persistence. The *FailureCommand* is a command-line command that should be executed if the service fails for whatever reason. This change can also help us maintain our access to a system; but the service needs to fail, or our command will never kick off.

All of these could be manipulated via the command-line utility sc.exe.

© 2024 Jonathan Reiter

© SANS Institute 2024



Service Failure

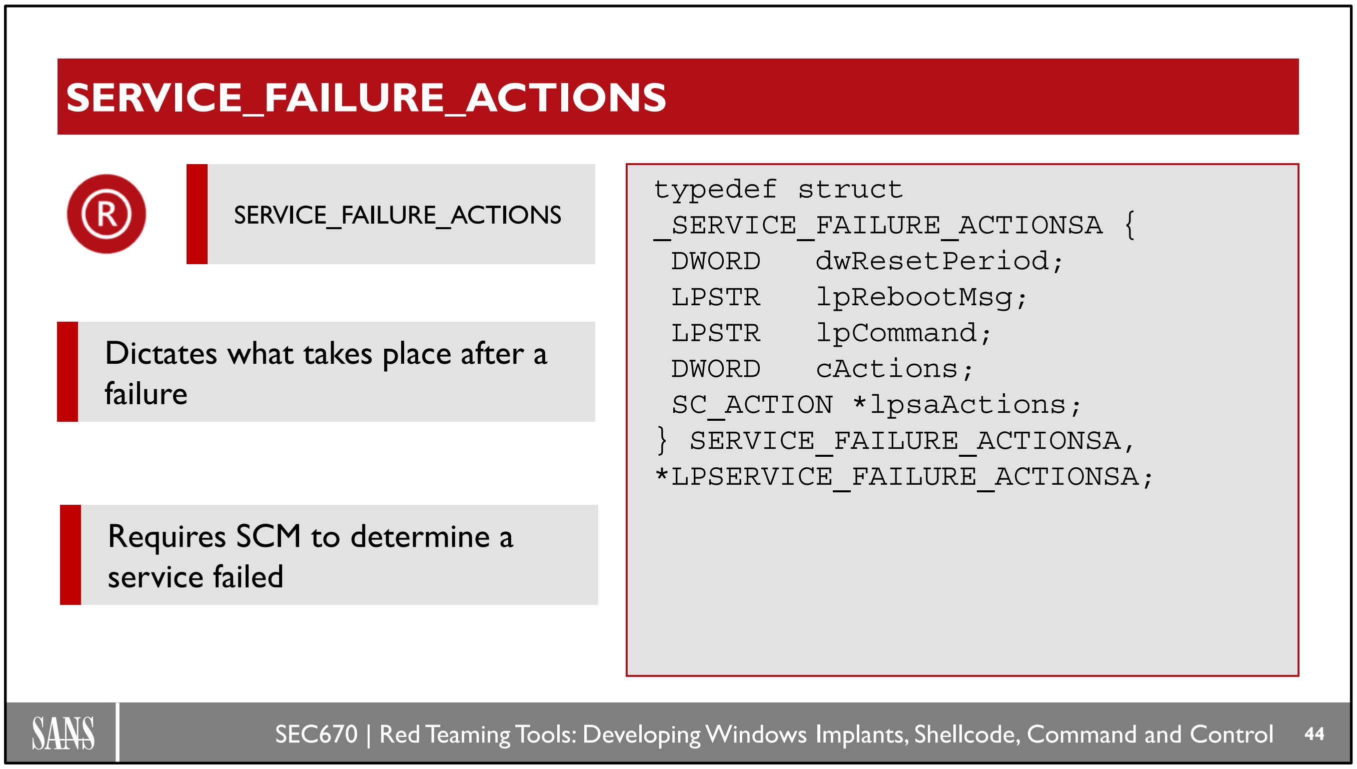
Failure is an option.

The Service Control Manager considers a service to have failed when a service terminates but does not report the SERVICE_STOPPED status to it. Furthermore, if the Win32ExitCode member of the SERVICE_STATUS structure does not indicate success (ERROR_SUCCESS), then it is also considered to have failed.

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 43

Service Failure
When does a service really fail and by whose standard? In the world of Windows services, the Service Control Manager (SCM) is the end-all-be-all. Services must answer to the SCM and if they do not, they are promptly terminated. If that terminated service is yours—well, there goes your persistence. Services are deemed as a failure when it does not start up properly, and starting up properly means sending the correct status to the SCM in time. When it comes to service termination, services must also send a status to the SCM. The SERVICE_STOPPED status must be sent before its function returns or else the SCM will follow any failure actions that have been registered for the service. Even more so, when a service exits but fails to set an ERROR_SUCCESS, or NO_ERROR (0) inside the SERVICE_STATUS structure, it is deemed a failure. It is very critical that services inform the SCM of what is happening and in a timely fashion because the SCM is not very forgiving. The failure actions the SCM looks for are defined in the SERVICE_FAILURE_ACTIONSA structure.

© SANS Institute 2024



SERVICE_FAILURE_ACTIONS

SERVICE_FAILURE_ACTIONS

Dictates what takes place after a failure

Requires SCM to determine a service failed

```
typedef struct
_SERVICE_FAILURE_ACTIONSA {
 DWORD   dwResetPeriod;
 LPSTR   lpRebootMsg;
 LPSTR   lpCommand;
 DWORD   cActions;
 SC_ACTION *lpsaActions;
} SERVICE_FAILURE_ACTIONSA,
*LPSERVICE_FAILURE_ACTIONSA;
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 44

SERVICE_FAILURE_ACTIONS
The SERVICE_FAILURE_ACTIONS structure is utilized by the SCM as a representation of what must happen when the SCM fails a service. The actions defined in the structure are executed by the SCM each time the service enters a failure state. Existing services can have these optional configuration parameters changed using the ***ChangeServiceConfig2*** API and passing the SERVICE_CONFIG_FAILURE_OPTIONS flag for the *dwInfoLevel* parameter. The members of the structure are described below.

dwResetPeriod, is the period that must expire before the failure count is decremented to 0. The SCM maintains a count of failures so that the correct failure actions can be executed after repeat failures. A common example would be for critical services that may delay their restart time with each successive failure.

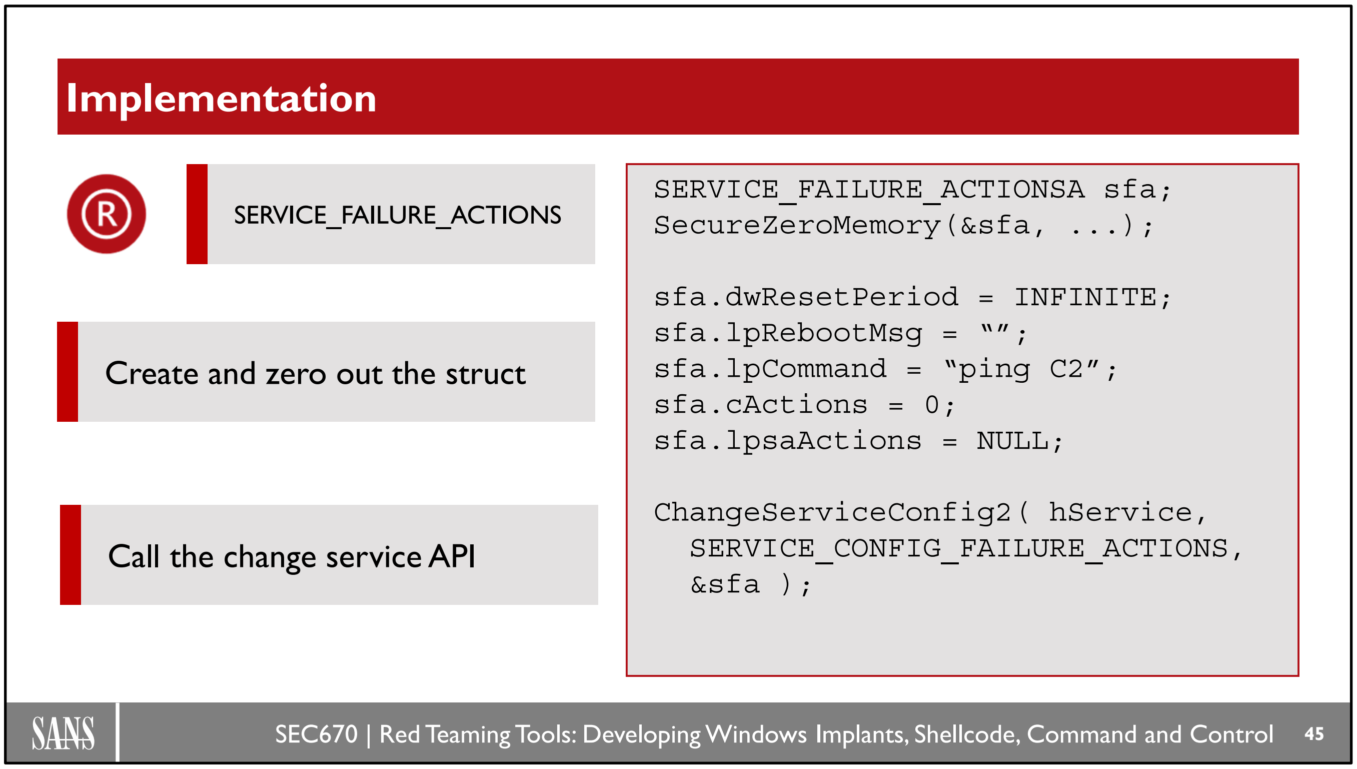
lpRebootMsg, is the message to be sent to the server to indicate to users that the system will be rebooted due to the service's failure. NULL here indicates to use the system's default message, but if you pass empty quotes (""), then the default message is overridden, and no message will be displayed to the users.

lpCommand, is the command line that is passed to the ***CreateProcess*** API. The command will be executed with the same permissions as the service.

cActions, is the number of elements that are in the *lpsaActions* array.

lpsaActions, is an array of SC_ACTION structures that indicate what action the SCM should take in the event the service fails for the nth time. For example, if there are four items in the array and the service failed for the fourth time, then that entry will be executed.

© SANS Institute 2024



Implementation

SERVICE_FAILURE_ACTIONS

Create and zero out the struct

Call the change service API

```
SERVICE_FAILURE_ACTIONSA sfa;
SecureZeroMemory(&sfa, ...);

sfa.dwResetPeriod = INFINITE;
sfa.lpRebootMsg = “”;
sfa.lpCommand = “ping C2”;
sfa.cActions = 0;
sfa.lpsaActions = NULL;

ChangeServiceConfig2( hService,
  SERVICE_CONFIG_FAILURE_ACTIONS,
  &sfa );
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 45

Implementation
The pseudo code one the slide shows just one of several ways you could go about implementing a service’s failure actions. It is best to do this after you have installed and created the service. The overall idea for this small example is to ping the configured C2 server if the service fails. The beginning of the code starts off with the SERVICE_FAILURE_ACTIONS struct. We make a variable called *sfa* of that type and securely zero out the memory of that struct by the size of the struct. The struct members are modified in the order they appear in the struct’s definition, to keep things easier to read. The reset period has an INFINITE value because we do not care if the failure count is reset or not—for this example, anyway. For the reboot message, we pass in an empty string because we do not want the default message or really any message at all. The empty string overrides that and NULL does not change the reboot message. The command is the heart and soul for this method. Our example simply pings our C2 server to demonstrate what could be done. There are no actions in the SC_ACTION array, so 0 and NULL are used. After you are done modifying the structure, the *ChangeServiceConfig2* API can be called.

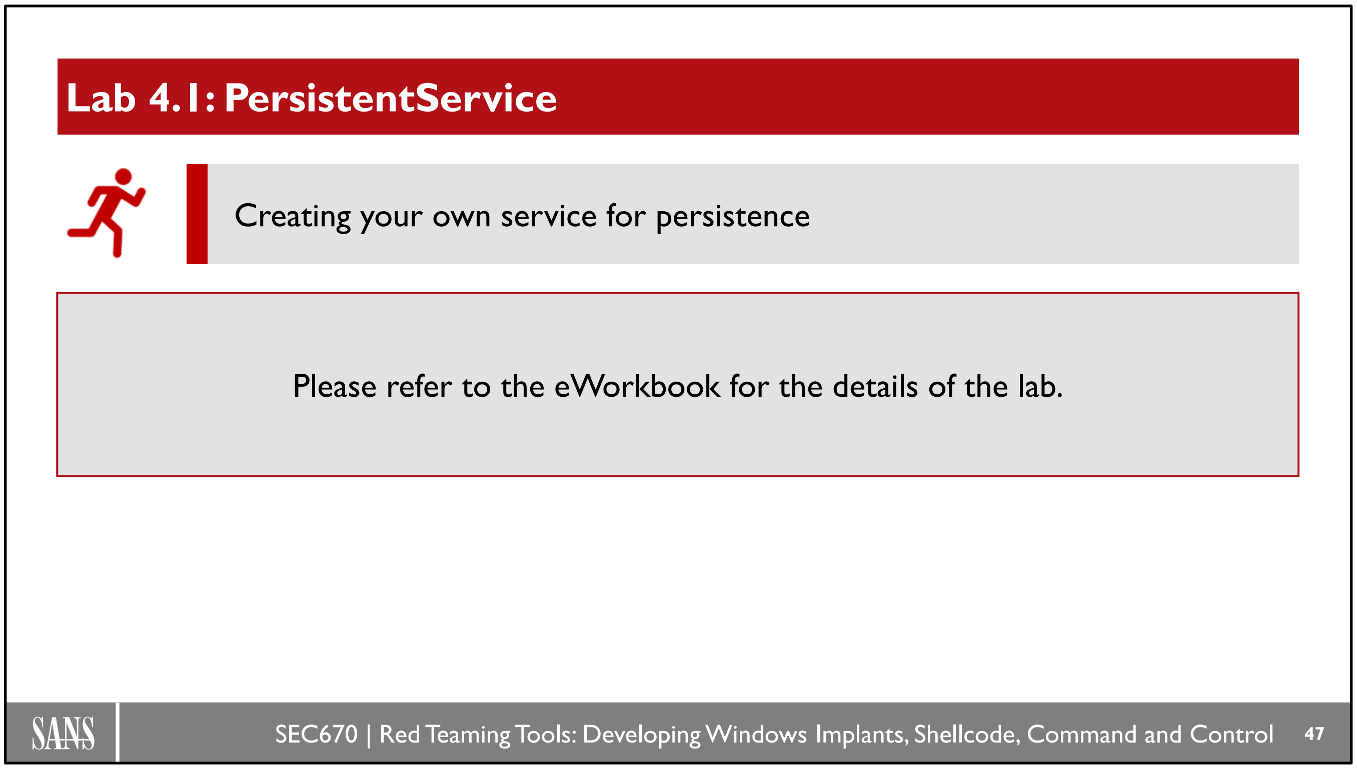
© SANS Institute 2024

Persistent Service

Source code review!

Persistent Service
Time to jump into the source code and explain it before you implement it on your own.

© SANS Institute 2024



Lab 4.1: PersistentService

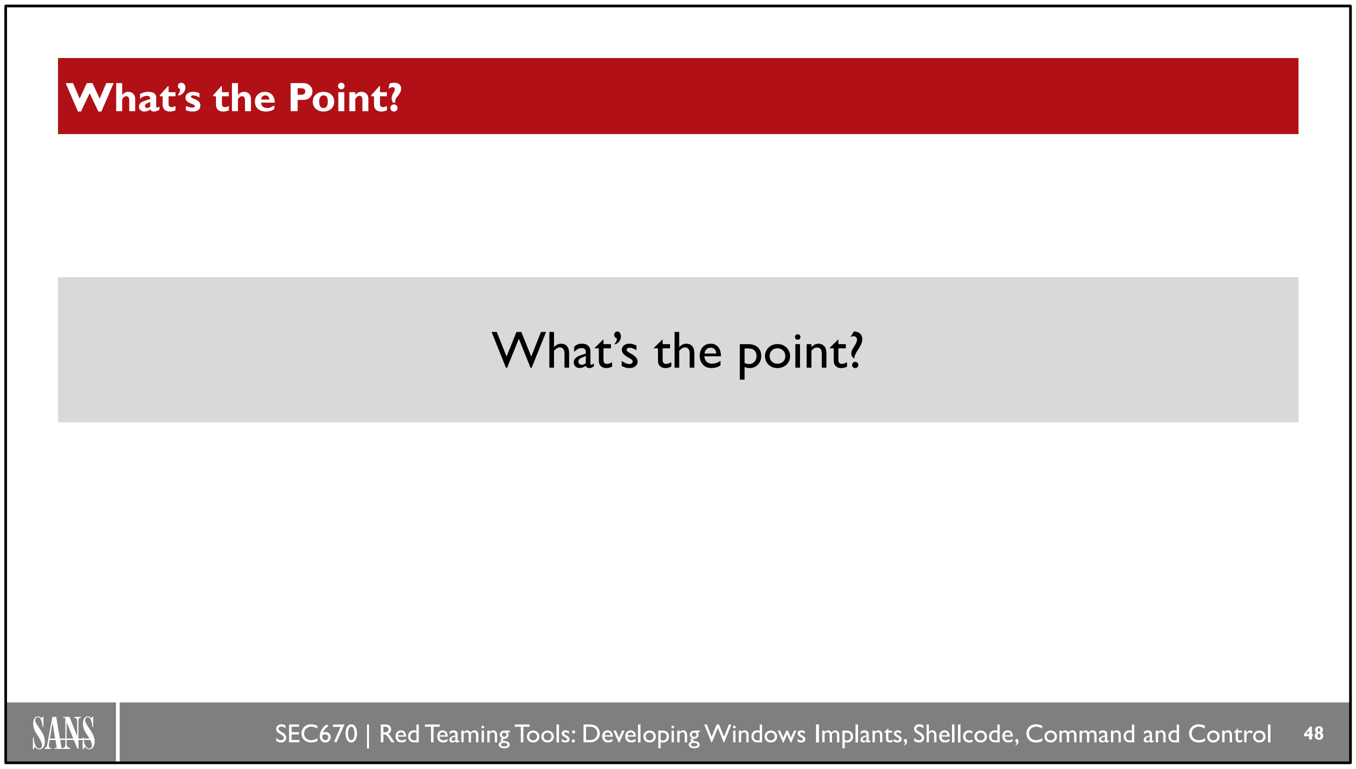
Creating your own service for persistence

Please refer to the eWorkbook for the details of the lab.

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 47

Lab 4.1: PersistentService
Please refer to the eWorkbook for the details of this lab.

© SANS Institute 2024



What's the Point?

What's the point?

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 48

What's the Point?
The point of this lab was to learn how to either create, install, or persist with your own service. When you have administrative and/or SYSTEM privileges, creating a service is a perfect action to take.

© SANS Institute 2024

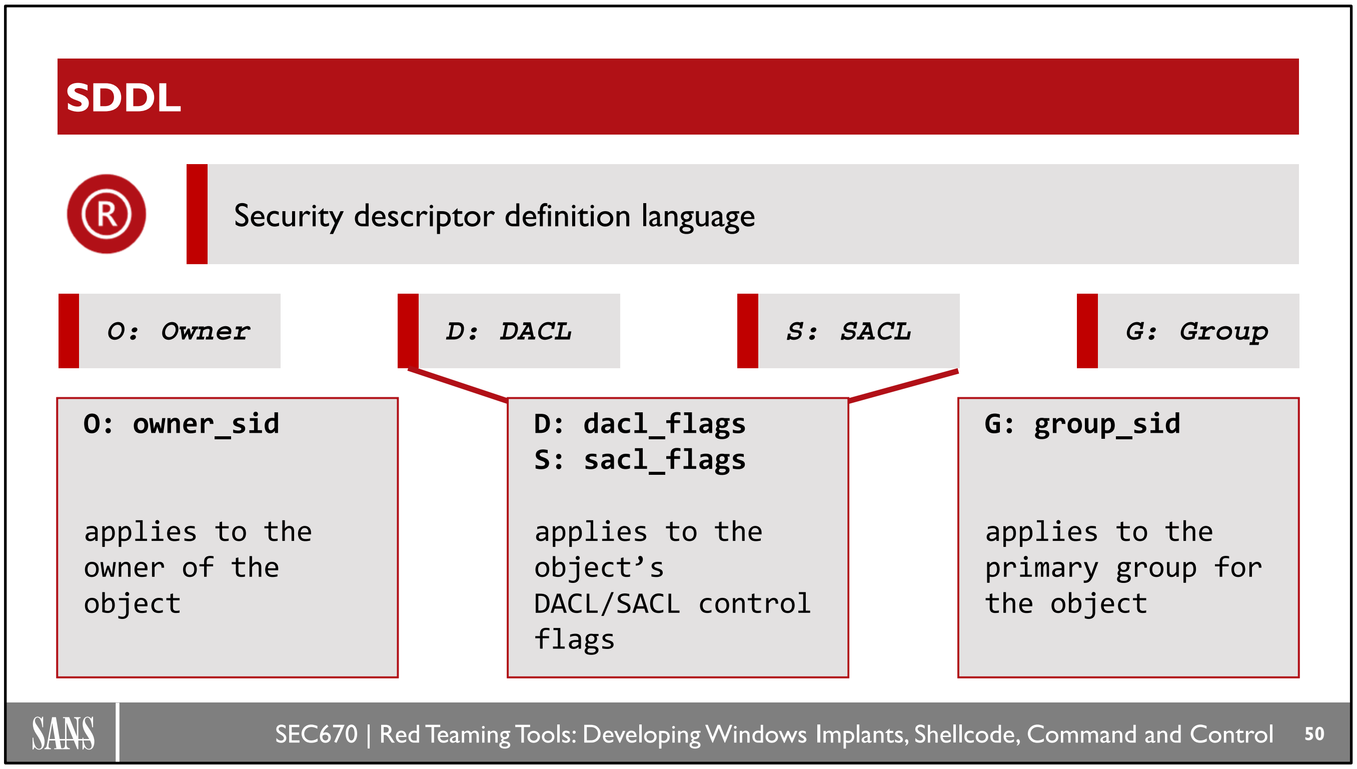
What Else?

Hiding a service

When the opportunity is there to hide ourselves from view, we tend to take advantage of it. We can hide a service similar to how we can hide a process from a user and other tools. The beauty about this is there is no need for a kernel driver of function hooking.

What Else?
What else can we do with services? Well, one thing we can do is we can hide a service. A service that we created for our own persistence needs can be hidden from the user and from several tools like the sc.exe command-line utility. We hide a service for similar reasons that we hide a process; stay hidden, stay safe. When you make it harder for users and tools to detect you, it helps you avoid detection longer. Hiding a service that implements your persistence portion could really help you stay on a system longer. With this technique, we do not need to hook any functions, nor do we need to install a kernel driver. Each service has a security descriptor tied to it and we can manipulate that object. The way we do this is through a Microsoft-specific language called SDDL, the security descriptor definition language.

© SANS Institute 2024



SDDL

Security descriptor definition language

O: Owner
D: DACL
S: SACL
G: Group
O: owner_sid
applies to the owner of the object
D: dacl_flags
S: sacl_flags
applies to the object's DACL/SACL control flags
G: group_sid
applies to the primary group for the object
SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 50

SDDL
The security descriptor definition language, or SDDL, is not really a language, per se, but something that is specific to the definition of a string format for two APIs: ***ConvertSecurityDescriptorToStringSecurityDescriptor*** and ***ConvertStringSecurityDescriptorToSecurityDescriptor***. The APIs are used to describe the security descriptor as a string. With SDDL, we can also hit specific components of a security descriptor just using strings. There is a specific format that must be followed for our hiding technique to work. At first, it might seem like it does not make any sense, but after a bit of practice crafting some strings, you will get the hang of it.

A security descriptor will have four main components, such as the Owner (O:), the SACL (S:) and DACL (D:), and the Primary Group (G:). The Owner is the owner of the object as identified by an *owner_sid*. The SACL has control flags as represented by *sacl_flags*. The DACL also has control flags as represented by *dacl_flags*. The Primary Group is the object's primary group as identified by the *group_sid*. There is also a *string_ace* that is enclosed in parenthesis that is used to describe the ACE for a SACL or DACL security descriptor. You must use an *ace_string* when changing the DACL or SACL of the object.

The control flags that can be used for the DACL or SACL are the following list of items:

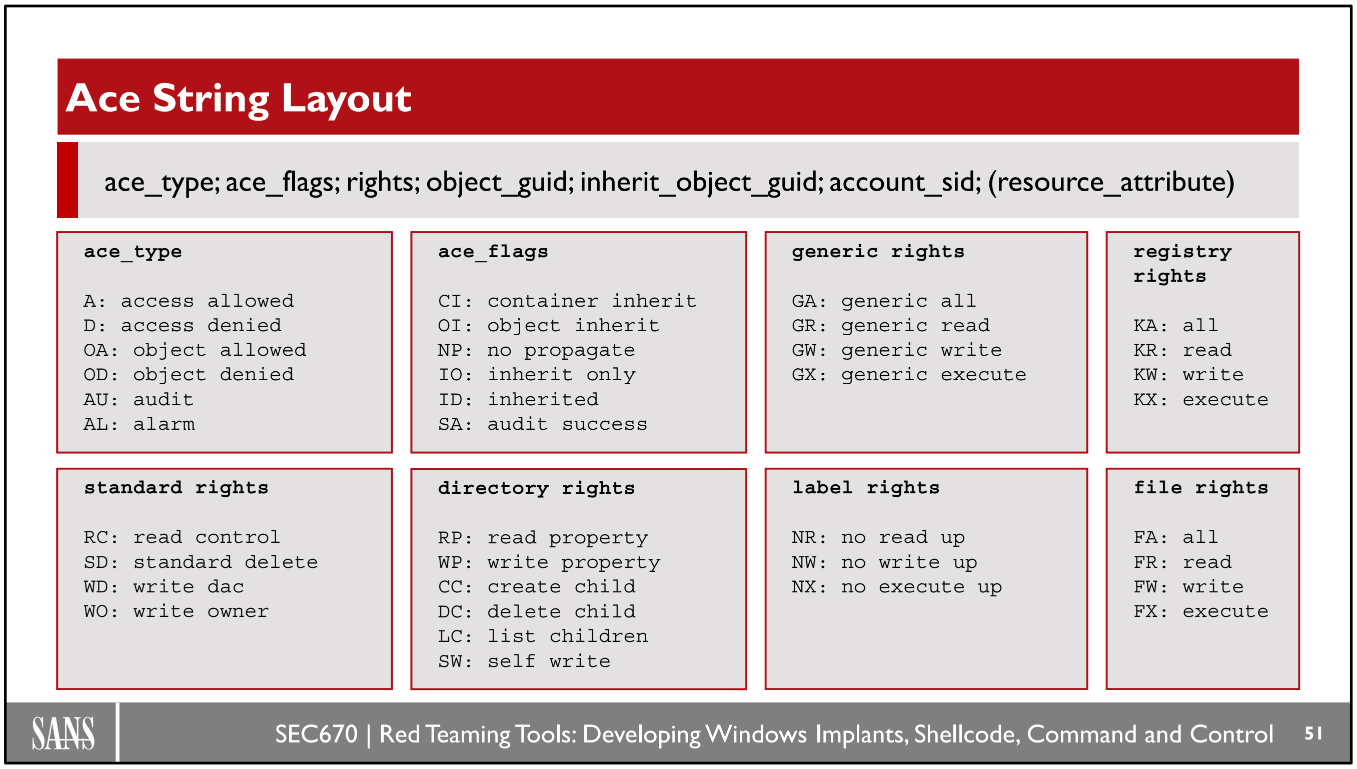
- **"P"** - *SDDL_PROTECTED* - SE_DACL_PROTECTED flag is set
- **"AR"** - *SDDL_AUTO_INHERIT_REQ* - SE_DACL_AUTO_INHERIT_REQ flag is set
- **"AI"** - *SDDL_AUTO_INHERITED* - SE_DACL_AUTO_INHERITED flag is set
- **"NO_ACCESS_CONTROL"** - *SDDL_NULL_ACL* - ACL is NULL

For reference, a DACL is a discretionary access control list and SACL is a system access control list.

Reference:
https://docs.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-string-format

© 2024 Jonathan Reiter

© SANS Institute 2024



Ace String Layout

ace_type; ace_flags; rights; object_guid; inherit_object_guid; account_sid; (resource_attribute)

ace_type	
A:	access allowed
D:	access denied
OA:	object allowed
OD:	object denied
AU:	audit
AL:	alarm

ace_flags	
CI:	container inherit
OI:	object inherit
NP:	no propagate
IO:	inherit only
ID:	inherited
SA:	audit success

generic rights	
GA:	generic all
GR:	generic read
GW:	generic write
GX:	generic execute

registry rights	
KA:	all
KR:	read
KW:	write
KX:	execute

standard rights	
RC:	read control
SD:	standard delete
WD:	write dac
WO:	write owner

directory rights	
RP:	read property
WP:	write property
CC:	create child
DC:	delete child
LC:	list children
SW:	self write

label rights	
NR:	no read up
NW:	no write up
NX:	no execute up

file rights	
FA:	all
FR:	read
FW:	write
FX:	execute

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 51

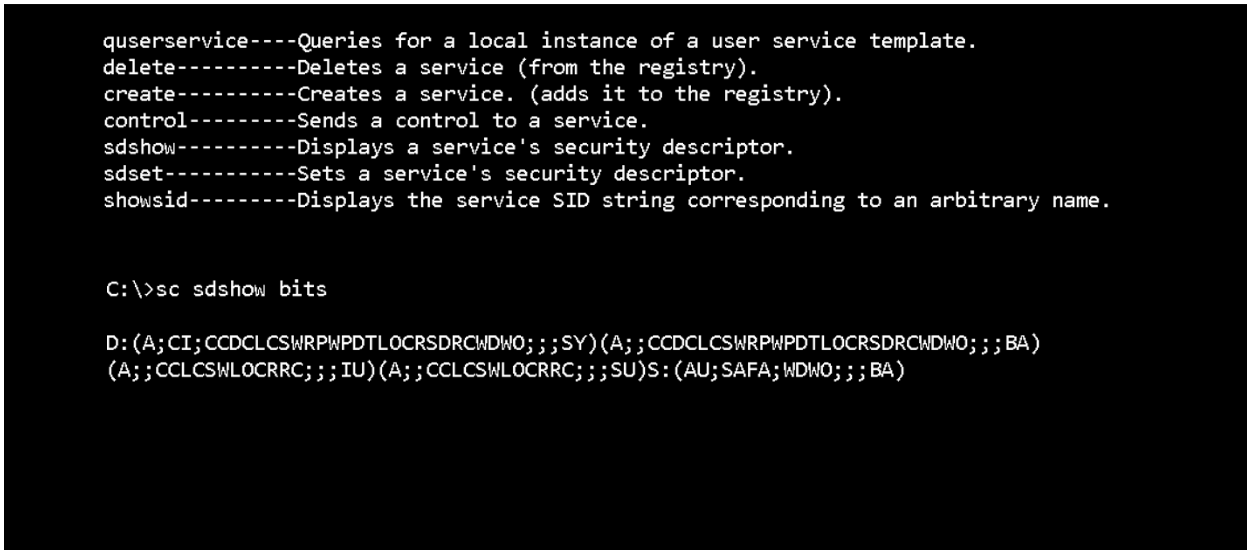
Ace String Layout
As mentioned on the previous slide, *ace_strings* are used whenever we want to change the object's DACL or SACL. This slide details the syntax used for creating an *ace_string* and a short description of several of the values. There is a lot on the slide, but there is even more documentation to be found on the MSDN site. At the top of the slide is the syntax for how an *ace_string* should be formatted. The *ace_type* is the first field in the string and can be any number of types that belong to that group. The simplest is to allow or deny access to an object. The ace_flags field is typically skipped since it does not directly apply to hiding a service. We indicate an empty field value or a skipped field like so, ";;". Generic access rights, standard access rights, directory service object access rights, file access rights, registry access rights should sound somewhat familiar to you with the APIs that have been used so far in the course. Not mentioned on the slide due to lack of room are the *account_sid* values. The *account_sid* values indicate the owner or primary group of the object. The famous SID strings (S-1-1-0) can be used here, or you can use the equivalent constant with some of them listed below.

