

University of Piraeus
Department of Digital Systems

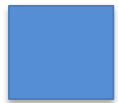


ROPInjector: Using Return-Oriented Programming for Polymorphism and AV Evasion

G. Poullos, C. Ntantogian, C. Xenakis
{gpoullos, dadoyan, xenakis}@unipi.gr

Objective of this research

- We propose **Return Oriented Programming (ROP)** as a **polymorphic alternative** to achieve **AntiVirus (AV) evasion**.



+



1 Portable Executable

1 well-known shellcode

Many different variations

- A malware is a piece of code
- Usually inside a benign executable (**Trojan**)
 - This will be our type of malware
- Can be written in high level language (C/C++, VB,...)
- BUT as any code, when compiled, an executable is created (files with extension .exe)
- This executable file includes a set of machine instructions → **Assembly**

- An AV uses a combination of two methods:
 1. Static analysis (i.e., AVs use a database of **signatures** of known malware including MD5 hashes and fixed strings).
 2. Dynamic analysis (i.e., when the file is executed, the AV monitors the behavior of the executable at **runtime** to detect any suspicious action).
- Each method has its own advantages and drawbacks!

- 32 bit CPU has many registers:
 - **eax, ebx, ecx, edx**
- Add eax, 1 \rightarrow eax=eax +1
- Sub eax, 1 \rightarrow eax=eax-1
- Mov eax, 4 \rightarrow eax=4

Example 1

Benign executable

```
Add eax, 1
Add eax, 6
Sub eax, 3
Mov eax, 3
Add eax, 2
Mov ecx, 2
Add ecx, 4
Sub eax, 7
Add eax, 4
Mov eax, 2
Add eax, 10
Mov ebx, 6
```

```
Sub eax, 7
Mov eax, 6
Add eax, 6
```

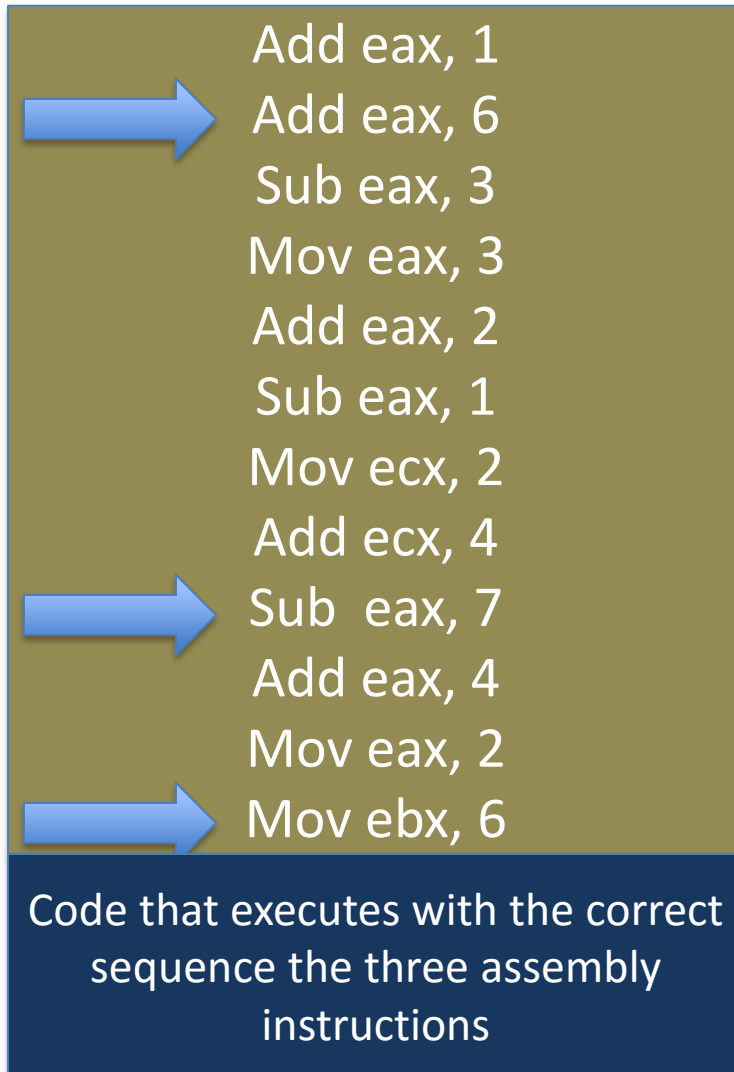
Malware (detected by AV)

```
Sub eax, 7
Mov ebx, 6
Add eax, 6
```

