PrivEsc: DLL Hijacking

Published on January 3, 2016 By HollyGraceful on Build Security

I posted earlier about Privilege Escalation through Unquoted Service Paths and how it's now rare to be able to exploit this in the real world due to the protected nature of the C:\Program Files and C:\Windows directories. It's still possible to exploit this vulnerability, but only when the service executable is installed outside of these protect directories which in my experience is rare. Writing that post though got me thinking about another method of privilege escalation which I think is a little more common to see – DLL Hijacking.

Again this vulnerability requires the software to be executed from a directory other than the protected directories, however it's not uncommon to see this being the case – either software has been installed to a non-default directory or alternatively software is being executed via a network share. I saw a lot of posts online that talk about this issue but nothing that really goes into the details leading up to proving the vulnerability, as would be expected by a security tester.

The vulnerability is a simple one, it abuses how Windows loads DLLs when EXEs are executed and how default folder permissions work on Windows. By default on Windows systems an authenticated user may add a file into a directory that they don't own, but cannot

overwrite or delete existing files. This means that if, for example, an Administrator has deployed a piece of software across a network either through deploying it to the local machine and setting up an autorun or by installing the software to a share and getting users to execute the software from that share then an attacker can cause a malicious DLL to be loaded when the software executes and this allows for arbitrary code execute at the privilege level of the executing user. Perfect for privilege escalation.

The way that Windows loads DLLs then, is to search the following directories in this order:

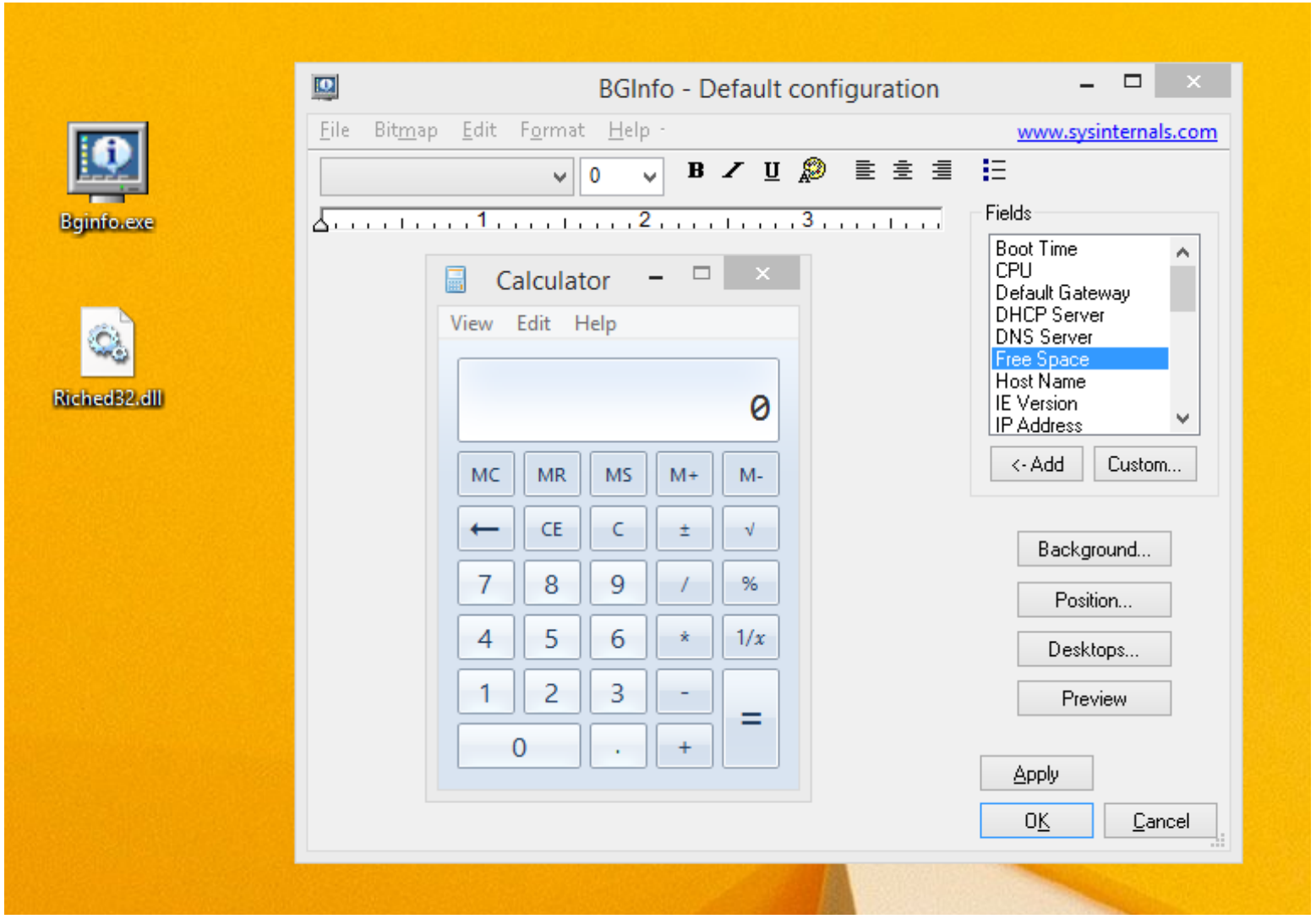
– The directory from which the application loaded
– C:\Windows\System32
– C:\Windows\System
– C:\Windows
– The current working directory
– Directories in the system PATH environment variable
– Directories in the user PATH environment variable

