CSE 127: Computer Security

Control Flow Hijacking

Kirill Levchenko

October 12, 2017

Stack Buffer Overflow

- **Stack buffer overflow:** writing past end of a stack-allocated buffer
  - Also called *stack smashing*
- One of the simplest control flow hijacking mechanisms
- Stack buffer overflow vulnerabilities still exist
  - Mainly embedded devices and legacy software
  - Newer OSes provide protection against common types

NOVEMBER 4, 2014

CVE-2014-1635 BELKIN N750 BUFFER OVERFLOW

1. Vulnerability Properties

Title: Belkin n750 buffer overflow

CVE ID: CVE-2014-1635

CVSSv2 Base Score: 10 (AV:N/AC:L/AU:N/C:P/I:N/A:N)

Vendor: BELKIN (http://www.belkin.com/)

Products: n750/F9K1103

Advisory Release Date: 2014-11-04

Advisory URL: https://labs.integrity.pt/advisories/CVE-2014-1635/

Credits: Discovery and PoC by Marco Vaz <mv[at]integrity.pt>

2. Vulnerability Summary

A remote unauthenticated attacker may execute commands as root by sending an unauthenticated crafted

Title: Belkin n750 buffer overflow

CVE ID: CVE-2014-1635

CVSSv2 Base Score: 10 (AV:N/AC:L/AU:N/C:P/I:N/A:N)

Vendor: BELKIN (http://www.belkin.com/)

Products: n750/F9K1103

Advisory Release Date: 2014-11-04

Advisory URL: https://labs.integrity.pt/advisories/CVE-2014-1635/

Credits: Discovery and PoC by Marco Vaz <mv[at]integrity.pt>

2. Vulnerability Summary

A remote unauthenticated attacker may execute commands as root by sending an unauthenticated crafted POST request to the httpd that serves authentication on the guest login network.

3. Technical Details

The vulnerability can be confirmed by sending a crafted POST request where the parameter "jump" takes 1379 bytes of padding concatenated with the commands to be executed and with content different from zero to overwrite an internal control variable.

Buffer Overflow Defenses

- Avoid unsafe functions
- Stack canary
- Separate control stack
- Address Space Layout Randomization (ASLR)
- Memory writable or executable, not both (W^X)
- Control flow integrity (CFI)

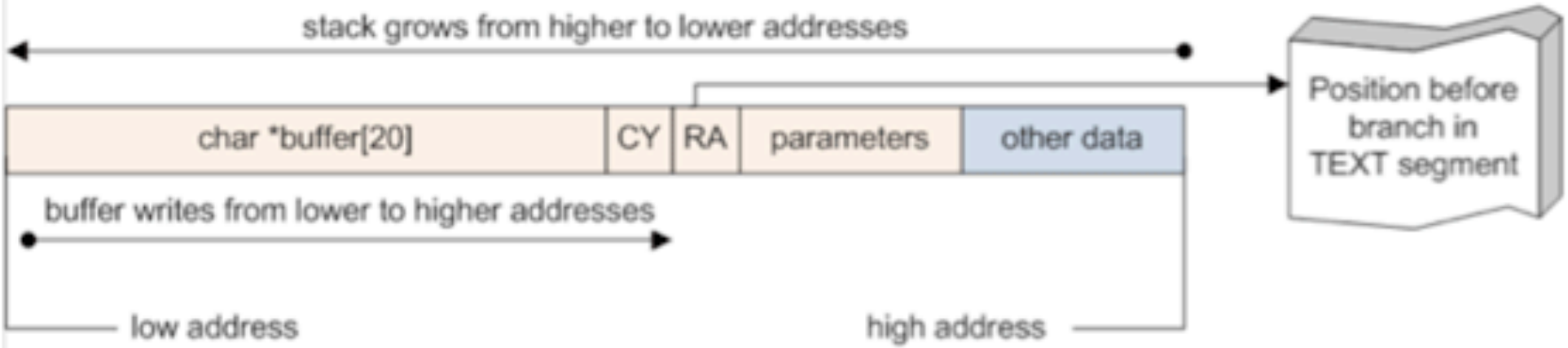
Avoiding Unsafe Functions

- `strcpy`, `strcat`, `gets`, *etc.*
- **Plus:** Good idea in general
- **Minus:** Requires manual code rewrite
- **Minus:** Non-library functions may be vulnerable also
  - E.g. user creates her own `strcpy`
- **Minus:** No guarantee you found everything