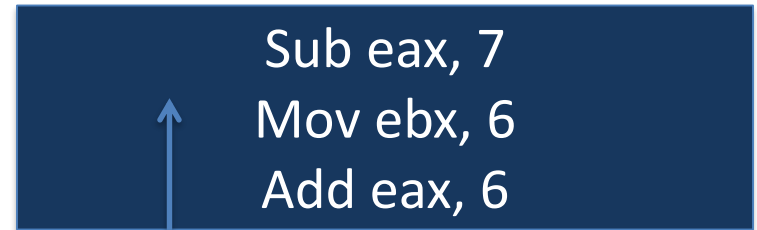
AVs detect it ! But not all of them!!

Example 2

Benign executable



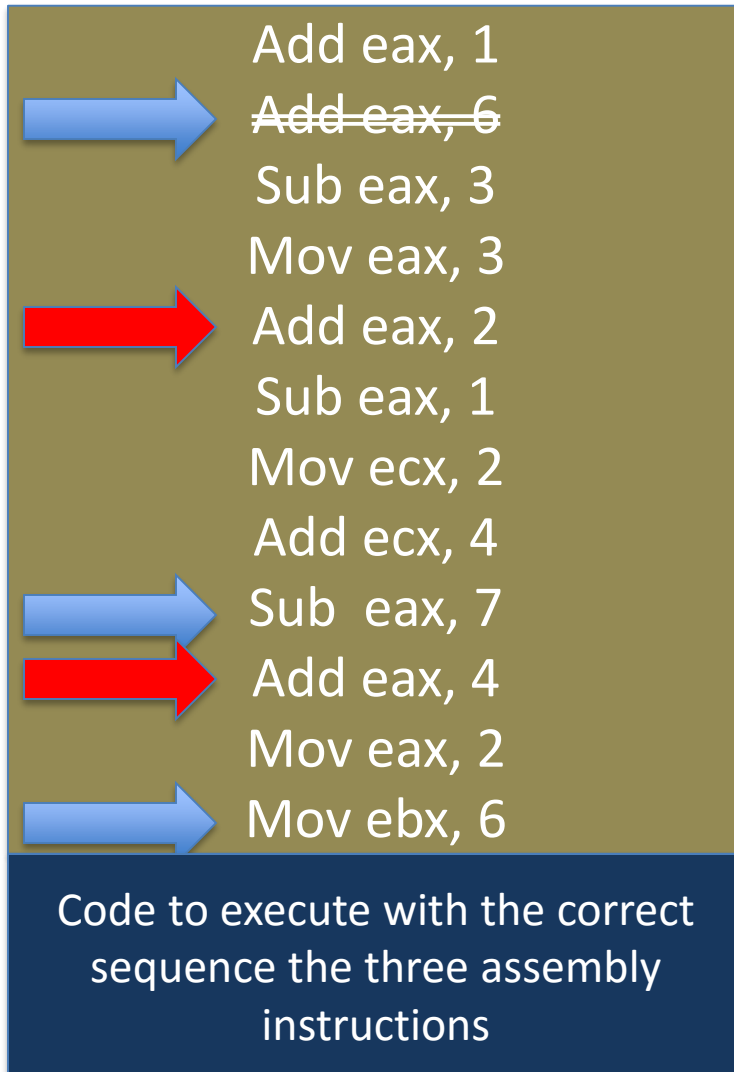
Malware (detected by AV)