- AU: Authenticated users
- BA: Built-in administrators
- DU: Domain users
- LA: Local admin
- SY: Local system

© 2024 Jonathan Reiter

© SANS Institute 2024

Viewing Security Descriptors

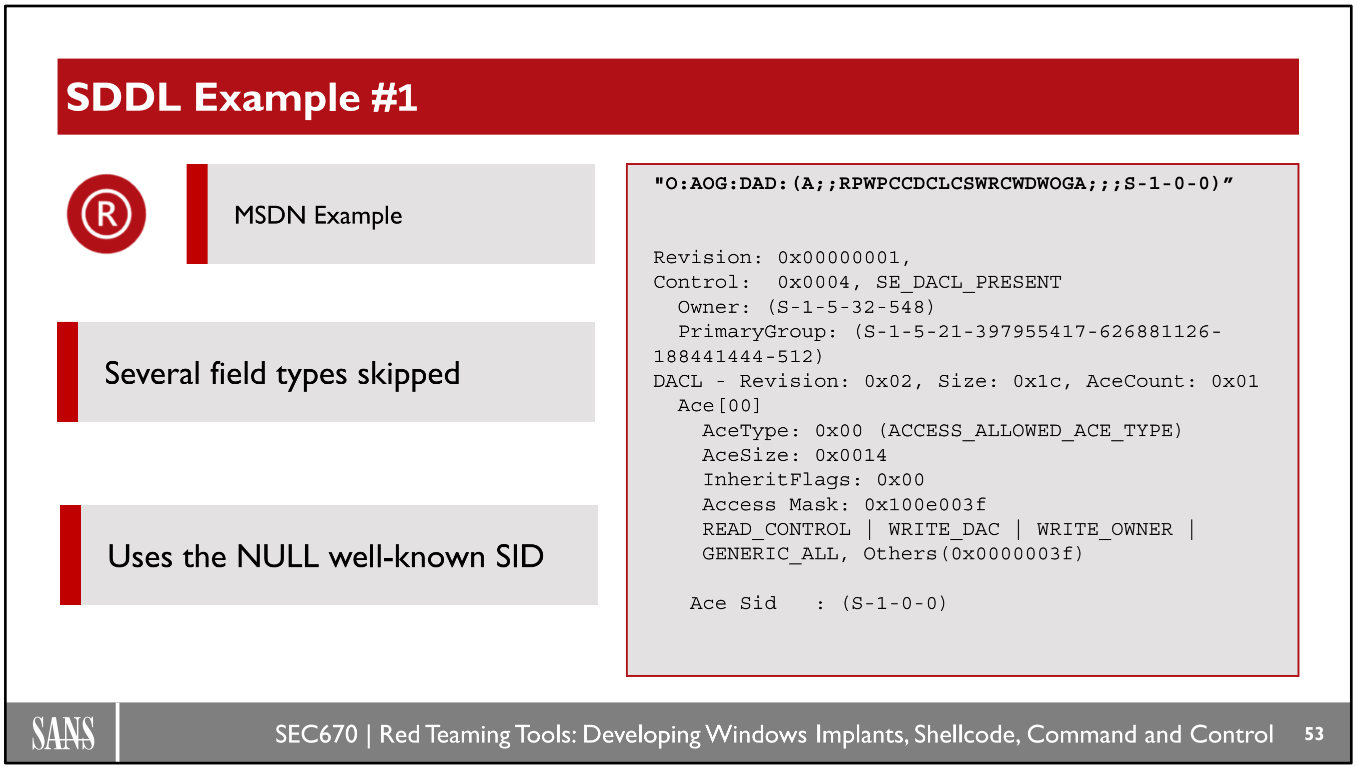


```
quserservice----Queries for a local instance of a user service template.
delete----------Deletes a service (from the registry).
create----------Creates a service. (adds it to the registry).
control---------Sends a control to a service.
sdshow----------Displays a service's security descriptor.
sdset-----------Sets a service's security descriptor.
showsid---------Displays the service SID string corresponding to an arbitrary name.


C:\>sc sdshow bits

D:(A;CI;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
(A;;CCLCSWLOCRRC;;;IU)(A;;CCLCSWLOCRRC;;;SU)S:(AU;SAFA;WDWO;;;BA)
```

Viewing Security Descriptors

Using the sc.exe command-line utility, we can view a service's security descriptor. Running the program with the /? argument shows the help menu. From the help menu we can see the argument *sdshow* and its description: Displays a service's security descriptor. This is what we want. From here we can choose a service, like BITS, and see what its security descriptor currently is. With the information we now know about SDDL and *ace_strings*, we can interpret the output without too much headache.

© SANS Institute 2024



SDDL Example #1

MSDN Example

Several field types skipped

Uses the NULL well-known SID

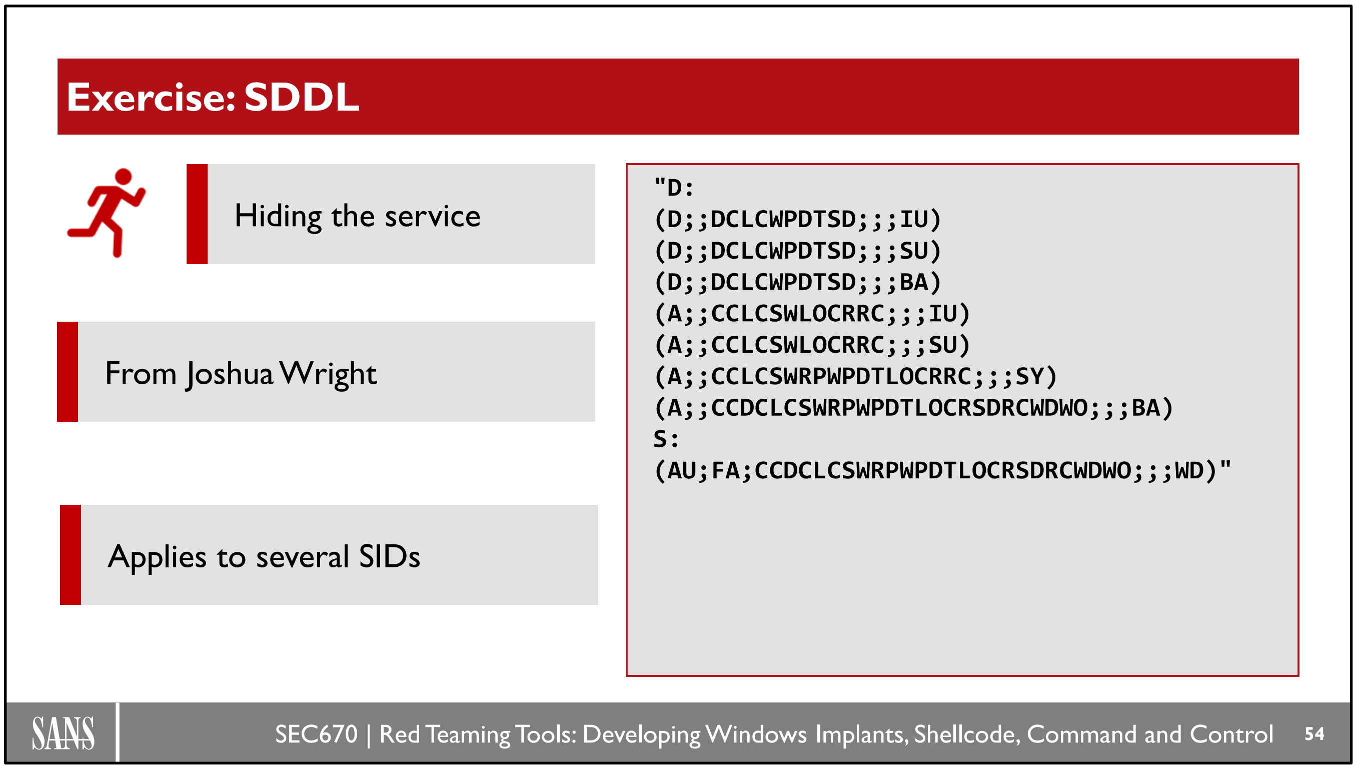
```
"O:AOG:DAD:(A;;RPWPCCDCLCSWRCWDWOGA;;;S-1-0-0)"


Revision: 0x00000001,
Control:  0x0004, SE_DACL_PRESENT
  Owner: (S-1-5-32-548)
  PrimaryGroup: (S-1-5-21-397955417-626881126-
188441444-512)
DACL - Revision: 0x02, Size: 0x1c, AceCount: 0x01
  Ace[00]
    AceType: 0x00 (ACCESS_ALLOWED_ACE_TYPE)
    AceSize: 0x0014
    InheritFlags: 0x00
    Access Mask: 0x100e003f
    READ_CONTROL | WRITE_DAC | WRITE_OWNER |
    GENERIC_ALL, Others(0x0000003f)

   Ace Sid    : (S-1-0-0)
```

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 53

SDDL Example #1
This first example is taken directly from MSDN documentation. The *ace_string* is the most interesting piece of the string and here is the breakdown. Since the first field of the *ace_string* is the *ace_type*, this is allowing access and more specifically the ACCESS_ALLOWED_ACE_TYPE. The next field is *ace_flags*, which is empty. The *rights* field is next and is where our focus is for truly understanding what is happening. We break this down in pairs starting with "RP", which indicates the read property (ADS_RIGHT_DS_READ_PROP), "WP" indicates write property (ADS_RIGHT_DS_WRITE_PROP), "CC" indicates create child (ADS_RIGHT_DS_CREATE_CHILD), "DC" indicates delete child (ADS_RIGHT_DS_DELETE_CHILD), "LC" indicates list children (ADS_RIGHT_ACTRL_DS_LIST), SW" indicates (ADS_RIGHT_DS_SELF), "RC" indicates read control (READ_CONTROL) and is a standard access right, "WD" indicates write DAC (WRITE_DAC), "WO" indicates write owner (WRITE_OWNER), and "GA" indicates generic all (GENERIC_ALL). The *object_gui* is skipped and the *inherit_object_guid* is also skipped. For the *account_sid* we see the well-known NULL SID is used because the actual SID might not have been known at the time. To our benefit, the interpretation of the string has been given for a full explanation of the effects.

Reference:
https://docs.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-string-format

© SANS Institute 2024



Exercise: SDDL

Hiding the service

From Joshua Wright

Applies to several SIDs

```
"D:
(D;;DCLCWPDTSD;;;IU)
(D;;DCLCWPDTSD;;;SU)
(D;;DCLCWPDTSD;;;BA)
(A;;CCLCSWLOCRRC;;;IU)
(A;;CCLCSWLOCRRC;;;SU)
(A;;CCLCSWRPWPDTLOCRRC;;;SY)
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
S:
(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)"
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 54

Exercise: SDDL
SANS instructor and author Joshua Wright crafted the SDDL string shown on the slide during an engagement and used it to hide a service he created. Take 10-15 minutes to break down this real-world example piece by piece just like we did together for the previous example from MSDN.

Reference:
https://www.sans.org/blog/red-team-tactics-hiding-windows-services/

© 2024 Jonathan Reiter

© SANS Institute 2024

Exercise: SDDL: The Solution

```
"D:        DACL
(D;;DCLCWPDTSD;;;IU)
  interactive user, deny: delete, list, write, delete tree, standard delete
(D;;DCLCWPDTSD;;;SU)
  service user, deny: delete, list, write, delete tree, standard delete
(D;;DCLCWPDTSD;;;BA)
  built-in admins, deny: delete, list, write, delete tree, standard delete
(A;;CCLCSWLOCRRC;;;IU)
  interactive user, allow: create, list, selfwrite, list obj, control access, read control
(A;;CCLCSWLOCRRC;;;SU)
  service user, allow: create, list, selfwrite, list obj, control access, read control
(A;;CCLCSWRPWPDTLOCRRC;;;SY)
  local system, allow: create, list, selfwrite, read/write property, delete tree, list obj...
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
  built-in admins, allow: create, delete, list, selfwrite, read/write property, delete tree...
S:         SACL
(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)"
  everyone,
```

Exercise: SDDL: The Solution
Here is the breakdown of the string applied to their service. To make this a bit easier, we can first look at the SIDs for which the rules will apply. Starting with the DACL portion (D:), we see "IU", "SU", "BA", and "SY" being used for the *account_sid,* which apply to the logged-on interactive user, the service logon user, the built-in administrators, and local system, respectively. The logged-on user is denied delete and list accesses to the object, cannot write or delete the directory recursively, and cannot even use standard delete. The service logon user is denied the same items as the logged-on user as well as the built-in admins. The SACL has an *ace_string* that applies everyone's access like so: audited, audited failure, create, delete, list, self-write, read/write property, delete tree, list object, control read, standard delete, read control, write dac, and write owner. If you were to apply this to a service that you created, try to query it with various tools like PowerShell's Get-Service cmdlet, the sc.exe utility, etc. and see what you are shown.

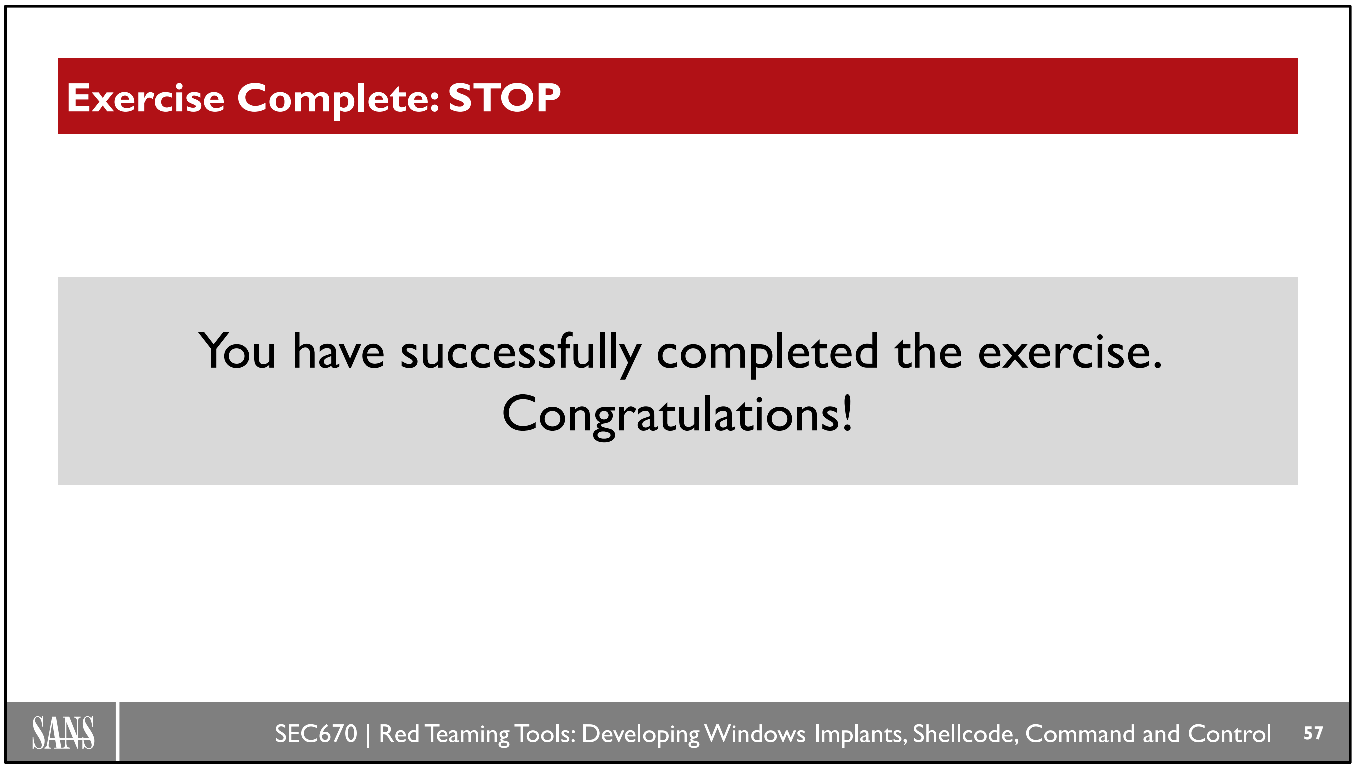
© SANS Institute 2024

Exercise: SDDL: Unhiding the Service

```
sc.exe sdset SWCUEngine
D:
(A;;CCLCSWRPWPDTLOCRRC;;;SY)
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
(A;;CCLCSWLOCRRC;;;IU)
(A;;CCLCSWLOCRRC;;;SU)
S:
(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)


The command on a single line
sc.exe sdset SWCUEngine \
D:(A;;CCLCSWRPWPDTLOCRRC;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)(A;;CCLCSWLOCRRC
;;;IU)(A;;CCLCSWLOCRRC;;;SU)S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)
```

Exercise: SDDL: Unhiding the Service
As professional red teamers, we need to clean up and one of the ways we can do that is to unhide the service. The command on the slide will restore the accesses back so that users and tools can “see” the service again. As can be seen on the slide, the command is broken out for readability. When running the command on target, it would all be on a single line as shown on the bottom of the slide.

© SANS Institute 2024



Exercise Complete: STOP

You have successfully completed the exercise.
Congratulations!

Exercise Complete: STOP
This marks the completion of the exercise. Congratulations on successfully completing all the exercise steps!

© SANS Institute 2024

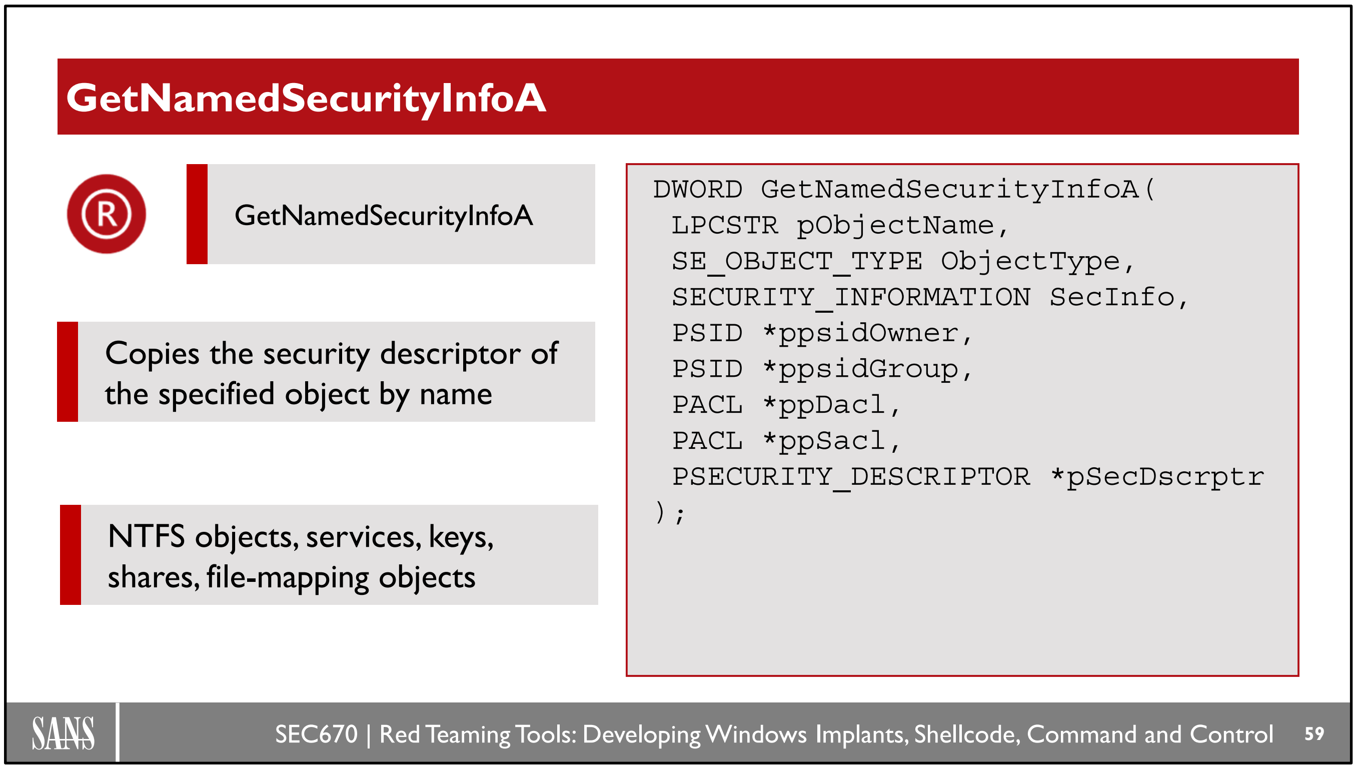
Programmatically Hide a Service

Manual versus programmatically

As with almost everything you do manually in a shell like the cmd prompt, there are Windows APIs on the back end that enable that effort. SDDL strings and ACE strings can be daunting to hand jam in an interactive session, but perhaps using the APIs is easier.

Programmatically Hide a Service
Crafting your SDDL string to change the permissions of an object is not intuitive by any means and rather annoying. This is largely due to the archaic SDDL syntax, but it can come in handy for those one-offs where you do not have tool that can implement this feature, nor do you have the time to create one. Thankfully, you are not stuck having to produce an ugly looking SDDL string. Windows has structures and APIs that can be configured and called to effectively hide a service. Knowing the various methods of how this is done via the various Windows APIs is important for developers—that way this feature can be added or created in our toolset. The securitybaseapi.h header file declares a very large number of APIs that are specific to changing the control bits of a security descriptor and then some. This is what we were doing manually with the SDDL strings and the *sc.exe sdse*t command-line utility. APIs such as ***SetSecurityDescriptorControl***, ***SetSecurityControlDacl***, ***SetSecurityDescriptorSacl***, ***SetSecurityDescriptorOwner***, etc. are where you would want to look first for this implementation in code. For the most part, they are all Boolean type functions and are very easy to implement. Another important header file to look at would be aclapi.h and, as the name implies, there will be APIs, such as ***GetNamedSecurityInfo*** and ***SetNamedSecurityInfo*** that directly tie to obtaining and or modifying ACLs.

© SANS Institute 2024



GetNamedSecurityInfoA

GetNamedSecurityInfoA

Copies the security descriptor of the specified object by name

NTFS objects, services, keys, shares, file-mapping objects

```
DWORD GetNamedSecurityInfoA(
 LPCSTR pObjectName,
 SE_OBJECT_TYPE ObjectType,
 SECURITY_INFORMATION SecInfo,
 PSID *ppsidOwner,
 PSID *ppsidGroup,
 PACL *ppDacl,
 PACL *ppSacl,
 PSECURITY_DESCRIPTOR *pSecDscrptr
);
```

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 59

GetNamedSecurityInfoA
The **GetNamedSecurityInfoA** API is used for when you would want to obtain the security descriptor of an object of interest. The great thing about this API is that is can be used for objects that are local to the system as well as objects that might be on a remote system. Because Windows has many object types, it can be daunting to fully understand each object type and how they can be secured. Thankfully, we can peek into them with this API. The SAL annotations have been omitted on the slide for brevity but have been included down below in the slide notes. There are three in parameters that are required and five out parameters that are optional. To get the most details about an object, you can pass in pointers for each of the out parameters. Passing in pointers for those extra parameters will tell the API that you want information about the Owner and Group SID, along with the DACL and SACL.

Here is a detailed breakdown of the API's parameters.

_In_ LPCSTR pObjectName, is the pointer to the name of the object that you are querying. Remember, this string must be a NULL-terminated string.

_In_ SE_OBJECT_TYPE ObjectType, is an enum entry from the enum SE_OBJECT_TYPE. This lets the API know the type of the object that pObjectName should be in the system.

_In_ SECURITY_INFORMATION SecurityInfo, is how you indicate to the API the security information set you are querying. Many of these can be combined using a bitwise OR like ATTRIBUTE_SECURITY_INFOMRATION | DACL_SECURITY_INFORMATION.

*_Out_opt_ PSID *ppsidOwner,* if desired, will hold the pointer to the owning SID if there is one, in the security descriptor pointed to by ppSecurityDescriptor. If there isn't one, then NULL will be returned.

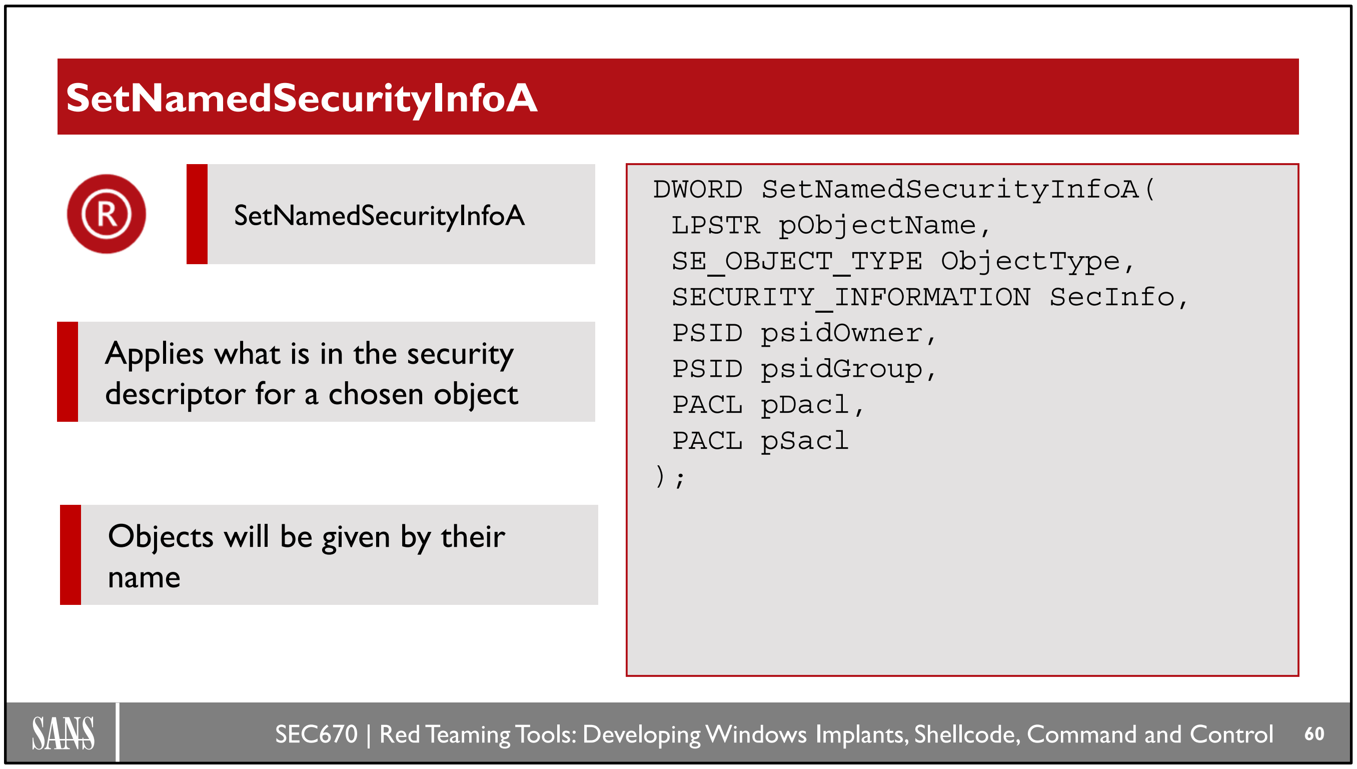
*_Out_opt_ PSID *ppsidGroup,* if desired, will hold the pointer to the primary group SID if there is one, in the security descriptor pointed to by ppSecurityDescriptor. If there isn't one, then NULL will be returned.

*_Out_opt_ PACL *ppDacl,* if desired, will hold the pointer to the DACL if there is one, in the security descriptor pointed to by ppSecurityDescriptor. If there isn't one, then NULL will be returned.

*_Out_opt_ PACL *ppSacl,* if desired, will hold the pointer to the SACL if there is one, in the security descriptor pointed to by ppSecurityDescriptor. If there isn't one, then NULL will be returned.

*_Out_opt_ PSECURITY_DESCRIPTOR *ppSecurityDescriptor,* will point to the security descriptor for the requested object. The buffer the API made for you must be freed by calling ***LocalFree***.

© SANS Institute 2024



SetNamedSecurityInfoA

SetNamedSecurityInfoA

Applies what is in the security descriptor for a chosen object

Objects will be given by their name

```
DWORD SetNamedSecurityInfoA(
 LPSTR pObjectName,
 SE_OBJECT_TYPE ObjectType,
 SECURITY_INFORMATION SecInfo,
 PSID psidOwner,
 PSID psidGroup,
 PACL pDacl,
 PACL pSacl
);
```

SetNamedSecurityInfoA
The ***SetNamedSecurityInfoA*** API is used for when you want to confirm or apply a change to security information in an object's security descriptor that you previously obtained. As with the ***GetNamedSecurityInfo*** API, this can deal with many types of objects. The objects can be local to the system or on a remote system given proper permissions or accesses to it. Let us break down the API's parameters. If you are modifying objects that do have existing ACEs, then you must call ***SetEntriesInAcl*** before you can call ***SetNamedSecurityInfo***. ACE entries will then be merged in the object's DACL.

_In_ LPSTR pObjectName, is the pointer to the name of the object that you are querying. Remember, this string must be a NULL-terminated string.

_In_ SE_OBJECT_TYPE ObjectType, is an enum entry from the enum SE_OBJECT_TYPE. This lets the API know the type of the object that pObjectName should be in the system.

_In_ SECURITY_INFORMATION SecurityInfo, is the combined flags using a bitwise OR for the information that is to be set.

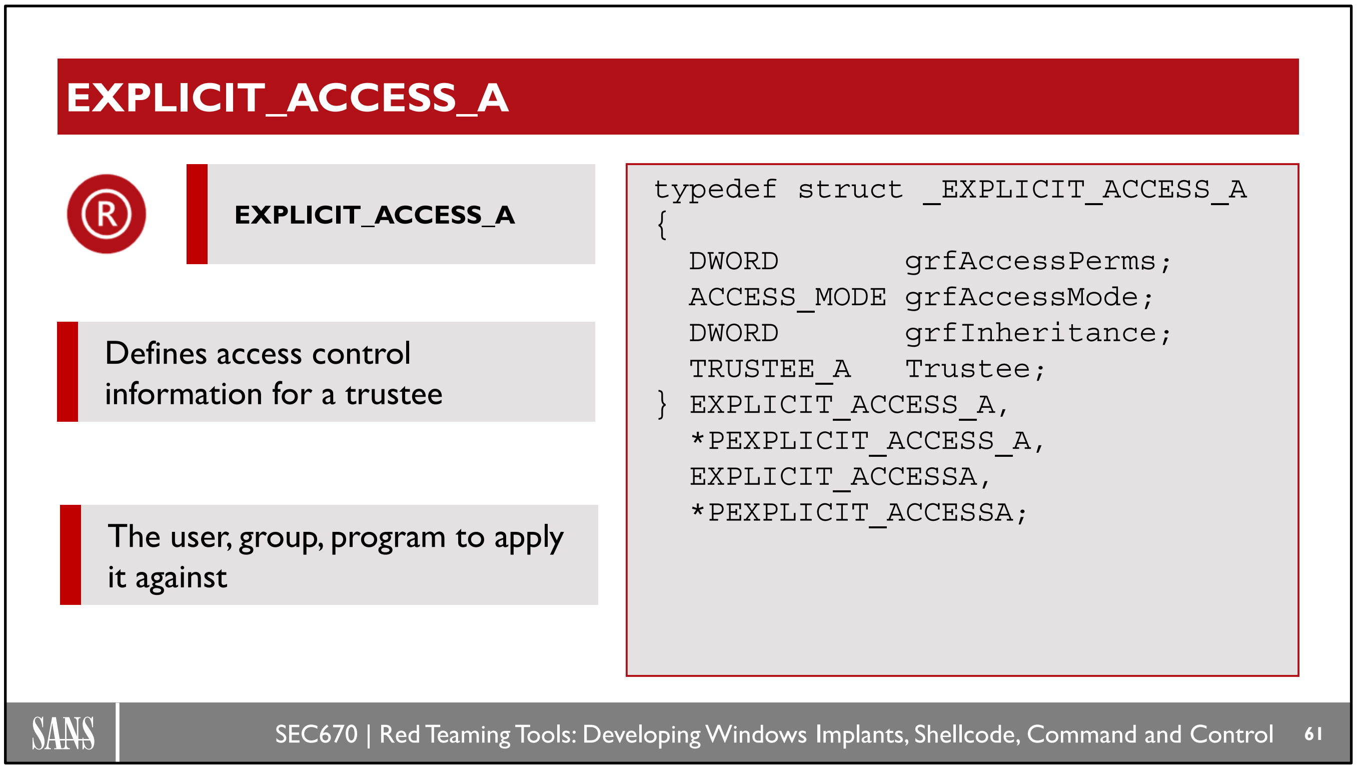
_In_opt_ PSID psidOwner, is a pointer to the SID of who the owner of the object should be. For this to take effect, you must first have WRITE_OWNER access or more like SE_TAKE_OWNERSHIP_NAME. Must use OWNER_SECURITY_INFORMATION for this to work.

_In_opt_ PSID psidGroup, if passed, can be used to indicate who the primary group of the object will be from here on out. Must use GROUP_SECURITY_INFORMATION for this to work.

_In_opt_ PACL pDacl, if passed, can be used to point to the new DACL of the object. Must use DACL_SECURITY_INFORMATION for this to work.

_In_opt_ PACL pSacl, if passed, can be used to point to the new SACL of the object. Must use SACL_SECURITY_INFORMATION, as well as a few others, for this to work.

© SANS Institute 2024



EXPLICIT_ACCESS_A

EXPLICIT_ACCESS_A

Defines access control information for a trustee

The user, group, program to apply it against

```
typedef struct _EXPLICIT_ACCESS_A
{
  DWORD       grfAccessPerms;
  ACCESS_MODE grfAccessMode;
  DWORD       grfInheritance;
  TRUSTEE_A   Trustee;
} EXPLICIT_ACCESS_A,
  *PEXPLICIT_ACCESS_A,
  EXPLICIT_ACCESSA,
  *PEXPLICIT_ACCESSA;
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 61

EXPLICIT_ACCESS_A
The ***EXPLICIT_ACCESS_A*** structure is heavily used whenever modifications are being made to the ACL of an object. The structure is used to describe to the system the information in an ACE that resides in an ACL. Let's jump right in and start to better understand the structure members and how you might want to set them to change an object.

DWORD grfAccessPermissions, is for a combination of flags that will be used to dictate the permissions granted or denied to the object; permissions like EVENT_ALL_ACCESS, SYNCHRONIZE, GENERIC_ALL, READ_CONTROL, and so many more.

ACCESS_MODE grfAccessMode, will be for one of the enum values in the ACCESS_MODE enum and depending on if you are messing with a DACL or SACL, this member will mean different things. For DACLS, the flag will indicate if there will be deny or allow access rights in the ACL. For SACLs, this will indicate if audit messages for access attempts should be generated.

DWORD grfInheritance, is used to indicate if the ACE of the primary object can be inherited by other containers or objects. Setting something like NO_INHERITANCE will indicate that the ACE cannot be inherited.

