

Binary Reverse Engineering And Analysis

Course 1: RE in Context

Caragea Radu

February 16, 2021

- Radu Caragea - Sr. Security Researcher at Bitdefender
- Binary Analysis and Exploitation, Forensics, Cryptanalysis
- Contact: rcaragea@bitdefender.com

About this course (PROs)

- Will teach you how to make sense of almost any executable
- Will teach you the dangers of code insecurity
- Will teach you how to map attack surface and exploit binaries

About this course (CONs)

- This course is HARD*
- Steep learning curve
- You must invest time at home

2. Date despre disciplină

2.1. Denumirea disciplinei		Inginerie inversă și tehnici de securizare a codului						
2.2. Titularul activităților de curs				Lector dr. Ruxandra-Florentina Olimid				
2.3. Titularul activităților de seminar / laborator / proiect				Lector dr. Ruxandra-Florentina Olimid				
2.4. Anul de studiu	II	2.5. Semestrul	II	2.6. Tipul de evaluare	E	2.7. Regimul disciplinei	Conținut ⁽¹⁾ Obligativitate ⁽²⁾	DS DI

3. Timpul total estimat (ore pe semestru al activităților didactice)

3.1. Număr de ore pe săptămână	3	din care: 3.2. curs	1	3.3. seminar/ laborator/ proiect	2
3.4. Total ore pe semestru	30	din care: 3.5. curs	10	3.6. SF	20
Distribuția fondului de timp					Ore
3.4.1. Studiul după manual, suport de curs, bibliografie și notițe – nr. ore SI					56
3.4.2. Documentare suplimentară în bibliotecă, pe platformele electronice de specialitate și pe teren					20
3.4.3. Pregătire seminare/ laboratoare/ proiecte, teme, referate, portofolii și eseuri					70
3.4.4. Examinări					4
3.4.5. Alte activități					
3.7. Total ore studiu individual	150				
3.8. Total ore pe semestru	180				
3.9. Numărul de credite	6				

Prerequisites

- Motivation
- Python (or learn fast)
- C and pointers (able to write a doubly linked list implementation)
- Linux CLI

Course Contents

- 01. Reverse Engineering in Context
- 02. 64-bit Assembly Crash Course
- 03. Static Analysis with IDA
- 04. Dynamic Analysis Using Debuggers

Course Contents

- 01. Reverse Engineering in Context
- 02. 64-bit Assembly Crash Course
- 03. Static Analysis with IDA
- 04. Dynamic Analysis Using Debuggers

- 05. Stack Constructs and Corruption
- 06. (NX/DEP), ASLR, ROP
- 07. SSP, RELRO, PIE
- 08. Heap Constructs and Corruption

Course Contents

- 01. Reverse Engineering in Context
- 02. 64-bit Assembly Crash Course
- 03. Static Analysis with IDA
- 04. Dynamic Analysis Using Debuggers

- 05. Stack Constructs and Corruption
- 06. (NX/DEP), ASLR, ROP
- 07. SSP, RELRO, PIE
- 08. Heap Constructs and Corruption

- 09. RE for other Programming Languages
- 10. Further topics on Exploitation

Grading

- Course attendance: $1p$
- Lab solutions sent: $3p$ (1 week to send in, solutions given after deadline)
- Two homework assignments: $2p \times 2 = 4p$ (3 weeks to send in)
- Final exam: $4p$

- Maximum grade: $12p$ (rounded to $10p$)
- Minimum pass grade: $4.90p$ (hard limit)

Register

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE		
Group	Final grade	Total points	Exam	Attendance	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	Lab grade	L01	L02	L03	L04	L05	L06	L07	L08	L09	L10	Assignments grade	Asg-1	Asg-2			
			40%	10%	Intro	ASM	IDA	DBG	Stack	ROP	PIE	Heap	RE++	Exp++	30%											40%					
	10	12	4	10	1	1	1	1	1	1	1	1	1	1	10	16	16	16	16	16	16	16	16	16	16	16	10	10	15		
510 (SLA)		0		0											0											0					
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0											0												0				
510 (SLA)		0		0																											

Why Reverse Engineering?

Why Reverse Engineering?

