Smashing the Stack

CeSeNA Security Team

University of Bologna
Ingegneria e Scienze Informatiche

December 13, 2014

Introduction I

Acknowledgement

A special thanks to **CeSeNA Security** group and *Marco Ramilli* our "old" mentor...

Where to find us

- Website: `http://cesena.ing2.unibo.it/`
- GitHub: `https://github.com/cesena`
- We have also an IRC channel on freenode and a Google Group!

Introduction II

Before smashing things

We need to say some words about security in general :) !

Introduction III

Security facts in modern era

- Each security breach costs over 500k to Corporates
  `http://goo.gl/RAUgOg`
- Cyber-Security market is growing (*63 billion in 2011, 120 billions in 2017*)
  `http://goo.gl/Zq8Efj`
- Zero-Day exploit black markets, and Bug-Bounty (*yes Microsoft is doing it too*)

Introduction IV

Is someone still using C

Lot of C/C++ out there.. `http://langpop.com/` `http://www.tiobe.com/`

Buffer OverFlows are old stuff

Who	*Adobe Reader and Acrobat*
What	*stack-based buffer overflow*
When	**2014**

Check CVE-2014-8460

Smash the stack I

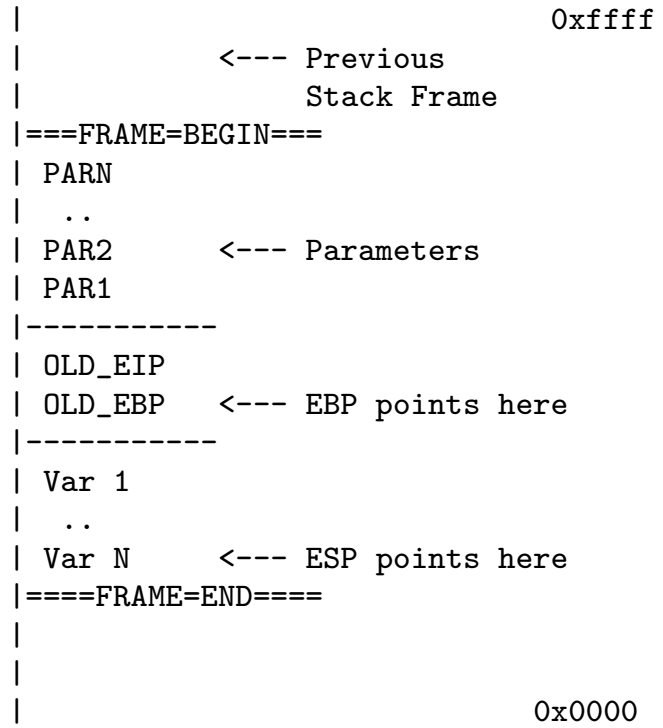
Smash The Stack [C programming] n.

- On many C implementations it is possible to **corrupt the execution stack** by writing past the end of an array declared auto in a routine.
- Code that does this is said to smash the stack, and *can cause return from the routine to jump to a random address*.

This can produce some of the most insidious data-dependent bugs known to mankind.

Stack Frame I

- Logical *frames* pushed during function calls and popped when returning.
- **stack frame** contains the function params, its local variables, and the necessary data for recovering previous frame.
- So it also contains the value of the **instruction pointer** at the time of the function call.
- Stack grows down (towards lower memory addresses)
- The stack pointer points to the last used address on the stack frame.
- The base pointer points to the bottom of the stack frame.



Stack Frame II

Stack in x86-x86_64

Stack grows in opposite direction w.r.t. memory addresses.
Also two registers are dedicated for stack management:

EBP/RBP , points to the **base** of the stack-frame (*higher address*)

EIP/RIP , points to the **top** of the stack-frame (*lower address*)

Who setup the stack frame?

Calling convention:

- Parameters are pushed by caller.
- *EIP* is pushed via *CALL instruction*.
- *EBP* and local vars are pushed by called function.

