CSA2 - Security

Software Security

Topics – today

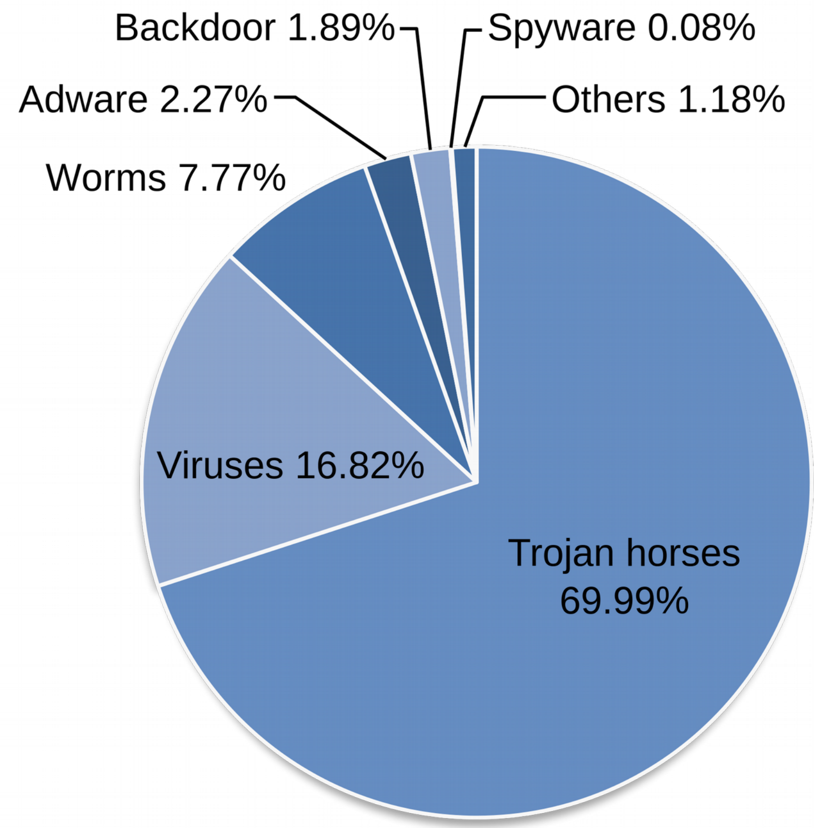
- **Malware / WannaCrypt**
- **Software Security**
  - Virtualization / Containerization / Sandboxing
- **Most common software vulnerabilities**
  - Buffer overflow / Buffer over-read
  - Denial of Service
- **Attack counter measures**
  - NX-Bit / ASLR / Protected Mode
  - Software Signing / Secure Boot

Software Security

- Software can attack system
- Software can attack other software
- Vulnerabilities allow attackers to change a software´s behavior

Types of Malware

- **Adware**
- **Ransomware**
- **Virus**
- **Worm**
- **Trojan Horse**
- **Spyware**



Malware by categories March 16, 2011

Short for "malicious software," malware refers to software programs designed to damage or do other unwanted actions on a computer system.

How WannaCrypt uses encryption

- **Aka:** WannaCry, WanaCrypt0r, Wana Decrypt0r 2.0
- **Ransomware**

- Existing **server public-private key pair**
- Generate **client public-private key pair** on infection
- Encrypt client private key with server public key
- Generate **AES key** for <u>each</u> file to encrypt
- Encrypt each AES key with **client public key**

