Lecture 1
Introduction. Basic Exploration Tools

Computer and Network Security
October 09, 2023

Computer Science and Engineering Department

Outline

On this class

- Computer and Network Security
- offensive security, hacking, reverse engineering, runtime application security
- programming/practical oriented
- focus on binary exploitation (pwn levels in CTFs)
- lecture: Monday, 4pm-6pm, room PR002, Edi
- labs:
  - Monday, 6pm-8pm, Adrian
  - Tuesday, 8pm-10pm, Mihai
- `http://ocw.cs.pub.ro/cns/`
- but first:
  `https://ocw.cs.pub.ro/courses/cns/need-to-know`
- labs start on Monday, October 9th, 2023, 6pm

The Team

- Eduard Stăniloiu: lectures, lecture tests, exam
- Mihai Dumitru: labs, infrastructure
- Adrian Șendroiu: labs, assignments, lectures

Resources

- wiki (content): `http://ocw.cs.pub.ro/cns/`
- MS Teams: live action
- Moodle (news, deadlines, exam, dicussions, links to content, feedback)
  - SRIC (not-enrolable, only used for feedback)
  - SCPD (not-enrolable, only used for feedback)
  - common/meta (enrolable, actually used)
- Facebook (news, trivia): `http://facebook.com/cns.upb`
- mailing list (news, dicussions): `https://ocw.cs.pub.ro/courses/cns/resources/mailing-list`
- assignment write-only mailing-list (assignments): `http://cursuri.cs.pub.ro/cgi-bin/mailman/listinfo/oss-support`
- calendar & planning: `https://ocw.cs.pub.ro/courses/cns/calendar`
- virtual machines (labs, assignments, CTFs): `https://ocw.cs.pub.ro/courses/cns/resources/vm`
- CTF platform (assignments, labs): `https://cns-ctf.security.cs.pub.ro/home`
- team: to yell at

Lab Split

- happens on the curs.upb.ro Discussion forum
- threads for each of the 4 lab slots
- almost complete
- you need to be enroled
- you can enrol by yourself by accessing the CNS curs.upb.ro instance
- limit is 16 students per lab slot

Class Keywords

- reverse engineering
- binary inspection
- stack overflow
- buffer overflow
- shellcode
- shell execution
- exploiting
- runtime application security
- return oriented programming
- CTF (*Capture the Flag*)

Table of Contents

planning: `https://ocw.cs.pub.ro/courses/cns/calendar`

Bibliography

- Robert Seacord – Secure Coding in C and C++, Addison Wesley Professional, 2005
- Robert Seacord – The CERT C Secure Coding Standard, Addison Wesley Professional, 2008
- Anton Chuvakin, Cyrus Peikari – Security Warrior, O'Reilly, 2004
- Grey Hat Hacking. The Ethical Hacker's Handbook, 3rd Edition, McGraw Hill, 2011
- Enrico Perla, Massimiliano Oldani – A Guide to Kernel Exploitation, Syngress, 2011
- Jon Erickson – The Art of Exploitation, 2nd Edition, No Starch, 2008
- Michael A. Davis, Sean M. Bodmer, Aaron LeMasters – Hacking Exposed. Malware and Rootkits, McGraw Hill, 2010
- Bruce Schneier – Applied Cryptography, John Wiley & Sons, 1996

Grading

- **2 points** – lab involvement
- **4.5 points** – 3 assignments
- **3.5 points** – final exam

Final Exam

- oral exam

CTF – Capture the Flag

- computer security competition
- educational, practice
- attack/defense vs. jeopardy
- web, stegano/forensics, crypto, binary/reverse, pwn/exploit, protocol, misc
- wargames
- may equate assignment points

Outline

- lingua franca of low-level programming
- powerful enough to build amazing software and flexible enough to shoot yourself in the foot
- close to hardware, everything is at some point coming from C code
- direct access to memory management (buffers, strings, arrays, pointers): mixed blessing

- move around quickly
- investigate, analyze system
- quickly develop, build, debug, analyze applications
- automate tasks

Assembly Language

- everything turns to machine code
- one may not have access to the source code, but it can be disassembled
- hardware specific – the "guts" of the computer
- required knowledge to fully be able to exploit and protect the system