As the application folder has a higher priority than the system folders, if an application is installed with the intention that it uses system DLLs an attacker may be able to plant a DLL in the installation directory (which is possible by default if it is installed outside of the protected C:\Program Files directory) and gain code execution.

A good tool to look for potentially vulnerable software is Autoruns (part of Sysinternals and available here), this will show any software that loads by default when the system loads and it will also expose the file path the software runs from to make it easier to determine if it's outside the protected structure. If you find a vulnerable piece of software then next stage is to determine which DLLs it loads by default and again there's a tool for this called Procmon (another Sysinternals and available here).

So if you find an application installed outside of the protected directory, for example such as running from a share or picked up by Autoruns, you can check for the DLLs it loads through ProcMon – you're looking for any QueryOpen operations that result in NAME_NOT_FOUND, that way you can hijack the loading by planting a DLL in that position so that it can find it!

As an example, the common tool bginfo requires the DLL Riched32.dll. Therefore if I plant a DLL called Riched32.dll in the same directory as bginfo.exe when that tool executes so will my malicious code. Therefore I created a proof-of-concept DLL which will execute calc.exe to prove that code execution is possible. Here's how it looks when bginfo.exe is loaded on my Windows 8.1 Pro machine:



The “malicious” Riched32.dll I created was done using the following code:

```
#include <windows.h>
int fireLazor()
{
 WinExec("calc", 0);
 return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL,DWORD fdwReason, LPVOID lpvReserved)
{
 fireLazor();
 return 0;
}
```

This can be compiled on Linux (I did this using Fedora) by saving the code in a file called main.cpp and compiling with the following commands:

```
i686-w64-mingw32-g++ -c -DBUILDING_EXAMPLE_DLL main.cpp
i686-w64-mingw32-g++ -shared -o main.dll main.o -Wl,--out-implib,main.a
```

However you'll need Mingw32 installed to do this, which isn't there by default so add it with:

```
sudo dnf install mingw32-gcc-c++.x86_64
You'll probably also need:
sudo dnf install mingw32-winpthreads*
```

The compilation commands will create a file called main.dll and you can simply rename it as required. When the DLL is loaded it will cause calc.exe to execute, proving that command execute was possible and that the vulnerability is legitimate and not a false positive.

Remediation

Perhaps the simplest remediation steps would be simply to ensure that all installed software goes into the protected directory c:\Program Files or c:\Program Files (x86). If software cannot be installed into these locations then the next easiest thing is to ensure that only Administrative users have “create” or “write” permissions to the installation directory to prevent an attacker from deploying a malicious DLL and thereby breaking the exploitation.

Tags: PrivEsc Privilege Escalation

PrivEsc: Insecure Service Permissions

PrivEsc: Unquoted Service Path

Theme Options

Default
Stereotype
Alternate
Beta

Tweets by @HollyGraceful

Tweets by @HollyGraceful

HollyGraceful Retweeted

FROVO
@fro_vo

POCAHONTAS: can you paint with all the colors of the wind

ME: *literally wrestling neil degrasse tyson to the ground* jus- just let her sing neil

4 May 2018

HollyGraceful
@HollyGraceful

Lying in the tattoo place again; hopefully only a few more hours and then the whole piece will be done!

8 May 2018

HollyGraceful
@HollyGraceful

Replying to @HollyGraceful

“Holly, you can’t render 4K footage on a MacBook Air from 2013.”
“WHAT?!”
“I SAID YOU CAN’T RENDER 4K FOOTAGE IN 2013!”

Embed View on Twitter