Virtualization

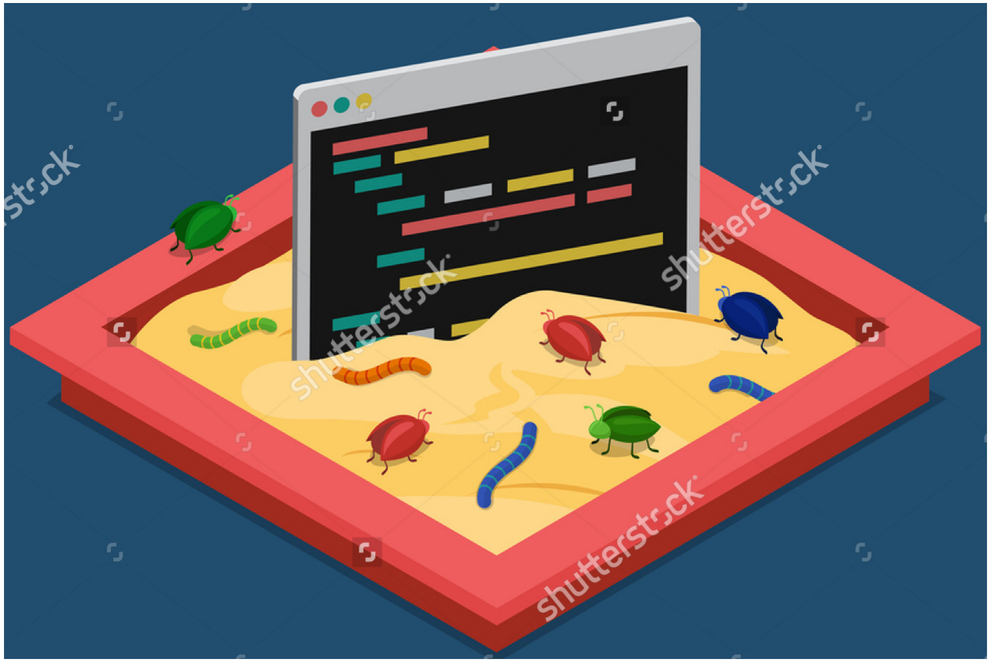
- **Fully independent execution environment**
- **Runs one or more separate systems**
  - Host System
  - Guest Systems
- **Can be supported by CPU**

Containerization

- **Partly independent execution environment**
- **Can use parts of host system**
- **Can share commonly used parts**
  - OS kernel, Device drivers
  - Libraries
  - Application Code

Sandboxing

- <u>**Managed**</u> **execution environment**
- **Independent of external applications**
- **The environment restricts**
  - Network Access
  - Memory Access
  - Access to kernel functions
  - Access to drivers
  - Access to file systems



Common types of vulnerabilities

- **Buffer overflow**
  - Stack overflow
- **Buffer over-read**
- **Memory Corruption**
  - Denial of Service (DoS)
- **Memory Leak**

Buffer Overflow

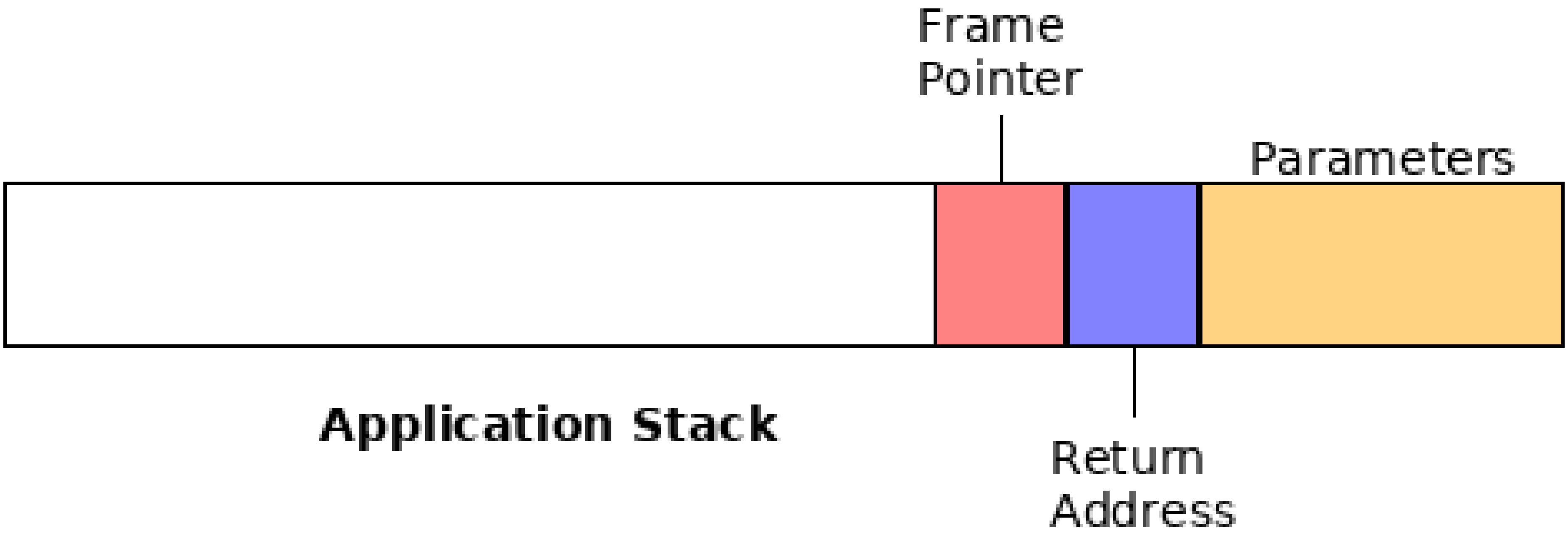
- **Allocate buffer of certain size**
- **Write more data than size into buffer**
  - Works on Stack & Heap
- **Influence information nearby in memory**
- Can happen with numbers on integer overflow