Stack Canary

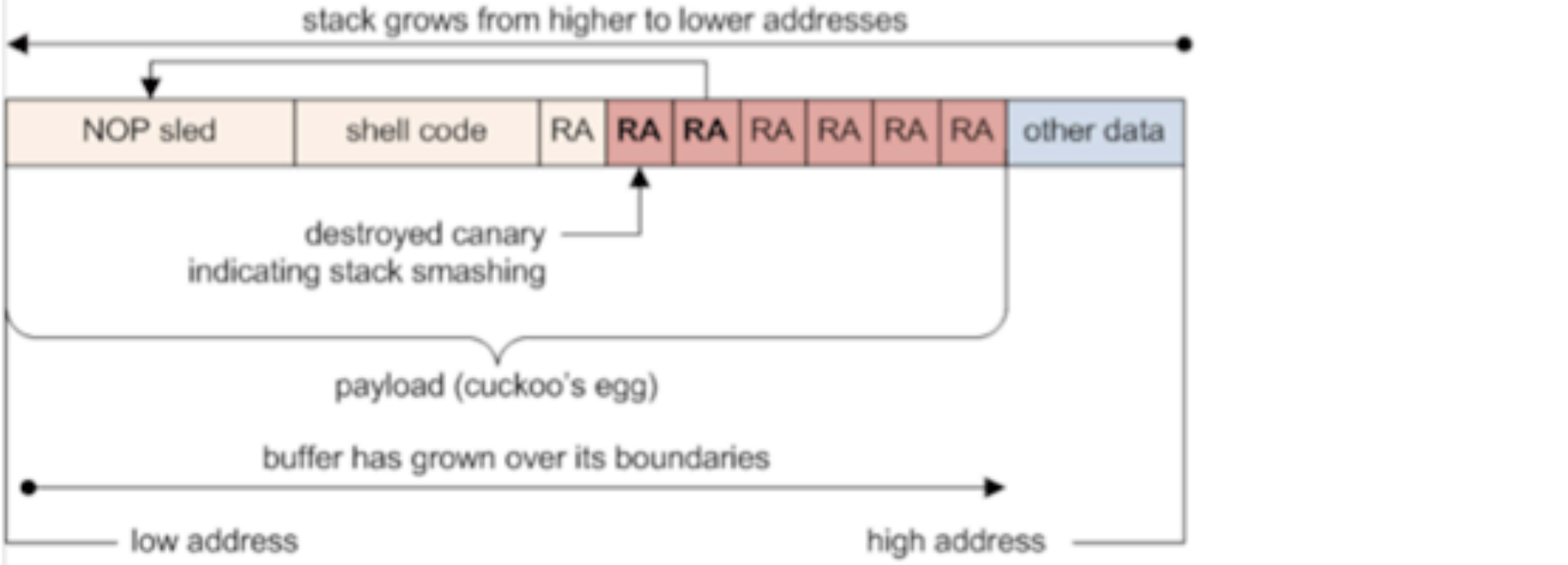
- Special value placed before return address on stack
  - Secret random value chosen at program start
  - String terminator '`\0`'
- Gets overwritten during buffer overflow
- Check canary before jumping to return address
- Automatically inserted by compiler
  - **GCC:** `-fstack-protector` or `-fstack-protector-strong`

Stack before overflow with canary



RA = return address
CY = canary

Stack after overflow attack with destroyed canary



Stack Canary

- **Plus:** No code changes required, only recompile
- **Minus:** Small performance penalty before every return
- **Minus:** Only protects against stack buffer overflows
- **Minus:** Random canary fails if attacker can read memory

Separate Stack

"SafeStack is an instrumentation pass that protects programs against attacks based on stack buffer overflows, without introducing any measurable performance overhead. It works by separating the program stack into two distinct regions: the safe stack and the unsafe stack. The safe stack stores **return addresses**, **register spills**, and **local variables that are always accessed in a safe way**, while the unsafe stack stores everything else. This separation ensures that buffer overflows on the unsafe stack cannot be used to overwrite anything on the safe stack."