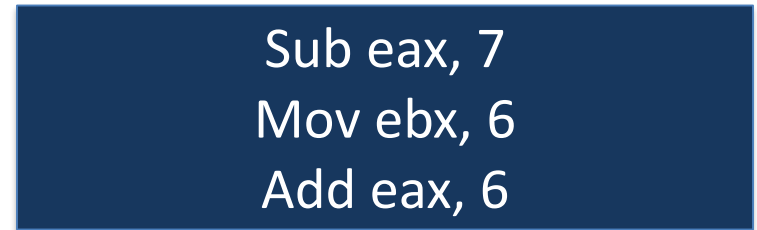
Return
Oriented
Programming
(ROP)!

Example 3

Benign executable

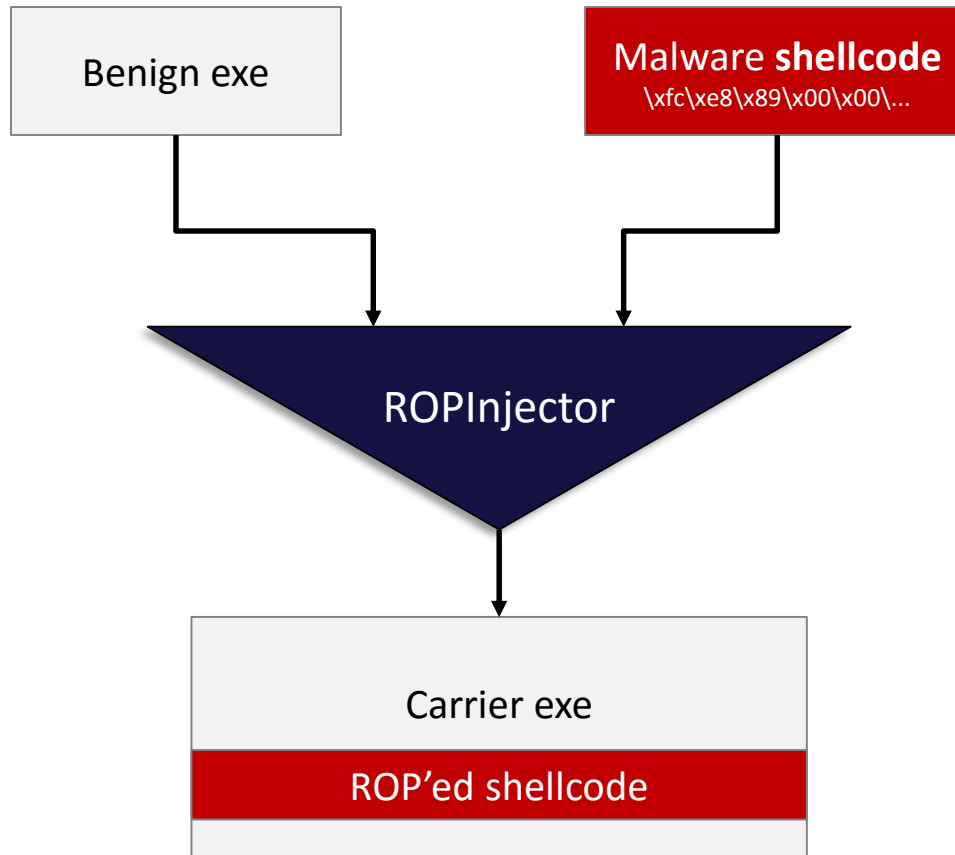


Malware (detected by AV)



Polymorphism!

Our Tool: ROPinjector



Why use ROP for AV evasion?