Stack overflow

- Write over the bounds of a buffer on the stack
- Overwrite return address
- Function will return to the injected address running the code there

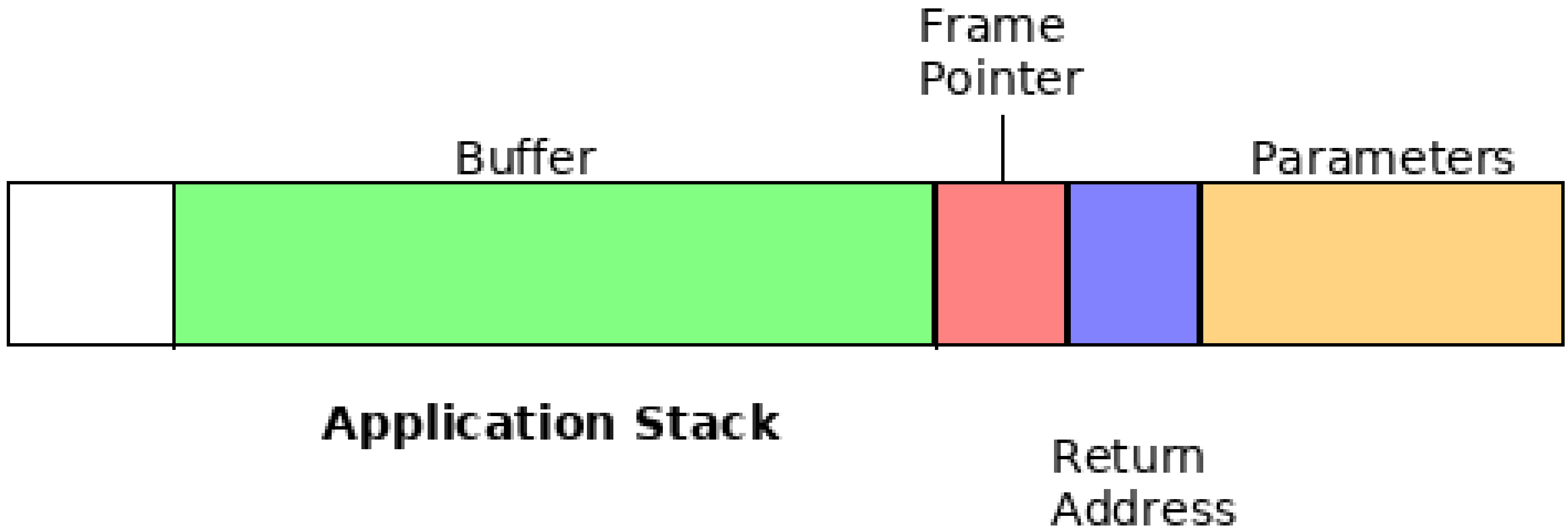
Stack overflow – Step 1

Stack contains return information.