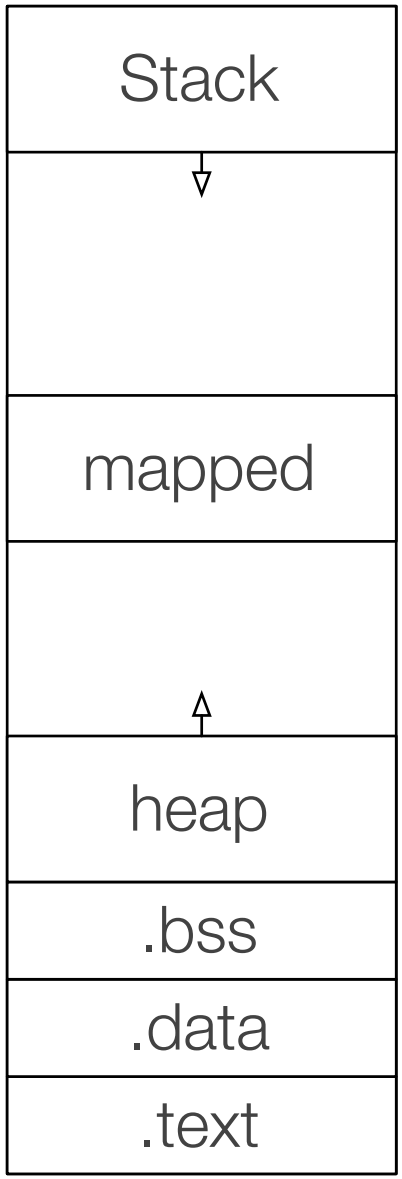
- Requires compiler support

ASLR

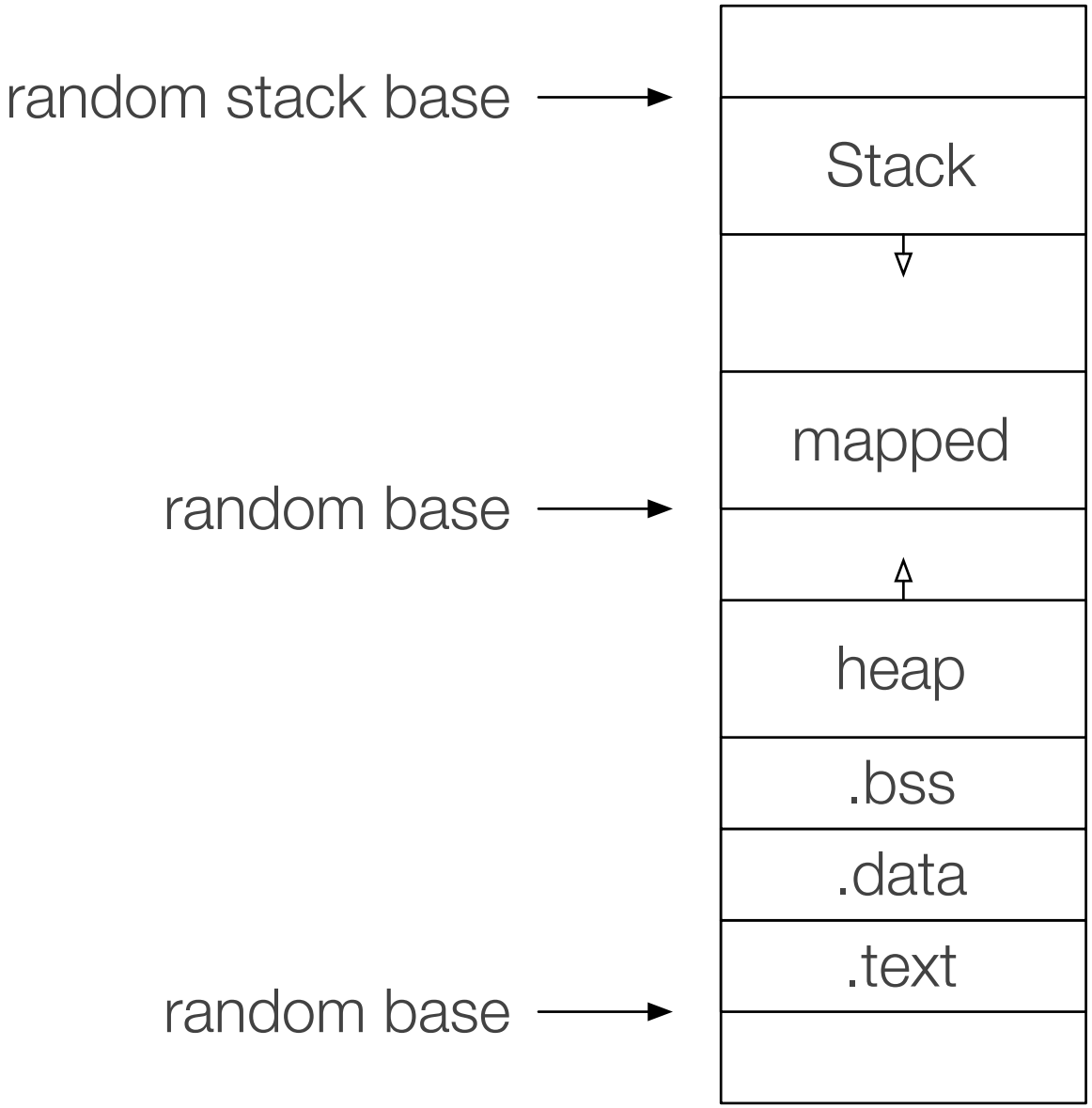
Address Space Layout Randomization

- Change location of stack, heap, code, static variables
- Works because attacker needs address of shellcode
- Layout must be unknown to attacker
  - Randomize on every launch (best)
  - Randomize at compile time
- Implemented on most modern OSes in some form