- a) We use **borrowed code** (i.e., ROP gadgets)
→ **Does Not raise any suspicion!**
- b) May transform any given **shellcode** to a
ROP-based equivalent → **Generic**
- c) May use **different ROP chain** →
Polymorphism

Challenges for our Tool

1. The new **resulting PE** should **evade AV detection**
2. PE should not be **corrupted/damaged**
3. The **tool** should be **generic** and **automated**



Steps of ROPInjector

1. Analyze the **shellcode**
2. Analyze the benign PE to find ROP chain
3. Transform the **shellcode** to an **equivalent ROP chain**
4. Inject into the PE **missing instructions** (*if required*)
5. Patch the PE with ROPed shellcode



STEP 1: Shellcode Analysis

- Aims to obtain the **necessary information** to safely replace **shellcode instructions** with **gadgets**
- For each **instruction**, **ROPInjector** likes to know:
 - what **registers** it **reads**, **writes** or **sets**
 - what **registers** are **free** to modify
 - its **bitness** (a `mov al,X` or a `mov eax,X` ?)
 - whether it is a **branch** (`jmp`, `conditional`, `ret`, `call`)
 - and if so, where it **lands**
 - whether it is a **privileged** instruction (e.g., `sysenter`, `iret`)
 - whether it contains a **VA reference**
 - whether it uses **indirect addressing mode** (e.g., `mov [edi+4], esi`)



STEP 2: Find ROP chain in PE

1. First, find **returns** of type:

- `ret (n)` or
- `pop regX`
 `jmp regX` or
- `jmp regX`



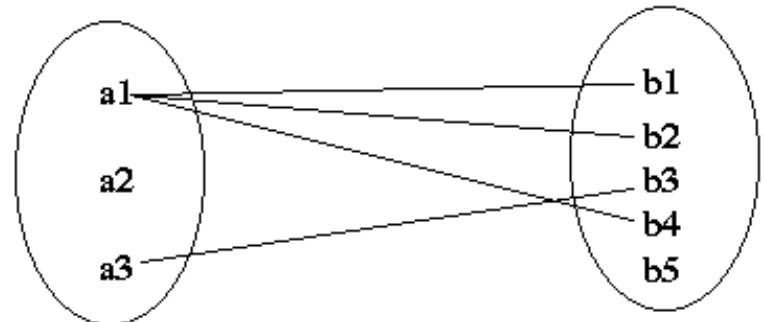
2. Then, search **backwards** for more **candidate gadgets**

STEP 3: Transform shellcode to ROP chain

- Initially, it translates **shellcode instructions** to an **Intermediate Representation (IR)**.
- Next, it translates the **ROP gadgets** found in PE to an **IR**.
- Finally, it provides a **mapping between the two IRs**
 - 1 to 1

or

- 1 to many



STEP 3: Intermediate Representation

IR Type (20 in total)	Semantics	Eligible instructions
ADD_IMM	regA += imm	<pre> add r8/16/32, imm8/16/32 add (e)ax/al, imm8/16/32 xor r8/16/32, 0 cmp r8/16/32, 0 inc r8/16/32 test r_a32, r_b32 (with r_a == r_b) test r8/16/32, 0xFF/FFFF/FFFFFFFF test (e)ax/al, 0xFF/FFFF/FFFFFFFF or r_a32, r_b32 (with r_a == r_b) </pre>
MOV_REG_IMM . . .	mov regA, imm	<pre> mov r8/16/32, imm8/16/32 imul r16/32, r16/32, 0 xor r_a8/16/32, r_a8/16/32 and r8/16/32, 0 and (e)ax/al, 0 or r8/16/32, 0xFF/FFFF/FFFFFFFF or (e)ax/al, 0xFF/FFFF/FFFFFFFF </pre>

STEP 3: Mapping examples

- 1-1 mapping example

- **Shellcode:**

`mov eax, 0` → `MOV_REG_IMM(eax, 0)`

- **Gadget in PE:**

`and eax, 0`
`ret` → `MOV_REG_IMM(eax, 0)`

1 to 1
IR
mapping



- 1-many mapping example

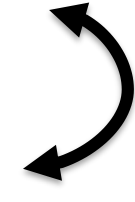
- **Shellcode:**

`add eax, 2` → `ADD_IMM(eax, 2)`

- **Gadget in PE:**

`inc eax`
`ret` → `ADD_IMM(eax, 1)`

1 to 2
IR
mapping



STEP 4: Gadget Injection

- If the PE does not include the required **ROP gadgets**
- By simply injecting **ROP gadgets** would raise **alarms**