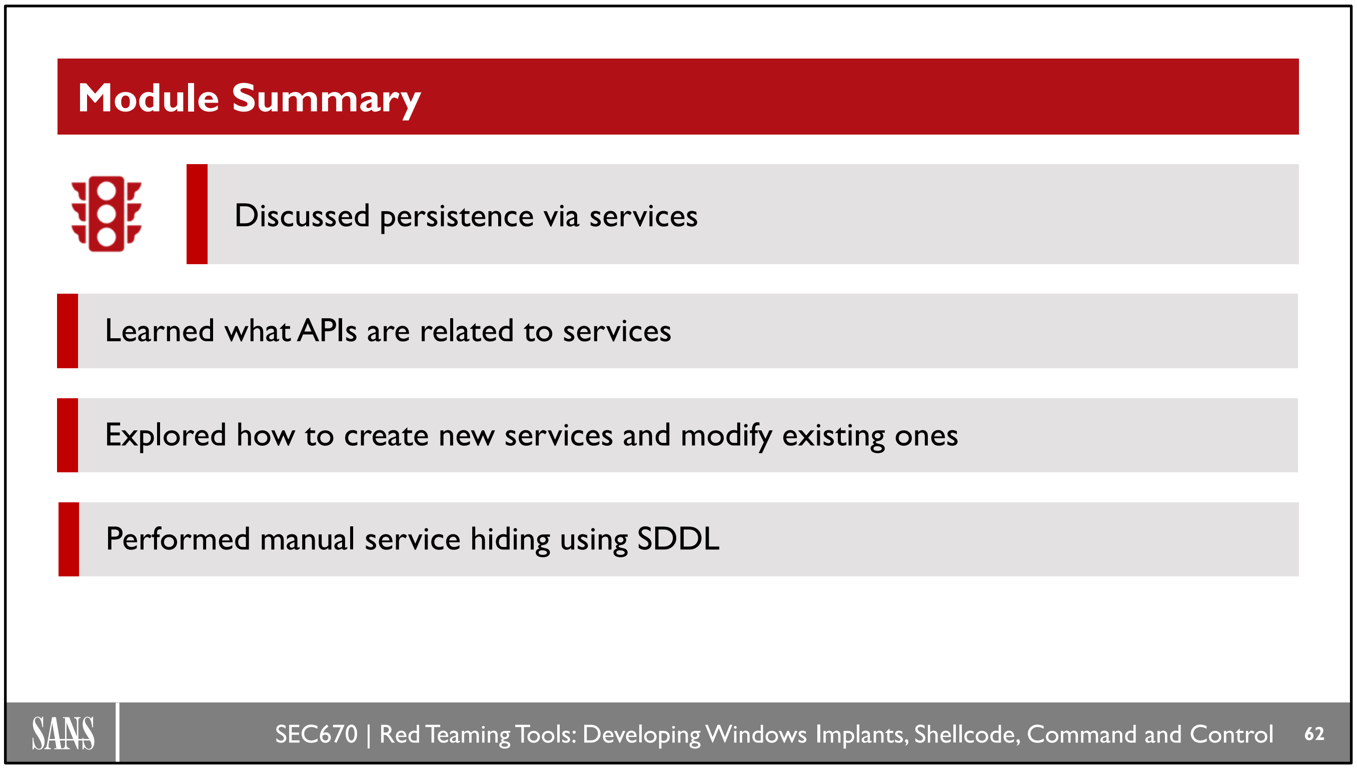
TRUSTEE_A Trustee, is a structure that is used to indicate how the ACE should apply. ACEs can be applied to a user, group, or a Windows service.

Below are some enums and structs that are used within the EXPLICIT_ACCESS struct discussed above.

```
typedef enum _ACCESS_MODE {
 NOT_USED_ACCESS,
 GRANT_ACCESS,
 SET_ACCESS,
 DENY_ACCESS,
 REVOKE_ACCESS,
 SET_AUDIT_SUCCESS,
 SET_AUDIT_FAILURE
} ACCESS_MODE;
```

```
typedef struct _TRUSTEE_A {
  struct _TRUSTEE_A   *pMultipleTrustee;
  MULTIPLE_TRUSTEE_OPERATION  MultipleTrusteeOperation;
  TRUSTEE_FORM  TrusteeForm;
  TRUSTEE_TYPE  TrusteeType;
  union {
    LPSTR  ptstrName;
    SID  *pSid;
    OBJECTS_AND_SID    *pObjectsAndSid;
    OBJECTS_AND_NAME_A  *pObjectsAndName;
  };
  LPCH  ptstrName;
} TRUSTEE_A, *PTRUSTEE_A, TRUSTEEA, *PTRUSTEEA;
```

© SANS Institute 2024



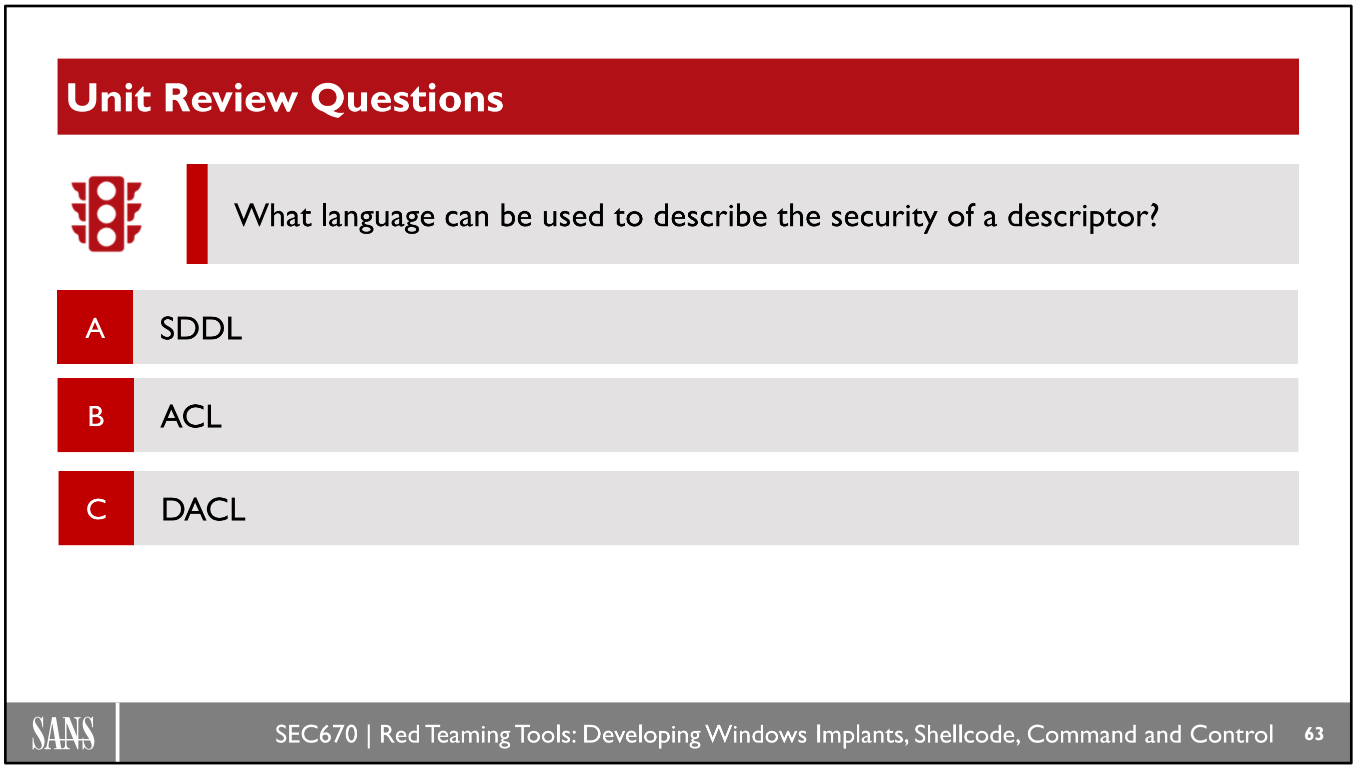
Module Summary

- Discussed persistence via services
- Learned what APIs are related to services
- Explored how to create new services and modify existing ones
- Performed manual service hiding using SDDL

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 62

Module Summary
In this module, spent a little bit of time discussing how services can be used for our persistence needs. Throughout that discussion, we discovered what APIs are related to services. We saw that actions can be taken based on the service starting successfully, as well as failing. One of the final items covered was manually changing the permissions of a service to hide it from the user.

© SANS Institute 2024



Unit Review Questions

What language can be used to describe the security of a descriptor?

A SDDL

B ACL

C DACL

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 63

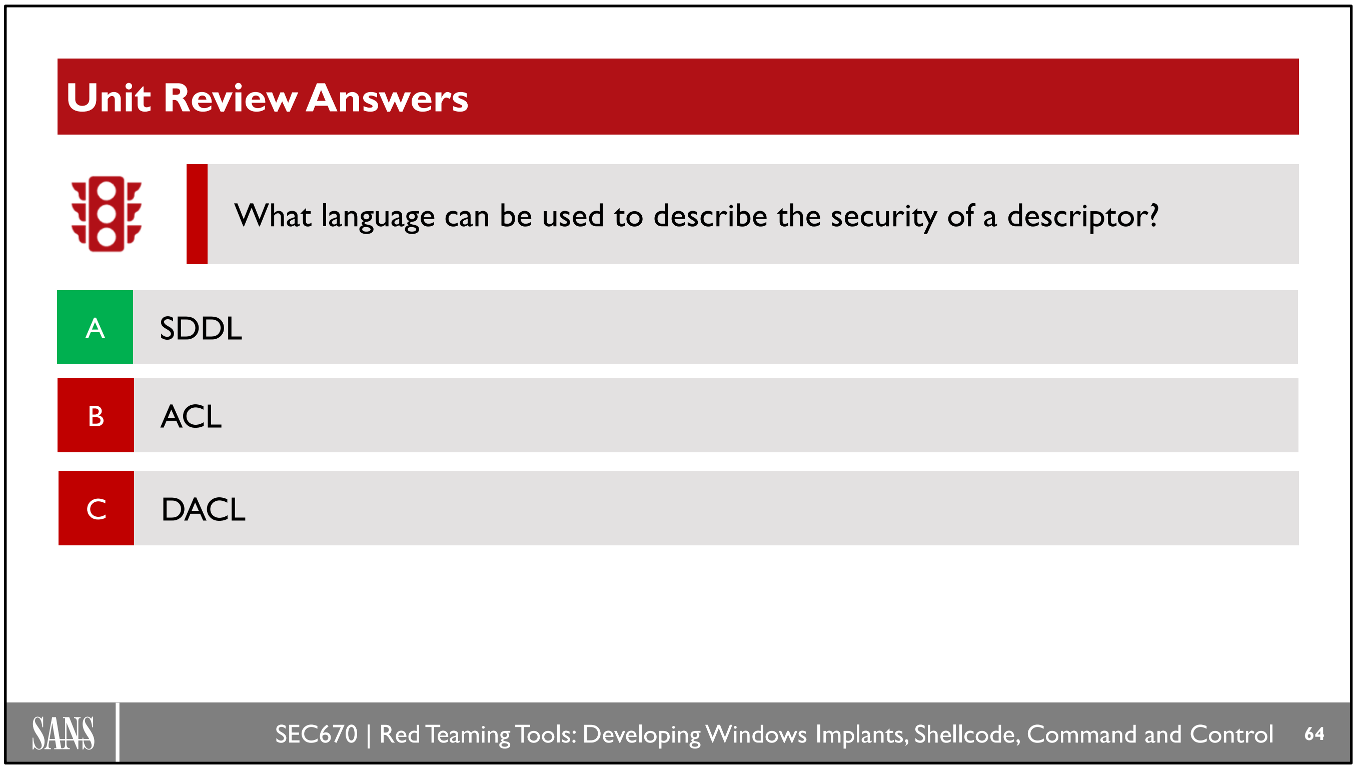
Unit Review Questions
Q: What language can be used to describe the security of a descriptor?

A: SDDL

B: ACL

C: DACL

© SANS Institute 2024



Unit Review Answers

What language can be used to describe the security of a descriptor?

A SDDL

B ACL

C DACL

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 64

Unit Review Answers
Q: What language can be used to describe the security of a descriptor?

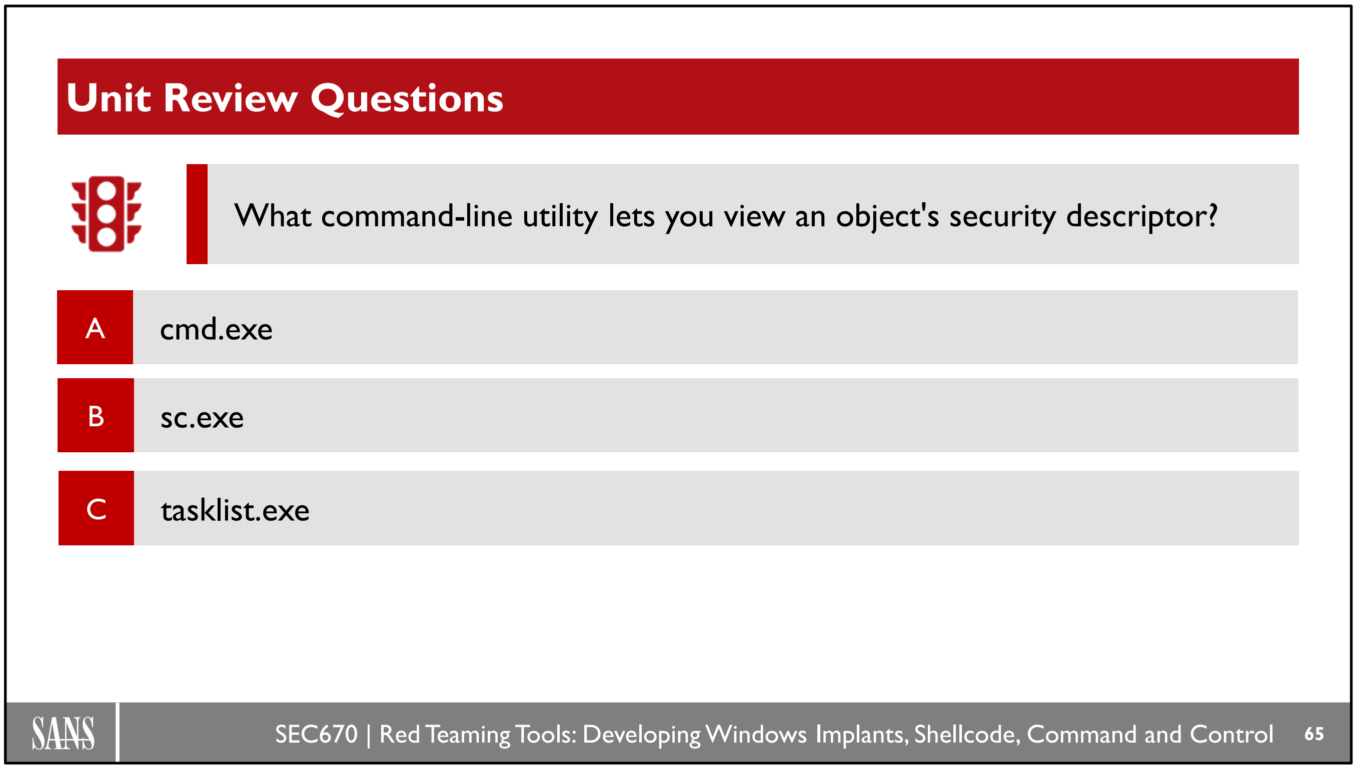
A: SDDL

B: ACL

C: DACL

© 2024 Jonathan Reiter

© SANS Institute 2024



Unit Review Questions

What command-line utility lets you view an object's security descriptor?

A cmd.exe

B sc.exe

C tasklist.exe

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 65

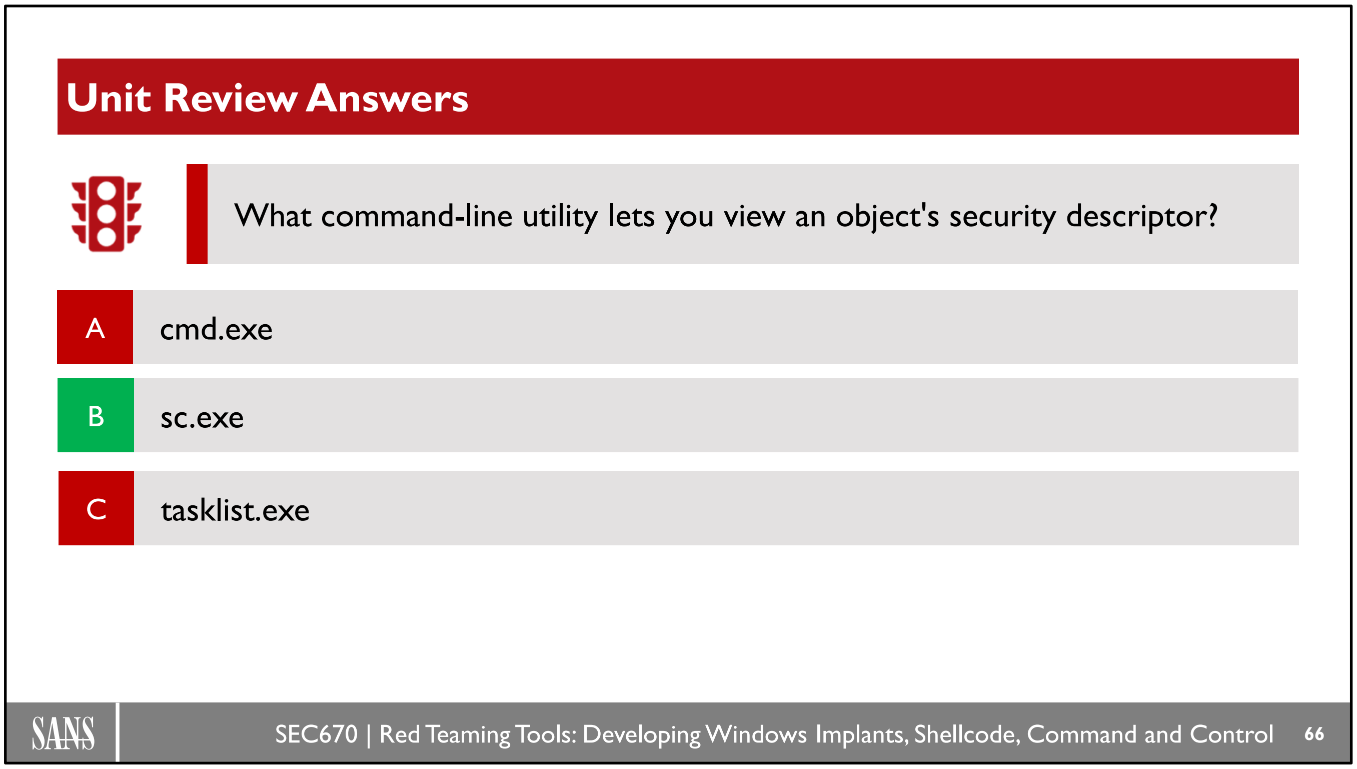
Unit Review Questions
Q: What command-line utility lets you view an object’s security descriptor?

A: cmd.exe

B: sc.exe

C: tasklist.exe

© SANS Institute 2024



Unit Review Answers

What command-line utility lets you view an object's security descriptor?

A cmd.exe

B sc.exe

C tasklist.exe

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 66

Unit Review Answers
Q: What command-line utility lets you view an object’s security descriptor?

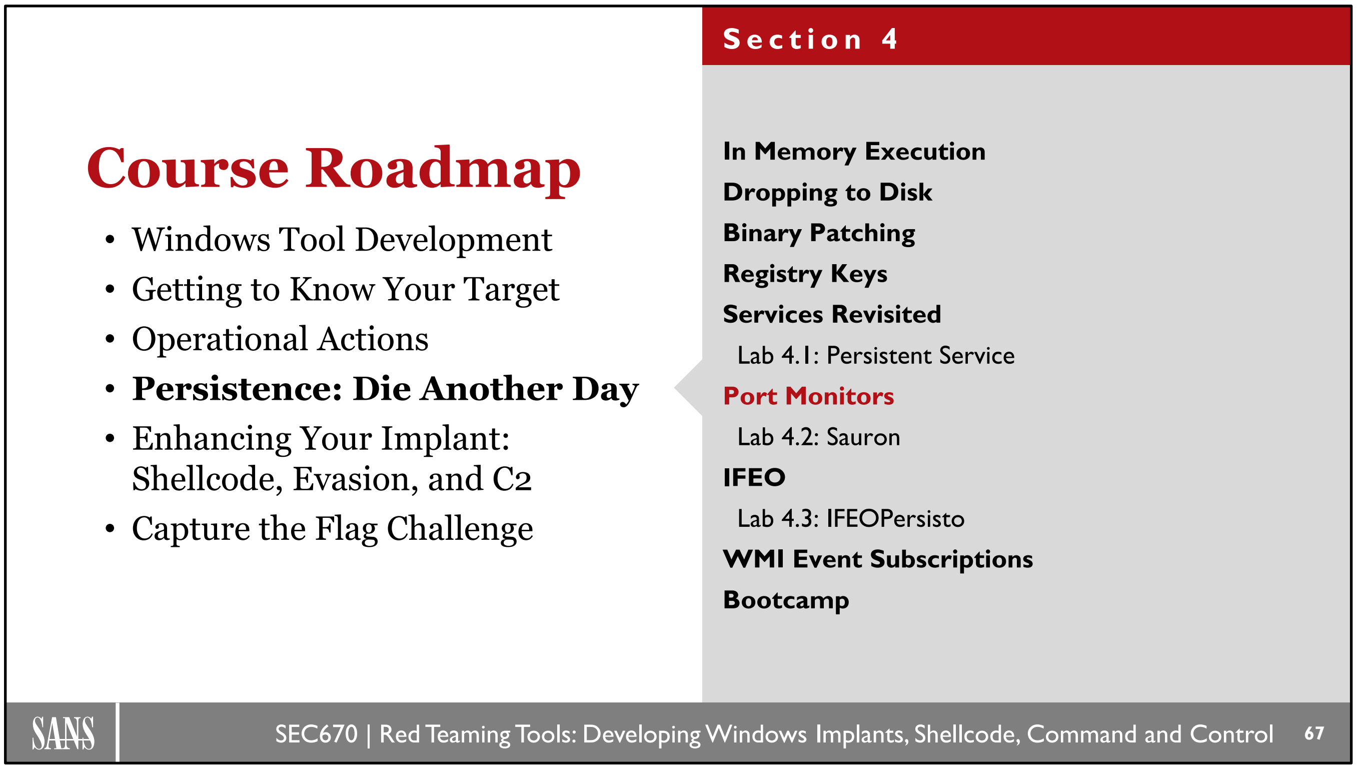
A: cmd.exe

B: sc.exe

C: tasklist.exe

© 2024 Jonathan Reiter

© SANS Institute 2024



Section 4

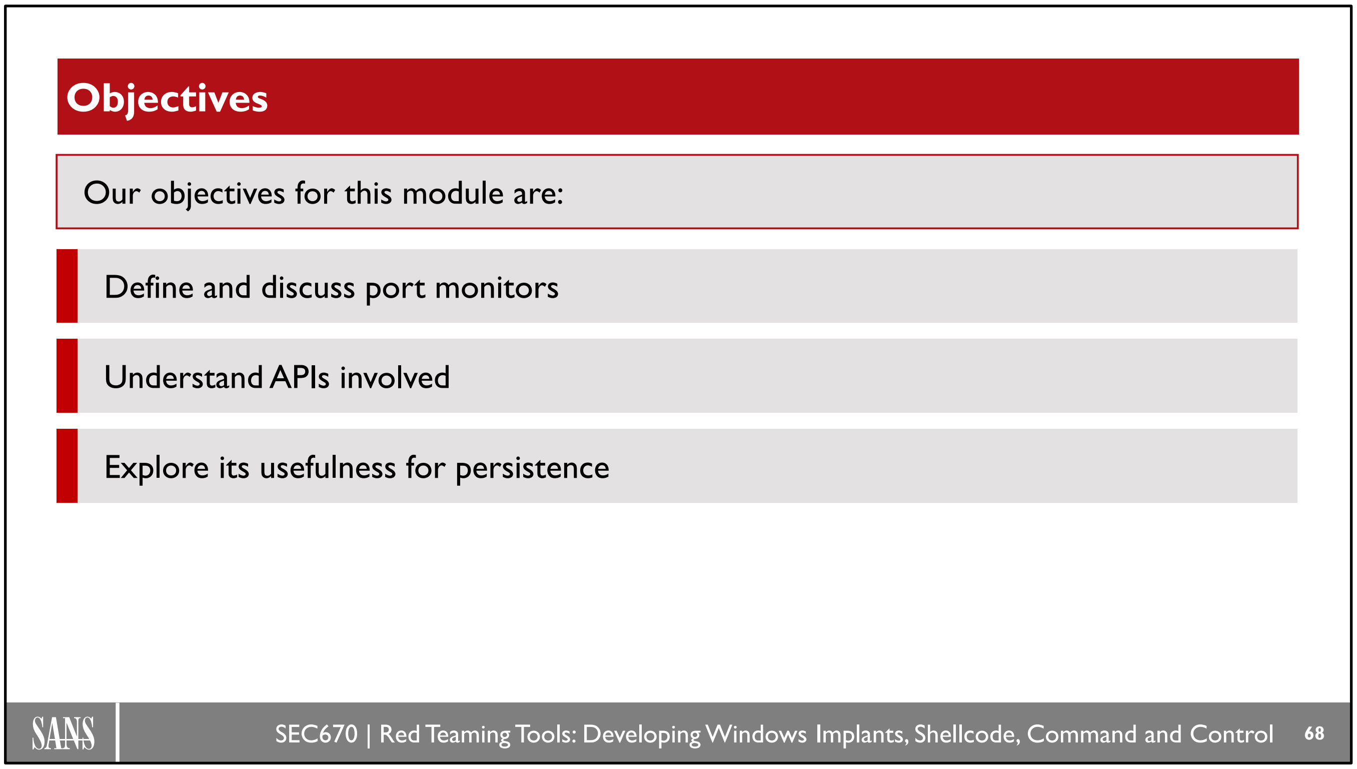
Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 67

In this module, we will discuss how to persist using port monitors.

© SANS Institute 2024



Objectives

Our objectives for this module are:

- Define and discuss port monitors
- Understand APIs involved
- Explore its usefulness for persistence

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 68

Objectives
The objectives for this module are to define and discuss port monitors; what they are and what they are used for. Next, we will look at the APIs involved with the implementation of this method before we move into exploring how useful it might be for our persistence needs.

© SANS Institute 2024

Port Monitors

What are port monitors?

Windows has two type of print monitors: language monitor and port monitor. Port monitors do what they say by monitoring a printer port and bridging the physical connection to the printer queue, which we see as a user.

Port Monitors
What exactly are port monitors? According to Microsoft, a port monitor acts like a bridge of sorts from user-mode to kernel-mode. The user-mode side comes from the spoolsv.exe image and it communicates with a port driver that resides in the kernel. The spoolsv.exe is a Windows service known as the Windows Print Spooler. The kernel port driver is what accesses the I/O port hardware. Port monitors are utilized to configure printer ports, and also manage them. As a user, you see the queue for a printer. The port is the physical connection that bridges that gap between the queue and the physical printer. Ports are assigned to port monitors using a Win32 API named *AddPrinter*. What we are doing, though, has nothing to do with printers or adding print job to a printer queue.

Reference:
https://docs.microsoft.com/en-us/windows-hardware/drivers/print/port-monitors

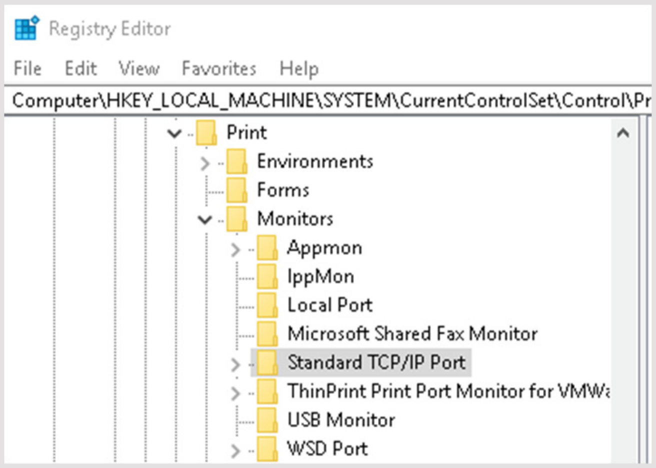
© SANS Institute 2024

Abusing Port Monitors: The Registry

Method one: Registry

Key holds the port monitors in place

Need local admin



Registry Editor

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Pr

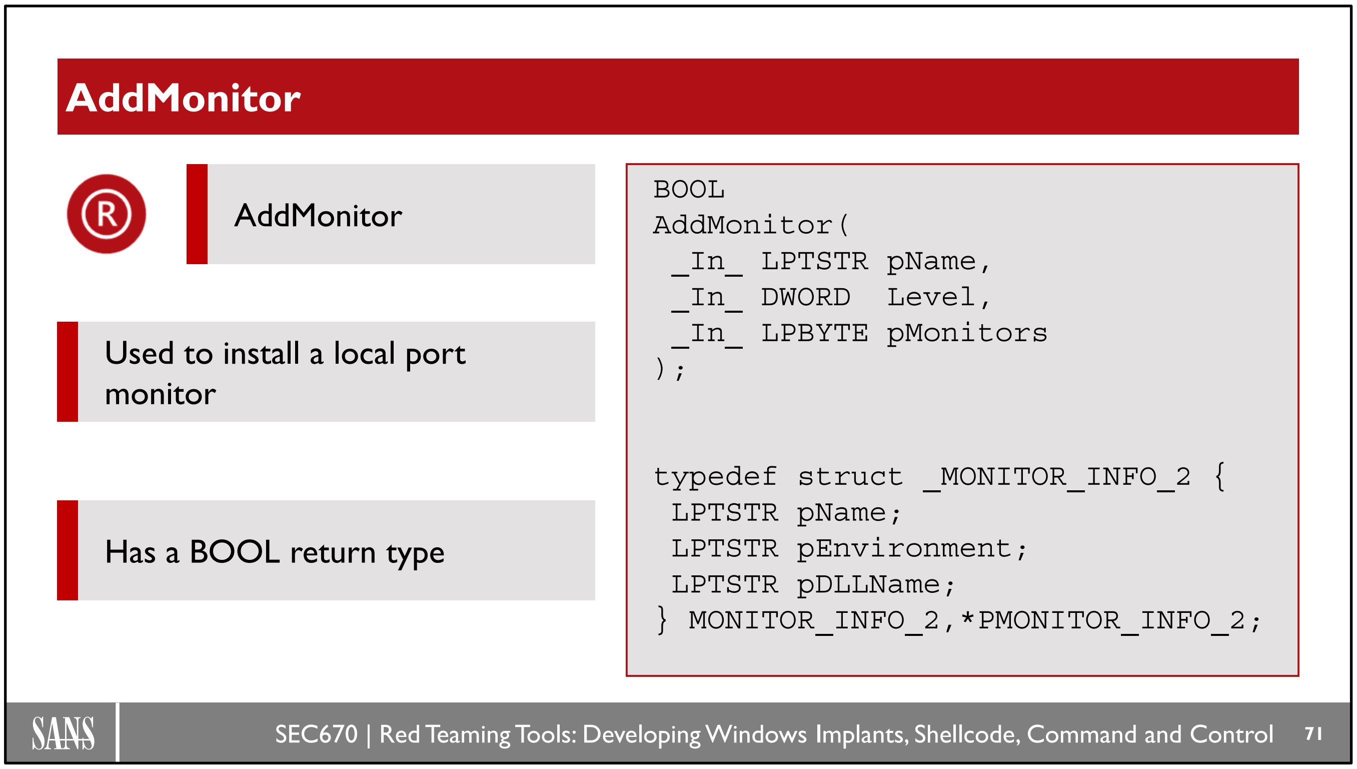
```
Print
  Environments
  Forms
  Monitors
    Appmon
    IppMon
    Local Port
    Microsoft Shared Fax Monitor
    Standard TCP/IP Port
    ThinPrint Print Port Monitor for VMWa
    USB Monitor
    WSD Port
```

Abusing Port Monitors: The Registry

There are two ways that we can leverage and abuse port monitors. The first method we are going to look at is using the Registry to modify the Print>Monitors hive. The full path to the key is **HKLM\SYSTEM\CurrentControlSet\Control\Print\Monitors** and the subkeys under it should be port monitors. We can manually view this using the regedit.exe GUI. Browsing to the key, we can select each subkey and view the values. Each one should have a DLL that it is using to monitor the port. Depending on your computer's configuration, you might see several subkeys like *Local Port*, or *USB Monitor*. Via the GUI, we can manually add a new subkey here and create our new port monitor. The new key that we create will have a new string value that will be the name of the DLL used to "monitor" the port. The catch is, the DLL being used for the "Driver" key value must reside in the C:\Windows\System32 folder. For this reason, we must have local admin privileges. The beauty behind this, though, is when the DLL gets kicked off, what comes back is SYSTEM. Not a bad deal at all. However, another downside to this method is that the system must be rebooted for this change to take effect. Probably not something you want to initiate on your own unless you know the user is gone for the day and will not notice. If you can determine that there will be a scheduled reboot to meet some corporate reboot policy, then that could work too.