A few examples:

- The online gaming industry
- The entertainment/multimedia industry
- The anti-malware industry
- The cybersecurity industry
- Academia/Research

Why Reverse Engineering? The Gaming Industry

Anti-cheat Software Engineer

 Blizzard Entertainment  Irvine, CA, United States

JOB DETAILS

The StarCraft II / Heroes of the Storm team is seeking an engineer experienced with preventing malicious software from accomplishing its objective. The ideal candidate would be equally comfortable combating these types of programs and collaborating with the engineering team as well as internal and external anti-cheat / security working groups to keep the game secure.

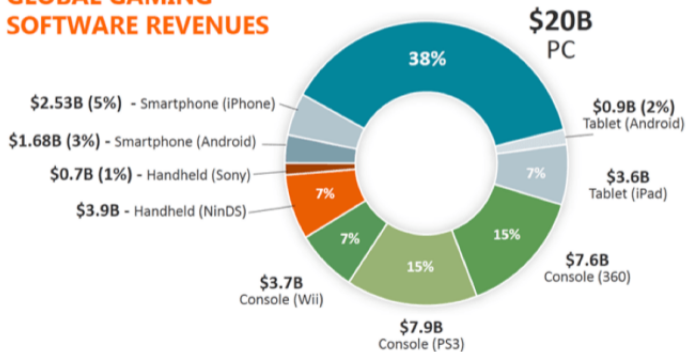
This is a full-time position at our office in Irvine, California

Requirements

- A passion to make life difficult for the “bad guys”
- Mastery of C++, disassembly, and reverse-engineering
- A strong motivation to analyze and improve systems and infrastructure
- Excellent organizational and communication skills
- Extensive Windows API and run-time model knowledge (memory, stack, and

Why Reverse Engineering? The Online Gaming Industry


GLOBAL GAMING SOFTWARE REVENUES



Why Reverse Engineering? The Entertainment Industry

Google's Widevine L3 DRM, used by Netflix, Hulu, and HBO, has been broken

45

 Ryne Hager
Jan 2, 2019

  160  100

Total Shares 260

GOOGLE NEWS SECURITY

WHO WOULD WIN?



NETFLIX HBO Disney

hulu Jio DIRECTV

WB prime video SHOWTIME

PS sling facebook

A FEW FLIPPY BOIS

Differential fault analysis (DFA)

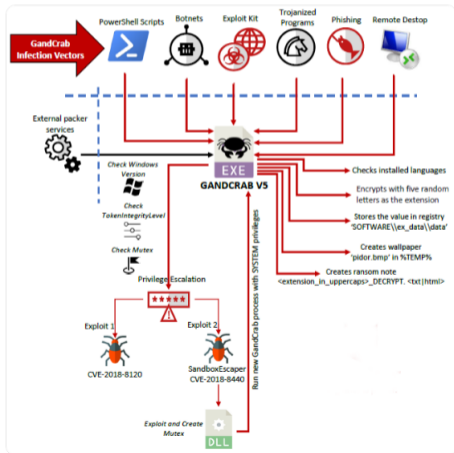
After ShiftRow	Fault injected 'E'	After MixColumns	K ₁
87 F2 40 97	87 F2 40 97	78 48 43 4C	AC 19 20 07
4E 92 90 8C	4E 92 90 8C	70 04 70 9F	77 FA 01 0C
44 E9 4A C3	44 E9 4A C3	8A 64 3A 42	46 0C 29 00
AA 0C 08 95	AA 0C 08 95	5E AS A3 0C	F3 21 A1 0E

After AddRoundKey	After SubBytes	After ShiftRows	K ₂
87 99 00 18	8E C8 30 AF	8E C8 30 AF	00 C9 03 84
2E A1 C3	2E 22 2E	2E 22 2E	14 0E 3F 43
03 38 13 42	03 07 70 2C	70 2C 07	FF 25 0C 0C
30 04 07 32	30 5F 94 05	05 00 9 94	A0 0F C8 AA