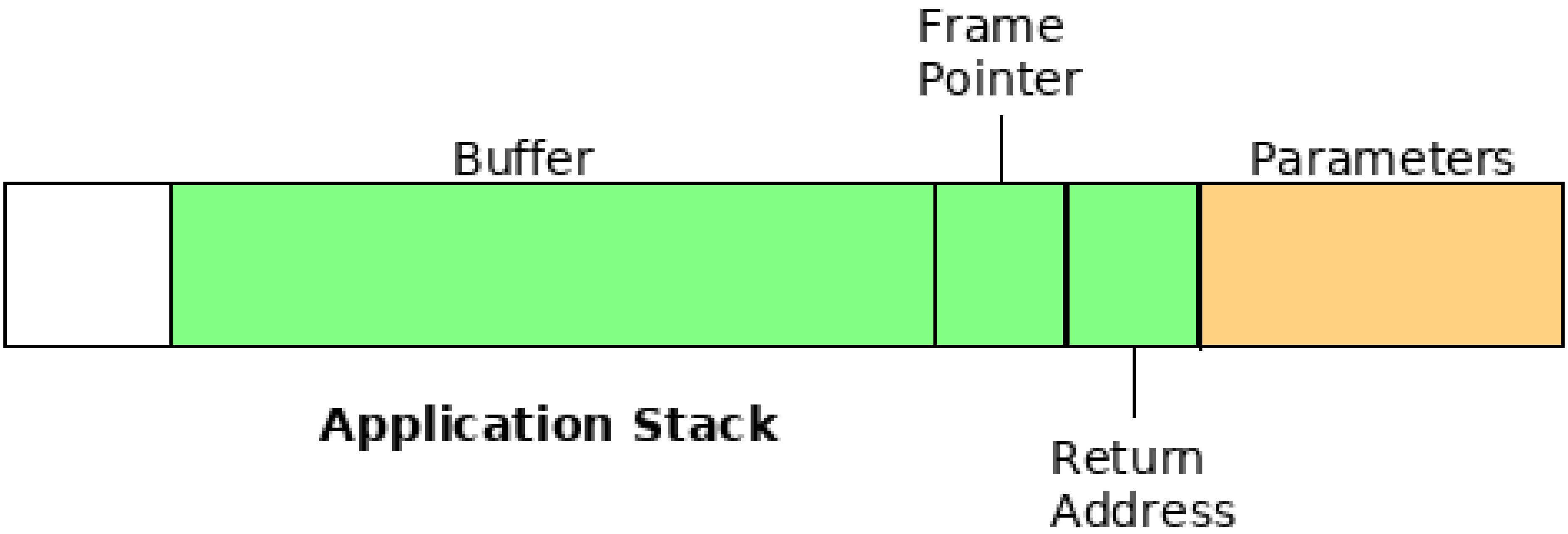
Stack overflow – Step 2

Buffer will be allocated behind return information.



Stack overflow – Step 3

Overwrite return address

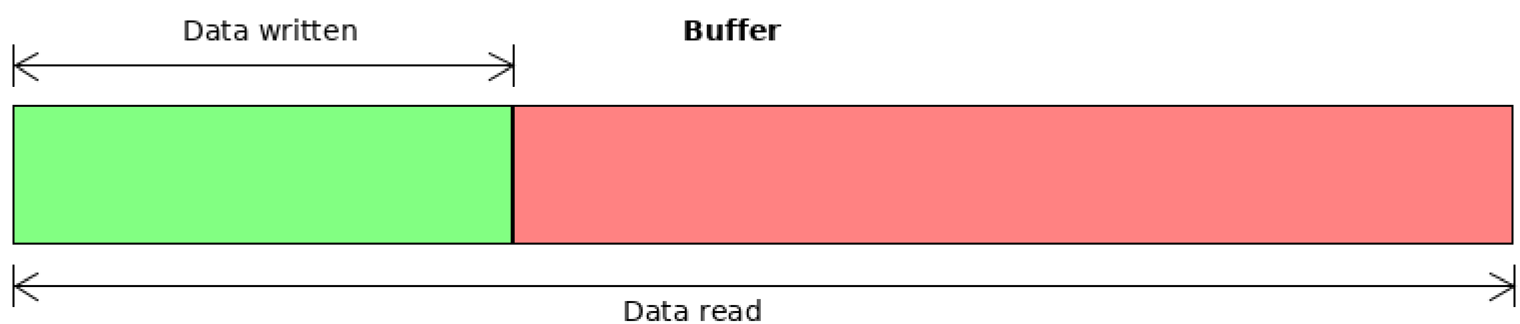


Application will execute code at new return address after function returns.

Buffer over-read

- **Allocate buffer of certain size**
- **Write less than allocated amount**
- **Read whole buffer**

Heartbleed



Memory Corruption / Memory Leak

- **Attacker is able to write arbitrary data into memory**
  - Can causes <u>Denial of Service</u>

- **Attacker is able to read data from memory**
  - Can cause <u>information disclosure</u>

Rowhammer

Attack counter measures

- **Technical measures to <u>prevent</u> or <u>complicate</u> software attacks**
- **Can be done in:**
  - Hardware
  - Operating System
    - Kernel
    - User Space
  - Compiler
  - Runtime Environment
    - JVM, ART, CLR, Node.js, Zend Engine, ...