© 2024 Jonathan Reiter

© SANS Institute 2024



AddMonitor

AddMonitor

Used to install a local port monitor

Has a BOOL return type

```
BOOL
AddMonitor(
 _In_ LPTSTR pName,
 _In_ DWORD  Level,
 _In_ LPBYTE pMonitors
);


typedef struct _MONITOR_INFO_2 {
 LPTSTR pName;
 LPTSTR pEnvironment;
 LPTSTR pDLLName;
} MONITOR_INFO_2,*PMONITOR_INFO_2;
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 71

AddMonitor
The *AddMonitor* API is used when you need to create and install a port monitor on the local machine. The API will fail if your monitor does not match the architecture of the system you are targeting. This means that for a 64-bit system, your environment would be "*Windows x64*". The parameters are described below.

pName, is a pointer to a NULL-terminated string for the server that we are installing the port monitor.

Level, is for the version, but the only option the API will accept is 2, so 2 it is.

pMonitors, is a pointer to the MONITOR_INFO_2 struct that we have set up before the API call.

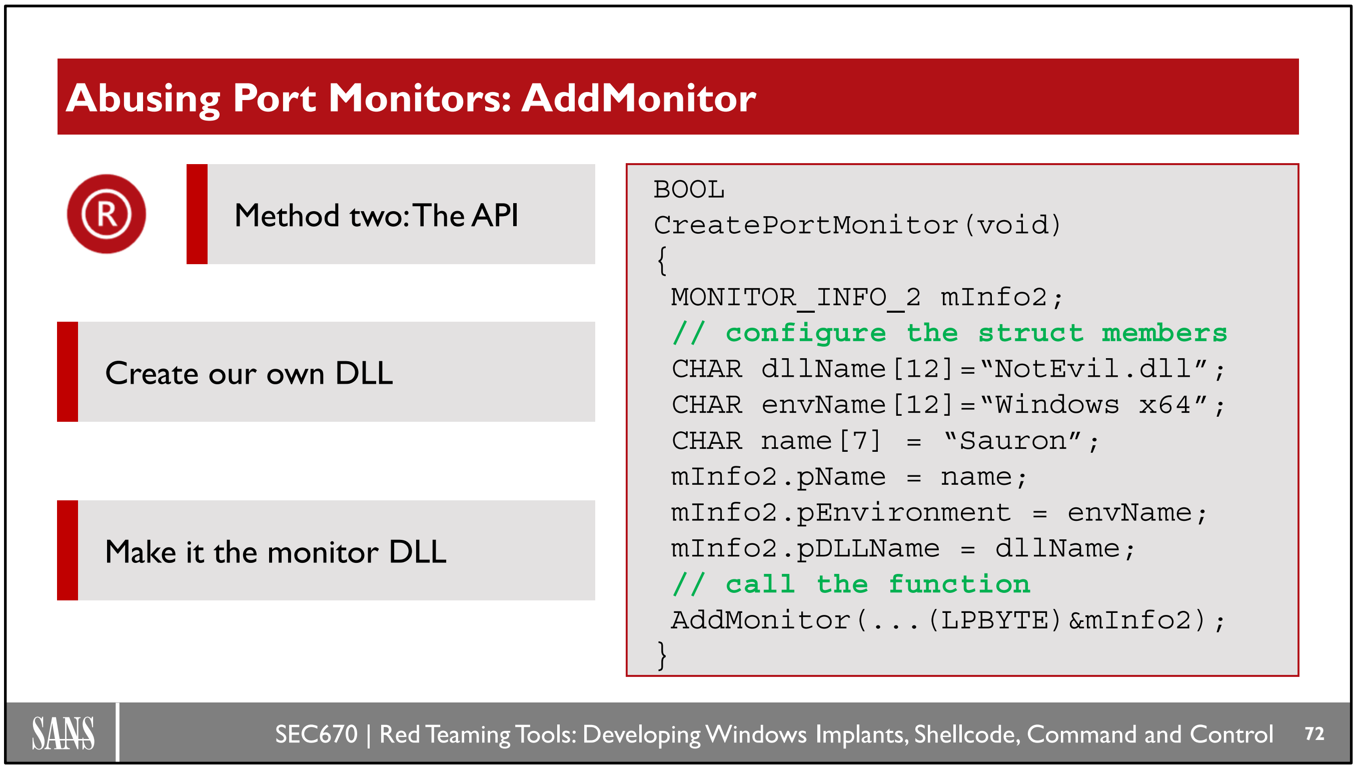
The MONITOR_INFO_2 struct is used by the API to identify the monitor that we are hoping to install. Here are the structure members.

pName is a pointer to a NULL-terminated string that will be the name of the monitor being installed.

pEnvironment is a pointer to a NULL-terminated string that indicates the environment that the monitor is being installed against. "Windows x64" and "Windows NT x86" are some examples of strings acceptable for the environment.

pDLLName is a pointer to a NULL-terminated string that is used to indicate the name of the monitor DLL.

© SANS Institute 2024



Abusing Port Monitors: AddMonitor

Method two: The API

Create our own DLL

Make it the monitor DLL

```
BOOL
CreatePortMonitor(void)
{
 MONITOR_INFO_2 mInfo2;
 // configure the struct members
 CHAR dllName[12]="NotEvil.dll";
 CHAR envName[12]="Windows x64";
 CHAR name[7] = "Sauron";
 mInfo2.pName = name;
 mInfo2.pEnvironment = envName;
 mInfo2.pDLLName = dllName;
 // call the function
 AddMonitor(...(LPBYTE)&mInfo2);
}
```

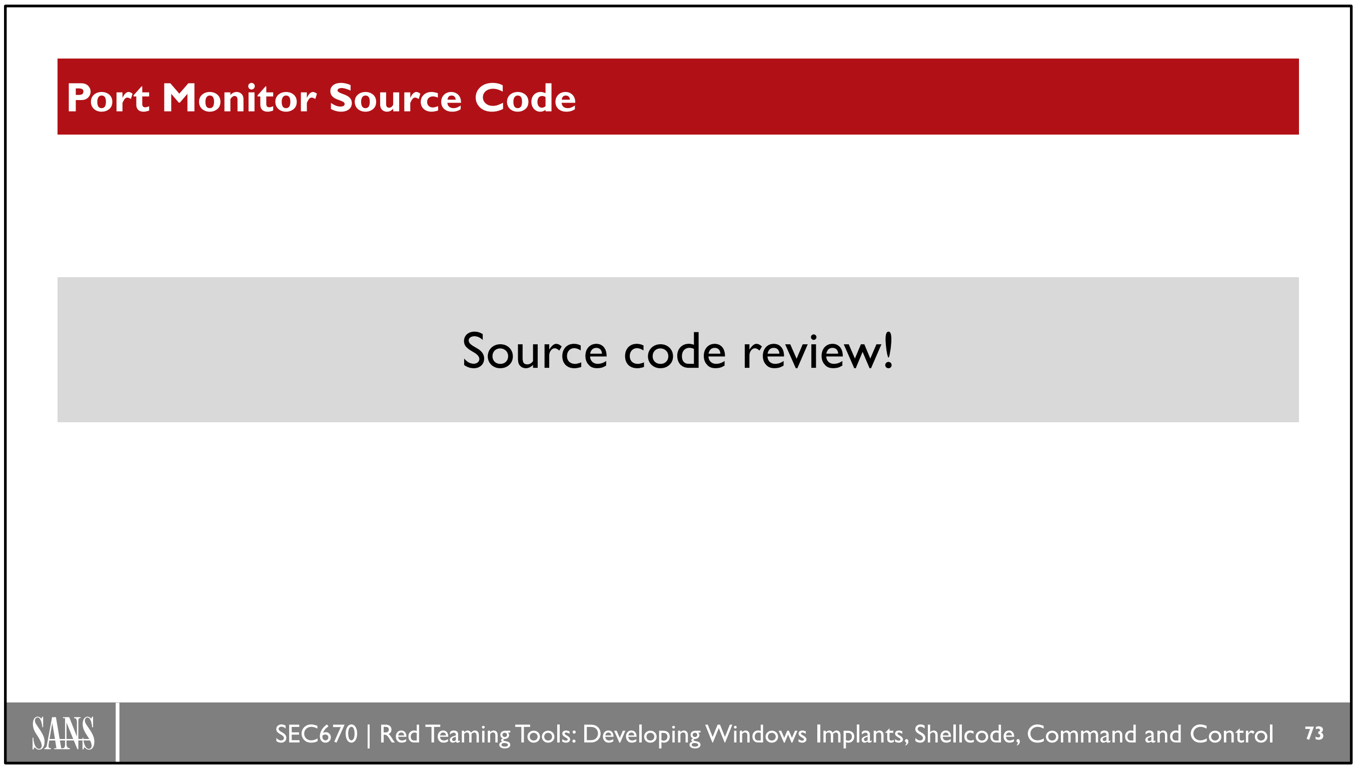
SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 72

Abusing Port Monitors: AddMonitor
Let us talk about the second method for abusing port monitors, using the *AddMonitor* API. The API was discussed in detail on the previous slide so that we can cover a snippet of pseudo code here showing a possible implementation of it. The overall goal with the code is to bring our own DLL to the game and use it to tell Windows that it needs to use our DLL to monitor a port. We can create a small function that can execute this for us and just call it from *main()* or another function. What is happening in the code is that we prepare the MONITOR_INFO_2 struct for use and give it a variable name. Before we start initializing each struct member, we make a few variables that will be assigned to the appropriate struct member. The name of our DLL will be "NotEvil.dll" because we will not be doing anything evil here. The environment this will be installed on will be "Windows x64" because it is a 64-bit OS. Finally, the name of the port monitor itself will be "Sauron" because it will act as the all-seeing eye for the port. With that out of the way, we can assign them to the proper members of the struct and make our API call. Done!

A nice effect with this method is there is no reboot required since the changes made here are immediate. A downside to this, though, is that you will still need to be a local admin to pull this off, but good thing we know what to look for when trying to elevate our privileges.

© 2024 Jonathan Reiter

© SANS Institute 2024



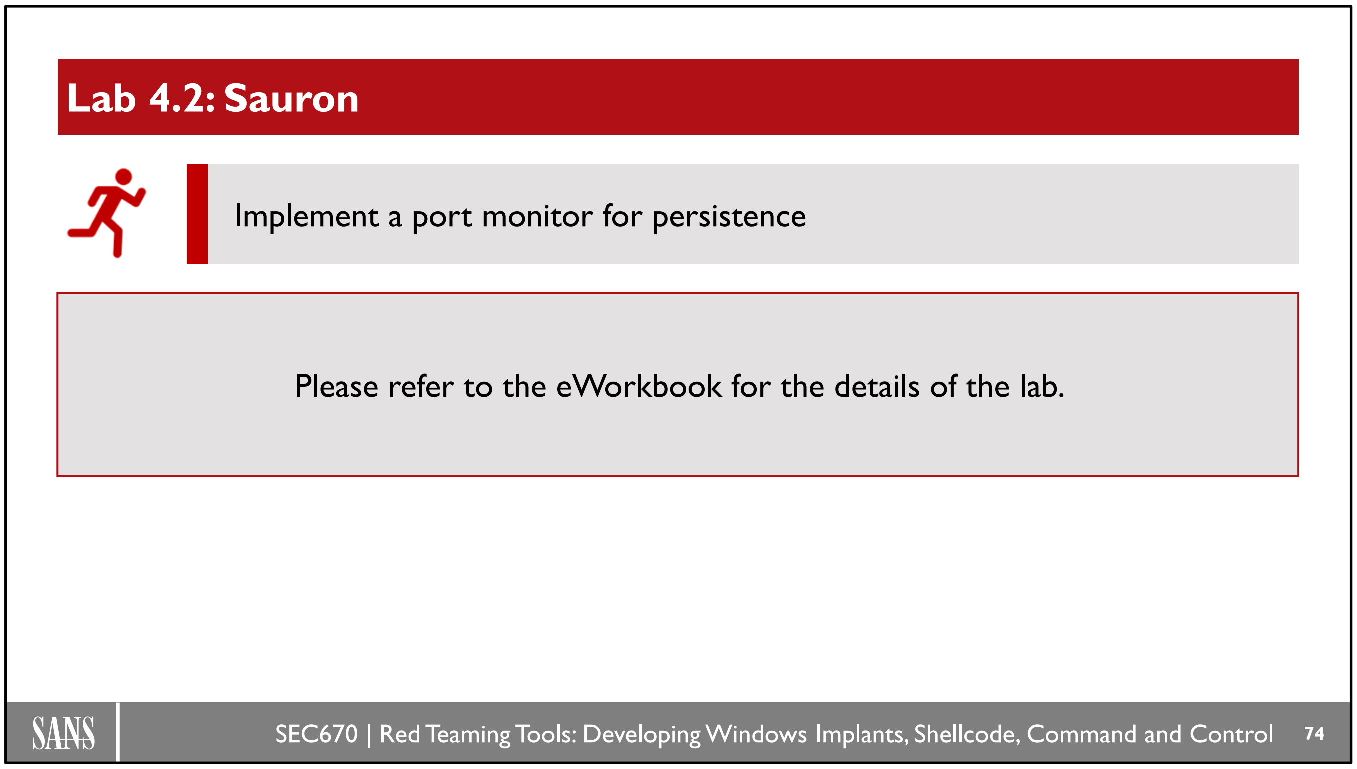
Port Monitor Source Code

Source code review!

SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 73

Port Monitor Source Code
Time to jump into the source code and explain it before you implement it on your own.

© SANS Institute 2024



Lab 4.2: Sauron

Implement a port monitor for persistence

Please refer to the eWorkbook for the details of the lab.

SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 74

Lab 4.2: Sauron
Please refer to the eWorkbook for the details of this lab.

© 2024 Jonathan Reiter

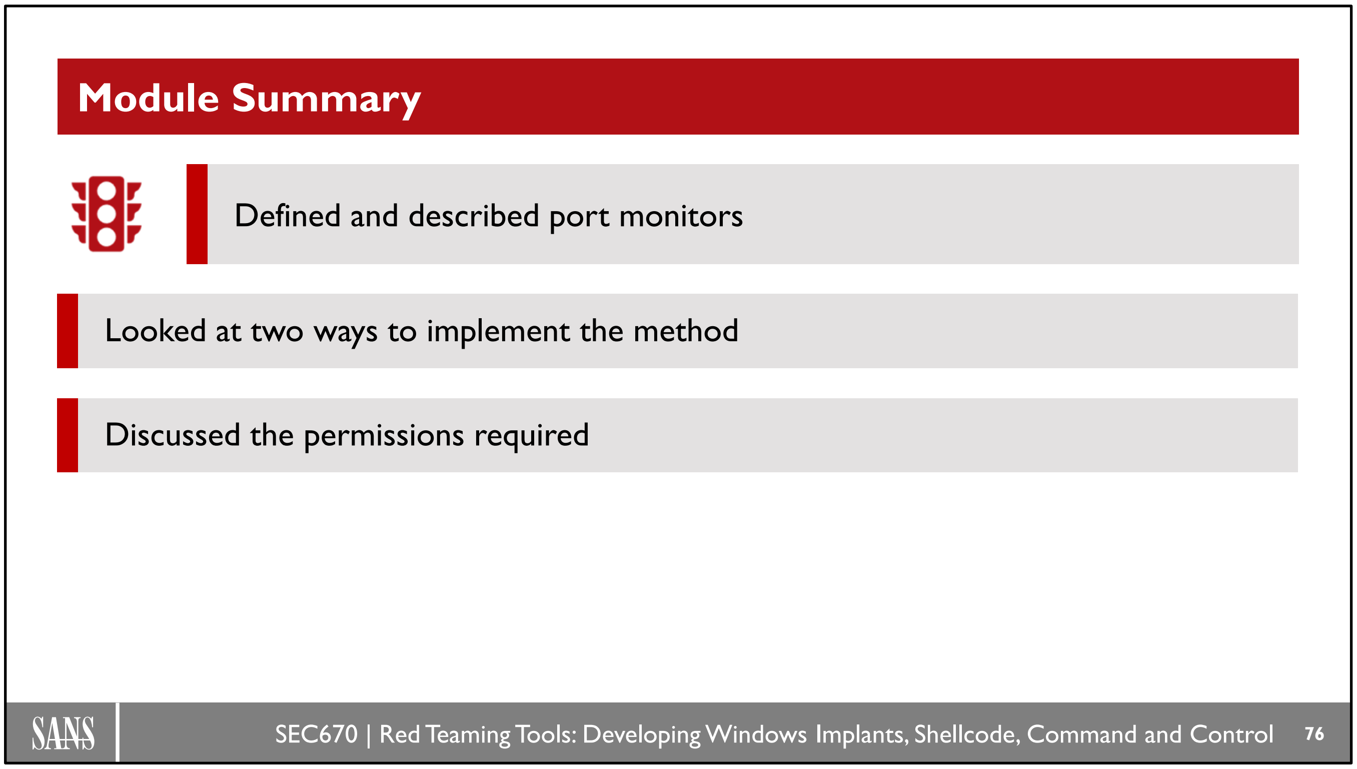
© SANS Institute 2024

What’s the Point?

What’s the point?

What’s the Point?
The point of this lab was to introduce you to another persistence technique that can be done programmatically. There was only one API that we needed to learn, and it was not a complicated one at all. We just had to set up a few things inside a structure, and everything was ready to execute.

© SANS Institute 2024



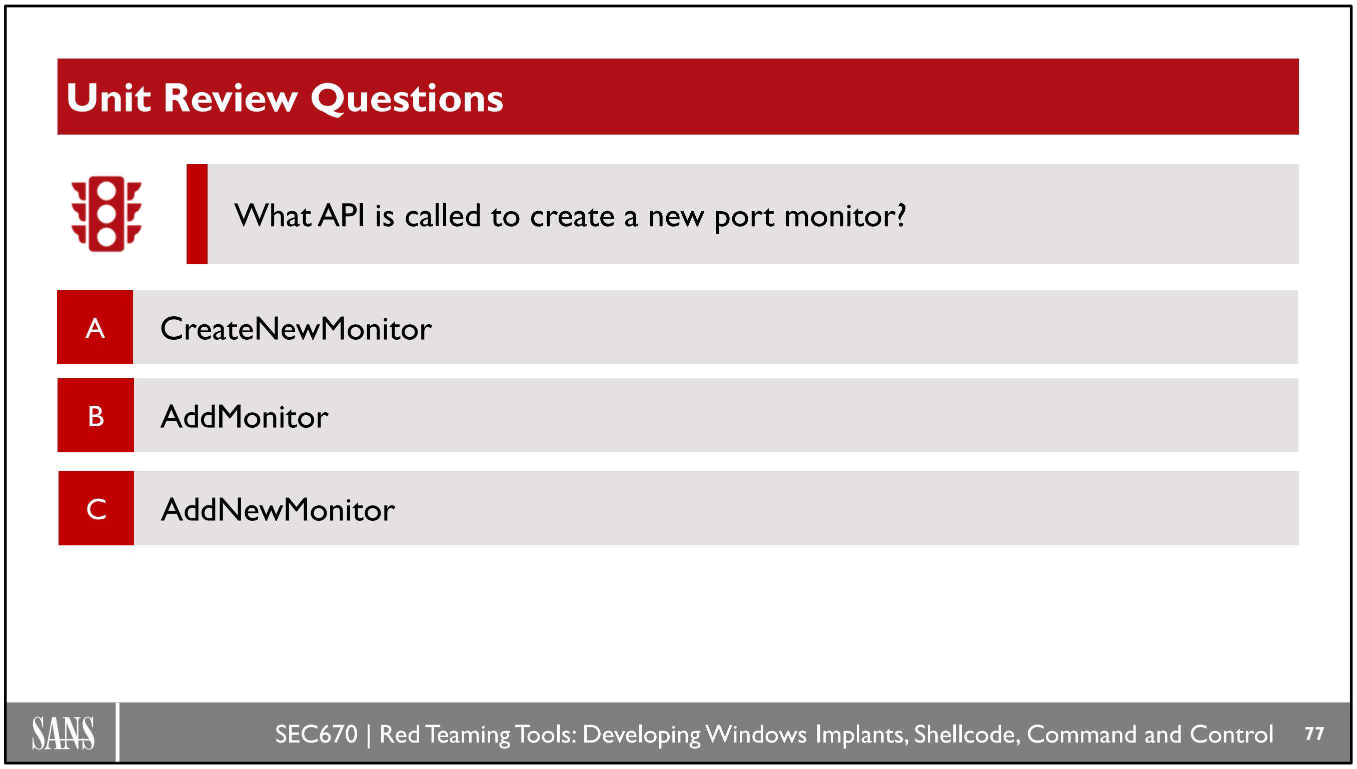
Module Summary

- Defined and described port monitors
- Looked at two ways to implement the method
- Discussed the permissions required

Module Summary
In this module, we defined port monitors and discussed how Microsoft intended for them to be used. Two ways of leveraging port monitors for persistence were explored from a manual method to a programmatic method. Lastly, we touched on a minor limiting factor for this method; the requirement to be a local admin.

© 2024 Jonathan Reiter

© SANS Institute 2024



Unit Review Questions

What API is called to create a new port monitor?

A CreateNewMonitor

B AddMonitor

C AddNewMonitor

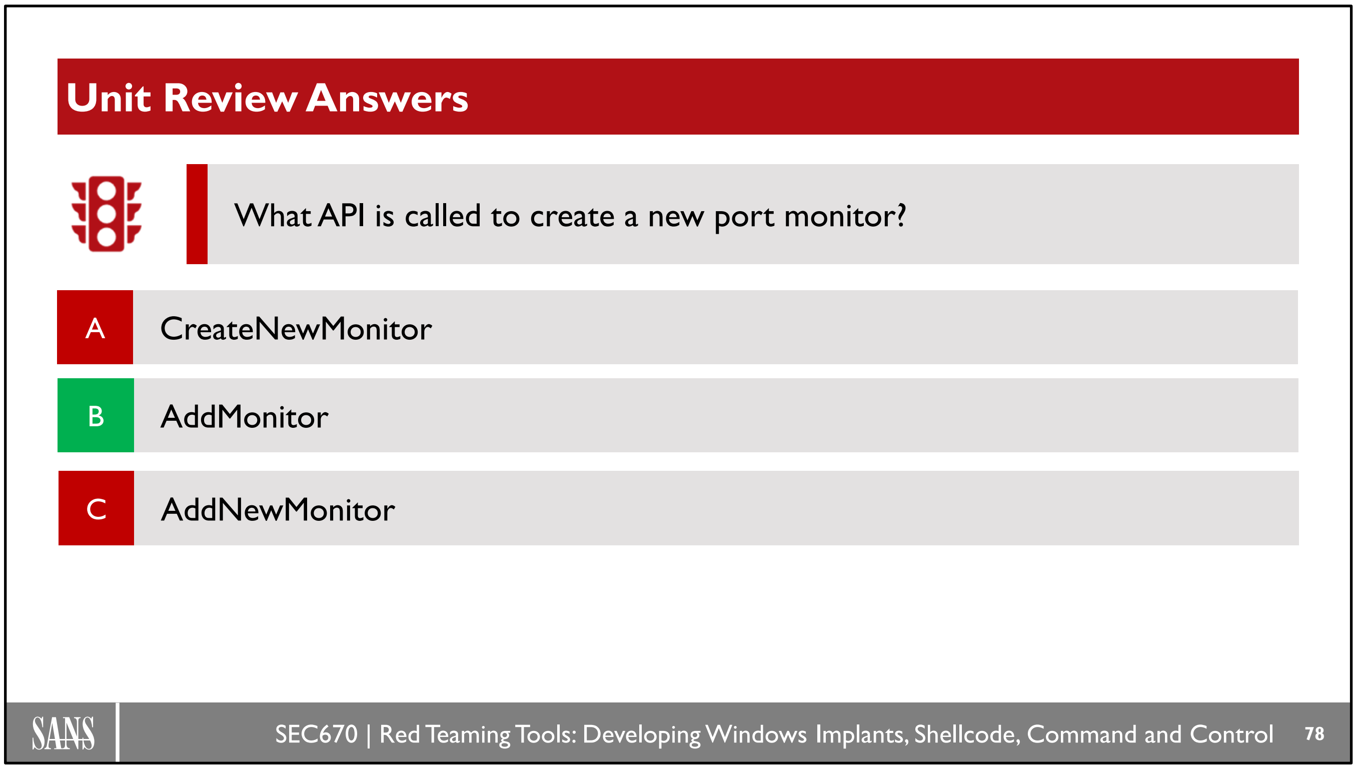
Unit Review Questions
Q: What API is called to create a new port monitor?

A: CreateNewMonitor

B: AddMonitor

C: AddNewMonitor

© SANS Institute 2024



Unit Review Answers

What API is called to create a new port monitor?

A CreateNewMonitor

B AddMonitor

C AddNewMonitor

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 78

Unit Review Answers
Q: What API is called to create a new port monitor?

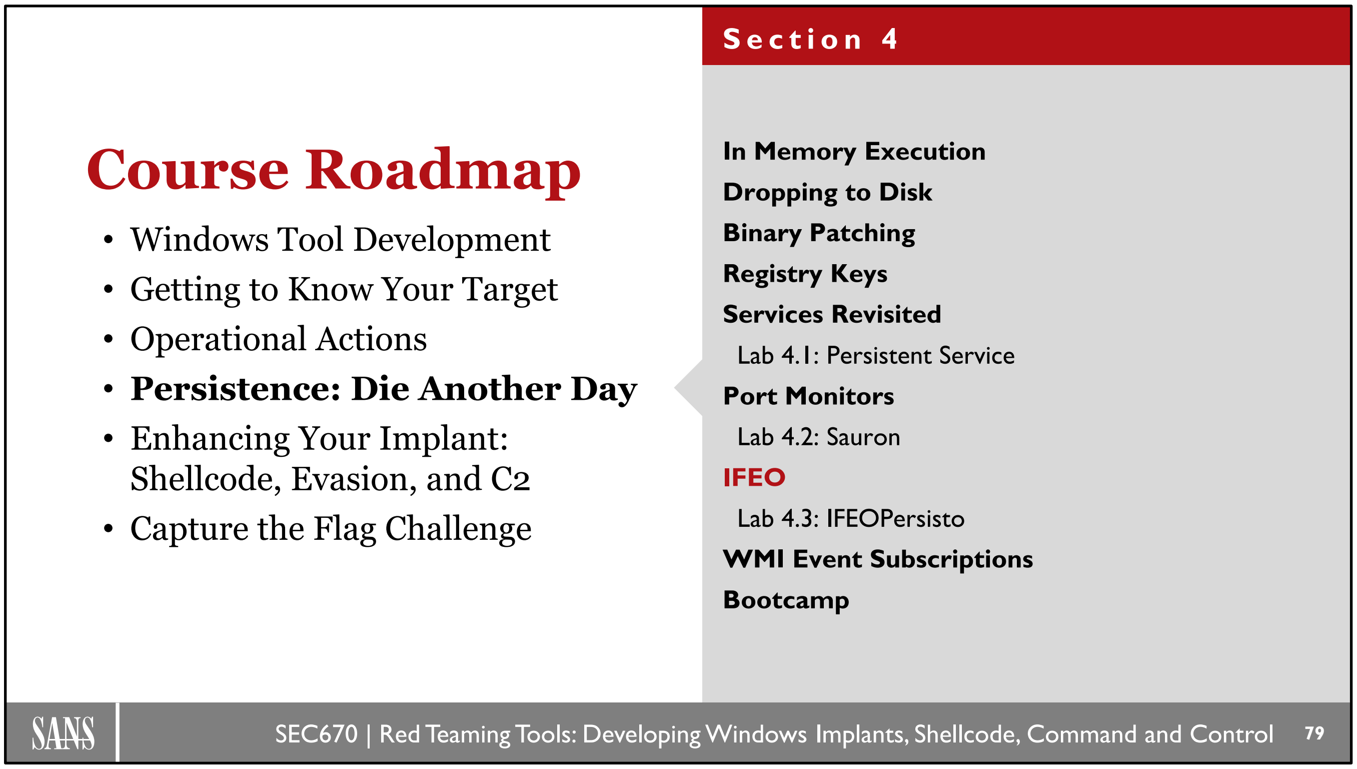
A: CreateNewMonitor

B: AddMonitor

C: AddNewMonitor

© 2024 Jonathan Reiter

© SANS Institute 2024



Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

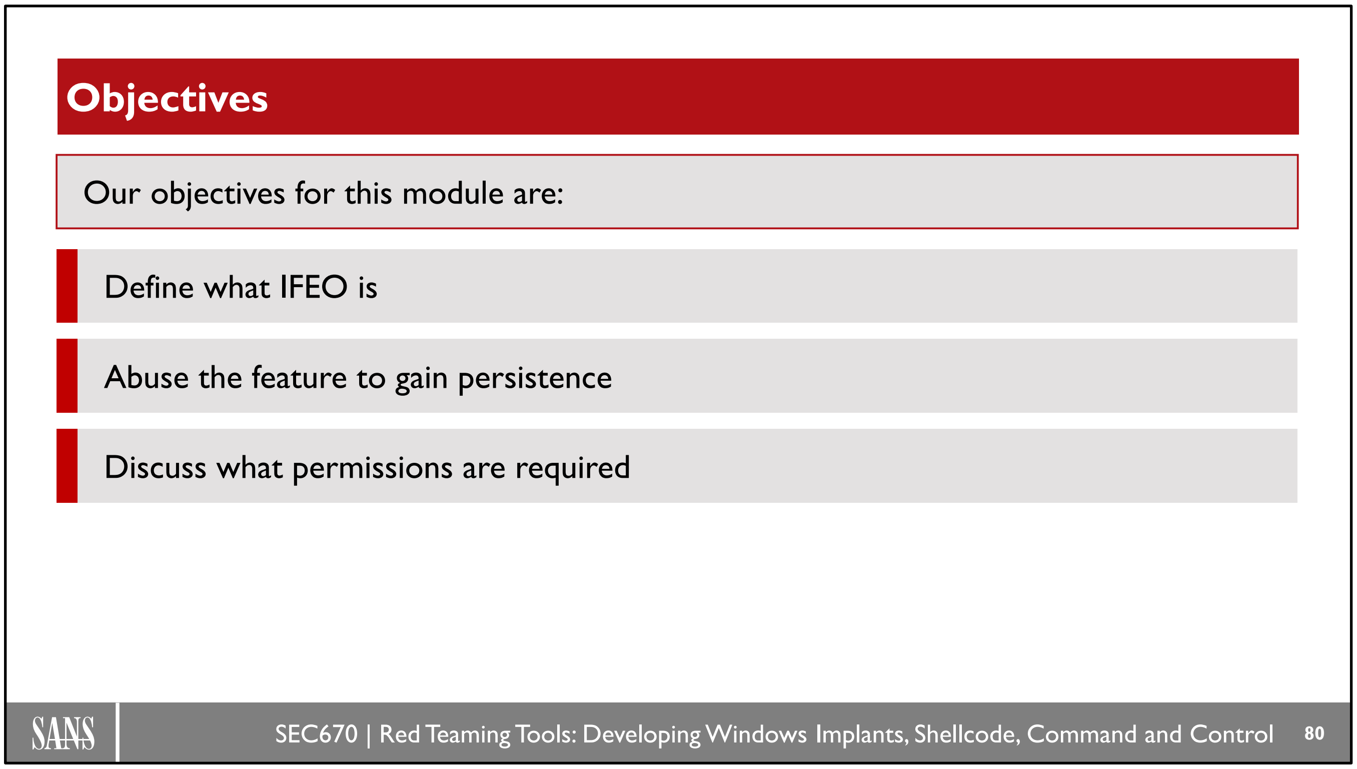
Section 4

SANS

SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 79

In this module, we will discuss how to persist using Image File Execution Options (IFEO).

© SANS Institute 2024



Objectives

Our objectives for this module are:

- Define what IFEO is
- Abuse the feature to gain persistence
- Discuss what permissions are required

SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 80

Objectives
The objectives for this module are to define Image File Execution Options and abuse the feature to service our persistence needs. We will also talk about what permissions are required for pulling this off.

© SANS Institute 2024

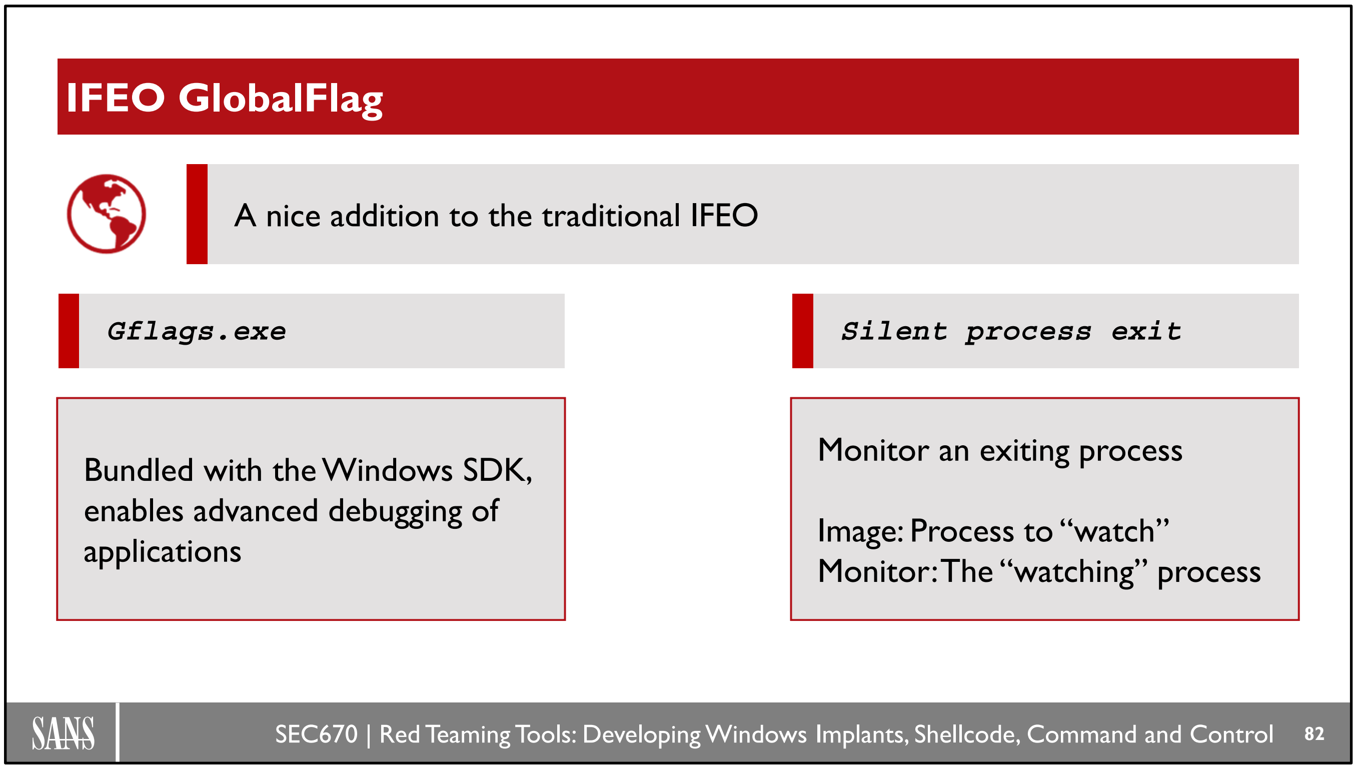
What Is IFEO?

Image File Execution Options

IFEO is a Windows Registry key that enables the debugging or tracing of a process when it is started. The IFEO key is a great for developers so their application can be debugged, but it is also great for malware authors looking to persist on the target.

What Is IFEO?
IFEO stands for Image File Execution Options and comes in the form of a Windows Registry key. The idea behind it is to give some more options for when a process begins execution. The options can be any number of actions, like having a debugger launch when the process does, or having your implant run when the process does. If you are a Sysinternals fan, you can easily have Process Explorer or Process Monitor launch when the Windows Task Manager is launched. In fact, Process Explorer gives you this option natively by choosing Options->Replace Task Manager. As a developer, we can take advantage of this key as a method to monitor our application. As an implant developer, we can take advantage of this key as a persistence mechanism same as malware authors have been doing. This method really works great for EXE files, and not so much for DLLs.

© SANS Institute 2024



IFEO GlobalFlag

A nice addition to the traditional IFEO

`Gflags.exe`

Bundled with the Windows SDK, enables advanced debugging of applications

`Silent process exit`

Monitor an exiting process

Image: Process to "watch"
Monitor: The "watching" process

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 82

IFEO GlobalFlag
To use GlobalFlags, the SDK must be installed. After installation is completed, the gflags binary should be located at: **C:\Program Files (x86)\Windows Kits\10\Debuggers\x64**. MSDN describes the binary as one that can enable more advanced debugging and is used to turn on other indicators that other tools track. Gflags has the power to set system-wide debugging settings under the following Registry key: **HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\GlobalFlag**. However, for this to take effect you must reboot the system. With it, we can also affect settings for a specific application. To make this setting stick, we would change the **HKLM\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Image File Execution Options*ImageFileName*\GlobalFlag**.

But wait, there's more! If you were to scroll down toward the bottom of the MSDN page, you would see something called Silent Process Exit. This feature allows you to "monitor" the silent exit of a process. This means we get to specify what actions we want to take when a process exits. The gflags.exe utility will make the necessary changes to this registry key: **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options*ProcessName*\GlobalFlag** and set the FLG_MONITOR_SILENT_PROCESS_EXIT flag. Simply replace "ProcessName" with the one that you want to "monitor" and then you have your persistence kicking off when the process exits.

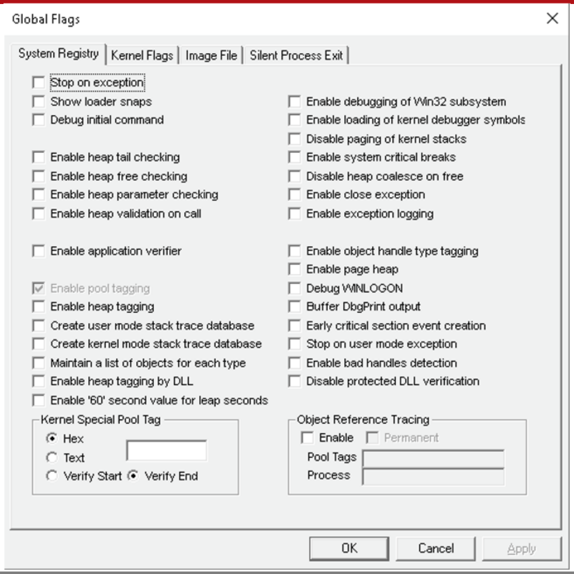
To summarize the relationship between the setting and the Registry Key, see the below list.

- Global settings: **HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\GlobalFlag**
- App-specific for all users: **HKLM\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Image File Execution Options*ImageFileName*\GlobalFlag**
- Silent exit, app-specific, for all users: **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit\*ImageFileName***
- App debugger: **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options*ImageFileName*\Debugger**

© 2024 Jonathan Reiter

© SANS Institute 2024

Running Gflags.exe


Global Flags

System Registry | Kernel Flags | Image File | Silent Process Exit

- [ ] Stop on exception
- [ ] Show loader snaps
- [ ] Debug initial command
- [ ] Enable heap tail checking
- [ ] Enable heap free checking
- [ ] Enable heap parameter checking
- [ ] Enable heap validation on call
- [ ] Enable application verifier
- [✓] Enable pool tagging
- [ ] Enable heap tagging
- [ ] Create user mode stack trace database
- [ ] Create kernel mode stack trace database
- [ ] Maintain a list of objects for each type
- [ ] Enable heap tagging by DLL
- [ ] Enable '60' second value for leap seconds
- [ ] Enable debugging of Win32 subsystem
- [ ] Enable loading of kernel debugger symbols
- [ ] Disable paging of kernel stacks
- [ ] Enable system critical breaks
- [ ] Disable heap coalesce on free
- [ ] Enable close exception
- [ ] Enable exception logging
- [ ] Enable object handle type tagging
- [ ] Enable page heap
- [ ] Debug WINLOGON
- [ ] Buffer DbgPrint output
- [ ] Early critical section event creation
- [ ] Stop on user mode exception
- [ ] Enable bad handles detection
- [ ] Disable protected DLL verification

Kernel Special Pool Tag: [✓ Hex] Text; Verify Start [✓ Verify End]

Object Reference Tracing: [ ] Enable [ ] Permanent; Pool Tags; Process

Running Gflags.exe
The GUI version of the gflags program looks similar to the screenshot on the slide. You can see the various tabs at the top of the window that are specific to categories like the Kernel, Image File, and Silent Process Exit. Feel free to explore the tool and the features that it provides. Once you have an understanding you can programmatically implement many of these items on your own.

© SANS Institute 2024

GflagsX

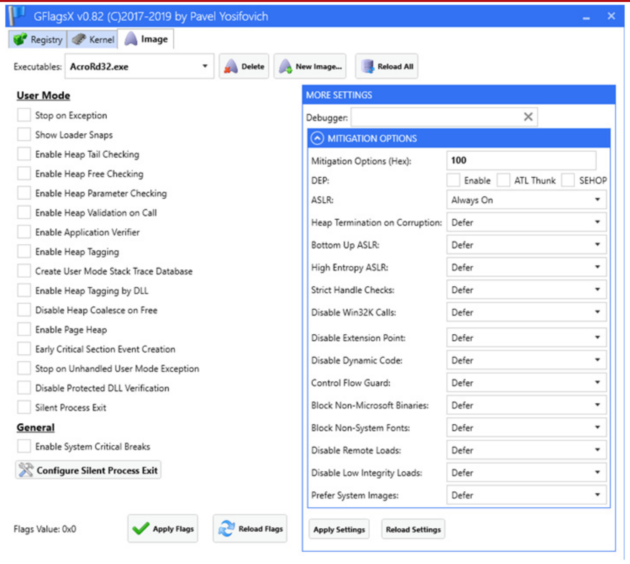
A modern take on the original gflags.exe utility

Pavel Yosifovich created a new version of gflags that offers a great new look to the tool. Check out his repo for this and other awesome tools: https://github.com/zodiacon.

GflagsX
Pavel Yosifovich has been recreating several tools with better features and a nicer looking GUI. Pavel has been posting them publicly on his repo https://github.com/zodiacon. If you get a chance, check them out for yourself.

© SANS Institute 2024

Running GflagsX



GFlagsX v0.82 (C)2017-2019 by Pavel Yosifovich

Executables: AcroRd32.exe

User Mode

- [ ] Stop on Exception
- [ ] Show Loader Snaps
- [ ] Enable Heap Tail Checking
- [ ] Enable Heap Free Checking
- [ ] Enable Heap Parameter Checking
- [ ] Enable Heap Validation on Call
- [ ] Enable Application Verifier
- [ ] Enable Heap Tagging
- [ ] Create User Mode Stack Trace Database
- [ ] Enable Heap Tagging by DLL
- [ ] Disable Heap Coalesce on Free
- [ ] Enable Page Heap
- [ ] Early Critical Section Event Creation
- [ ] Stop on Unhandled User Mode Exception
- [ ] Disable Protected DLL Verification
- [ ] Silent Process Exit

General

- [ ] Enable System Critical Breaks

Flags Value: 0x0

MORE SETTINGS

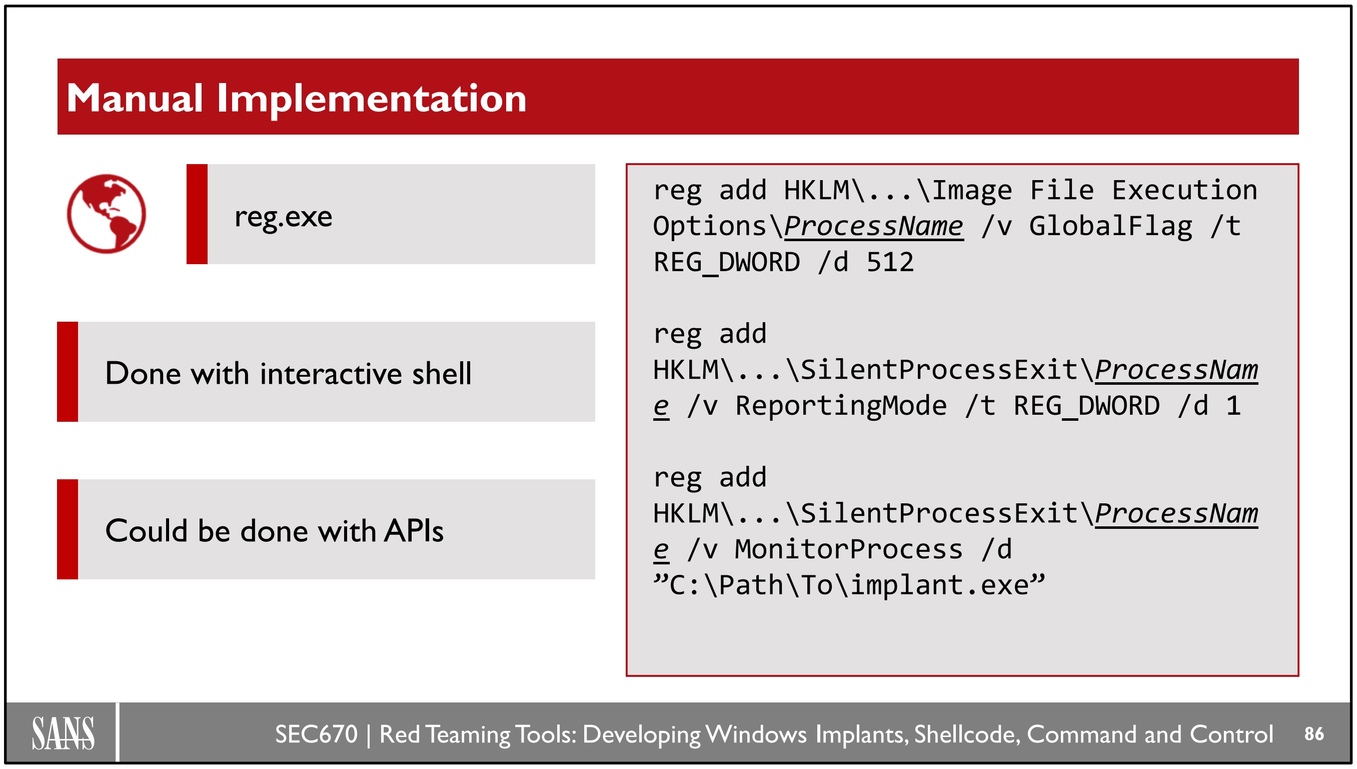
Debugger:

MITIGATION OPTIONS

Mitigation Options (Hex):	100
DEP:	[ ] Enable [ ] ATL Thunk [ ] SEHOP
ASLR:	Always On
Heap Termination on Corruption:	Defer
Bottom Up ASLR:	Defer
High Entropy ASLR:	Defer
Strict Handle Checks:	Defer
Disable Win32K Calls:	Defer
Disable Extension Point:	Defer
Disable Dynamic Code:	Defer
Control Flow Guard:	Defer
Block Non-Microsoft Binaries:	Defer
Block Non-System Fonts:	Defer
Disable Remote Loads:	Defer
Disable Low Integrity Loads:	Defer
Prefer System Images:	Defer

Running GflagsX
The GUI for Pavel's tool looks much nicer and more modern than the legacy tool. There are not as many tabs at the top of the window, but many of those features are consolidated. For example, the Silent Process Exit tab and its options are located under the Image tab for GflagsX.

© SANS Institute 2024



Manual Implementation

reg.exe

Done with interactive shell

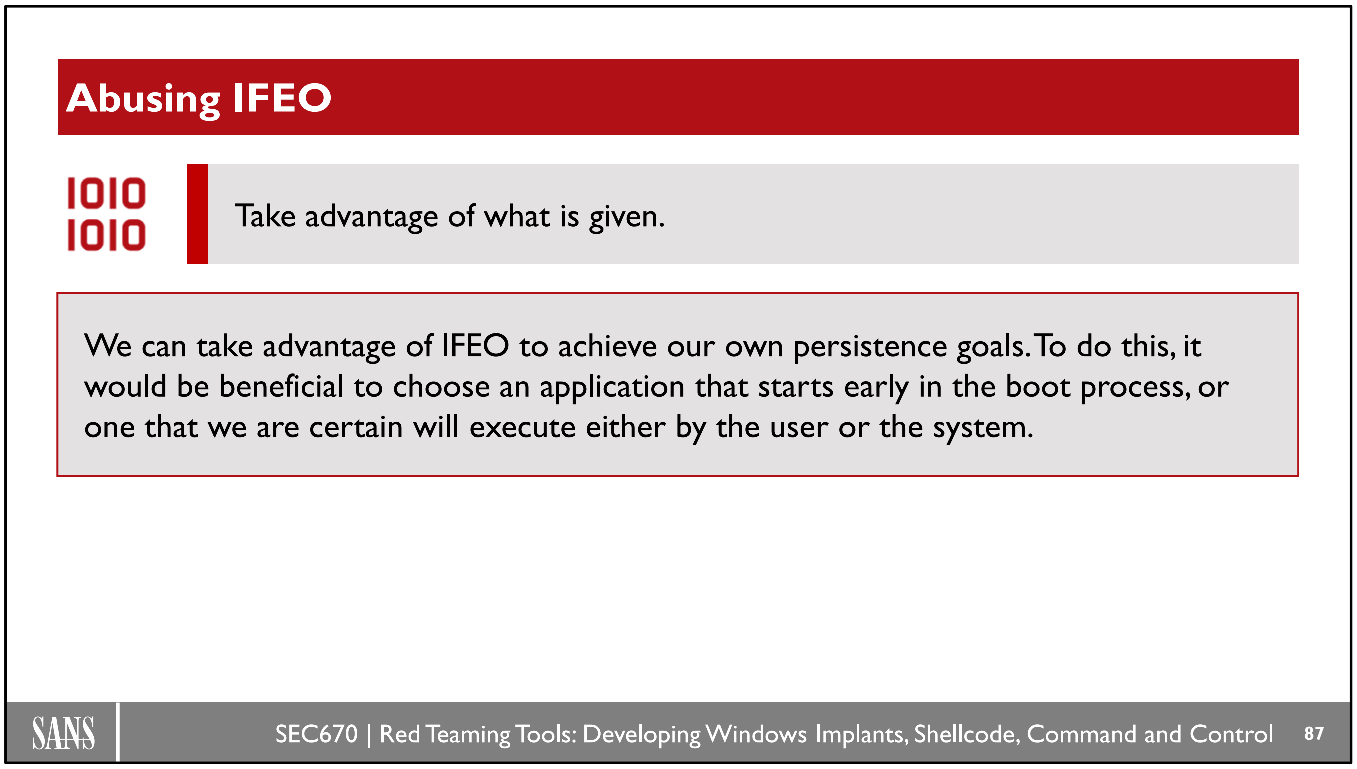
Could be done with APIs

