Disclaimer

- Reverse engineering is a very very broad topic.
- A lot of overhead knowledge is required.
- There are no click and it’s reverse engineered tools.
- Please ask questions. Some of these topics are esoteric and will need explanations.
Who Am I.

- Reverse engineer malware at my $dayjob
- Been programming in Python since 2010.
- Wrote the *The Beginner’s Guide to IDAPython*, xxxswf.py and author of the blog *Hooked on Mnemonics*.
- Not 1337.
- @nullandnull
Outline

● Why Python?
  ○ History from an RE perspective.
  ○ Why it was adopted.
● What kind of tasks?
● What modules?
● Awesome links!
Why Python? - History

- Python has been in use by the reverse engineering community for over a decade.
- In 2005 on OpenRCE a number of influential tools were released in Python.
  - IDAPython and Pefile are the most popular.
- In 2005 and 2006 security companies started posting Python as a desired skill.
Why Python? - Because...

- Python is friendly.
- Prototyping in Python is quick.
  - Continuous recompiling isn’t.
- Adding modules is painless.
- Python is everywhere.
- Batteries included.
  - “import zlib” vs Visual Studio’s Project properties -> Linker -> Input -> Additional dependencies.. etc
What kind of tasks?

- It depends on the type of research.
- Python is commonly used for:
  - Binary analysis
  - Forensics
  - Malware analysis
  - Network analysis
  - Exploring file formats
  - Vulnerability and exploit analysis
Binary Analysis - Tasks

- Disassembling code
  - Binary to Assembly
- Automating the analysis of code
- Deobfuscating code
- Solving Cracks Me
- Full binary analysis frameworks have been written in Python
Binary Analysis - Example

```
push    ebx
push    esi
push    dword ptr [ebp+6Ch]
call    ds:off_880008
jmp     loc_879130
```

```
db      53h ; S
```

```
loc_879130:
    cmp     eax, ebx
    mov     [ebp+50h], eax
    jz      loc_871941
    jmp     loc_87A1D0
```

```
db      50h ; P
    aNeha   db 'laha',0,0
```

```
loc_877CD4:
    mov     esi, eax
    lea     eax, [ebp+68h]
    jmp     loc_87914F
```

```
db      76h ; V
db      6
```
Binary Analysis - Example

```python
# Python code to remove JMPs from obfuscated code in IDA
# created by alexander dot hanel at gmail dot com
# Note you will need to have your cursor at what is the start
# of the function or at least in the path.

from idaapi import *
import idautils
import idc
import sys

class JMPJMP:
    def __init__(self):
        self.es = ScreenEA()
        self.errorStatus = 'Good'
        self.funcStartAddr = GetFunctionAttr(self.es, FUNCATTR_START)
        self.checkFunctionStart()
        self.buffer = []
        self.count = 0
        self.condJmpsAddr = set([])
        self.retn = ['ret', 'ret', 'retf']
        self.callAddr = set([])
        self.call = 'call'
        self.callByte = 0xe8
        selfjmp = 'jmp'
        self.visitedAddr = set([])
        self.target = set([])

    def getJmpAddress(self, addr):
        "returns the address the JMP instruction jumps to"
        return GetOperandValue(addr, 0)

    def checkFunctionStart(self):
        'checks if the address is valid'
        if self.funcStartAddr is BADADDR:
            print "Could not find function start address"
            self.errorStatus = 'Bad!'"
Binary Analysis - Example