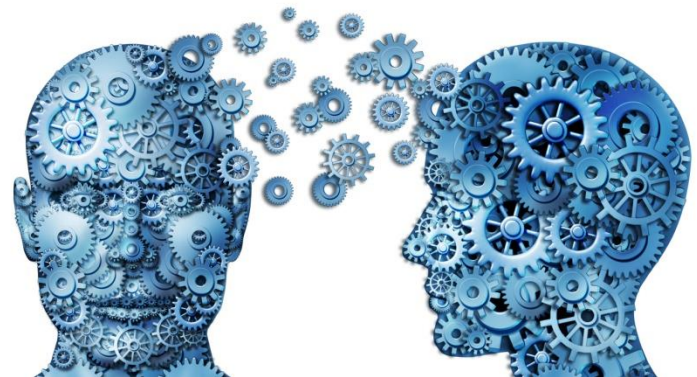
Statistics (presence of successive **ret** instructions)

- Therefore, **ROPInjector** inserts **ROP gadgets** **scattered** in a **benign looking way** avoiding alarms:
 - 0xCC caves in **.text** section of PEs (*padding space left by the linker*)
 - Often preceded by a **ret** (*due to function epilogue*)

```
00000640 FC 1E 00 00 E9 19 31 00 00 E9 44 09 00 00 CC CC .....1...D....
00000650 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC .....
00000660 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC .....
00000670 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC .....
```

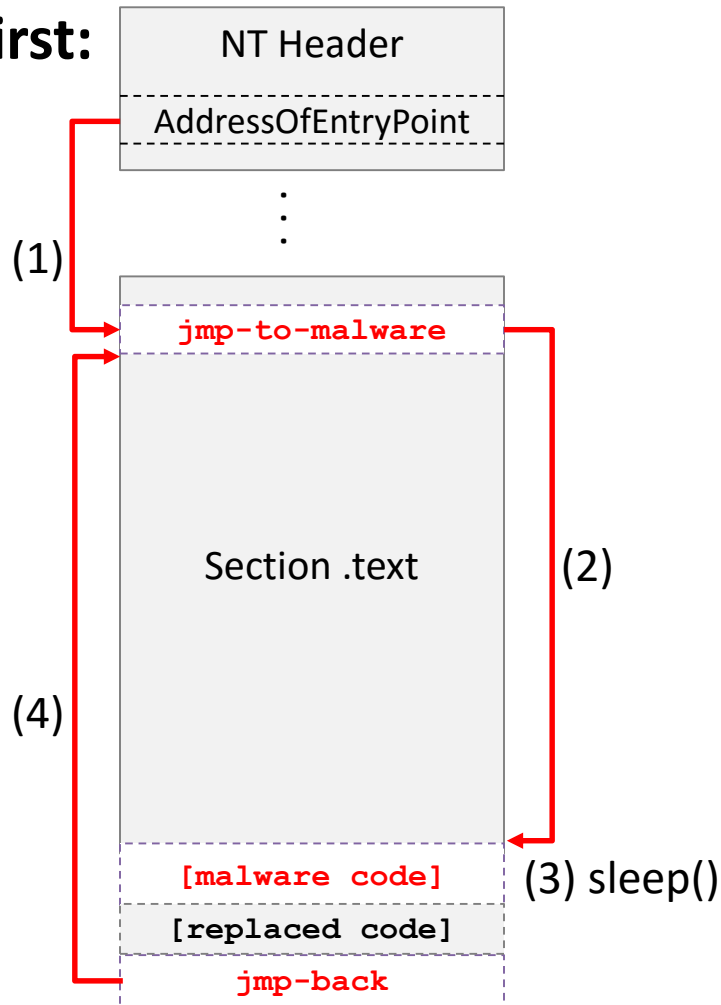

STEP 5 : Assemble and patch the ROP chain into the PE

- Step 5: Insert the **code** that loads the **ROP chain** into the **stack** (*mainly PUSH instructions*)
- Step 6 patch the new PE: Extends the **.text** section (instead of adding a new one), and, then, **repair** all **RVAs** and **relocations** in the **PE**.
- **ROPInjector** includes **two** different methods to **pass control** to the **ROPed shellcode**
 - Run first + delay execution via `sleep()`
 - Run last