```
reg add HKLM\...\Image File Execution
Options\ProcessName /v GlobalFlag /t
REG_DWORD /d 512

reg add
HKLM\...\SilentProcessExit\ProcessNam
e /v ReportingMode /t REG_DWORD /d 1

reg add
HKLM\...\SilentProcessExit\ProcessNam
e /v MonitorProcess /d
"C:\Path\To\implant.exe"
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 86

Manual Implementation
It is possible to manually make the Registry keys using the reg.exe command-line utility. The examples on the slide demonstrate how we could implement this if we were using an interactive shell on the target. The other option is to do this programmatically with the appropriate Win32 Registry APIs we covered earlier in the course, like ***RegOpenKeyExA***. After obtaining a handle to a key, we can then call ***RegCreateKeyExA*** so we can create or modify a key, and then call ***RegSetValueEx*** so we can write the values to the proper keys, etc. Needless to say, there is more than one way that we can go about getting this done.

© SANS Institute 2024



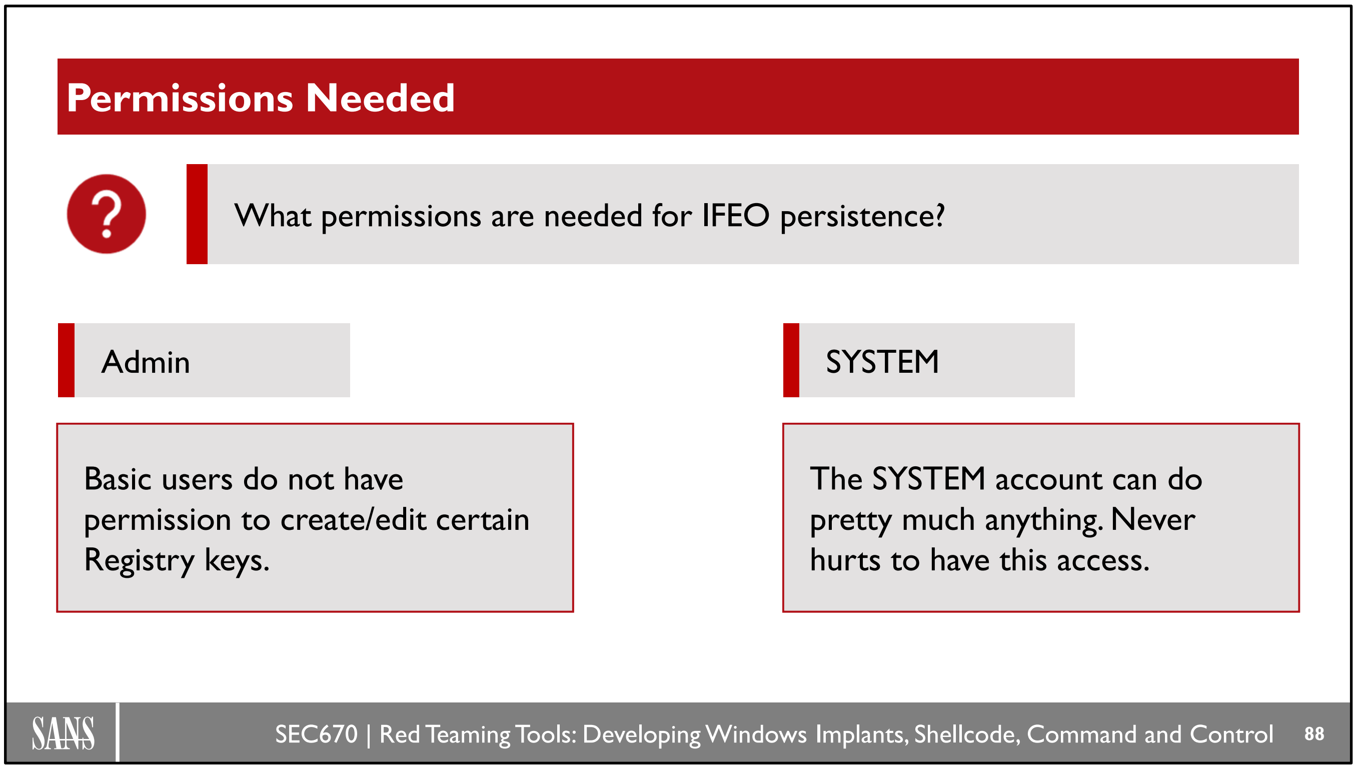
Abusing IFEO

Take advantage of what is given.

We can take advantage of IFEO to achieve our own persistence goals. To do this, it would be beneficial to choose an application that starts early in the boot process, or one that we are certain will execute either by the user or the system.

Abusing IFEO
Since we are interested in persisting on the target, we are going to abuse this option. The trick is choosing the best target process to do this against. We should choose one that kicks off when the system is starting up. A good possibility that fits into that role would be userinit.exe as it starts when the system does. You can leverage your knowledge of Windows internals for other options. There could also be third party applications that start when the system does, so they can serve as good options too. Perhaps you know that a user opens a particular process as part of their daily work routine; we can leverage that as well. In the end, pick something that you do not have to trigger manually.

© SANS Institute 2024



Permissions Needed

What permissions are needed for IFEO persistence?

Admin

Basic users do not have permission to create/edit certain Registry keys.

SYSTEM

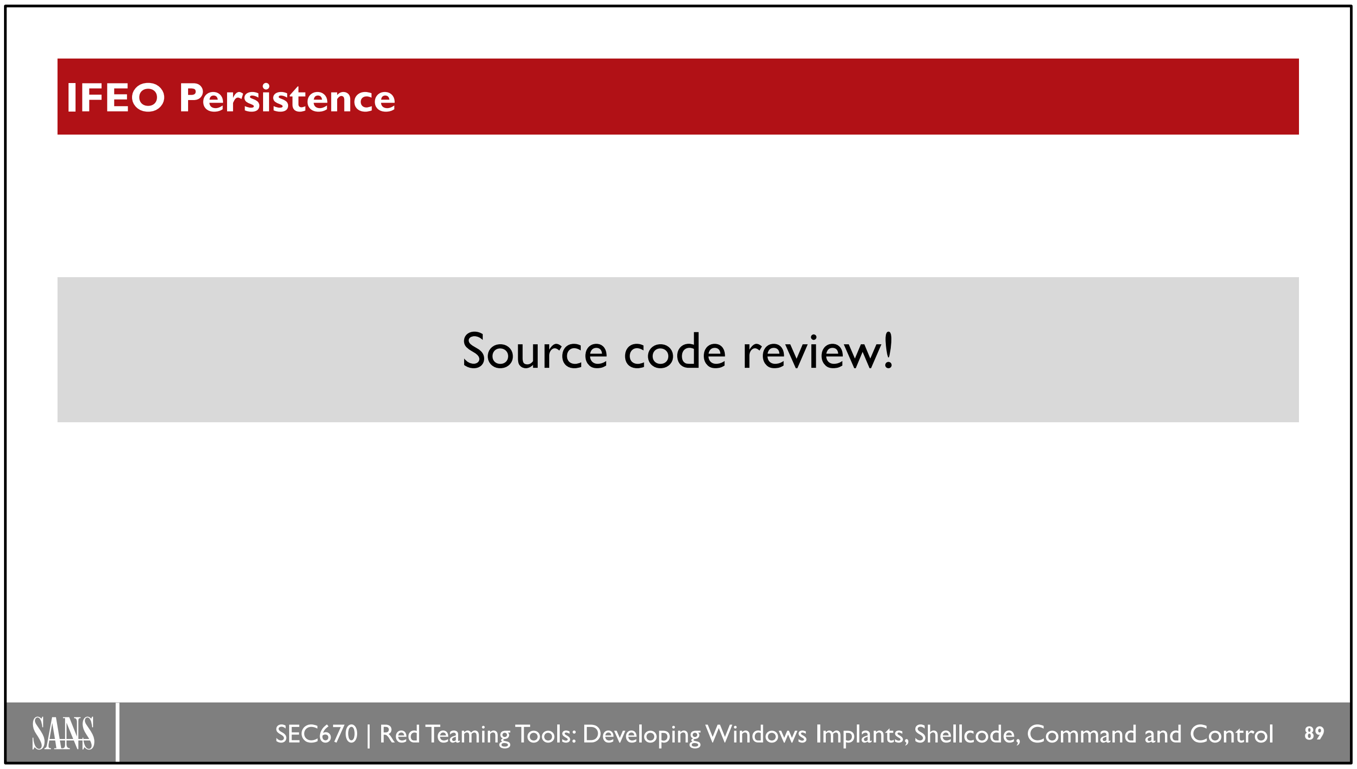
The SYSTEM account can do pretty much anything. Never hurts to have this access.

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 88

Permissions Needed
Sadly, if we only have permissions as a basic user, then we will be denied access when trying to modify the HKLM Registry keys needed for the IFEO persistence method. Since we have already discussed a few ways that we can escalate our privileges, there is no need to bring that discussion here, but we can use any technique we can to become Admin. We might be lucky and find the path to Admin and perhaps even SYSTEM is trivial. Regardless of the path to gain higher privileges, now that we have them, we can modify the Registry as we see fit. One thing to remember is to properly clean up should you need to get off the target. We would want to put things back where they belong and leave the system how it was before we came on the box, so disabling IFEO for an image would need to be done. The logic could be built into an uninstall command, or similar command, and sent to the implant so that when sent, it will go through and reverse many of the actions taken, like Registry key modifications.

© 2024 Jonathan Reiter

© SANS Institute 2024

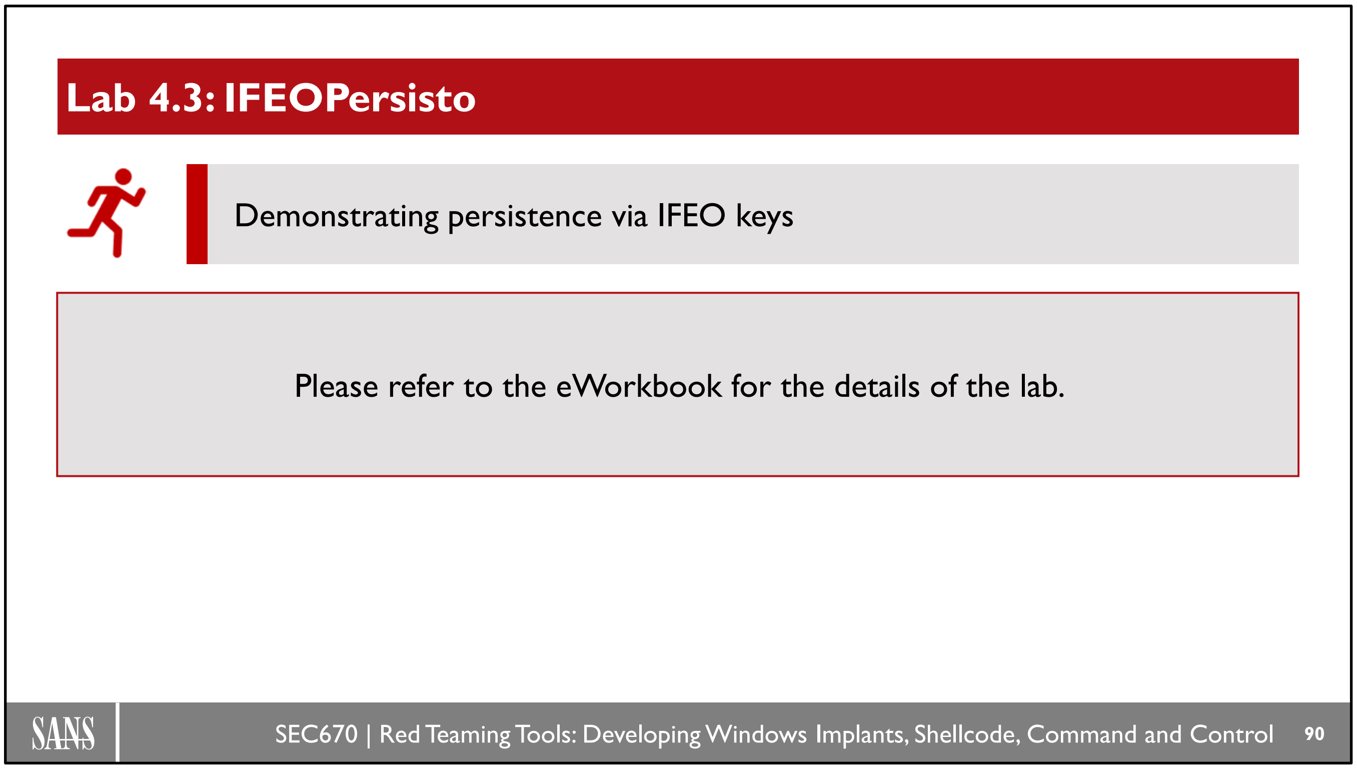


IFEO Persistence

Source code review!

IFEO Persistence
Time to jump into the source code and explain it before you implement it on your own.

© SANS Institute 2024



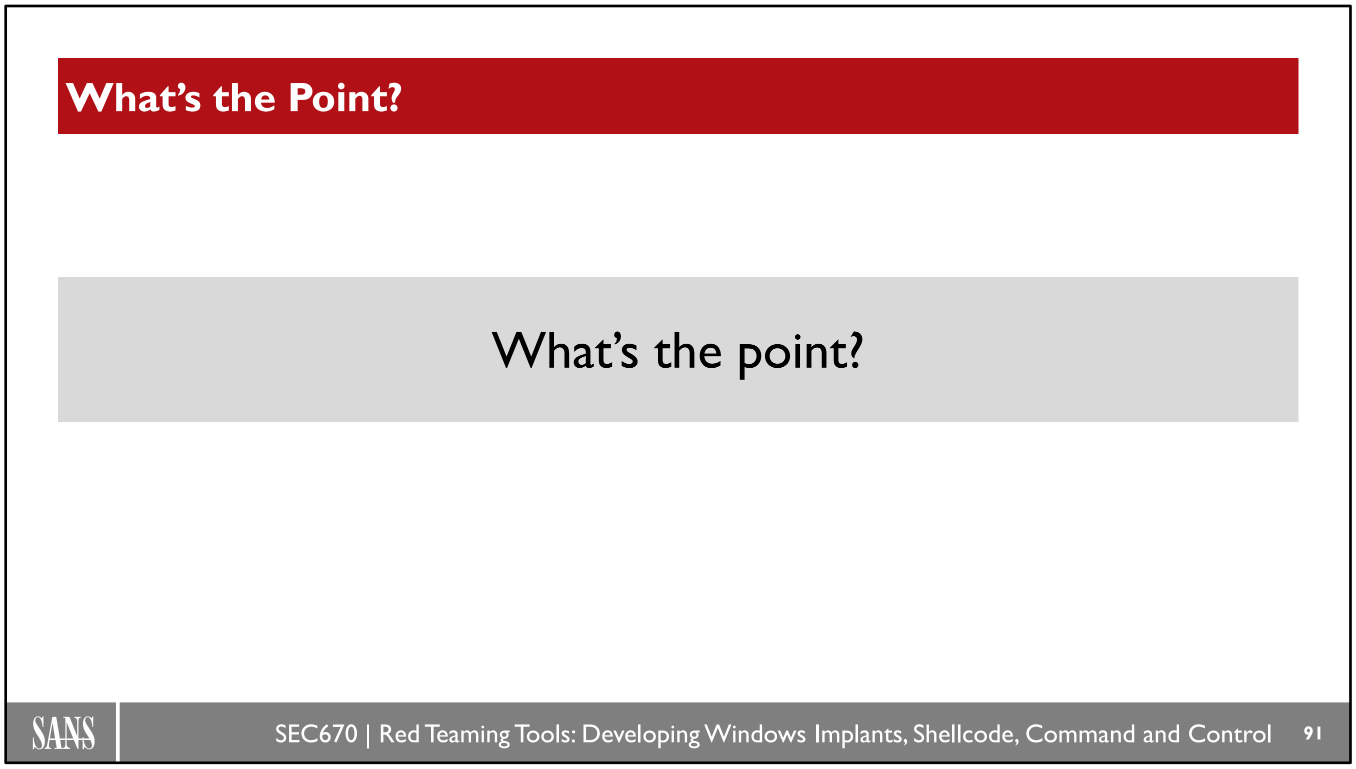
Lab 4.3: IFEOPersisto

Demonstrating persistence via IFEO keys

Please refer to the eWorkbook for the details of the lab.

Lab 4.3: IFEO Persistence
Please refer to the eWorkbook for the details of this lab.

© SANS Institute 2024

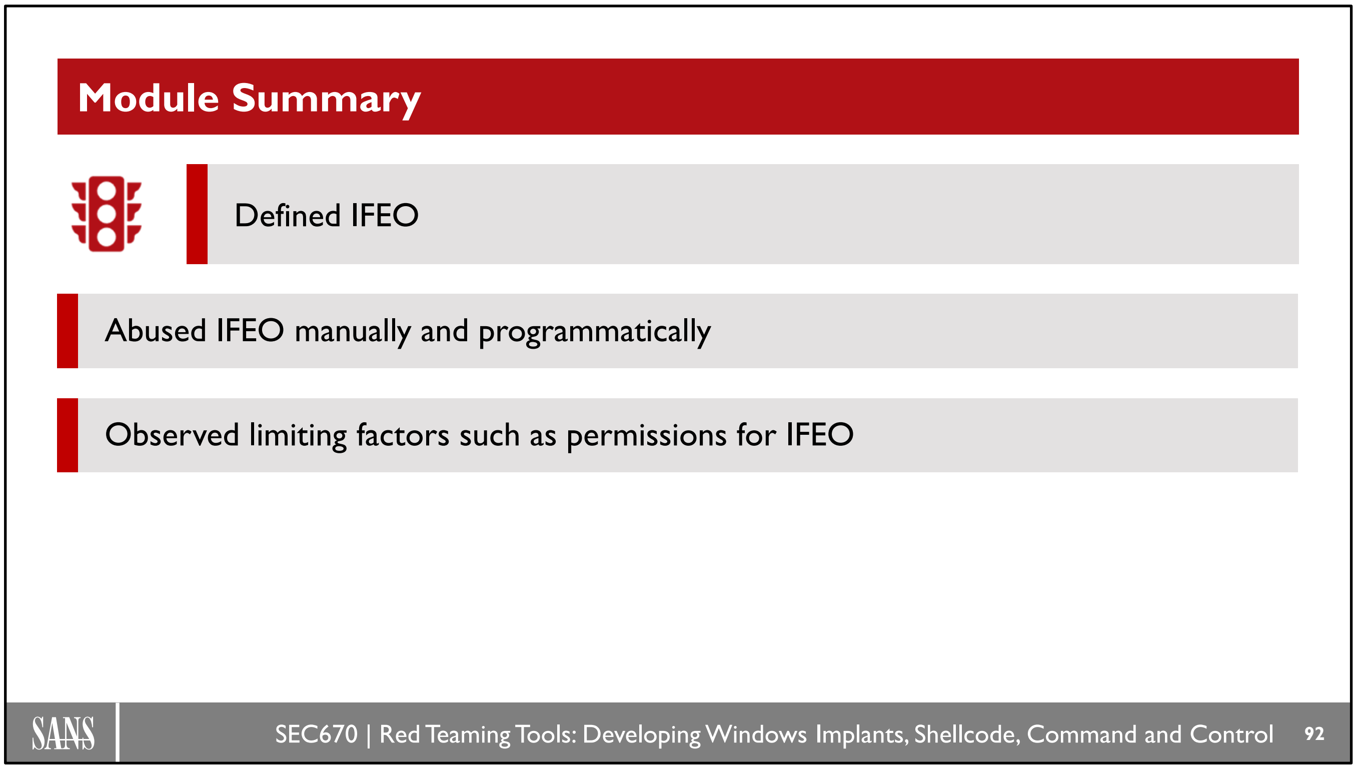


What's the Point?

