Product Solutions Resources Open Source Enterprise Pricing

Sign in Sign up

connormcgarr / **LittleCorporal** Public

Notifications Fork 61 Star 226

Code Issues Pull requests Actions Projects Security Insights

main

Code

45 Commits

LittleCorporal

README.md

README

LittleCorporal

LittleCorporal: A C# Automated Maldoc Generator

```
C:\LittleCorporal\bin\Release>LittleCorporal.exe C:\beacon.bin explo
.____    .__  __    __  .__         _________
|    |   |__|/  |__/  |_|  |   ____ \_   ___ \  _________________  .
|    |   |  \   __\   __\  | _/ __ \/    \  \/ /  _ \_  __ \____ \ /
|    |___|  ||  |  |  | |  |_\  ___/\     \___(  <_> ) | \/  |_> >
| _______ \__||__|  |__| |____/\___  >\______  /\____/|__|  |   __/
        \/                        \/        \/             |__|


             ________
            /        \
         __ /       (o)\__
        /     ______\   \                   I am the strongest debat
        | ____/__  __\____|                 I let myself be attacked
           [  --~~--  ]
            | (  L   )|
      ___----\  __  /----___
     /   |  < \____/ >   |   \
    /    |   < \--/ >    |    \
    ||||||    \ \/ /     ||||||
    |          \  /   o       |
    |     |     \/   === |    |
    |     |      |o  ||| |    |
    |     \______|   +#* |    |
    |            |o      |    |
     \           |      /     /
     |\__________|o    /     /
     |           |    /     /

[+] Parsed Arguments:
   [>] Shellcode Path: C:\beacon.bin
   [>] Target Process: explorer.exe

[+] Embedded shellcode in Loader.cs!
[+] Generated C# Loader artifact!
[+] Ran the .NET assembly through Donut!
[+] Donut artifact is located at: C:\LittleCorporal\Artifacts\payloa
[+] Path to Word document: C:\LittleCorporal\Artifacts\Form.doc
```

PLEASE READ THE ENTIRE README BEFORE USAGE!

How It Works?

About

LittleCorporal: A C# Automated Maldoc Generator

Readme

Activity

226 stars

4 watching

61 forks

Report repository

Releases

No releases published

Packages

No packages published

Contributors 2

Languages

● C# 100.0%

LittleCorporal accepts a user-supplied argument for a process to inject into on a *remote* machine, in which you plan to execute the malicious Word document on, and also accepts a path to a local shellcode file stored in `.bin` format - such as a Beacon Stageless shellcode blob on the machine you are running LittleCorporal from. So, if you would like to use the maldoc generated from this project, you will need to specify an already running process on the machine you intend to run the maldoc on (be it the local machine or a different machine. `explorer.exe` is always going to have one instance, so use this if you do not care about which process you inject into).

LittleCorporal embeds the shellcode and the target process name into `Loader.cs`, compiles `Loader.cs` on the fly into a .NET `.exe` artifact, and then utilizes thread hijacking to perform remote process injection. The .NET `.exe` artifact, which is the thread hijacking loader, is sent through Donut to generate position independent shellcode, which will execute the `.exe`. The shellcode generated by Donut is then base64 encoded, a Word document is generated, and the final Donut blob is stored in an InlineShape.AlternativeText Word property, which is able to hold the entire payload. This is done by inserting an image (currently a blank image, giving the document a "blank" look) property into the Word document, as alternative text on the image, which contains the payload. LittleCorporal then leverages a VBA "template", contained in this project as a text file, and injects this Macro into the newly generated Word document. The Macro is named `autoopen`, so it opens upon the document opening, and then is configured to extract the value of the alternative text of the previously generated image, which contains the final payload, base64 decodes it, and finally uses Windows API calls, in VBA, to perform *local* injection into Word. In essence, this project uses a simple "loader" in VBA to perform local injection into Word, which is a bit less scrutinized than remote process injection, and then uses execution from the simple local injection injection to execute the Donut shellcode, which is *another* loader that performs thread hijacking for the final remote process injection of the user-supplied shellcode into the user-specified process. This is all done in an automated fashion, including generation of the Word document.

Requirements and Limitations

1. LittleCorporal assembles the .NET thread hijacking artifact using .NET v2. Please make sure .NET v2 is installed on the machine you are generating the Word document with (make sure `C:\Windows\Microsoft.NET\Framework64\v2.0.50727\` exists) and on the target machine you plan to run the document on.
2. You must also have Office installed on the machine you plan on running `LittleCorporal.exe` on. This is because LittleCorporal needs to interact via COM with Word. Please also note that the generated Word doc pulls down a blank image from the internet - meaning you will need to have internet access on the computer you run LittleCorproal with *and* on the remote machiny you plan to detonate the Word doc on.
3. Unfortunately, LittleCorporal can *only* be used with 64-bit versions of Word. Why is this? Only a 64-bit system can identify structures used in the thread hijacking loader capability - such as a 64-bit `CONTEXT` structure, which has 64-bit specific data types. This is because thread hijacking is currently only supported on 64-bit systems, as it requires custom shellcode which adheres specifically to the `__fastcall` calling convention. I do not plan to implement a 32-bit version of the thread hijacking capability, but if there is eventually a pull request for 32-bit support, as many installations of Microsoft Word are 32-bit, I would not totally count this out of the realm of possibility. It currently shouldn't be a big fix from a logical perspective, this issue becomes the burden and hassle of maintaining track of the stack when moving from `__fastcall` to `__stdcall` and computing new offsets. This isn't complex from a technical perspective, but is a bit arduous.
4. The thread hijacking code first performs a check to see if the machine executing the Word document is domain joined. If running on a non-domain joined machine, please edit this line of code to `bool FUNC1 = true;` before running `LittleCorporal.exe` to generate a Word document

Recommendations

If you plan on using this project for active red team operations, please consider setting the "Exit" functionality of your shellcode to perform a "clean" exit with a thread exit, instead of completely killing the process in which the shellcode resides in. This can be configured with msfvenom via `EXITFUNC=thread`, and can also be configured in Cobalt Strike as such via Aggressor.

Usage

1. **YOU MUST FETCH THE ENTIRE PROJECT IN ORDER TO USE**! LittleCorporal uses relative paths for additional resources, such as Donut.
2. Once obtaining the entire project, change your working directory to the `bin\Release` directory (`cd C:\Path\to\LittleCorporal\bin\Release`). It is recommended that if you choose to re-compile this project that you use a Release build instead of a Debug build.
3. Specify the path to your shellcode file on the machine you are executing `LittleCorporal.exe` from and the already running process on the machine which you would like to execute the Word document on. (`LittleCorporal.exe C:\Path\To\Shellcode.bin explorer.exe`)
4. LittleCorporal will then output the path to the final Word document
5. To "clean" the Artifacts directory, use the following command: `LittleCorporal.exe clean`