Lecture 1:
Introduction and Overview

Sanchuan Chen

schen@auburn.edu

8/16/2023

Software Becomes Inceasingly Sophiscated

Software Becomes Inceasingly Sophiscated

Software Becomes Inceasingly Sophiscated

System Software Developed w/ Unsafe Languages

C was designed in 1970s

1. Less networked, less attacks
2. Favor performance and flexibility
3. Lack of security features
   - No automatic memory management
   - No strong typing
   - No bounds checks
   - No overflow checks

System Software Developed w/ Unsafe Languages

C was designed in 1970s

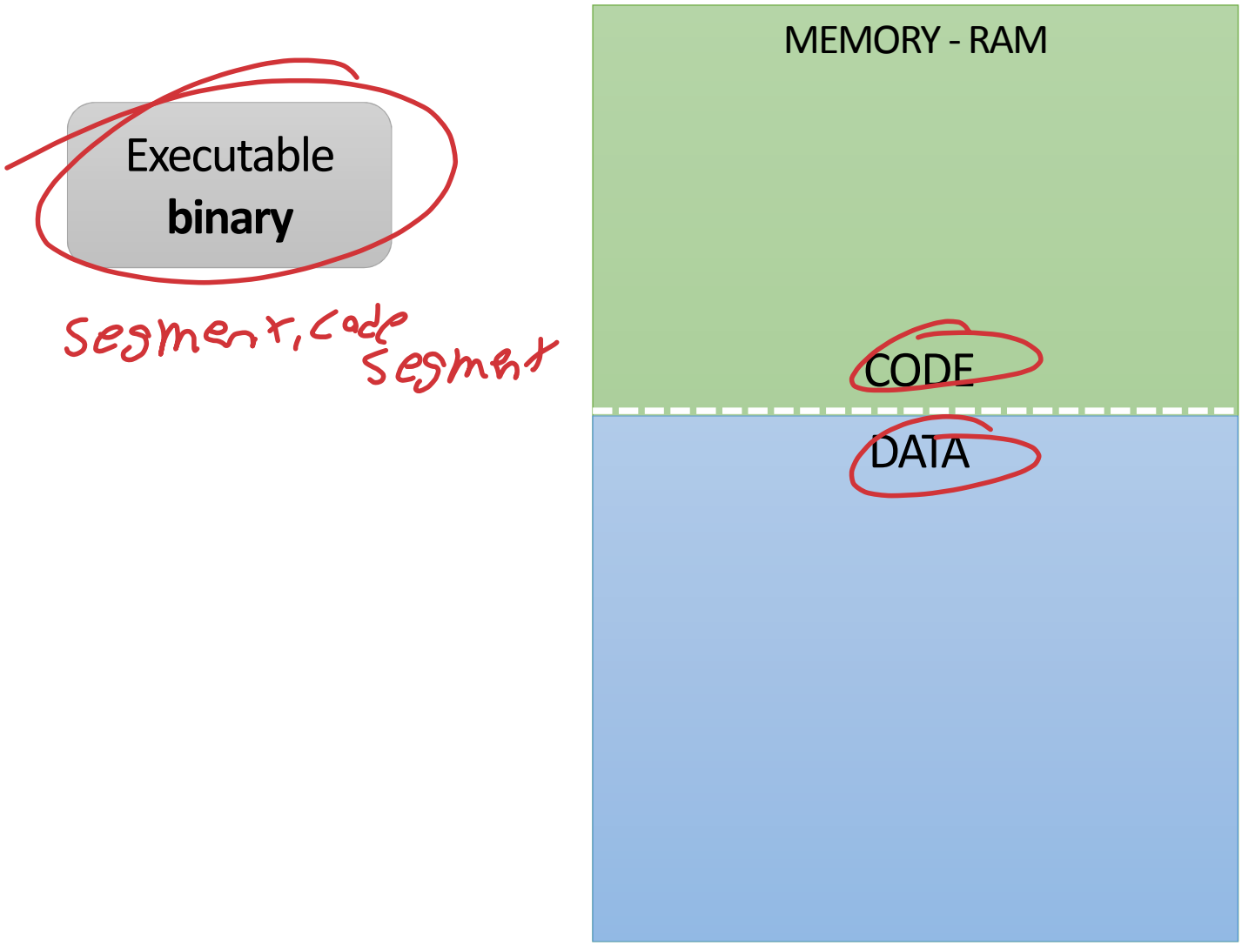
1. Less networked, less attacks
2. Favor performance and flexibility
3. Lack of security features
   - No automatic memory management
   - No strong typing
   - No bounds checks
   - No overflow checks

Why still being used?