What's the point?

What's the Point?
The point of this lab was to become familiar with the persistence method of Image File Execution Options and the two variants: process start and silent exits.

© SANS Institute 2024



Module Summary

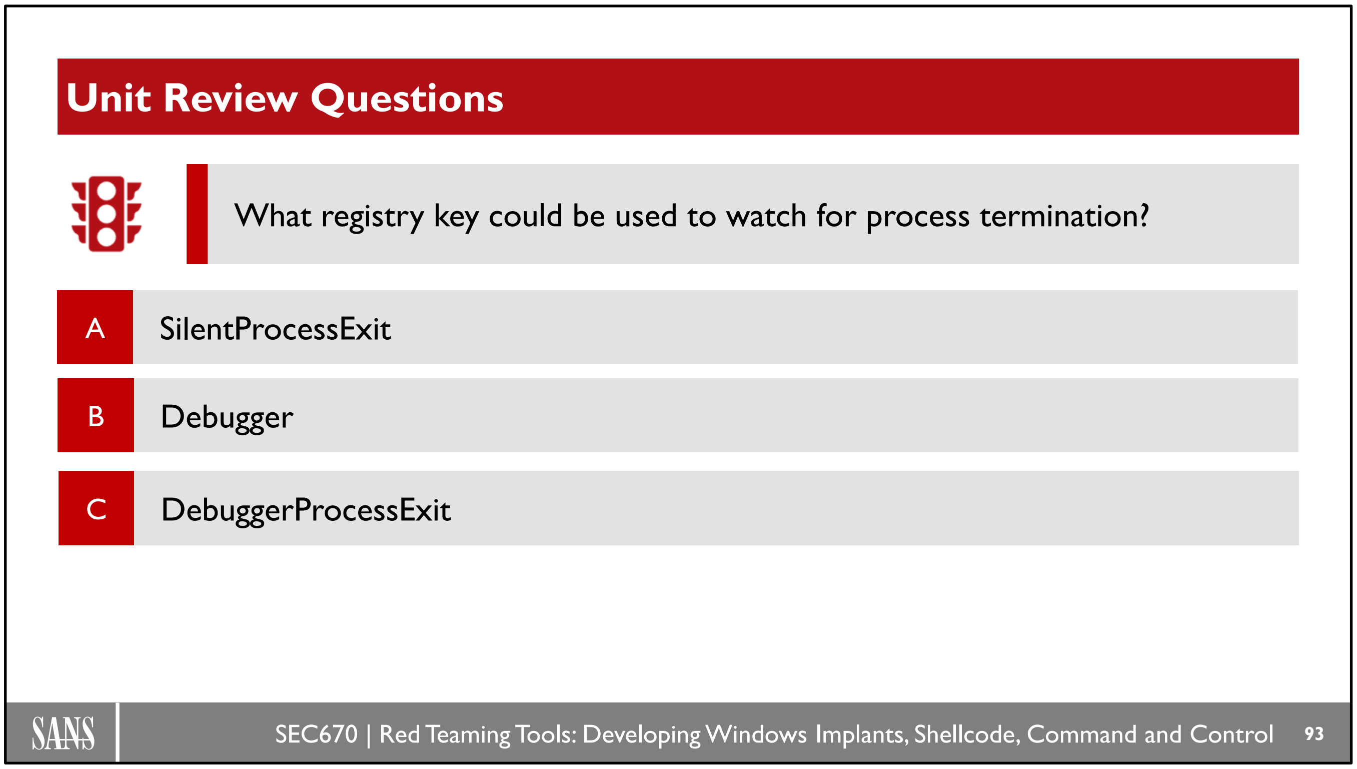
- Defined IFEO
- Abused IFEO manually and programmatically
- Observed limiting factors such as permissions for IFEO

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 92

Module Summary
In this module, we defined IFEO and discussed how Microsoft intended for it to be used. We also moved into discussing another variant of IFEO, which was the SilentProcessExit option. From there we looked at how we can abuse both variants while at the same time observing permissions issues if not being done with elevated permissions like Admin or SYSTEM.

© 2024 Jonathan Reiter

© SANS Institute 2024



Unit Review Questions

What registry key could be used to watch for process termination?

A SilentProcessExit

B Debugger

C DebuggerProcessExit

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 93

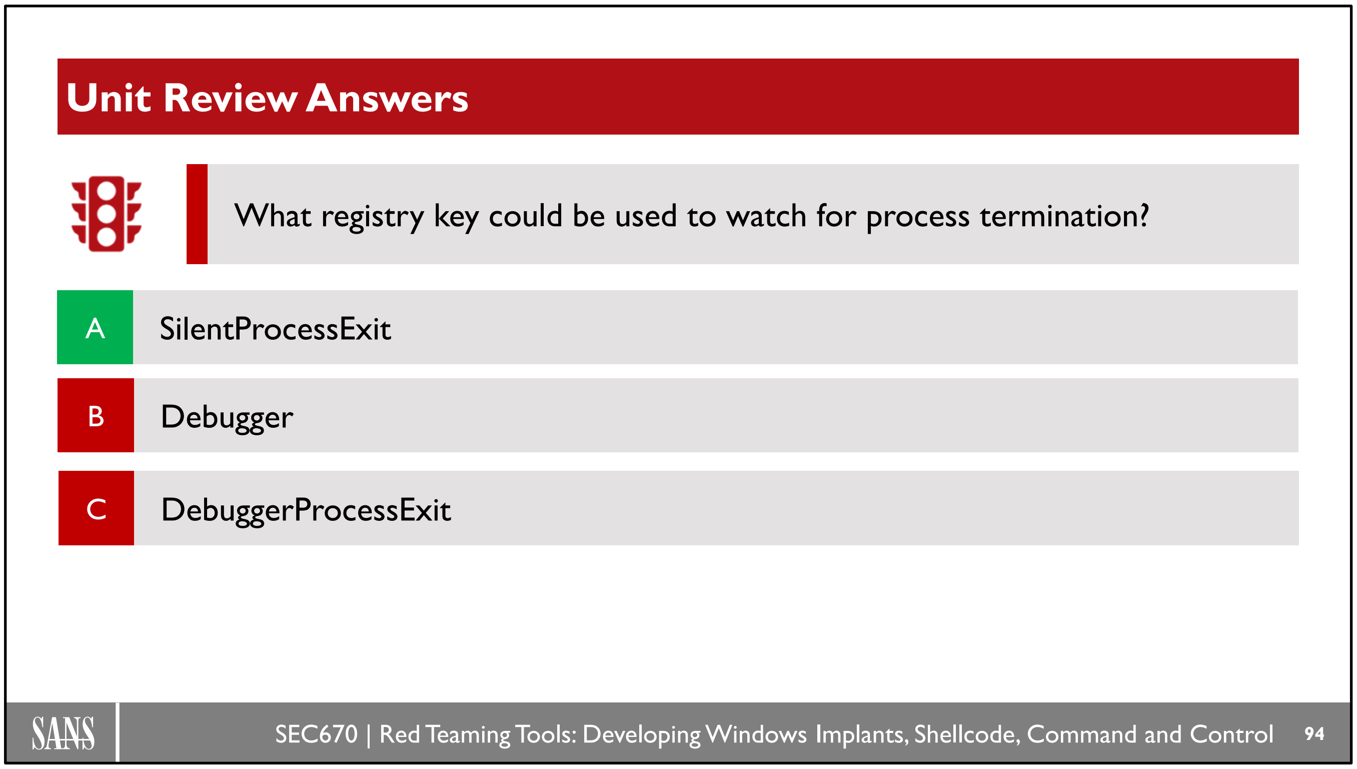
Unit Review Questions
Q: What registry key could be used to watch for process termination?

A: SilentProcessExit

B: Debugger

C: DebuggerProcessExit

© SANS Institute 2024



Unit Review Answers

What registry key could be used to watch for process termination?

A SilentProcessExit

B Debugger

C DebuggerProcessExit

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 94

Unit Review Answers
Q: What registry key could be used to watch for process termination?

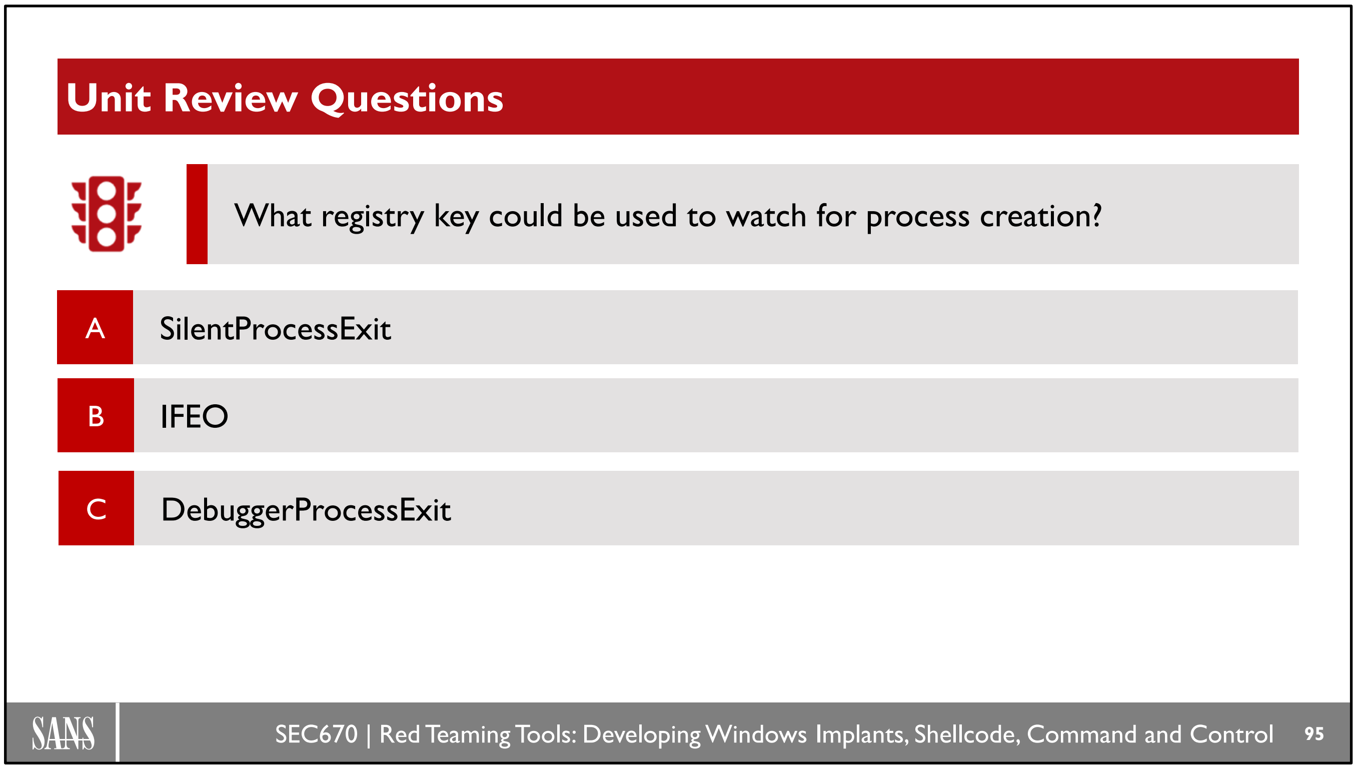
A: SilentProcessExit

B: Debugger

C: DebuggerProcessExit

© 2024 Jonathan Reiter

© SANS Institute 2024



Unit Review Questions

What registry key could be used to watch for process creation?

A	SilentProcessExit
B	IFEO
C	DebuggerProcessExit

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 95

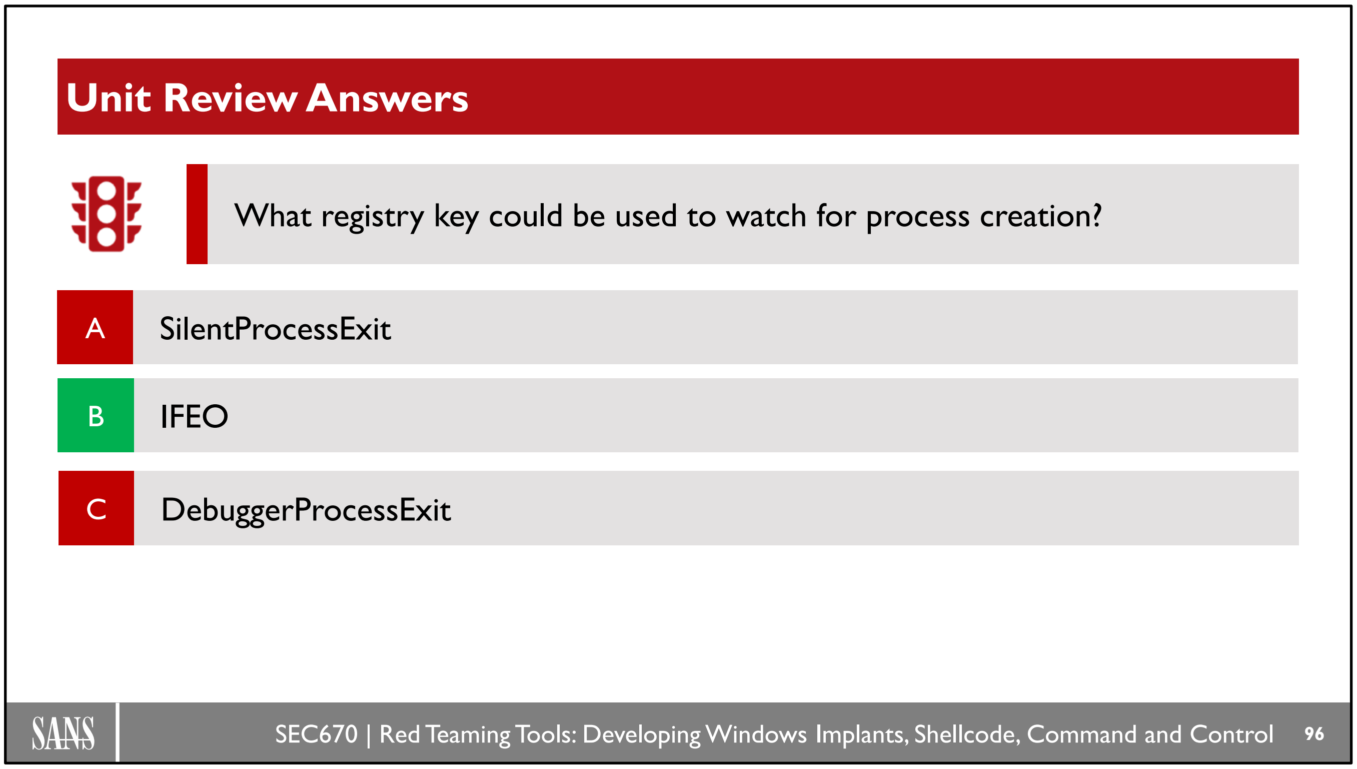
Unit Review Questions
Q: What registry key could be used to watch for process creation?

A: SilentProcessExit

B: IFEO

C: DebuggerProcessExit

© SANS Institute 2024



Unit Review Answers

What registry key could be used to watch for process creation?

A	SilentProcessExit
B	IFEO
C	DebuggerProcessExit

Unit Review Answers
Q: What registry key could be used to watch for process creation?

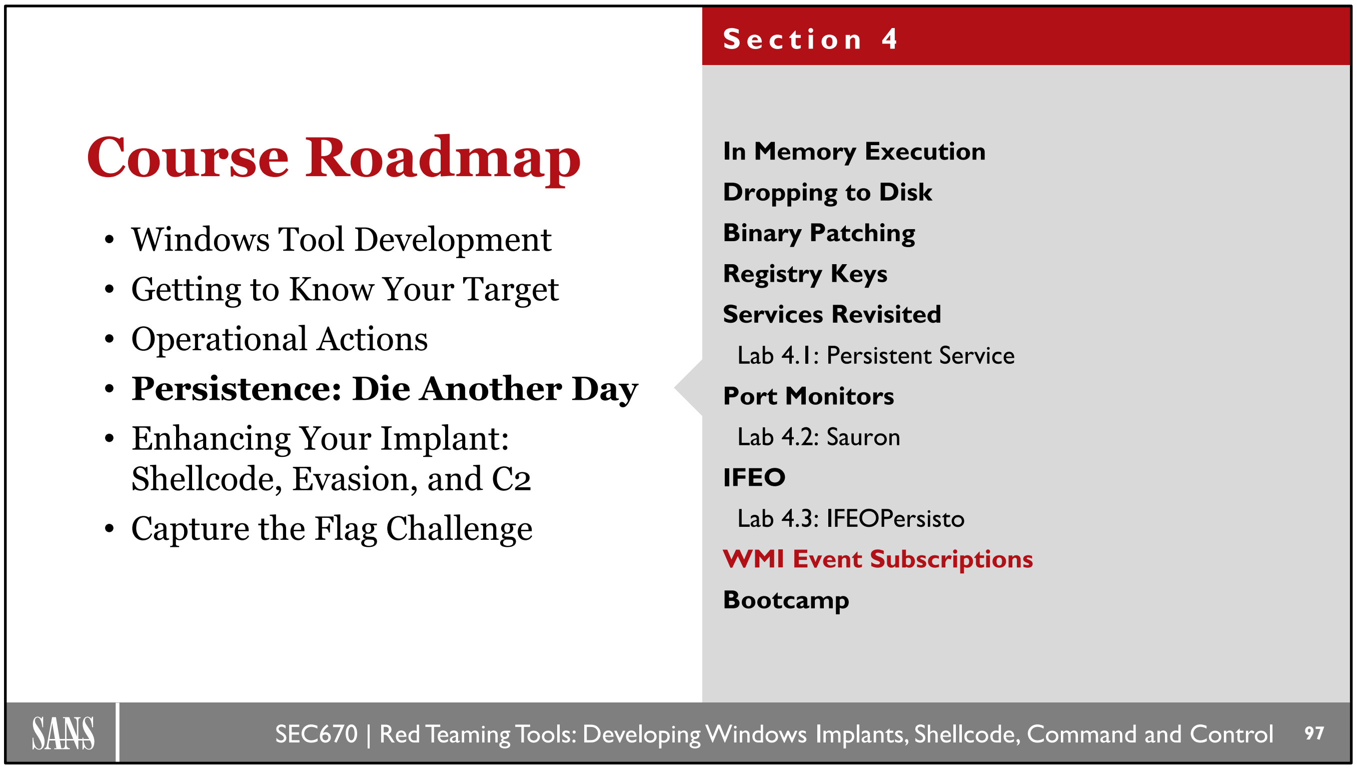
A: SilentProcessExit

B: IFEO

C: DebuggerProcessExit

© 2024 Jonathan Reiter

© SANS Institute 2024



Section 4

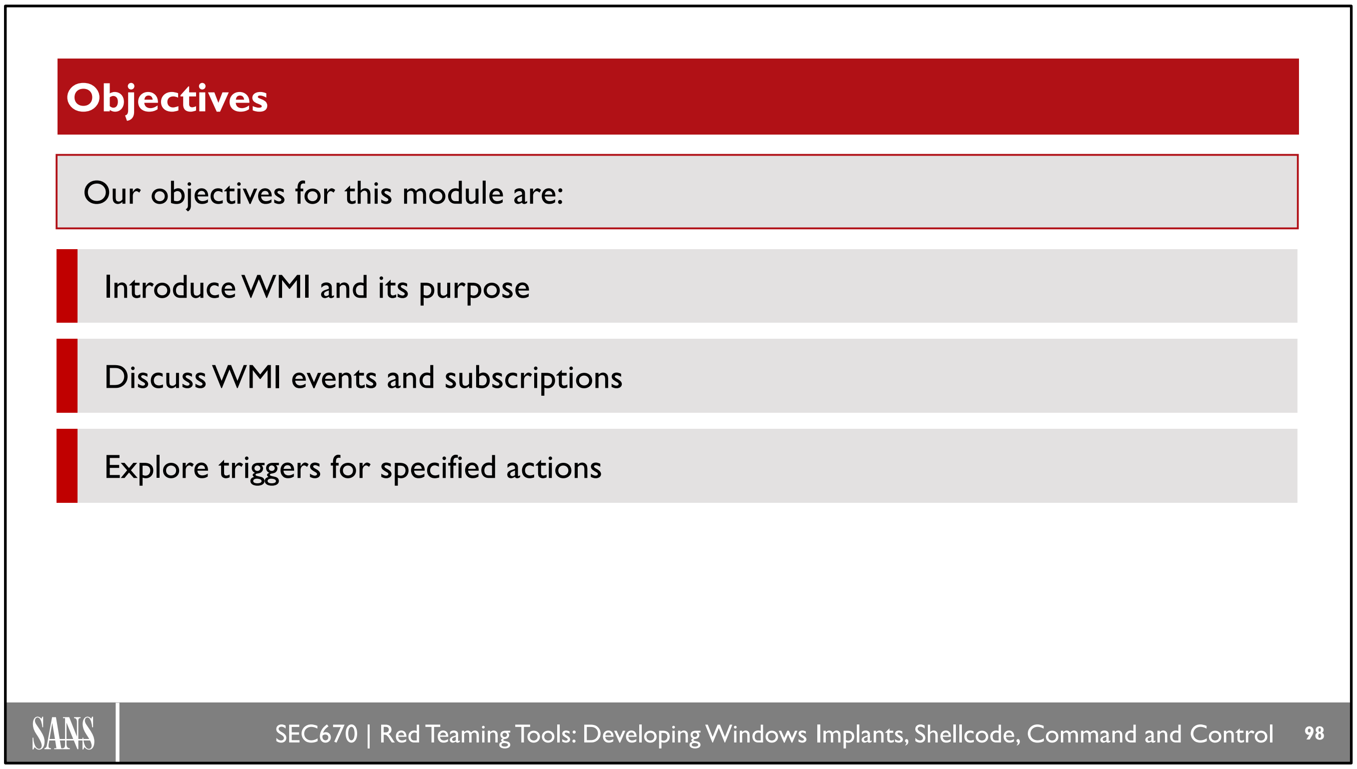
Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 97

In this module, we will discuss why you would need to drop something to disk, where to drop, cleaning up, and more.

© SANS Institute 2024



Objectives

Our objectives for this module are:

- Introduce WMI and its purpose
- Discuss WMI events and subscriptions
- Explore triggers for specified actions

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 98

Objectives
The objectives for this module are to start off with an introduction to the Windows Management Instrumentation (WMI) and its intended usage. We will also get into WMI events, filters, and subscribing to receiving events of interest. Lastly, we will look at the triggers to kick off an event, like the creation of a new process, a new logical drive being loaded, or a failed logon attempt. WMI abuse is an active technique still being used by nation state actors today. WMI attacks are not new at all and yet they are still being overlooked by cybersecurity defenders and AV/EDR solutions (although these groups are getting better at noticing these attacks).

© 2024 Jonathan Reiter

© SANS Institute 2024

What Is WMI?

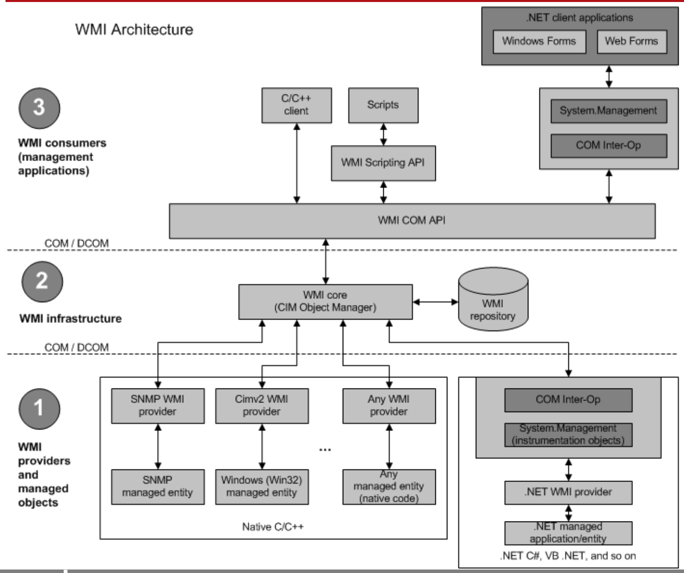
The Windows Management Instrumentation (WMI)

The Windows OS must manage the data and operations not only for itself but also for remote systems. WMI is the method that instruments this management and is designed for developers and administrators to use with ease via C++ development or PowerShell scripting.

What Is WMI?
Windows generates an enormous amount of data and executes many operations. Because of this, it needs something to assist with the management of said data. Enter the Windows Management Instrumentation (WMI). WMI enables developers and administrators to query data that might be held in the WMI database. For administrators, WMI enables you to connect to and query large numbers of remote computers for whatever data you might need. In addition, you can invoke WMI methods to execute something locally or against a remote system, like creating a new process that may have crashed and did not start up again, or you need to create some new Registry keys. WMI becomes even easier to manage with PowerShell natively supporting WMI with several cmdlets, like Get-WmiObject for one. For developers, WMI is perfect and catered for our C++ development. We can create WMI providers that provide data to consumers, and we can also create our own consumers to subscribe to events, or data, that existing providers are pushing out. Before we get into events and filters, we need to understand a bit more about the back end of WMI.

© SANS Institute 2024

WMI Architecture



- **Providers and Objects**
- **WMI Infrastructure**
- **WMI Consumers**

WMI Architecture
The graphic on the slide is the representation of the architecture of WMI, according to MSDN online documentation. You can follow this in stages starting at the very bottom with area 1. Area 1 is where WMI providers are doing their job of providing data that can be stored in the WMI repository, and area 2 is where data is sent to WMI consumers in area 3. One very interesting WMI provider is the Win32 provider that can give consumers a list of processes. This could prove to be another method used for process enumeration—it's just noisy. Area 2 houses the core of WMI, and its repo can store static data to be queried by consumers. This data is persistent across reboots and offers an interesting persistence mechanism that we will explore. As you might have noticed on the graphic, consumers can be created by C++ programs or scripting with PowerShell, which makes it extremely easy to utilize. In the end they all go through COM interfaces, but regardless, they enable us to create a consumer that can query events given by providers. Because there can be so many events, we must filter them by creating a filter, and filters are created using a specific query language named WQL, or the Windows Query Language. WMI adheres to a universal standard known as CIM, the Common Interface Model.

© SANS Institute 2024

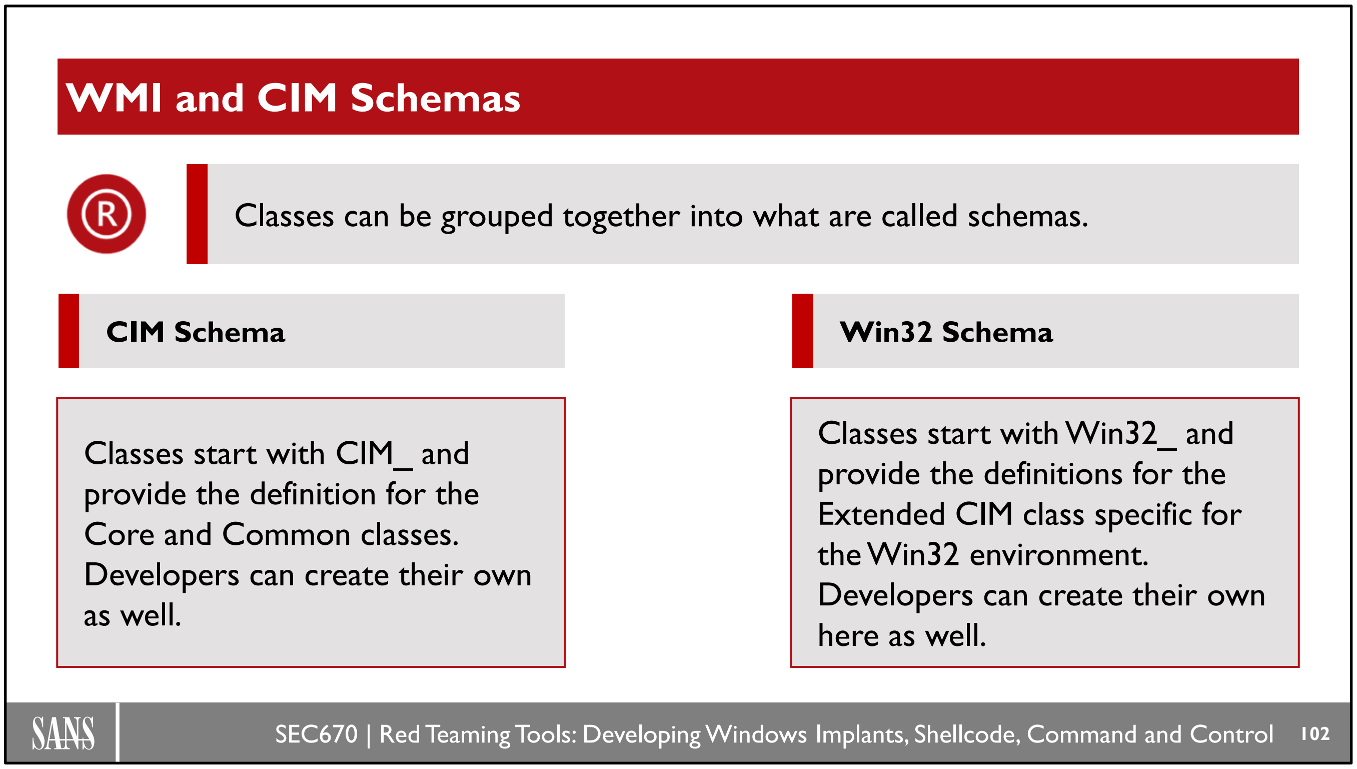
What Is CIM?

The Common Information Model (CIM)

The CIM is an industry standard that is used by WMI to represent various items, like systems, processes, devices, and more. CIM is object oriented and gives the look and feel of a C++ class. CIM gives us three levels of classes: Core, Common, and Extended.

What Is CIM?
Standards typically have models to go by and WMI is no exception because it follows the industry standard called the Common Information Model (CIM). The CIM is used to represent data in a uniformed way, data such as systems, processes/applications, networks, devices, and more. There are numerous existing CIM classes, but developers can create new ones that can be used to represent completely customized products like networked ovens or HVAC units. These classes are very similar to that of C++ classes developers would create. At its core, CIM provides various levels of classes—three of them to be exact. The first one is Core, which is used to represent the managed objects for managed systems. Next is the Common class. These classes are an extension of the Core classes and are not tied down to any specific technology. Finally, we have the Extended class, which unlike the Common class is specific to certain technologies like operating systems. One example of a Windows Extended class is Win32_ComputerSystem. These levels of classes make CIM cross-platform, except for the Extended class, of course.

© SANS Institute 2024



WMI and CIM Schemas

Classes can be grouped together into what are called schemas.

CIM Schema

Classes start with CIM_ and provide the definition for the Core and Common classes. Developers can create their own as well.

Win32 Schema

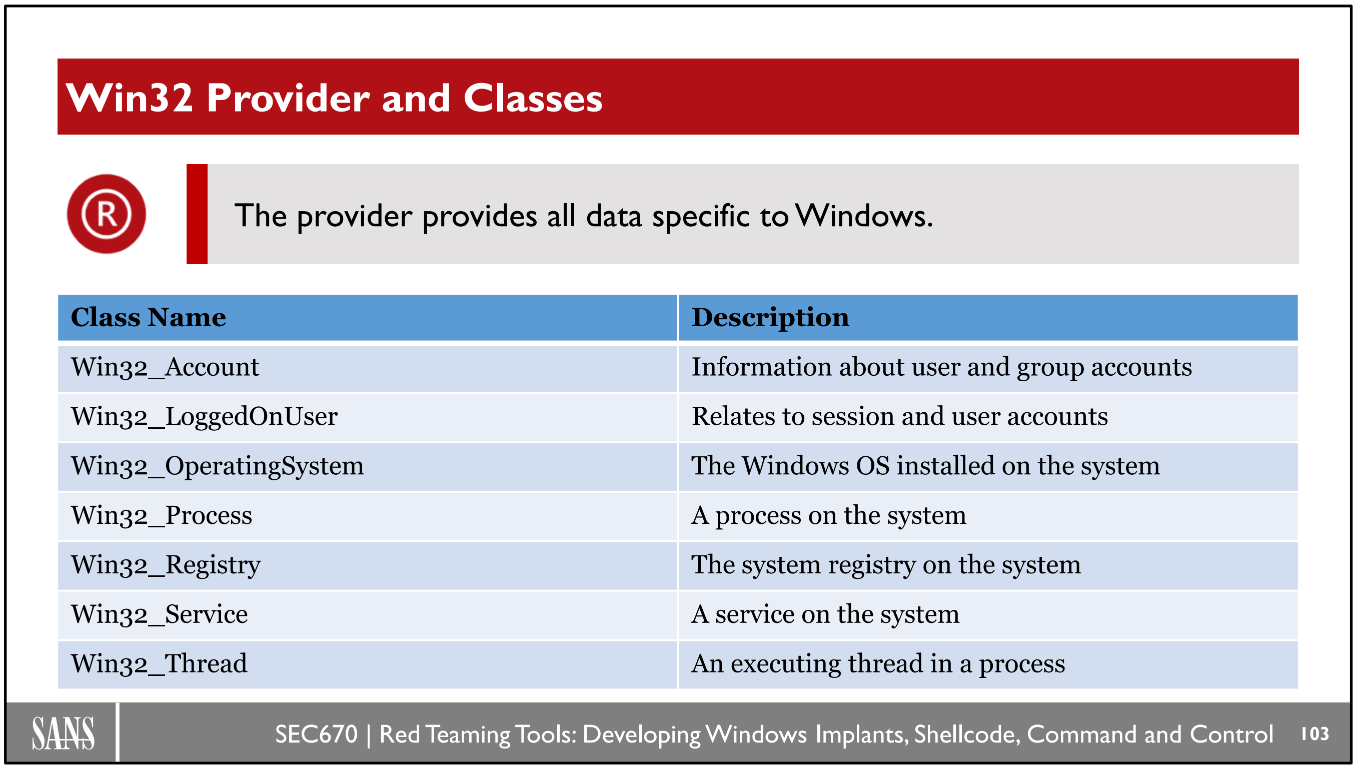
Classes start with Win32_ and provide the definitions for the Extended CIM class specific for the Win32 environment. Developers can create their own here as well.