```
; _start: 00877e05
push  ebx
push  esi
push  edx
push  ecx
push  eax
mov  edx, [ebp+0x6Ch]
call  00877e08
mov  [ebp+4], eax
mov  [ebp+8], eax
mov  [ebp+14h], ecx
call  00877e68
lea   ecx, [(ebp+14h)]
call  sub_871188
mov  edx, [ebp+6Ch]
call  00877e08
mov  [ebp+0x6Ch], edx
```

Binary Analysis - Cool Examples!

- Using Microsoft's Z3 Theorem Prover to solve a CrackMe **.
- Deobfuscation: recovering an OLLVM-protected program **.
- Breaking Kryptonite's Obfuscation: A Static Analysis Approach Relying on Symbolic Execution **.
Obfuscated

Deobfuscated

Via Quarksblog
Forensics - Why & Tasks

● Most tools are *not* platform dependent.
  ○ Example: Analyzing a Windows memory dump in Linux

● File and disk analysis.
  ○ Timelines

● Parsing the registry.
  ○ Extracting shellbag data to see folders accesses.

● Memory analysis.
  ○ Analyzing memory dumps
Error is caused by regedit unable to display invalid characters written by poweliks. An analyst could copy %userprofile%/ntuser.dat to a separate machine and parse the hive using python-registry.
Forensics - Cool Examples!

- python-registry Introduction by Willi Ballenthin **.
- Automating DFIR Series by David Cowen **.
- Stuxnet’s Footprint in Memory with Volatility 2.0 **.
- Extracting the Powelik's DLL from the Registry **.
Malware Analysis- Tasks

- Automating the analysis of samples in a sandbox environment.
- Deobfuscation, decompressing and decoding data.
- Debugging and disassembling.
- Scanning files.
- Extracting data.
- Hard to describe so many amazing projects.
Malware Analysis - Sample Automation

Cuckoo Sandbox - Mozilla Firefox

Recent Files

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Filename</th>
<th>MD5</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-05-17</td>
<td>cracime.pdf</td>
<td>9a8c90f3547d8fd3c065cd74782af600</td>
<td>reported</td>
</tr>
<tr>
<td>2015-05-17</td>
<td>0db9979ed1b8077127e5530202d392797cob16052aa9972156c5197ce732b.bin</td>
<td>e4c68f3564858969aa37c5fb7a6ce363</td>
<td>reported</td>
</tr>
<tr>
<td>2015-05-17</td>
<td>0db9979ed1b8077127e5530202d392797cob16052aa9972156c5197ce732b.bin</td>
<td>e4c68f3564858969aa37c5fb7a6ce363</td>
<td>reported</td>
</tr>
<tr>
<td>2015-05-17</td>
<td>ad2caa30a9f85a8f210d3b945d812011b874c77358ae98ee391b1f61aac7a6.bin</td>
<td>85e9061625762439ae5de766b952229f</td>
<td>reported</td>
</tr>
<tr>
<td>2015-05-17</td>
<td>0db9979ed1b8077127e5530202d392797cob16052aa9972156c5197ce732b.bin</td>
<td>e4c68f3564858969aa37c5fb7a6ce363</td>
<td>reported</td>
</tr>
<tr>
<td>2015-05-17</td>
<td>a.exe</td>
<td>7c68feaa6ac68243e4c468a752753d74</td>
<td>reported</td>
</tr>
<tr>
<td>2015-05-17</td>
<td>b622e73b8b4be93a8c3fe57ca9a357d8bae695</td>
<td>e85d64dd896e967de0771321d6222d19</td>
<td>reported</td>
</tr>
</tbody>
</table>
Malware Analysis - Deobfuscation Example

```java
// obfuscated
package
{
    import flash.utils.*;

    public class II1l1IIII1II1II1 extends ByteArray
    {
        public function II1l1IIII1II1II1()
        {
            return;
        } // end function

        public function +IlIlIII111II11(): void
        {
            return;
        } // end function

        public function 11lIII1II1II11(): int
        {
            return 0;
        } // end function
    }
}
```

```java
// de-obfuscated
package
{
    import flash.utils.*;

    public class _tare extends ByteArray
    {
        public function _tare()
        {
            return;
        } // end function

        public function _secant() : void
        {
            return;
        } // end function

        public function _carat() : int
        {
            return 0;
        } // end function
    }
}
```
Malware Analysis - Example