Valid for x86
x86-64 uses different convention (FAST-CALL)

Stack Frame III

Call Prologue and Epilogue

```
1  call fun    ;push EIP
```

```
1  fun:
2  push EBP ; prologue
3  mov EBP, ESP
4  sub ESP,<paramspace>
5  ...
6  ; epilogue
7  mov ESP, EBP
8  pop EBP ;restore EBP
9  ret      ;pop EIP
```

Stack Frame IV

Stack Frame: Recap

Logical stack frames that are *pushed in the .stack segment* on function call, popped when returning.
A stack frame contains:

- Parameters (depends on calling convention, not true for linux64)
- **Data for previous frame recovering, also old Instruction Pointer value**.
- Local variables

Stack Frame V

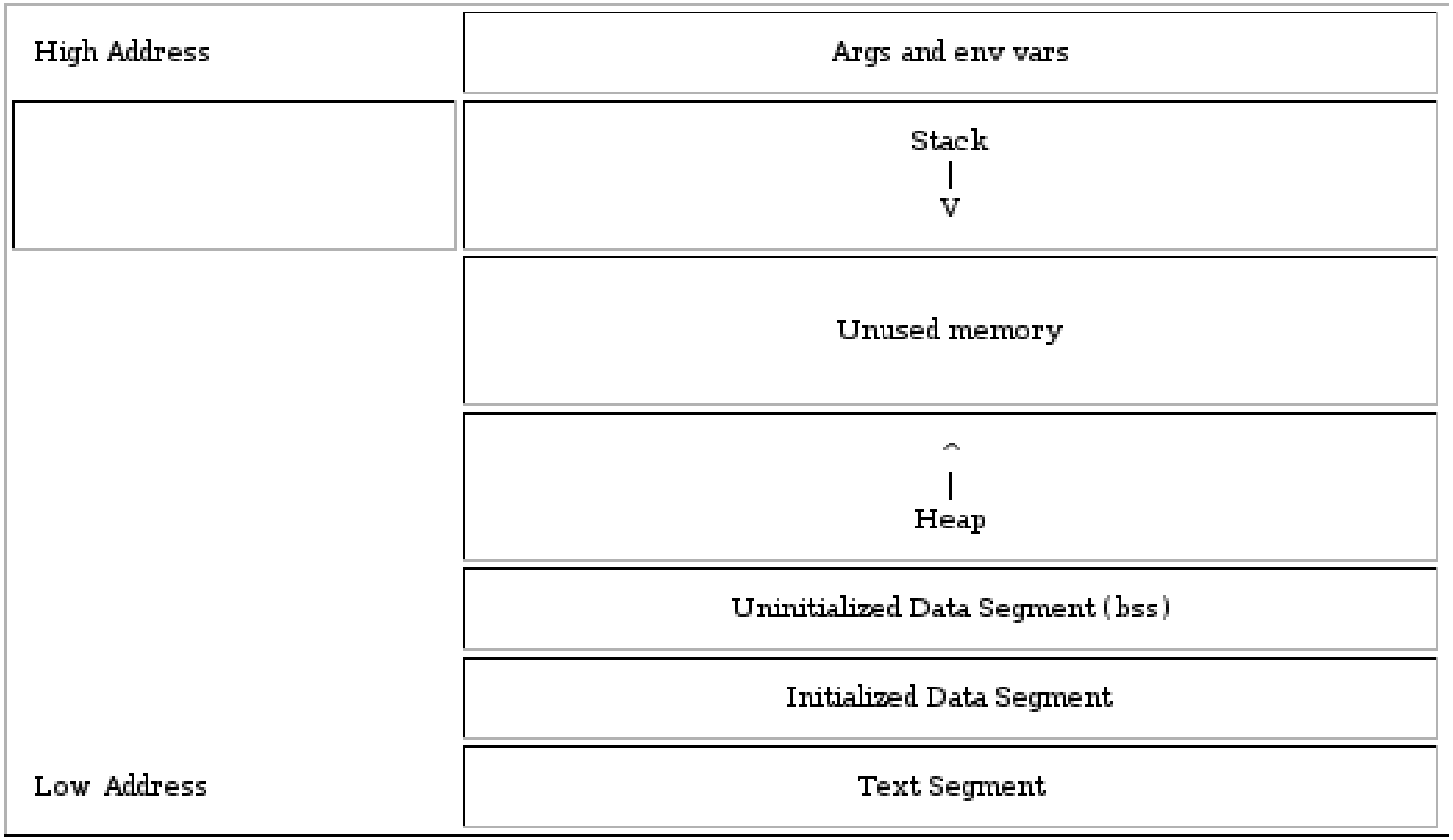


Figure: VM space