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 102

WMI and CIM Schemas
WMI and CIM classes are often grouped together to form what are called schemas, and they are typically specific to certain managed objects. Specific to the Windows SDK, there are two schemas in place. The first schema is the CIM schema, and its classes are easy to identify as they begin with the prefix CIM_. The Core and Common classes are defined as part of the CIM schema. There are many CIM classes, like CIM_Battery, CIM_Memory, CIM_ComputerSystem, CIM_Directory, etc. Perhaps the more interesting schema is the Win32 schema. It is here that the Extended class is defined that provides classes specific to the Win32 environment. Win32 is also a provider, but more on that shortly. Developers can create their own schemas, classes, etc. in either the CIM schema or the Win32 schema.

© 2024 Jonathan Reiter

© SANS Institute 2024



Win32 Provider and Classes

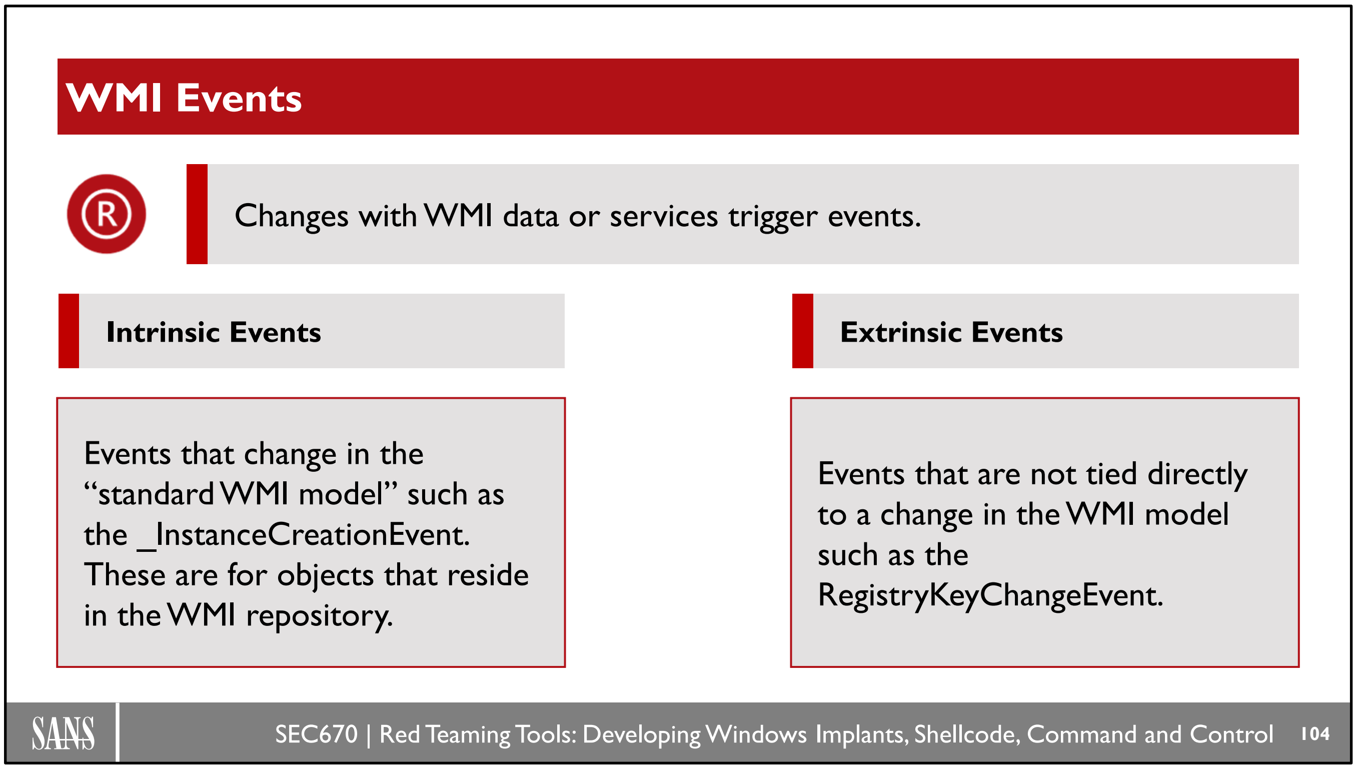
The provider provides all data specific to Windows.

Class Name	Description
Win32_Account	Information about user and group accounts
Win32_LoggedOnUser	Relates to session and user accounts
Win32_OperatingSystem	The Windows OS installed on the system
Win32_Process	A process on the system
Win32_Registry	The system registry on the system
Win32_Service	A service on the system
Win32_Thread	An executing thread in a process

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 103

Win32 Provider and Classes
The Win32 provider is where we can fetch any data that might relate to the Windows operating system. The data can be static or dynamic, such as a list of running processes. The classes shown on the slide are but a small subset of classes that are defined by the Win32 provider. As you can see, there are several classes that can be used for very specific purposes. We can use specific classes for recon about users of the system, who is logged on like an Admin or Domain Admin, what processes are running, and more. Several classes have methods that can be invoked like the Win32_Process class. We could use it to create a new instance of the class, which in this case would be a new process. To narrow things down and cut out some of the noise, we could create filters to focus on certain events from a specific class. We can filter on any number of items, like filtering on processes using the Win32_Process class, we can filter for account logon attempts, etc. Once a filter is created, it can be used to trigger events if the filter criteria is met.

© SANS Institute 2024



WMI Events

Changes with WMI data or services trigger events.

Intrinsic Events

Events that change in the "standard WMI model" such as the _InstanceCreationEvent. These are for objects that reside in the WMI repository.

Extrinsic Events

Events that are not tied directly to a change in the WMI model such as the RegistryKeyChangeEvent.

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 104

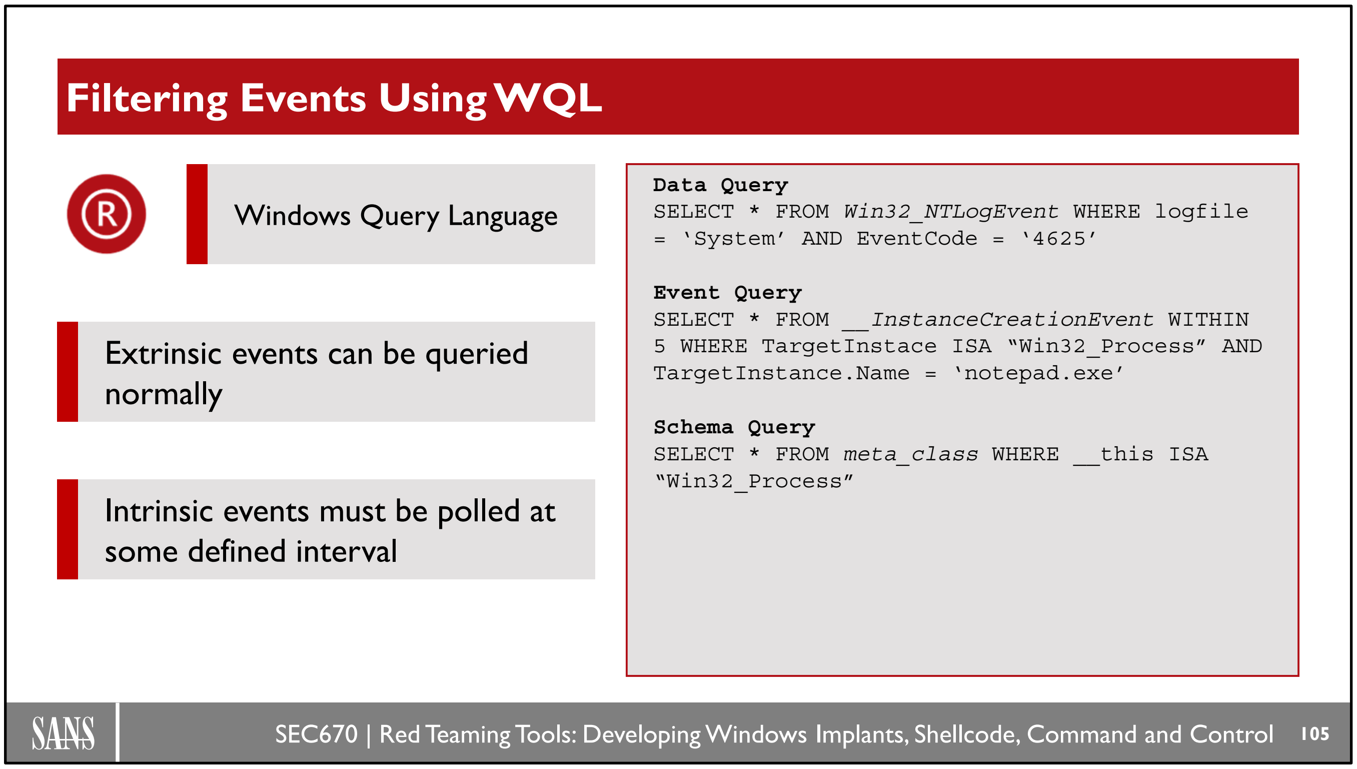
WMI Events

Before we would ever create a subscription to an event, we need to determine what type of event we will be receiving. WMI has intrinsic and extrinsic events with each type being quite different. The intrinsic events are created when there is some kind of change in the WMI model. The WMI repository holds objects inside of it, and WMI providers can create, delete, or modify the objects. An example of such objects are classes, instances of classes, or namespaces. When WMI providers make changes to objects, an intrinsic event occurs. To these changes, WMI conducts a polling operation like *_InstanceCreationEvent* for when an instance of a class is created. Another example of an event could be *_InstanceModificationEvent* for when an instance is modified.

Extrinsic events are not triggered by changes to WMI repository objects. Specifically, when an event cannot be tied directly to a change in the data model it must be an extrinsic event. One such example would be events that are related to the Registry. The *RegistryKeyChangeEvent* is an extrinsic event that can notify consumers that a certain Registry key has been changed. This could possibly be useful for protecting your Registry persistence mechanism.

© 2024 Jonathan Reiter

© SANS Institute 2024



Filtering Events Using WQL

Windows Query Language

Extrinsic events can be queried normally

Intrinsic events must be polled at some defined interval

```
Data Query
SELECT * FROM Win32_NTLogEvent WHERE logfile
= 'System' AND EventCode = '4625'

Event Query
SELECT * FROM __InstanceCreationEvent WITHIN
5 WHERE TargetInstace ISA "Win32_Process" AND
TargetInstance.Name = 'notepad.exe'

Schema Query
SELECT * FROM meta_class WHERE __this ISA
"Win32_Process"
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 105

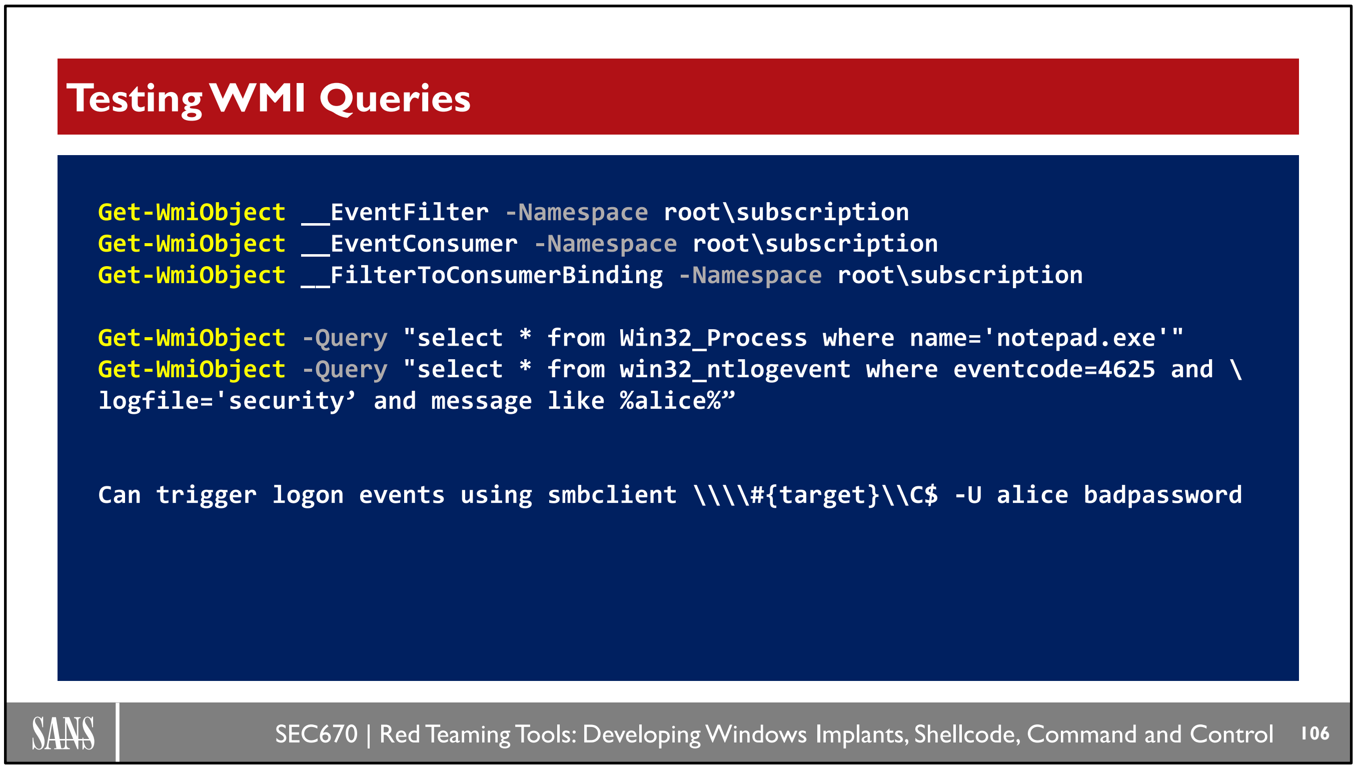
Filtering Events Using WQL
How are events filtered? The Windows Query Language, WQL, might look similar to that of the Structured Query Language, SQL, and that is because WQL is a subset of SQL. There are certain types of queries that we can create such as Data queries, Event queries, or Schema queries. We can start off our query using the *SELECT* statement. The *SELECT* statement will be specific to the type of query, Data, Event, or Schema. It takes quite a bit of research to determine what classes you will need to query. The determination must also be made if the query will be for intrinsic or extrinsic events. Remember, intrinsic events require polling like the Event query example on the slide and polls must be done on a frequency.

The event query example on the slide is polling for *__InstanceCreationEvents* every 5 seconds. The new instance that is created must be an instance of the *Win32_Process* class with a process name of notepad.exe. These types of queries can get very complex rather quickly but yet specific for your persistence needs.

The Data query example is making a query from the *Win32_NTLogEvent* class. The logfile being chosen is the System logfile and it is looking for a specific event code of 4625, the event for failed logon attempts.

The last example query is for a Schema query to better understand the *Win32_Process* class definition. Schema queries are used for understanding the definition of a class. This type of query is quite different because we do not rely on instances of a class being present. Schema queries only support the *SELECT ** statement unlike the other query types. Also, by saying *meta_class* we indicate that this is a schema query being executed.

© SANS Institute 2024

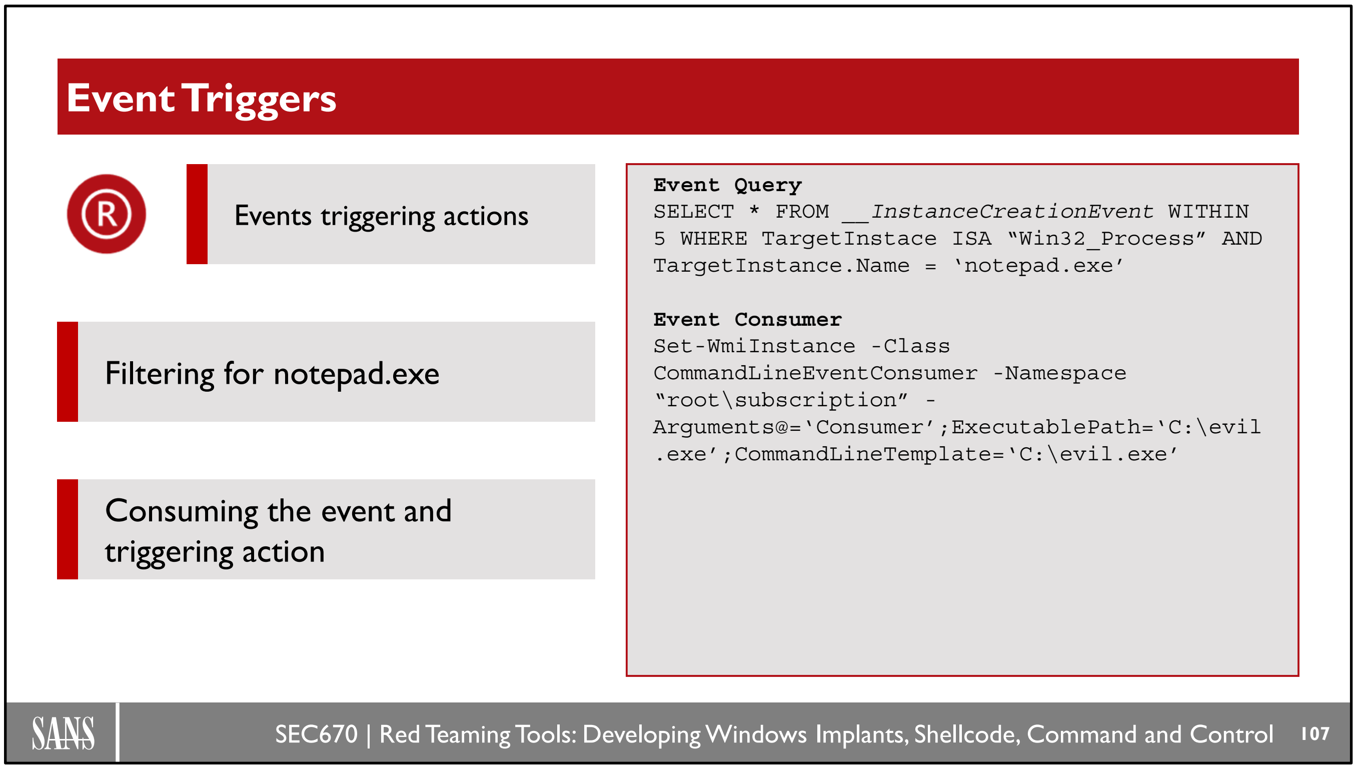


Testing WMI Queries

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

Get-WmiObject -Query "select * from Win32_Process where name='notepad.exe'"
Get-WmiObject -Query "select * from win32_ntlogevent where eventcode=4625 and \
logfile='security' and message like %alice%"


Can trigger logon events using smbclient \\\\#{target}\\C$ -U alice badpassword
```

Testing WMI Queries
PowerShell offers developers with perhaps the easiest method for testing WMI queries, thus saving us from having to develop what could be hundreds of lines of code for queries that do not work. Thanks to the *Get-WmiObject* cmdlet and we can pass a crafted query to the cmdlet's -Query parameter.

© SANS Institute 2024



Event Triggers

Events triggering actions

Filtering for notepad.exe

Consuming the event and triggering action

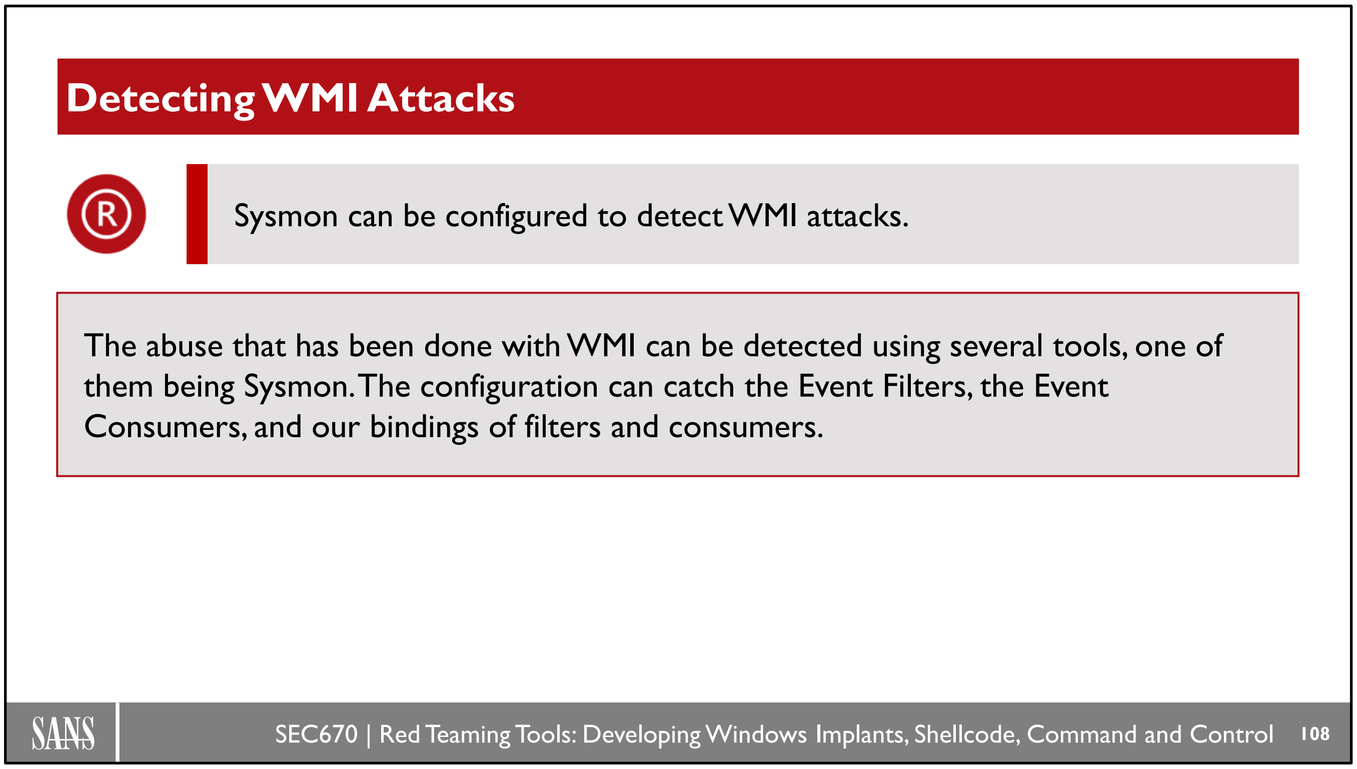
Event Query
```
SELECT * FROM __InstanceCreationEvent WITHIN
5 WHERE TargetInstace ISA "Win32_Process" AND
TargetInstance.Name = 'notepad.exe'
```

Event Consumer
```
Set-WmiInstance -Class
CommandLineEventConsumer -Namespace
"root\subscription" -
Arguments@='Consumer';ExecutablePath='C:\evil
.exe';CommandLineTemplate='C:\evil.exe'
```

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 107

Event Triggers
When it comes to event triggers, nothing has been triggered yet. The only thing done so far is filtering out a specific event. The next thing is to set up our Event Consumer that will contain the action we wish to execute. When registering the consumer, there is an Action block that must be filled out. Specifically, there are five Consumer classes that we can implement. *ActiveScriptEventConsumer*, *CommandLineEventConsumer*, *LogFileEventConsumer*, *NTEventLogEventConsumer*, *SMTPEventConsumer* are the five consumers that we could use, but perhaps the most interesting to us are the *CommandLineEventConsumer* and the *ActiveScriptEventConsumer* classes. The *CommandLineEvent* is an interesting class because it will create a process when the filter it is being bound to is triggered. One of the properties of the class is the *CommandLineTemplate* that specifies the binary that is to be executed. There is also the *ExecutablePath* property that is used to specify the absolute path for the executable. The filter on the slide polls from the *__InstanceCreationEvent* class for the notepad.exe process every five seconds. The instance that will consume that event is shown using PowerShell for easier readability. It will execute the evil.exe file whenever the notepad.exe process is created.

© SANS Institute 2024



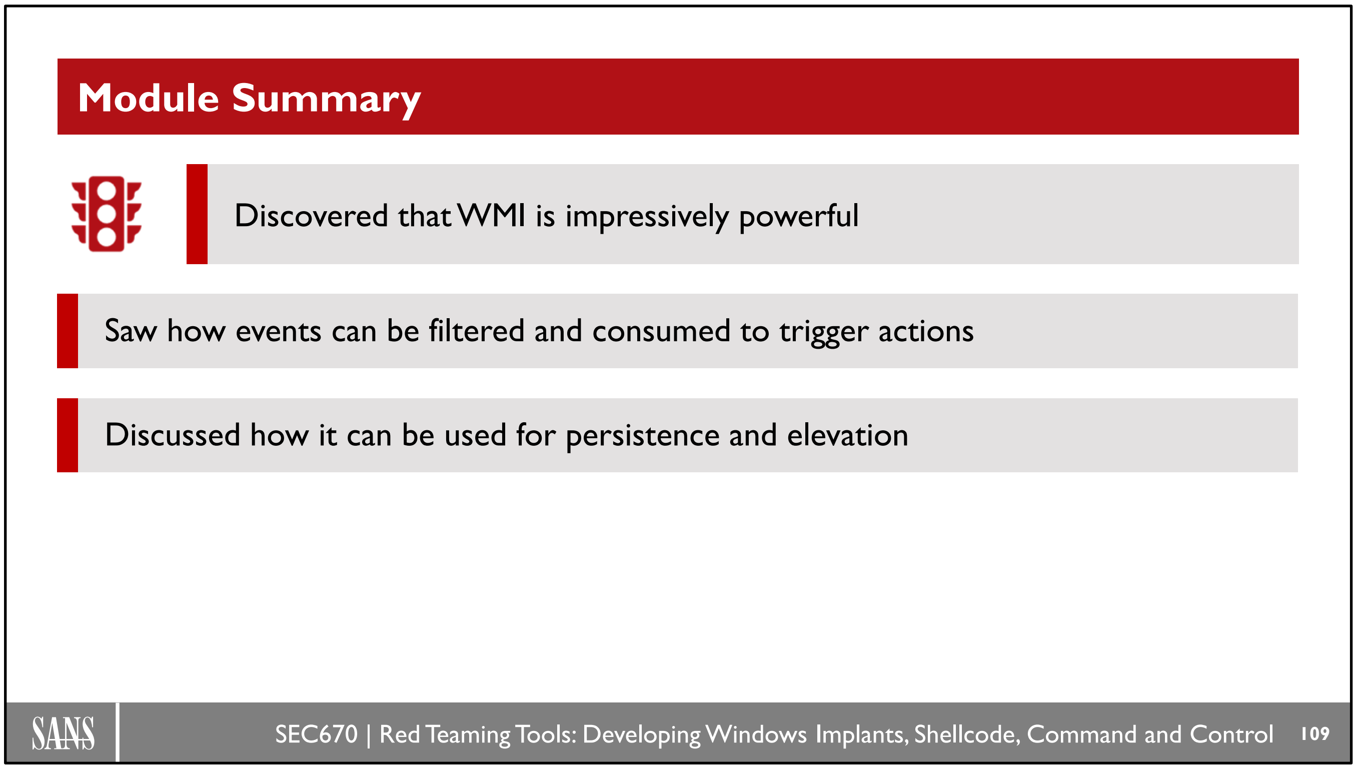
Detecting WMI Attacks

Sysmon can be configured to detect WMI attacks.

The abuse that has been done with WMI can be detected using several tools, one of them being Sysmon. The configuration can catch the Event Filters, the Event Consumers, and our bindings of filters and consumers.

Detecting WMI Attacks
There are several methods and tools for detecting these style of attacks, so be careful before you implement this method. One of the more popular tools today is one put out by Microsoft called Sysmon. It has proven to be very formidable, and when configured correctly, can make our job much more challenging. One of the configurations for Sysmon is for detecting WMI attacks, which should be on by default, making this easier for the defenders and harder for us attackers. As an example, Sysmon would let you know when a *WmiEventConsumer* event has been created. Despite the logs being created, the decision must be made to determine if the event was malicious. Logs are one thing, but the categorization of the event is another.

© SANS Institute 2024

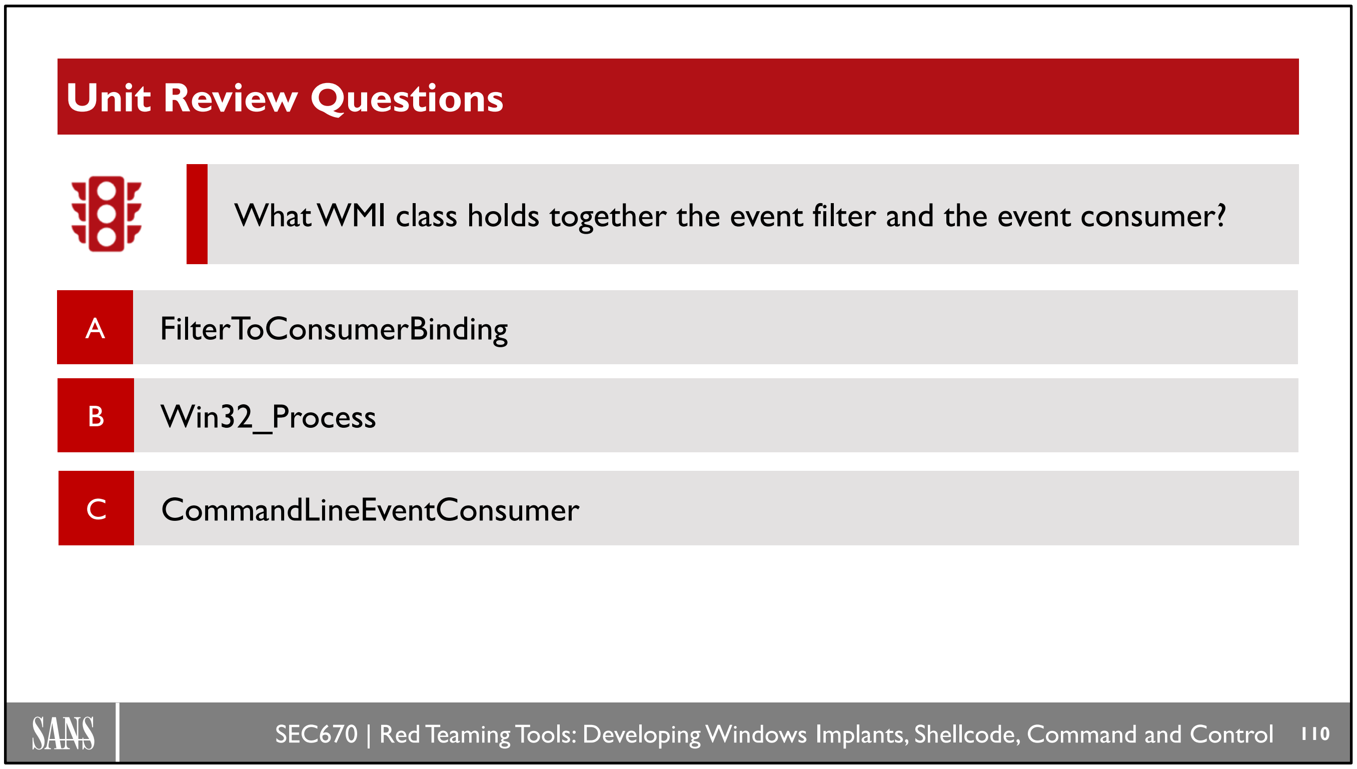


Module Summary

- Discovered that WMI is impressively powerful
- Saw how events can be filtered and consumed to trigger actions
- Discussed how it can be used for persistence and elevation

Module Summary
In this module, we took a deep dive into WMI, how it is designed, how events are created, how events can be consumed, and how we can trigger process creation when a certain event is kicked off. WMI, aside from being an amazing tool for sysadmins, is also an amazing tool for attackers. Not only can we persist, but we can elevate from Admin to SYSTEM at the same time since our action in our consumer will kick off as SYSTEM.

© SANS Institute 2024



Unit Review Questions

What WMI class holds together the event filter and the event consumer?

A FilterToConsumerBinding

B Win32_Process

C CommandLineEventConsumer

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 110

Unit Review Questions
Q: What WMI class holds together the event filter and the event consumer?