1. Performance
   - C/C++/Object-C
   - Hand-written assembly
   - Optimal use of the hardware
2. Backwards-compatibility
   - Legacy systems
   - Most of them were developed in C

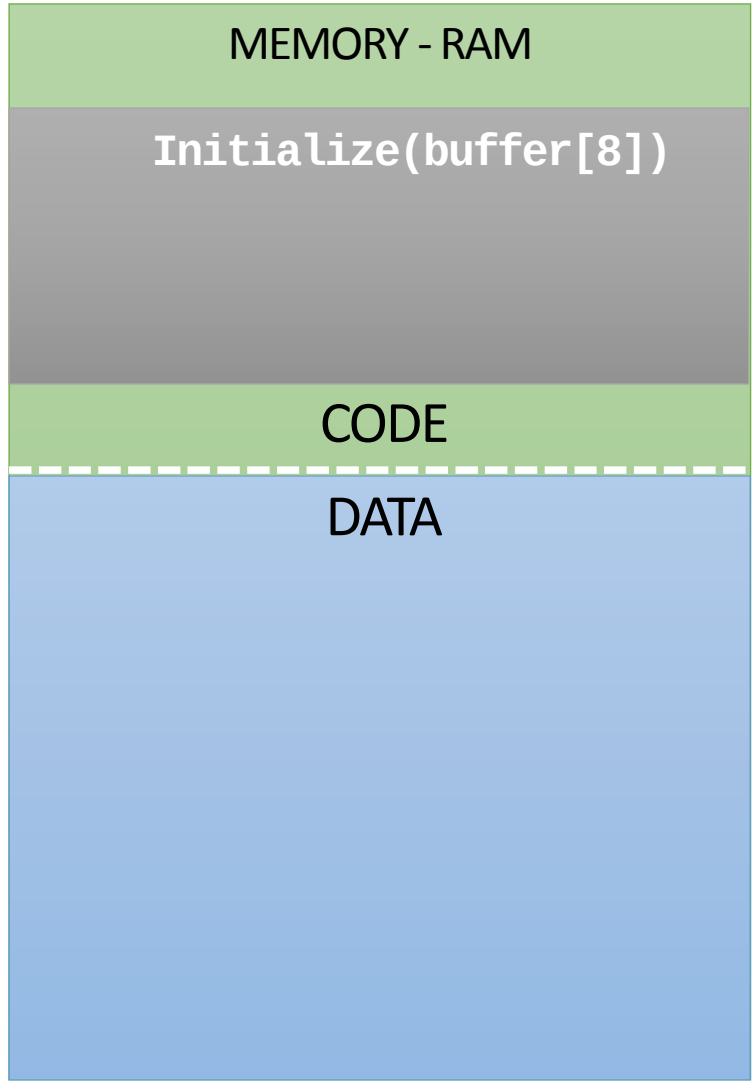