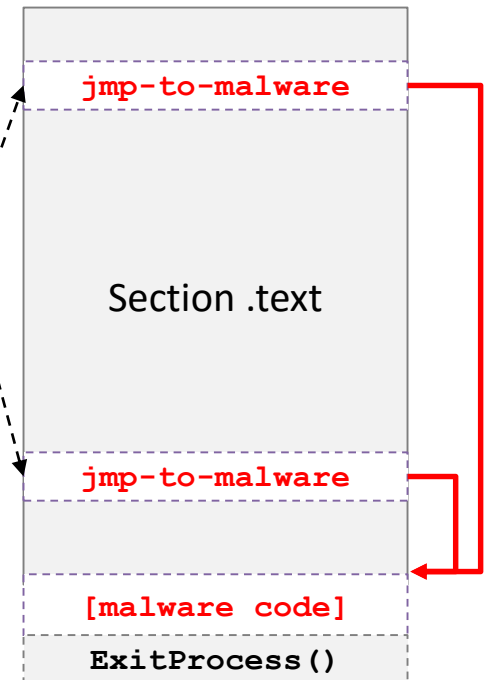
STEP 6: PE Patching (2/2)

Run first:



Run last:

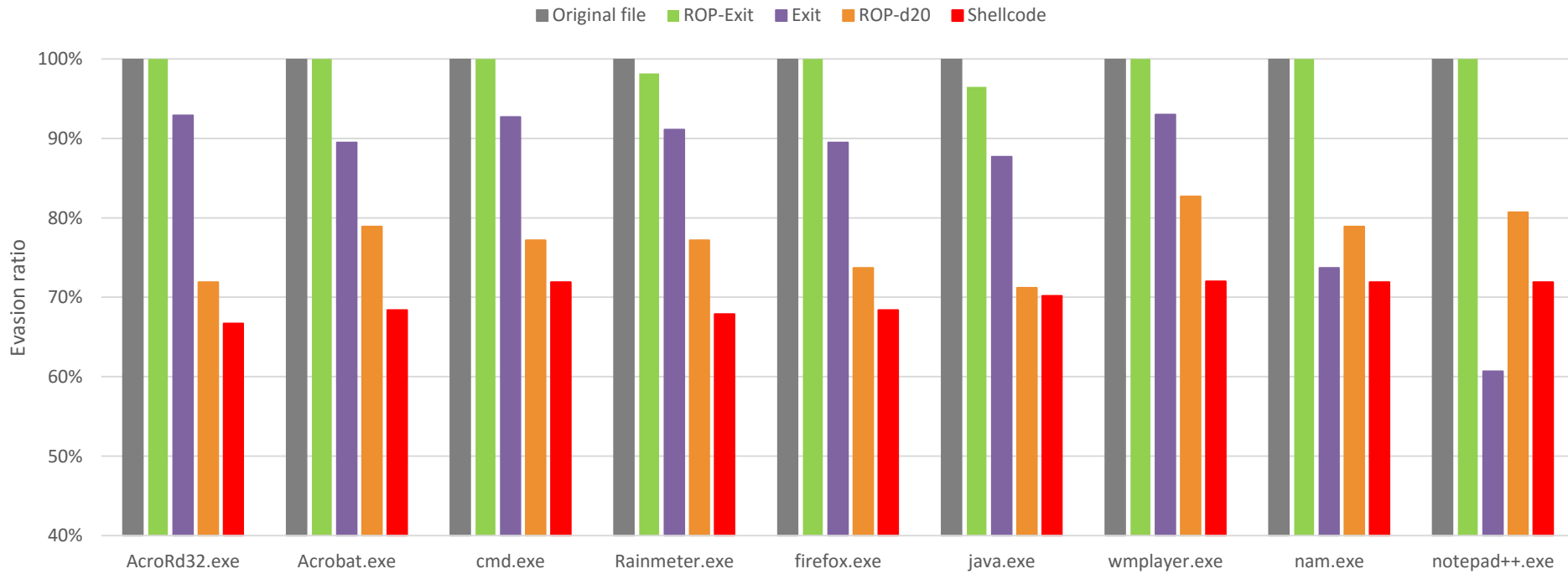
Previous calls to
ExitProcess()
/ exit()