- binary, octal, hexadecimal
- ASCII
- signed / unsigned integers: size, range, 2's complement representation
- endianess
- there are 10 types of people in the world ...
- disassembled code, addresses and hardware instructions are shown in hexadecimal
- one is required to easily convert hexadecimal to decimal and the other way around

- system and application inner workings
- process virtual address space
- application run time: CPU, memory, I/O usage
- system calls, kernel space

Process Investigation

- processes and resource usage: `ps`, `pstree`, `pgrep`, `procfs` filesystem
- memory mappings: `pmap`
- open file descriptors: `lsof`

If You Feel Lacking

- this is a master class, you need to be on the level
- work, work, work
- C programming:
  `https://ocw.cs.pub.ro/courses/programare`
- Linux / Unix CLI, shell scripting:
  `https://ocw.cs.pub.ro/courses/uso`
- assembly language + hexadecimal:
  `https://ocw.cs.pub.ro/courses/iocla`
- operating systems + process investigation:
  `https://ocw.cs.pub.ro/courses/so`

Outline

- Python, Perl
- automation
- generate/print binary data and feed it to an executable
- generate strings, generate variating integers & addresses
- do redirects, make conversions, process strings

- quick'n'dirty scripting language
- more powerful than shell scripting
- create binary payloads (use `struct` package)
- convert data
- work with strings
- work with files
- work with processes (use `subprocess` package)
- advanced exploit techniques (use `pwn` package)
- use Python3, FTW!!!

Hex Viewers and Editors

- dump and edit data in binary files (object files, executables, encrypted files)
- `hexdump`, `xxd`, `od`: make hexdumps
- `hte`: terminal hex editor
- `ghex`, Bless: GUI hex editor

- dynamic analysis
- default debugger on Unix systems
- may be used to trace programs, check variables and return values

PEDA

- *Python Exploit Development Assistance for GDB*
- enhance GDB for exploit development
- improved commands
- improved views
- search for ROP gadgets
- generate shellcodes
- generate buffer cyclic patterns
- `http://ropshell.com/peda/`

Binary Code Analysis

- inspect object and executable files
- disassembling: `objdump`
- forensics: `strings`
- executable parsing: `readelf`, `nm`
- dependencies: `ldd`

Call Tracing

- dynamic analysis
- capture system calls, function callls of program
- check out system call arguments
- check out system call return values
- see whether process blocks in a system call
- `strace`, `ltrace`

- **IDA**
  - IDA 7.0 freeware
  - different executable formats for different processors
  - debugger
  - decompiler
  - interactive
  - plugins
- Ghidra
  - open source
  - similar to IDA
- radare2
  - disassemble, debug
  - static and dynamic analysis
  - CLI
- capstone
  - "lightweight multi-platform, multi-architecture disassembly framework"
  - open source

Other Binary-related Tools

- Binary Ninja: `https://binary.ninja`
- BinNavi: `http://www.zynamics.com/binnavi.html`
- Hopper: `http://www.hopperapp.com/`

Emulators

- run executables for different architectures
- QEMU: emulates MIPS, ARM, PowerPC, SPARC
- Unicorn Engine, based on QEMU

- CTF framework and exploit development library
- Python
- connections to local and remote processes
- packing / unpacking
- assemby and disassembly
- ELF manipulation
- shellcode generation
- Return Oriented Programming
- `https://github.com/Gallopsled/pwntools`
- `https://docs.pwntools.com/en/stable/`

Others

- brain
- will
- perseverance
- will
- perseverance
- perseverance
- perseverance
- Did we mention perseverance?