```
class ObfStrReplacer():
    #
    # A module that can be used to de-obfuscate code by searching for strings
    # that match a regular express pattern and replace them with more readable
    # characters.
    #
    def __init__(self):
        self.regex_pattern = None
        self.compiled_regex = None
        self.file_glob_pattern = None
        self.test_regex = False
        self.script_name = None
        self.globbed_files = None
        self.word_list = [
            "abacus", "iota", "nu", "baryon", "ceres", "dean", "zipf",
            "mur", "epsilon", "luna", "format", "gamma", "carat", "gaudy",
            "ides", "alpha", "iris", "julia", "tare", "omicron", "pascal",
            "kappa", "aeon", "umbra", "secant", "lambda", "beta", "lema",
            "eta", "mars", "moebio", "occam", "chaos", "arc", "omega",
            "xenon", "pareto", "locus", "psi", "rho", "delta", "sigma",
            "pi", "sionson", "tau", "gnomen", "theta", "atlas", "upsilon",
            "phi", "venus", "oigive", "surd", "xi", "zeta", "sabot", "chi",
            "kite"
        ]
        self.match_set = set({})
        self.names = []
        self.name_mapping = {}  
    
    def __get__(self):
        #
        # gets the command line arguments.
        #
        parser = argparse.ArgumentParser(
            description='Replaces strings matched by a regular express with more 
                        distinguishable text/strings.
        )

    def run(self):
        #
        #
        #
        #
        if __name__ == "__main__":
            xx = ObfStrReplacer()
            xx.run()
```
Malware Analysis- Projects!

- My Favorites or at least should be mentioned
  - Cuckoo Sandbox
  - Yara
  - winappdbg, pydbg, pykd and vivisect
  - Capstone Project
  - IdaPython
  - pefile
  - IDAScope
Network Analysis- Tasks

- Protocol and decoding analysis
- Network and browser emulation.
- PCAP parsing.
- Packet creation, sniffing and manipulation.
- Custom passive DNS tool.
- Automating URL lookups
Network Analysis- Projects!

- Scapy or Dpkt
- Chopshop built on top of Pynids
- fakedns.py or Fakenet (python bindings)
- jsunpack
- Malcom
Network Analysis - Example

dpkt - example code and documentation.
Network Analysis - Example

Malcom - graphical view
Exploring file formats - Tasks

- Carving out embedded files in a data streams.
- Exploring structured data.
- Decompressing files.
  - SWF files are compressed zlib.
- Writing binary parsers
- Analyzing and extracting firmware.
Exploring file formats - Projects

- pdf-parser.py or peepdf
- oletools or oledump.py
- xxxswf.py
- pe-carv.py
- hachoir
  - hachoir-urwid
  - hachoir-subfile
- construct & vstruct
Exploring file formats - Example

File carving with pecarv.py
Exploring file formats - Example

Non-obfuscated JavaScript
Exploring file formats - Example

Obfuscated JavaScript
Task - Vulnerability & Exploit Analysis

- Small subset of a very complex area.
- Fuzzing
  - Providing invalid, unexpected or random data as input to see if the data invokes exceptions or crashes
  - Projects
    - Sulley
    - Peach 2
    - python-afl (for fuzzing python code)
Task - Vulnerability & Exploit Analysis

- Auditing Binaries
  - Scripts in IDAPython commonly used.
  - Simple search for commonly buggy functions
    - Enumerating suspicious function calls (strcpy, fgets, etc)
    - Enumerating file and network input and output
  - Diffing the assembly of patched and vulnerable executables
  - Analyzing data flow and allocation of variables
Links

- ObfStrReplacer - http://hooked-on-mnemonics.blogspot.com/2015/06/obfstrreplacer-extractsubfile-snippets.html
- Cuckoo Sandbox - https://github.com/cuckoobox/cuckoo
- dpkt - https://github.com/jeffsilverm/dpkt_doc
- malcom - https://github.com/tomchop/malcom
- The Very Unofficial Dummies Guide To Scapy - https://theitgeekchronicles.files.wordpress.com/2012/05/scapyguide1.pdf
Books!
- Hacking Secret Ciphers with Python
- Gray Hat Python: Python Programming for Hackers and Reverse Engineers
- The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory
- Black Hat Python: Python Programming for Hackers and Pentesters
- Python Forensics: A workbench for inventing and sharing digital forensic technology
- The Beginner’s guide to IDAPython :)
Questions?

Thanks!