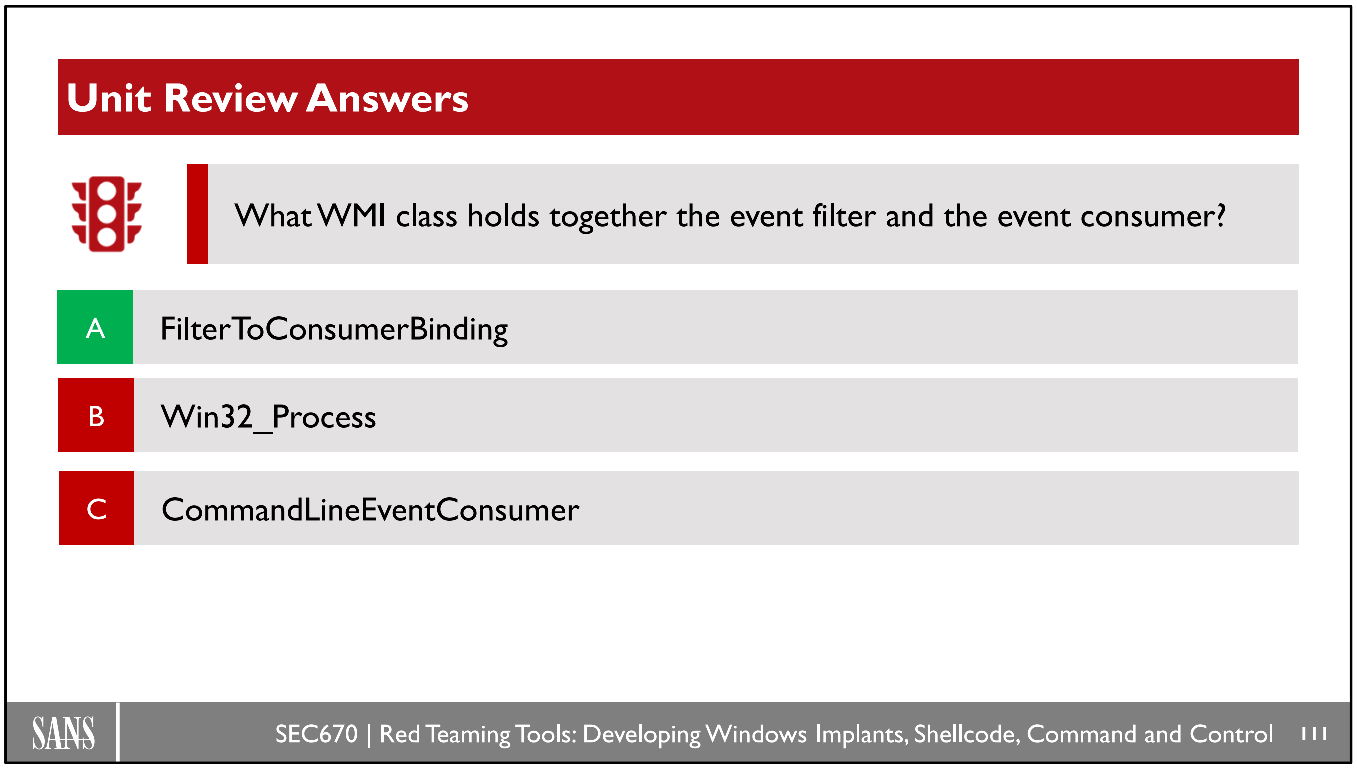
A: FilterToConsumerBinding

B: Win32_Process

C: CommandLineEventConsumer

© 2024 Jonathan Reiter

© SANS Institute 2024



Unit Review Answers

What WMI class holds together the event filter and the event consumer?

A	FilterToConsumerBinding
B	Win32_Process
C	CommandLineEventConsumer

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 111

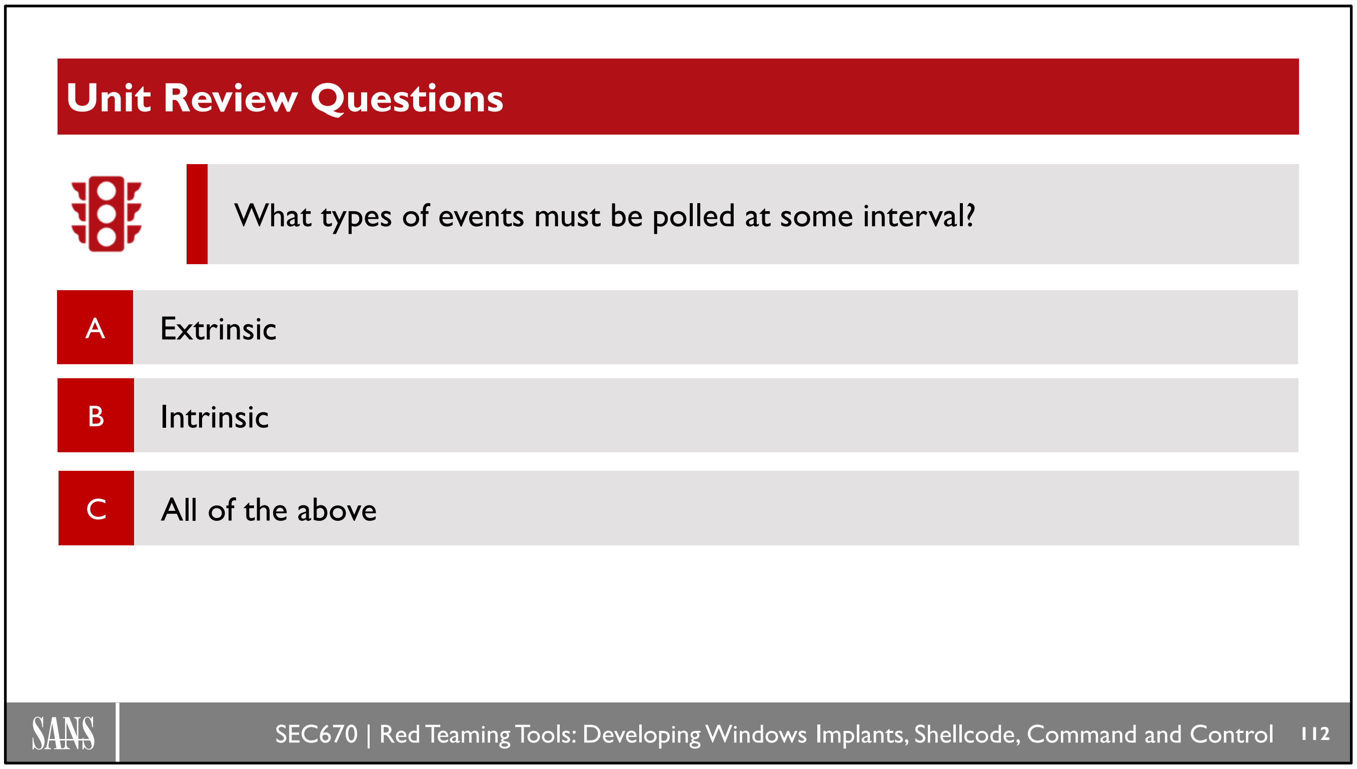
Unit Review Answers
Q: What WMI class holds together the event filter and the event consumer?

A: FilterToConsumerBinding

B: Win32_Process

C: CommandLineEventConsumer

© SANS Institute 2024



Unit Review Questions

What types of events must be polled at some interval?

A Extrinsic

B Intrinsic

C All of the above

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 112

Unit Review Questions
Q: What types of events must be polled at some interval?

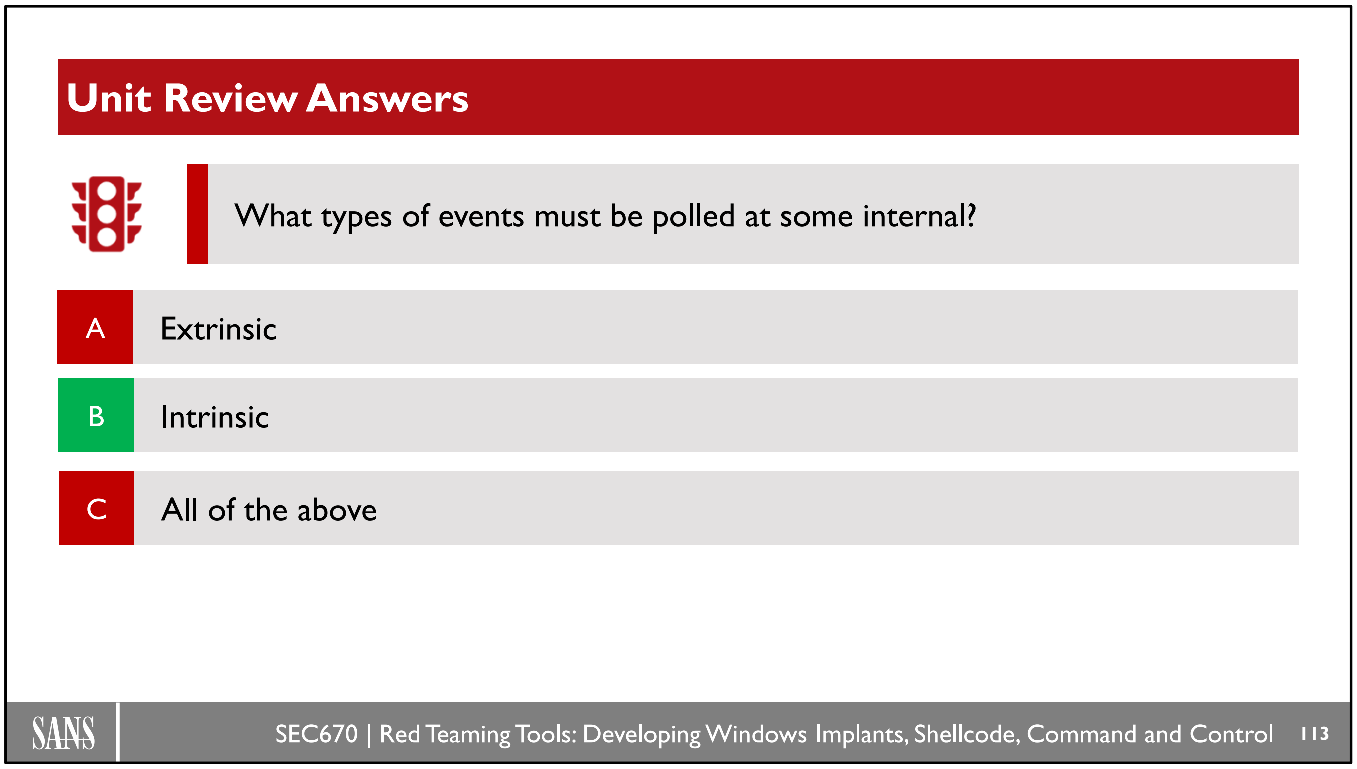
A: Extrinsic

B: Intrinsic

C: All of the above

© 2024 Jonathan Reiter

© SANS Institute 2024



Unit Review Answers

What types of events must be polled at some internal?

A	Extrinsic
🟩 B	Intrinsic
C	All of the above

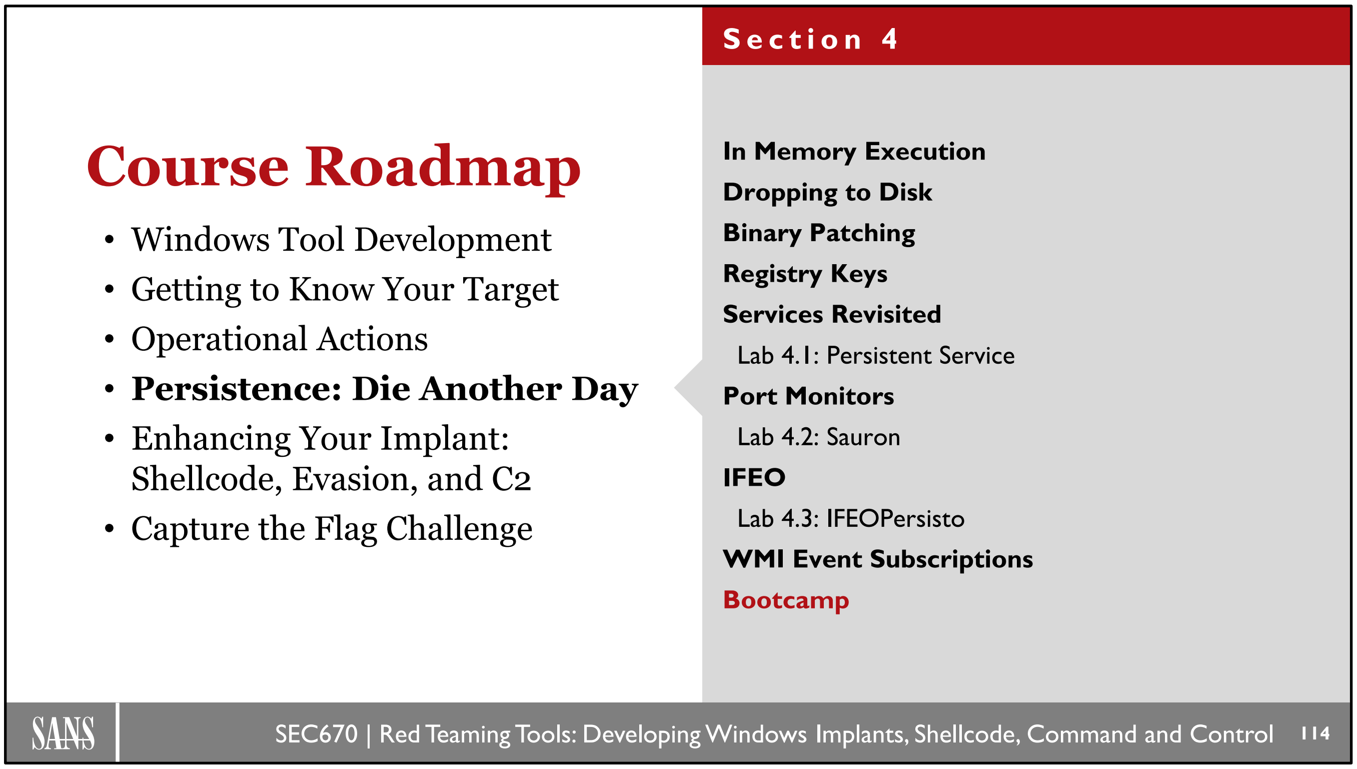
Unit Review Answers
Q: What types of events must be polled at some interval?

A: Extrinsic

B: Intrinsic

C: All of the above

© SANS Institute 2024



Course Roadmap

- Windows Tool Development
- Getting to Know Your Target
- Operational Actions
- **Persistence: Die Another Day**
- Enhancing Your Implant: Shellcode, Evasion, and C2
- Capture the Flag Challenge

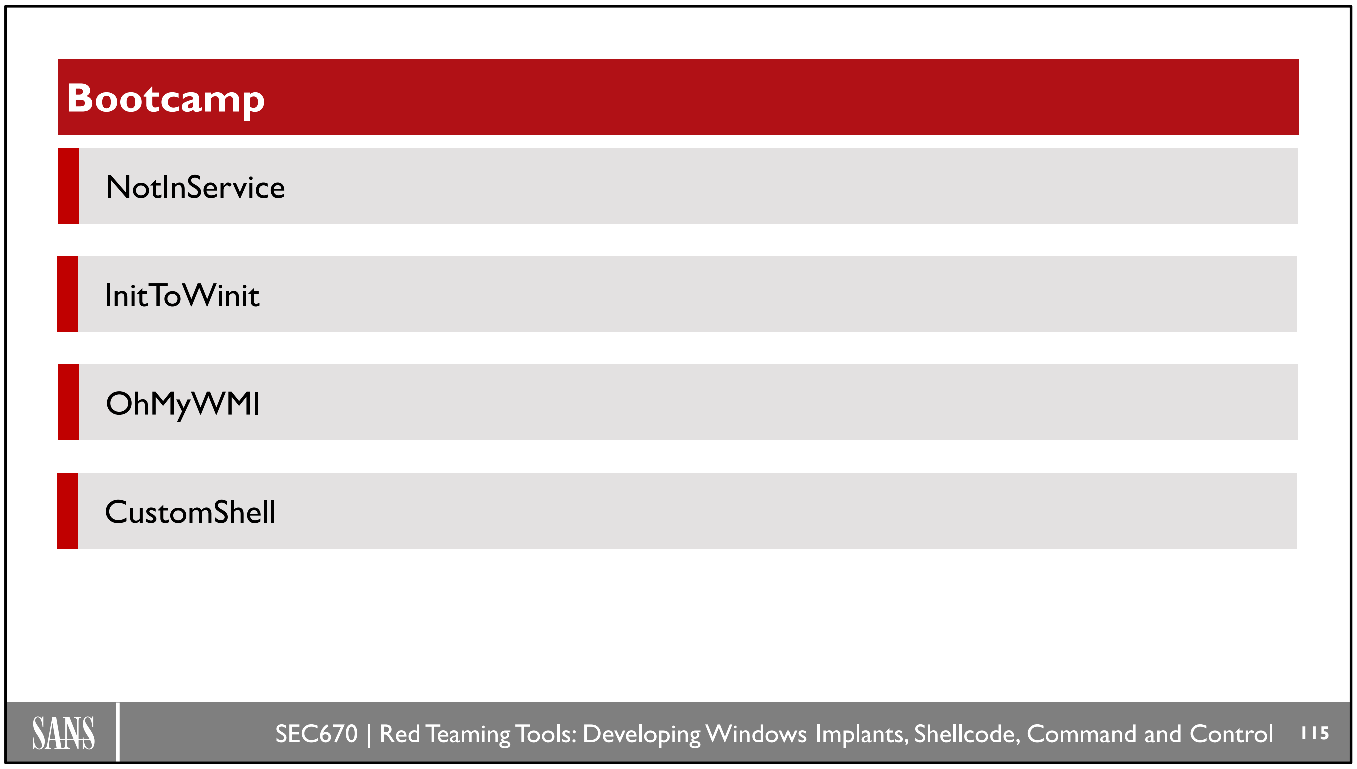
Section 4

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 114

Welcome to the bootcamp for Section 4! The challenges during the bootcamp will be very challenging but have fun with them and do not hesitate to reach out for assistance, guidance, or if you have any questions.

© 2024 Jonathan Reiter

© SANS Institute 2024



Bootcamp

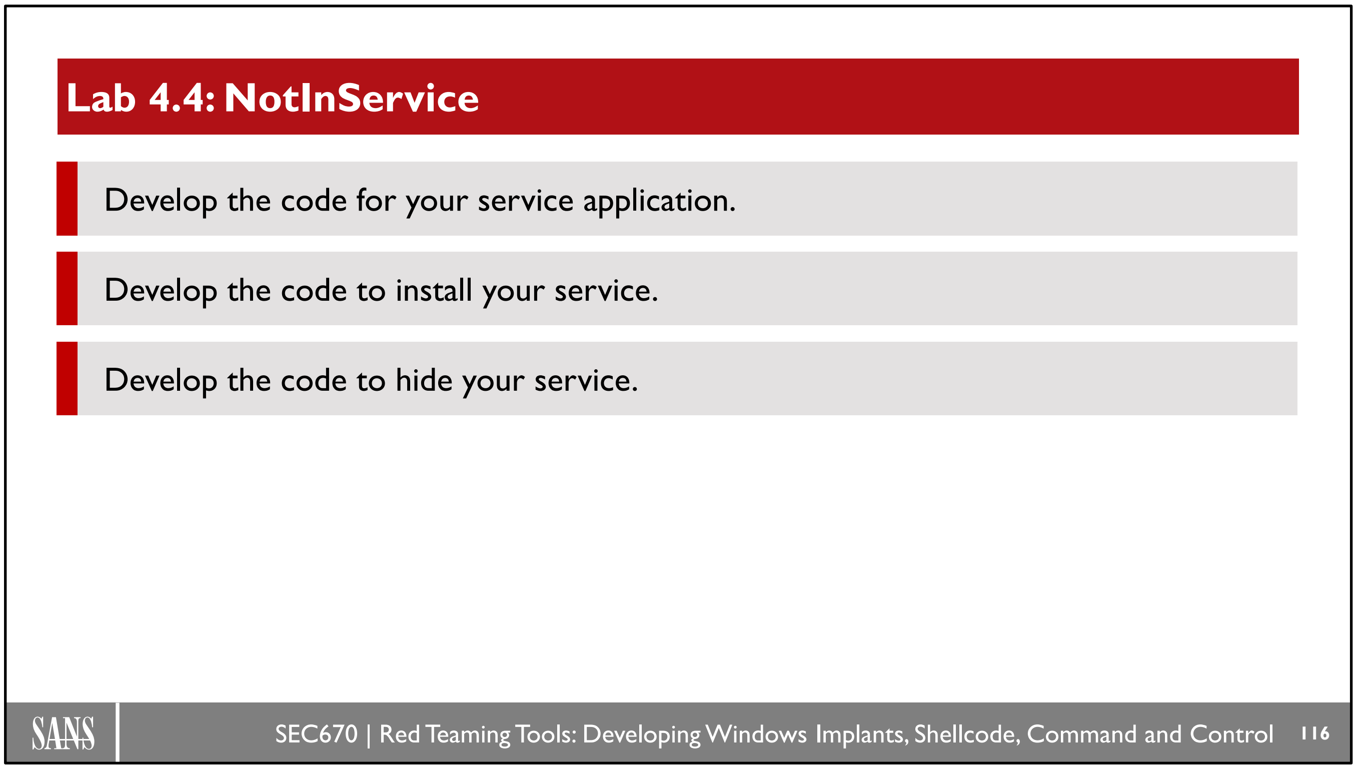
- NotInService
- InitToWinit
- OhMyWMI
- CustomShell

SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 115

Bootcamp
The bootcamp challenges today will really test your knowledge of Windows APIs. Services are important and as such, the first challenge is about services. The "Not in service" challenge requires you to create, install, and then hide your own service. The second challenge is about the *AppInit* method where you create the AppInit_DLLs key accordingly to get your DLL payload to execute without getting stuck in the infinite loop mentioned during that section. The third challenge is about using WMI to programmatically establish your foothold. You can use PowerShell for your initial testing or to get it going but the final product should be done in C or C++ as a compiled binary.

The last challenge of the day is to combine everything you have learned so far into a baby implant. The baby implant should be able to do full recon, inject into other processes, persist across reboots, create a log file, etc.

© SANS Institute 2024



Lab 4.4: NotInService

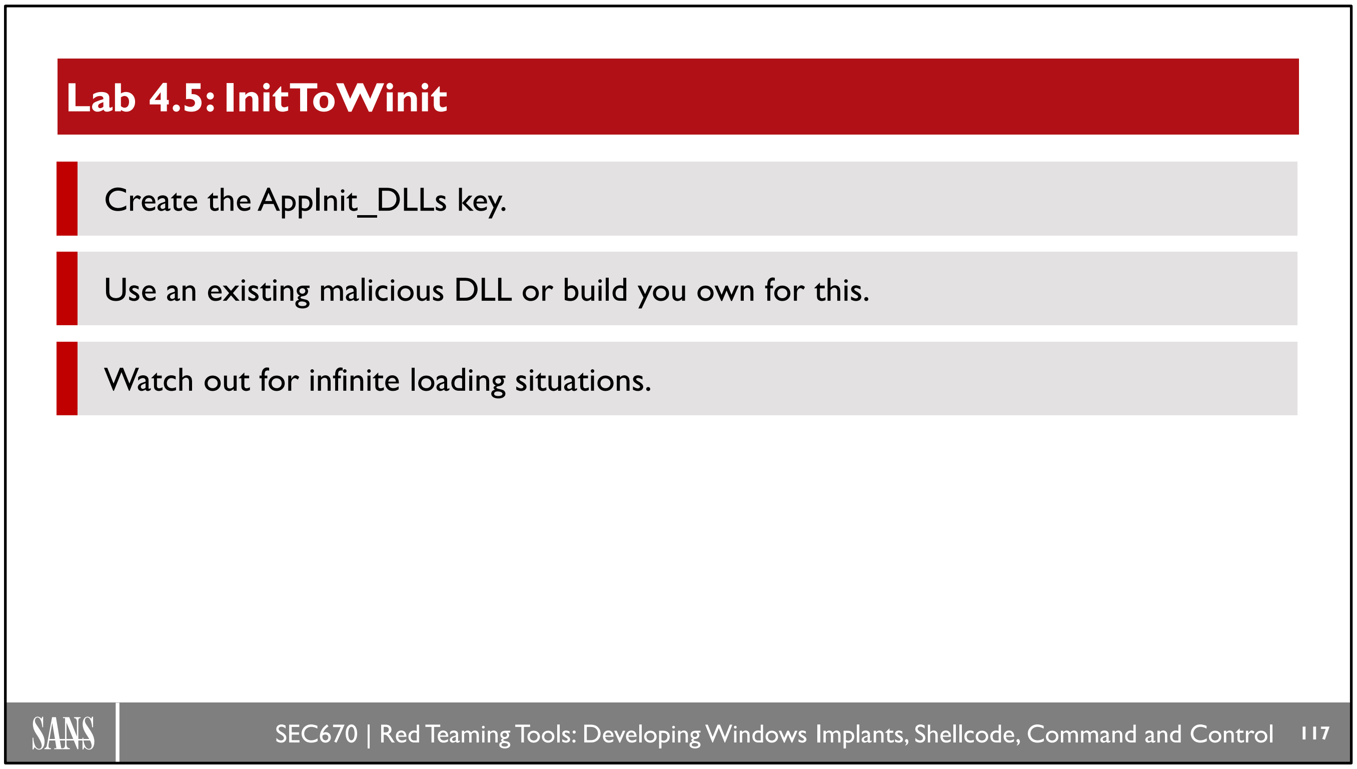
- Develop the code for your service application.
- Develop the code to install your service.
- Develop the code to hide your service.

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 116

Lab 4.4: NotInService
For “Not in service,” you will be developing a custom service to be installed on the target system. Once done, you will also create the code that will install your service. Finally, after the service has been installed, you will create the code to hide it. There are a couple of ways to go about accomplishing this challenge, but the most straightforward is to use the proper Win32 APIs. Your service does not have to perform anything specific as long as it runs successfully and hidden is the main point.

Have fun!

© SANS Institute 2024



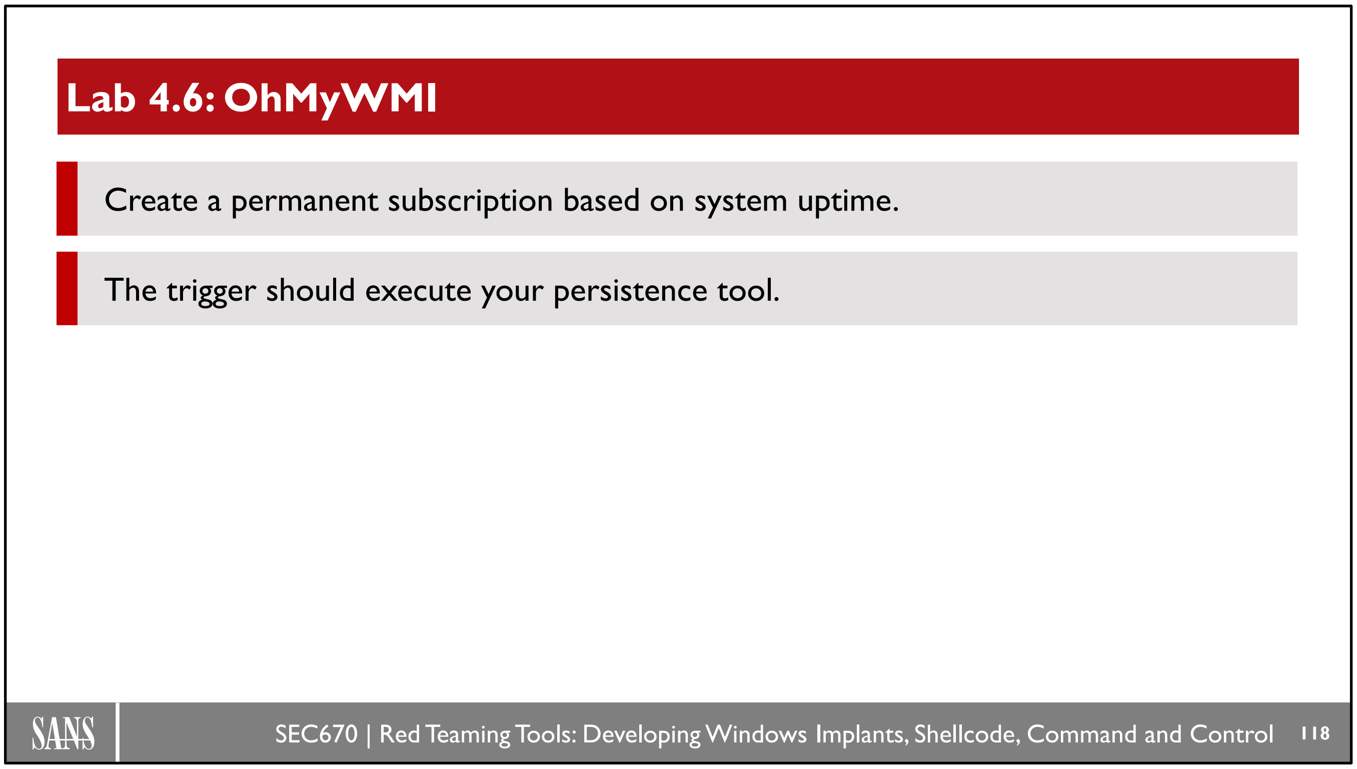
Lab 4.5: InitToWinit

- Create the AppInit_DLLs key.
- Use an existing malicious DLL or build you own for this.
- Watch out for infinite loading situations.

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 117

Lab 4.5: InitToWinit
You can get carried away with this challenge quickly if you are not paying close attention to the details. If you need to, refer back to the Registry keys section when we discussed this method and what it can be used for. The main goal here is to become familiar with the code to implement the method, while at the same time not getting caught in a never-ending loop situation. The DLL to use should be one that you have either created already or one that you create specifically for this lab. Avoid using DLLs made by msfvenom, but for a learning experience, try one out and see what happens.

© SANS Institute 2024



Lab 4.6: OhMyWMI

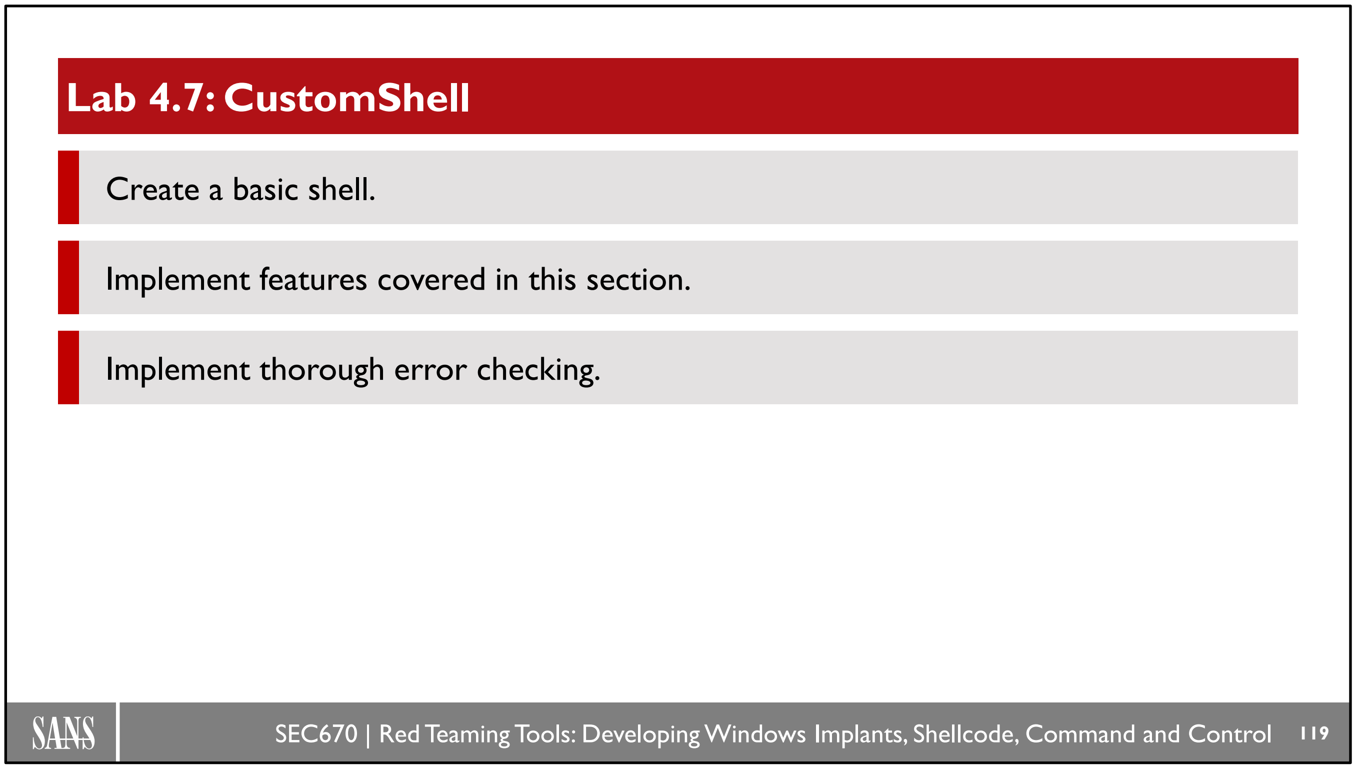
- Create a permanent subscription based on system uptime.
- The trigger should execute your persistence tool.

SANS SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 118

Lab 4.6: OhMyWMI
WMI was discussed in depth earlier in this section, and now it is time to put your foundations to the test. For this challenge, you are to create an installer that will establish your persistence based on system uptime. PowerShell will be your friend for this challenge as you should use it to test your queries. Once you have your query finalized, you can implement it programmatically in your code. If you finish early, you can implement more options that change what events are being subscribed to and what the trigger is.

© 2024 Jonathan Reiter

© SANS Institute 2024



Lab 4.7: CustomShell

- Create a basic shell.
- Implement features covered in this section.
- Implement thorough error checking.

SANS | SEC670 | Red Teaming Tools: Developing Windows Implants, Shellcode, Command and Control 119

Lab 4.7: CustomShell
Please refer to the eWorkbook for the details of this bootcamp challenge.