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

6	46	Laptop
Day Program	CPEs	Required

You Will Be Able To

- Create custom compiled Windows implants
- Collect target information
- Hide processes from user mode tools
- Hook and unhook functions for AV bypasses
- Generate and execute custom shellcode
- Escalate privileges from medium integrity levels to high (NT AUTHORITY/SYSTEM)
- Persist across reboots
- Beacon out to configured C2 infrastructure

Prerequisites

It is strongly recommended that students have experience with developing programs in C/C++ for either Linux or Windows platforms. Additionally, students should have C/C++ experience programming loops, conditional statements, and switch statements, creating functions and function pointers, and using pointers, linked lists, and type casting.

- Courses that lead into SEC670:
- SEC560: Network Penetration Testing and Ethical Hacking
- SEC660: Advanced Penetration Testing, Exploit Writing, and Ethical Hacking
- SEC760: Advanced Exploit Development for Penetration Testers

SEC670 gives alumni of these courses a more in-depth look at how the tools used in them operate. It also shows students how to make their own tools and add missing features that other compiled tools might not have. As an example, in SEC660, there is a brief mention of tampering with Windows AMSI and AMSI bypasses. SEC670 will take you behind the scenes where students will create their own, fully customized AMSI bypass.

Learning how to develop custom-compiled tools for Windows is a skillset that is not being taught by universities or other academic organizations and, as a result, the cybersecurity industry has a severe skills deficit that is limiting the overall capability of red team operations. Defense contractors and industries looking to hire Windows tools developers are facing a severe shortage of talent and are unable to further hone their defenses.

SEC670 is the first course of its kind, giving students hands-on lab experience creating custom-compiled programs specifically for Windows using the C/C++ programming languages. Students will learn the internal workings of existing offensive tools that offer capabilities such as privilege escalation, persistence, and collection by creating their own tools using Windows APIs. Windows defenses have become more robust, and cloud-connected AV solutions are making it more challenging to operate under the radar. In response, this course introduces students to techniques that real nation-state malware authors are currently implementing in their implants.

The course starts with an introduction to developing Windows Computer Network Operations (CNO) tools. You will explore current offensive and defensive tools like Moneta and PE-Sieve that are designed to detect malicious actions. Students will then quickly ramp up to creating their first compiled program. Students will move through the course learning how to obtain target information, what operational actions (such as injection and privilege escalation) can be carried out using this information, and how to take advantage and maintain system access through persistence. You will also learn how to take shellcode, encrypted or otherwise, and execute it in a process using the C programming language and leveraging compiler tricks. Finally, students will learn how to evade AV solutions by bypassing their function-hooking engine, patching key functions like AmsiScanBuffer and code caves. The course will even discuss scenarios where going after low-hanging fruit is preferred to dropping more complicated and sensitive implant capabilities.

SEC670 culminates with an immersive Capture-the-Flag event that will challenge students like no other event ever has. Students must leverage the tools and capabilities they have built during the week to solve complex challenges like getting information from a remote process memory. By the end of the course, students will have built a lightweight Windows implant that can enumerate the Windows Registry, files, folders, network connections, users, and processes; bypass UAC and AV products; escalate privileges; persist across reboots; inject into other processes; and hide from users and other tools.

Author Statement

Penetration testers, red team operators (RTO), exploit developers, and those in the Intel Community (IC) have all used amazing tools and frameworks to get their jobs done. These amazing tools have one thing in common: they were developed by an effective team or by one dedicated individual. The developers are the enablers of operations, and without them we would not be where we are today. Creating offensive tools is a broad task and can have many areas of focus. One particularly important area is building implants or agents that are dropped on a victim computer to establish that shell with an operator. This course will focus on building implants for Windows targets using the C/C++ programming languages. The course is heavy on labs and hands-on development, giving you ample time to fully grasp how Windows does things differently than other operating systems. By the end of the week, you should have a fully functioning Windows implant that you can continue to tweak well beyond the course.
—Jonathan Reiter

Section Descriptions

SECTION 1: Windows Tool Development

The course begins by introducing students to Windows Internals, starting with a high-level overview and gradually diving deeper into some of the core mechanisms that make the Operating System tick. We will discuss key differences between offensive and defensive tools as well as the need for them. Equipped with a solid understanding of Windows programming, students can choose to create offensive or defensive tools for Windows. However, the course will only focus on creating offensive cyber capabilities using the C/C++ programming languages. Key differences between Linux and Windows are important and help to ease the transition from Linux to Windows. The C programming language has core data types, but Windows introduces its own data types, which are presented in this course section. Students will be introduced to calling conventions and how Windows brought its own into the programming arena. At the end of the section, we will present the Windows Application Programming Interface (API) more formally by having students use key Windows APIs in a lab to bring it all together.