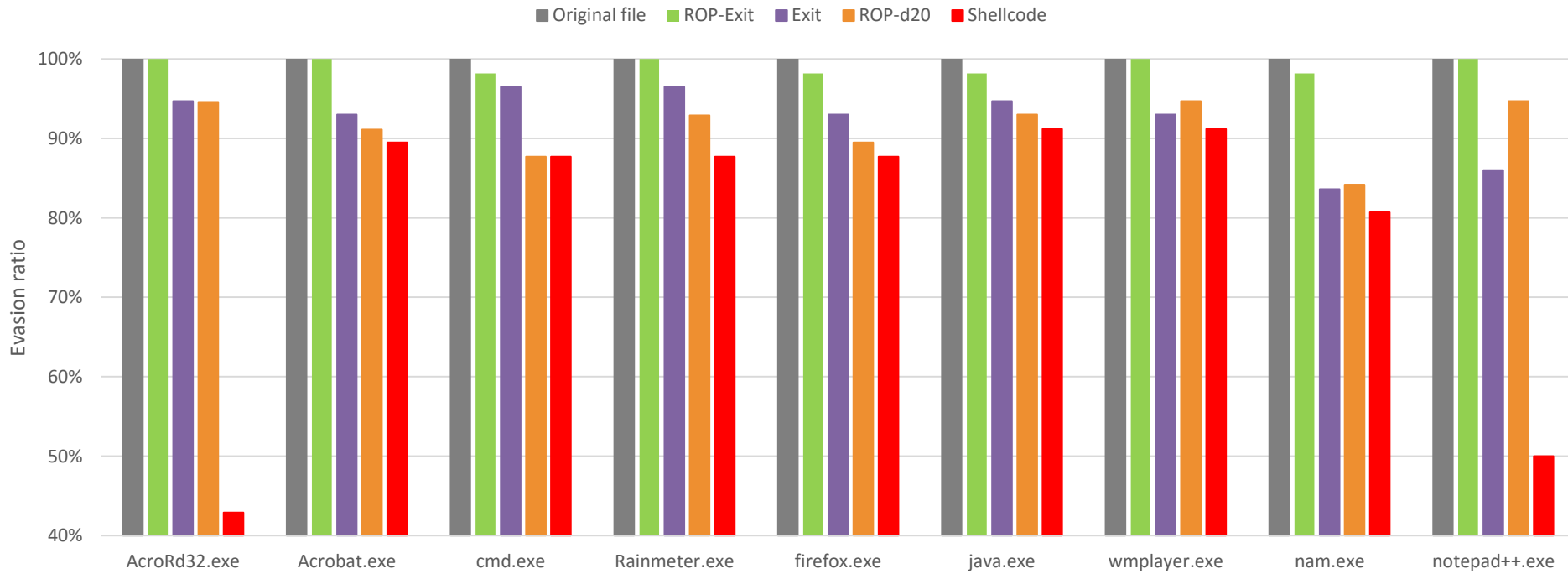
- ROPInjector is implemented in **native Win32 C**
- Nine (9) **32-bit** popular executables
 - firefox.exe, java.exe, AcroRd32.exe, cmd.exe, notepad++.exe and more
- 2 of the most popular Metasploit payloads
 - **reverse TCP shell**
 - **meterpreter reverse TCP**
- VirusTotal
 - at the time it employed **57 AVs**