Traditional Memory Layout

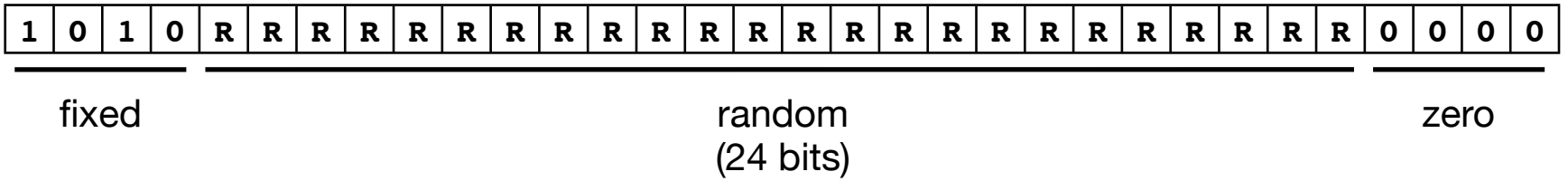


PaX Memory Layout

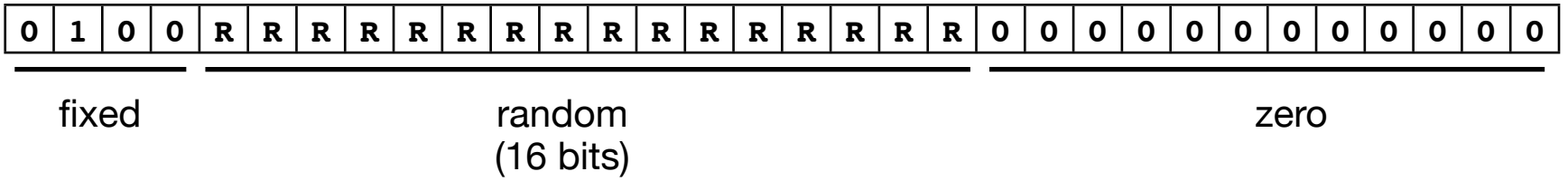


32-bit PaX ASLR (x86)

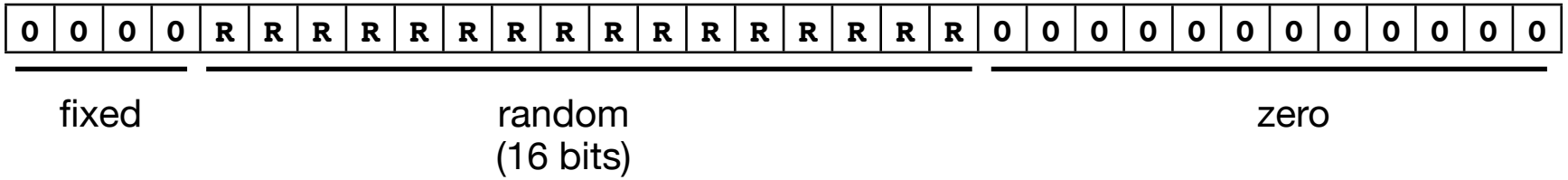
Stack:



Mapped area:



Executable code, static variables, and heap:



ASLR

Address Space Layout Randomization

- **Plus:** No code changes or recompile required
- **Minus:** 32-bit architectures get limited protection
- **Minus:** Fails if attacker can read memory
- **Minus:** Load-time overhead for strongest protection
  - No executable image sharing between processes

W^X

"Write XOR Execute"

- Use hardware memory protection to ensure memory cannot be both writeable and executable at same time
- Code is executable, not writeable
- Stack, heap, static variables writeable, not executable
- Supported by most modern processors
  - AMD64 and later; also available via segments since 80286
- Implemented by modern operating systems

W^X

- **Plus:** No code changes or recompile required
- **Minus:** Requires hardware support
- **Minus:** Breaks self-modifying code (extremely rare)
- **Minus:** Defeated by return-oriented programming
- **Minus:** Does not protect just-in-time compiled code

Control Flow Integrity

- Check destination of every indirect jump
  - Function returns
  - Function pointers
  - Virtual methods
- What are the valid destinations?
  - Caller of every function known at compile time
  - Class hierarchy limits possible virtual function instances

CFI

- **Plus:** No code changes or hardware support
- **Plus:** Protects against many vulnerabilities
- **Minus:** Performance overhead
- **Minus:** Requires smarter compiler
- **Minus:** Requires having all code available