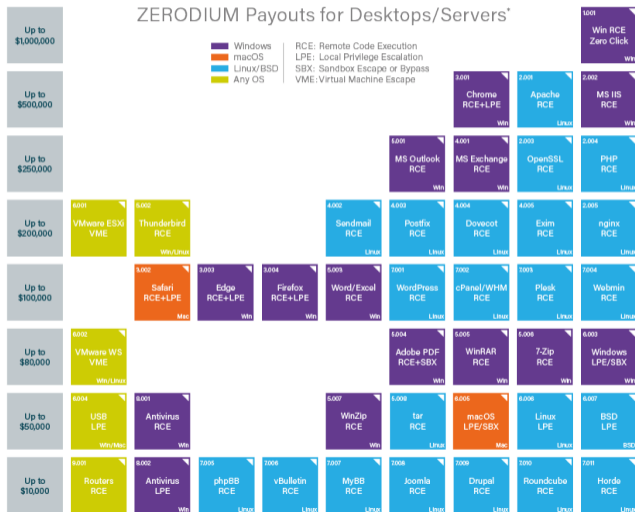
Output with fault	Output without fault	Error
02 0C 19	02 0C 19	00 00 00
25 0C 11 0A	25 0C 11 0A	00 00 00
84 19 00 00	84 19 00 00	00 00 00
10 00 97 32	10 00 97 32	00 00 00

Why Reverse Engineering? The Antimalware Industry

Rapidly Evolving Ransomware GandCrab Version 5 Partners With Crypter Service for Obfuscation

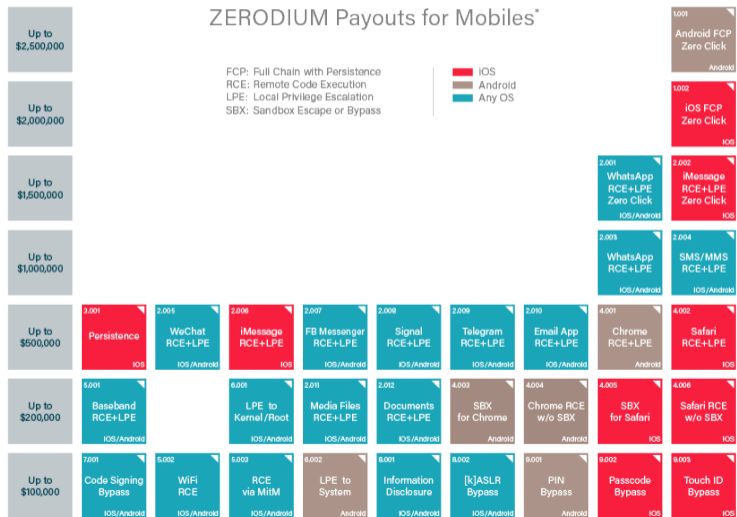


Why Reverse Engineering? The 0day "Industry" (desktop/server)



* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

Why Reverse Engineering? The 0day "Industry" (mobile)



* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

How to RE? Methodology

- First, you need an executable (ideally one you can run)
- Ultimately, all CPU instructions are contained in it

How to RE? Methodology

- First, you need an executable (ideally one you can run)
- Ultimately, all CPU instructions are contained in it
- Method 1 - exhaustively analyze the machine code (what ends up in the CPU)

How to RE? Methodology

- First, you need an executable (ideally one you can run)
- Ultimately, all CPU instructions are contained in it
- Method 1 - exhaustively analyze the machine code (what ends up in the CPU)
- Method 2 - follow along the execution paths to see how it works

How to RE? Methodology

- First, you need an executable (ideally one you can run)
- Ultimately, all CPU instructions are contained in it
- Method 1 - exhaustively analyze the machine code (what ends up in the CPU)
- Method 2 - follow along the execution paths to see how it works
- Method 3 - watch its behaviour from outside and interactions (with the OS)

Method 3 a.k.a. Black Box Analysis

- Investigate the system call/library call surface
- Windows tools:
 - ProcMon (SysInternals)
 - API Monitor
- Linux tools:
 - strace (syscalls)
 - ltrace (library calls)

References

- <https://www.velvetjobs.com/job-posting/anti-cheat-software-engineer-250735>
- mcvuk.com/development/defending-online-games-from-piracy-cheating-and-fraud
- androidpolice.com/2019/01/02/googles-widevine-l3-drm-used-by-netflix-hulu-and-hbo-has-been-broken
- <https://zerodium.com/program.html>

Practice

- Any Questions?
- Start lab tasks
- https://pwnthybytes.ro/unibuc_re/01-lab.html