Outline

strings

- search for ASCII strings in binary data
- `strings /path/to/binary/file`
- `man ascii` to show ASCII table

Let's print shellcode from `http://www.shell-storm.org/shellcode/files/shellcode-827.php`:

Shellcode Sample

```
char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
      "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";
```

Do it in several scripting languages:

Print Shellcode in Bash, Python, Perl

```
(Bash) echo -e '\x31\xc0\x50\x68\x2f...'
(Python) python -c 'print "\x31\xc0\x50\x68\x2f..."'
(Perl) perl -e 'print "\x31\xc0\x50\x68\x2f..."'
```

Printing Binary/Hex Data (2)

Dump binary data in hex and binary:

Using xxd

```
$ echo -en '\x31\xc0\x50\x68...' | xxd
0000000: 31c0 5068 2f2f 7368 682f 6269 6e89 e350  1.Ph//shh/bin..P
0000010: 5389 e1b0 0bcd 80                        S......

$ echo -en '\x31\xc0\x50\x68...' | xxd -g 4
0000000: 31c05068 2f2f7368 682f6269 6e89e350  1.Ph//shh/bin..P
0000010: 5389e1b0 0bcd80                      S......

$ echo -en '\x31\xc0\x50\x68...' | xxd -g 1
0000000: 31 c0 50 68 2f 2f 73 68 68 2f 62 69 6e 89 e3 50  1.Ph//shh/bin..P
0000010: 53 89 e1 b0 0b cd 80                             S......

$ echo -en '\x31\xc0\x50\x68...' | xxd -b
0000000: 00110001 11000000 01010000 01101000 00101111 00101111  1.Ph//
0000006: 01110011 01101000 01101000 00101111 01100010 01101001  shh/bi
000000c: 01101110 10001001 11100011 01010000 01010011 10001001  n..PS.
0000012: 11100001 10110000 00001011 11001101 10000000           .....
```

Using strace

- `strace ./executable`
- `strace -e write ./executable` – print `write` syscalls
- `strace -e trace=file ./executable` – print syscall taking a filename as argument
- `strace -f ./executable` – trace child processes
- `strace -p PID` – trace existing process by PID
- `strace -s strsize` – trace using a different size for strings

ltrace

- see library calls
- `ltrace -p PID` – trace process
- `ltrace -t` – show timestamp

Passing Binary Data as Argument

- `command $(python -c 'print ...')`

Passing Binary Data as Standard Input

- `python -c 'print ...' | command`
- `cat file - | command`
- `cat <(python -c 'print ...') - | command`

List Open Files

- `lsof`
- `lsof -p PID` – show open files for process
- shows file descriptors: standard input/output, sockets, pipes

Process Address Space

- `pmap`
- `pmap PID` – show address space mappings for process
- shows permissions and addresses

Show Library Dependencies

- `ldd /path/to/executable`
- useful to check if an executable may run on a given system, what library version is it using

Installing 32bit Development Libraries on Debian

Installing 32bit Development Libraries

```
# dpkg --add-architecture i386
# apt update
# apt install gcc-multilib g++-multilib libc6:i386 libc6-dev:i386 \
      libstdc++6:i386 libstdc++6-dev:i386
```

Outline

- use `objdump` to disassemble binary
- use `man ascii` or `hex printing` to print password

Outline

Keywords

- offensive security
- runtime application security
- table of contents
- grading
- CTF (*Capture the Flag*)
- tools of the trade
- hex editors
- scripting language
- disassemblers
- exploration
- hex/binary data
- Python
- `strings`
- `objdump`
- `strace`, `ltrace`
- `ldd`, `lsof`, `pmap`
- IDA
- Ghidra, radare2
- GDB, PEDA
- pwntools

Useful Links

- http://reverseengineering.stackexchange.com/
- http://security.cs.pub.ro/hexcellents/wiki/
- http://web.cecs.pdx.edu/~jrb/cs201/lectures/handouts/gdbcomm.txt
- http://ctftime.org/
- https://picoctf.com/
- http://captf.com/practice-ctf/
- https://io.netgarage.org/
- http://www.overthewire.org/wargames/
- http://ctf365.com/
- PEDA: https://github.com/longld/peda
- IDA: https://www.hex-rays.com/products/ida/
- Ghidra: https://ghidra-sre.org
- Radare: http://rada.re/r/
- pwntools: https://docs.pwntools.com/en/stable/

References

- Security Warrior
  - Chapter 1. Assembly Language
  - Chapter 2. Windows Reverse Engineering
  - Chapter 3. Linux Reverse Engineering
- The Ethical Hacker's Handbook, 3rd Edition
  - Chapter 10: Programming Survival Skills
  - Chapter 20: Passive Analysis
  - Chapter 21: Advanced Static Analysis with IDA Pro