TOPICS: Developing Offensive Tools; Developing Defensive Tools; Setting up Your Development Environment; Similarities and Difference with *Nix Dev; Windows Data Types; Call Me Maybe; SAL Annotations; Windows API

SECTION 2: Getting to Know Your Target

Section 2 introduces students to the art of on-target reconnaissance. One of the first actions red team operators might take after gaining initial access is to conduct in-depth enumeration, or recon, against the target. This step is often overlooked, since it can take a while to perform and it is not as glamorous as using a 0-day to exploit something. However, it is vital to know what type of environment you are on and what you can do next. Students will learn how to programmatically survey the lay of the land using a detailed approach, then prepare a final product to obtain information about the operating system version build, patches, and processes, installed applications, the filesystem, users and groups, the network, services, tasks, the registry, and more.

TOPICS: Gathering Operating System Information; Service Packs/Hotfixes/Patches; Process Enumeration; Installed Software; Directory Walks; User Information; Services and Tasks; Network Information; Registry Information

Who Should Attend

- Red Team operators
- Exploit developers
- Penetration testers
- Linux computer network operations developers
- Windows developers
- AV/EDR developers

SECTION 3: Operational Actions

Section 3 focuses on actions that can be taken after initial access. Red team operators typically leverage process injection to execute desired actions. This section will teach students how to programmatically implement those capabilities, starting with a deep dive into the format of the Portable Executable (PE) header. Students will learn how to parse important sections of the PE header, which is a valuable skill that will allow them to create their own version of Windows APIs such as GetProcAddress. After mastering the PE format, we will look at the internals of Threads, their structure, and how they are created. The section will also explore how Asynchronous Procedure Calls can be queued to a Thread to aid in process injection. We will cover several process injection methods, including the classic DLL injection where we force a Thread to load our malicious DLL in a target process. Another action is to escalate privileges to enable an operator to be more effective on target, so students will programmatically create privilege escalation modules for their implant and test them out on their target system.

TOPICS: Understanding the PE Format; Creating Custom Equivalents to Win32 APIs; Exploring Thread Internals; Exploring Process Injection Methods; Programmatically Interacting with Remote Processes; Creating Custom Tools for Privilege Escalation

SECTION 4: Persistence: Die Another Day

Section 4 focuses exclusively on various methods to achieve persistence by surviving reboots. Gaining initial access is a great place to start, but steps must be taken to maintain that access in the event something unforeseen happens like a loss of power, unscheduled reboots, the initial access process dying, etc. Typically, operators will use persistence methods baked into existing tools or frameworks, but those tools had to have been developed at some point. During bootcamp challenges in this course section, students will programmatically implement persistence tools and then test the compiled products against their Windows 10 Test VM to see if access is maintained after a reboot.

TOPICS: In-memory Execution; Dropping to Disk; Binary Patching; Registry Keys; Services Revisited for Persistence; Port Monitors; Image File Execution Options

SECTION 5: Enhancing Your Implant: Shellcode, Evasion, and C2

This course section will cover more advanced techniques that developers must master to be successful. One great feature for implants is the ability to execute position-independent shellcode. The shellcode can come from existing tools like msfvenom, donut, or shellter, or it can be hand-crafted using the C programming language. Students will explore in detail how to execute shellcode locally in their own process as well as remotely across process address space boundaries. Since shellcode can be caught quickly unless it is obfuscated or encrypted, students will learn how to decrypt shellcode right before execution. Another implant capability is evading AV products. Evasion can be done via several methods, but this course section will focus on unhooking functions, restoring system calls, and implementing our own hooks. Executing shellcode and lowering AV detection rates are great, but they can be useless if there is no method of collecting information from the target machine. Students will explore how implants can send back information to an operator for offline analysis and accept tasking for what to do next.

TOPICS: Shellcode Generation and Execution; Hiding Processes; Doppelganging; Unhooking Hooks; Code Caves; AV Product Bypasses; Calling Home; Writing Shellcode in C

SECTION 6: Capture the Flag

This Capture-the-Flag event involves solving an array of hands-on challenges that mimic real-world events. Students play the role of senior developers as part of a nation-state team to create cyber capabilities to leverage against their targets. The challenges will require students to apply everything that they learned during the week. However, that foundation of skills will be just the starting point. Since not every target will be the same, students will have to take their initial access to a target and then expand on that access using only tools that have been custom developed for that specific target.

TOPICS: Target Survey and Recon; AV Bypass; Privilege Escalation; Persistence; Hooking; Code Injection