Threat Intelligence

SharPersist: Windows Persistence Toolkit in C#

September 3, 2019

Mandiant

Written by: Brett Hawkins

Background

PowerShell has been used by the offensive community for several years now but recent advances in the defensive security industry are causing offensive toolkits to migrate from PowerShell to reflective C# to evade modern security products. Some of these advancements include Script Block Logging, Antimalware Scripting Interface (AMSI), and the development of signatures for malicious PowerShell activity by third-party security vendors. Several public C# toolkits such as Seatbelt, SharpUp and SharpView have been released to assist with tasks in various phases of the attack lifecycle. One phase of the attack lifecycle that has been missing a C# toolkit is

Team called SharPersist.

Windows Persistence

During a Red Team engagement, a lot of time and effort is spent gaining initial access to an organization, so it is vital that the access is maintained in a reliable manner. Therefore, persistence is a key component in the attack lifecycle, shown in Figure 1.

Figure 1: FireEye Attack Lifecycle Diagram

Once an attacker establishes persistence on a system, the attacker will have continual access to the system after any power loss, reboots, or network interference. This allows an attacker to lay dormant on a network for extended periods of time, whether it be weeks, months, or even years. There are two key components of establishing persistence: the persistence implant and the persistence trigger, shown in Figure 2. The persistence implant is the malicious payload, such as an executable (EXE), HTML Application (HTA), dynamic link library (DLL), or some other form of code execution. The persistence trigger is what will cause the payload to execute, such as a scheduled task or Windows service. There are several known persistence triggers that can be used on Windows, such as Windows services, scheduled tasks, registry, and

ATT&CK persistence page.

Figure 2: Persistence equation

SharPersist Overview

SharPersist was created in order to assist with establishing persistence on Windows operating systems using a multitude of different techniques. It is a command line tool written in C# which can be reflectively loaded with Cobalt Strike's "execute-assembly" functionality or any other framework that supports the reflective loading of .NET assemblies. SharPersist was designed to be modular to allow new persistence techniques to be added in the future. There are also several items related to tradecraft that have been built-in to the tool and its supported persistence techniques, such as file time stomping and running applications minimized or hidden.

SharPersist and all associated usage documentation can be found at the SharPersist Mandiant GitHub page.

SharPersist Persistence Techniques

There are several persistence techniques that are supported in SharPersist at the time of this blog post. A

Technique	**Description**	**Technique Switch Name (** **t)**
KeePass	Backdoor KeePass configuration file	keepass
New Scheduled Task	Creates new scheduled task	schtask
New Windows Service	Creates new Windows service	service
Registry	Registry key/value creation/modification	reg
Scheduled Task Backdoor	Backdoors existing scheduled task with additional action	schtaskbackdo
Startup Folder	Creates LNK file in user startup folder	startupfolder

Figure 3: Table of supported persistence techniques

SharPersist Examples

On the SharPersist GitHub, there is full documentation on usage and examples for each persistence technique. A few of the techniques will be highlighted below.

Registry Persistence

The first technique that will be highlighted is the registry persistence. A full listing of the supported registry keys in SharPersist is shown in Figure 4.

Registry Key Code (-k)	**Registry Key**
hklmrun	HKLM\Software\Microsoft\Windows\C
hklmrunonce	HKLM\Software\Microsoft\Windows\C

userinit	HKLM\SOFTWARE\Microsoft\Window NT\CurrentVersion\Winlogon
hkcurun	HKCU\Software\Microsoft\Windows\C
hkcurunonce	HKCU\Software\Microsoft\Windows\C
logonscript	HKCU\Environment
stickynotes	HKCU\Software\Microsoft\Windows\C

Figure 4: Supported registry keys table

In the following example, we will be performing a validation of our arguments and then will add registry persistence. Performing a validation before adding the persistence is a best practice, as it will make sure that you have the correct arguments, and other safety checks before actually adding the respective persistence technique. The example shown in Figure 5 creates a registry value named “Test” with the value “cmd.exe /c calc.exe” in the “HKCU\Software\Microsoft\Windows\CurrentVersion\Run” registry key.

Figure 5: Adding registry persistence

Figure 6. We are removing the "Test" registry value that was created previously, and then we are listing all registry values in "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" to validate that it was removed.

Figure 6: Removing registry persistence

Startup Folder Persistence

The second persistence technique that will be highlighted is the startup folder persistence technique. In this example, we are creating an LNK file called "Test.lnk" that will be placed in the current user's startup folder and will execute "cmd.exe /c calc.exe", shown in Figure 7.

Figure 7: Performing dry-run and adding startup folder persistence

The startup folder persistence can then be removed, again using the "-m remove" argument, as shown in Figure 8. This will remove the LNK file from the current user's startup folder.

Figure 8: Removing startup folder persistence

The last technique highlighted here is the scheduled task backdoor persistence. Scheduled tasks can be configured to execute multiple actions at a time, and this technique will backdoor an existing scheduled task by adding an additional action. The first thing we need to do is look for a scheduled task to backdoor. In this case, we will be looking for scheduled tasks that run at logon, as shown in Figure 9.

Figure 9: Listing scheduled tasks that run at logon

Once we have a scheduled task that we want to backdoor, we can perform a dry run to ensure the command will successfully work and then actually execute the command as shown in Figure 10.

Figure 10: Performing dry run and adding scheduled task backdoor persistence

As you can see in Figure 11, the scheduled task is now backdoored with our malicious action.

Figure 11: Listing backdoored scheduled task

Figure 12: Removing backdoored scheduled task action

Conclusion

Using reflective C# to assist in various phases of the attack lifecycle is a necessity in the offensive community and persistence is no exception. Windows provides multiple techniques for persistence and there will continue to be more discovered and used by security professionals and adversaries alike.

This tool is intended to aid security professionals in the persistence phase of the attack lifecycle. By releasing SharPersist, we at FireEye Mandiant hope to bring awareness to the various persistence techniques that are available in Windows and the ability to use these persistence techniques with C# rather than PowerShell.

Posted in Threat Intelligence—Security & Identity

Related articles

Threat Intelligence

Hybrid Russian Espionage and Influence Campaign Aims to Compromise Ukrainian Military Recruits and Deliver Anti-Mobilization Narratives

By Google Threat Intelligence Group • 10-minute read

Threat Intelligence

Investigating FortiManager Zero-Day Exploitation (CVE-2024-47575)

By Mandiant • 19-minute read

Threat Intelligence

How Low Can You Go? An Analysis of 2023 Time-to-Exploit Trends

By Mandiant • 10-minute read

Threat Intelligence

capa Explorer Web: A Web-Based Tool for Program Capability Analysis

By Mandiant • 6-minute read