- Various combinations
 - Original-file (no patching at all)
 - ROPShellcode-Exit (ROP'ed shellcode + run last)
 - Shellcode-Exit (intact shellcode passed control +run last)
 - ROPShellcode-d20-Exit (ROP'ed shellcode + run first with delayed execution for 20 secs)
 - Shellcode (intact shellcode)

Evasion rate: reverse TCP shell

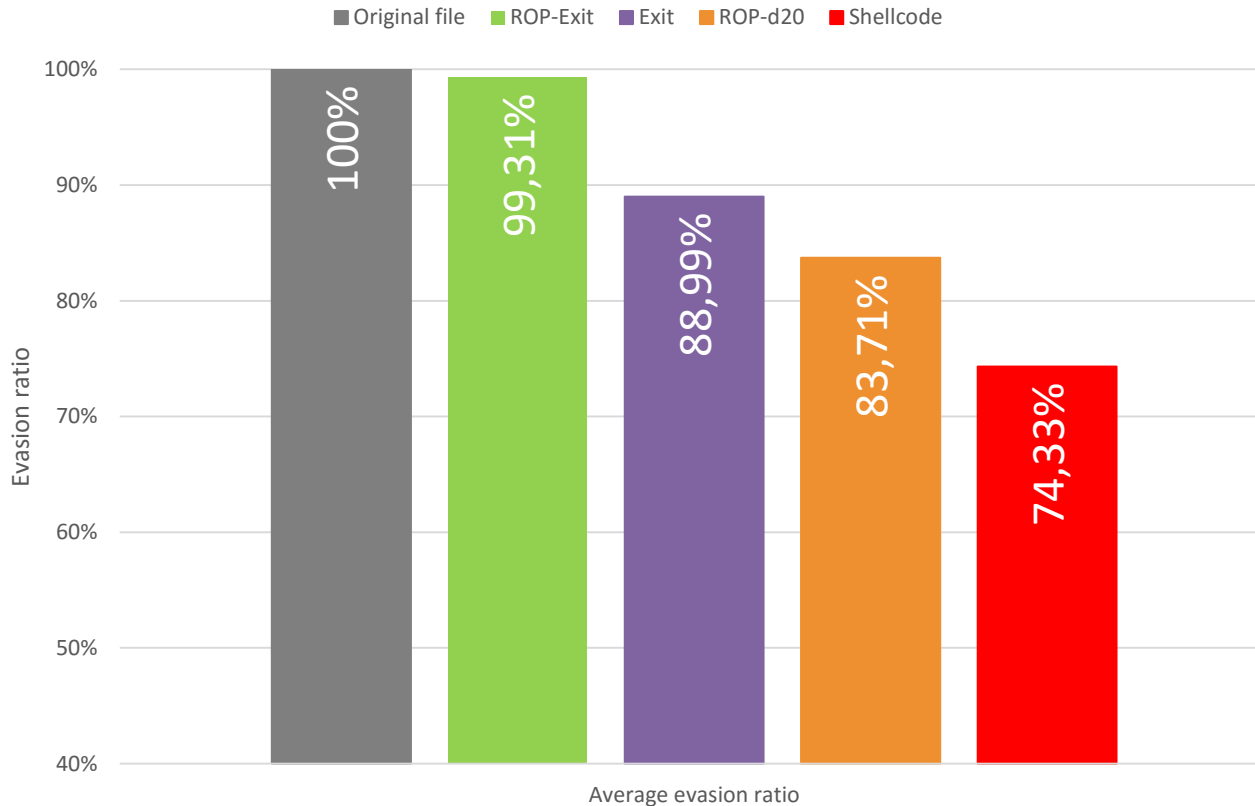


Evasion rate: meterpreter reverse TCP



Overall evasion results

- 100% most of the times
- 99.31% on average



- Microsoft's Enhanced Mitigation Experience Toolkit (**EMET**) is a freeware security toolkit for Microsoft Windows .
- It can be used as an extra layer of defense against malware attacks, after the firewall and before antivirus software.
- It can be used to detect ROP based exploits.

Thank you!

Questions?