No Execution Bit (NX-Bit)

- Mark data with special bit
- Done in page table
- Marked address space will not be executed
- Not used anymore

Compiler Stack Protector

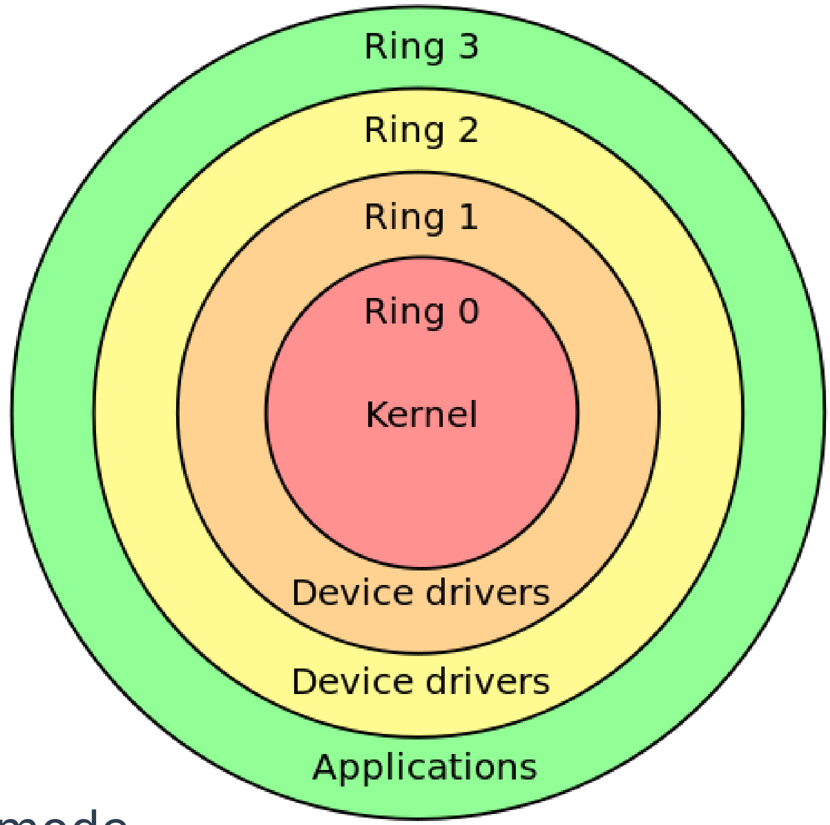
- **Compiler includes stack protection**
  - Checks for boundaries & overflows
- **Compiler places buffers after pointer**
  - Overflows do not affect return address pointer
- **Compiler includes random data into stack**
  - Data will be checked for changes
- **Automatically done by all modern compilers**
  - GCC, LLVM, Microsoft VS, Intel Compiler, ...

Address Space Layout Randomization

- **Randomize addresses in memory**
- **Complicate attacks using overflow flaws**
- **Randomize location of:**
  - Executable (code to run)
  - Stack
  - Heap
  - Libraries
- **Done by most modern operating systems**
  - Windows, Linux, Android, iOS, OS X, ...

Protected Mode

- **CPU can run on different modes** (protection rings)
- **Kernel Mode** (Ring 0)
- **Protected Mode** (Ring 3)
- Virtualization Mode (Ring -1)



- Restricts access to **hardware**
- Restricts access to **memory**
- Restricts access to **CPU functions**

Access via **system calls**

Disadvantage: **Context Switch** for changing mode

Software Signing / Code signing

- **Executable will be signed by CA**
  - e.g. Microsoft WHQL
- **Executable signed with own certificate**
  - Public Key will be integrated into executable
  - OS can check signature and ask to trust developer

What happens to certificates who run out of life?

Secure Boot

- **Kernel is <u>signed</u>**
- **All drivers are <u>signed</u>**
- **Bootloader is <u>signed</u>**

Where are the certificates?

Problems?

1. **BIOS** checks bootloader´s signature
2. **Bootloader** checks kernel´s signature
3. **Kernel** checks drivers signatures
4. **Kernel** checks software signatures

Certificates are included in UEFI BIOS.

Who decides which certificates are installed?

Questions?

Examples

- Examples available on Github: https://github.com/lukeelten/csa2-examples