The Memory Corruption Vulnerability

mem corruption

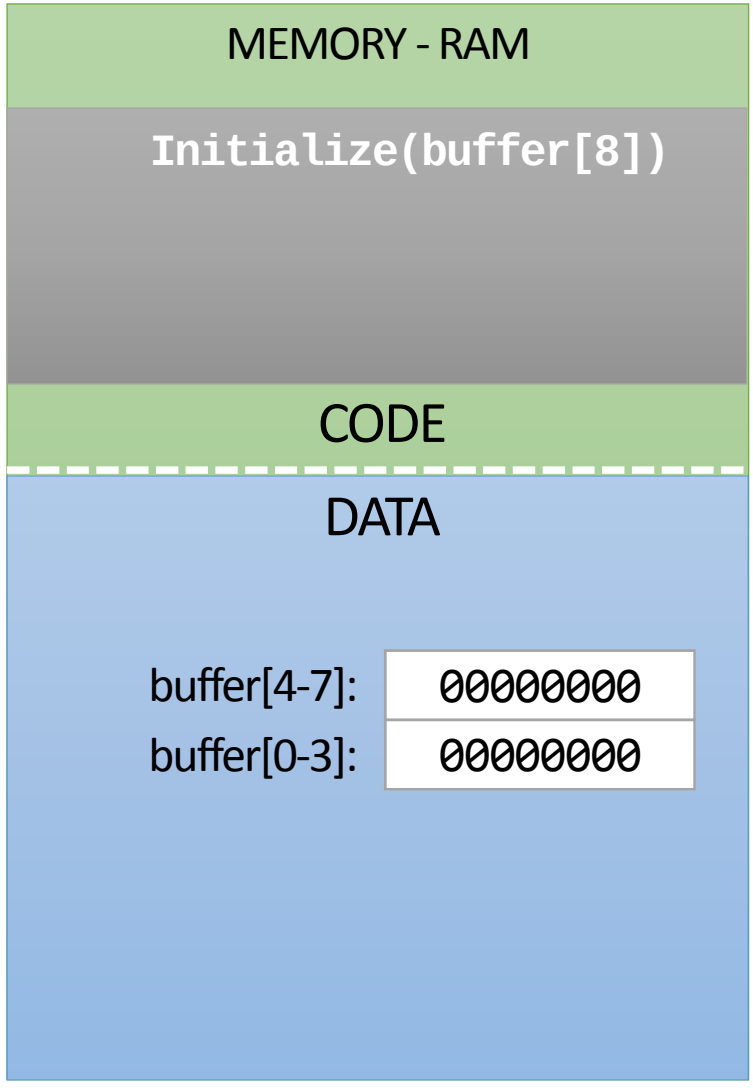


The Memory Corruption Vulnerability

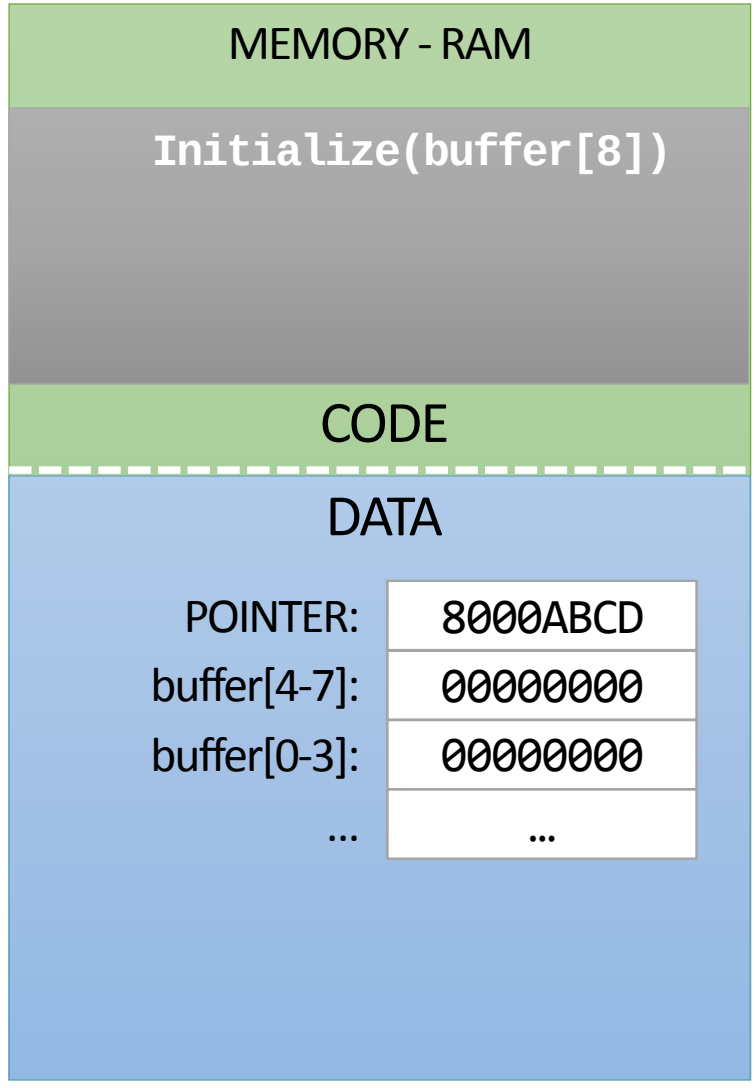
High level description



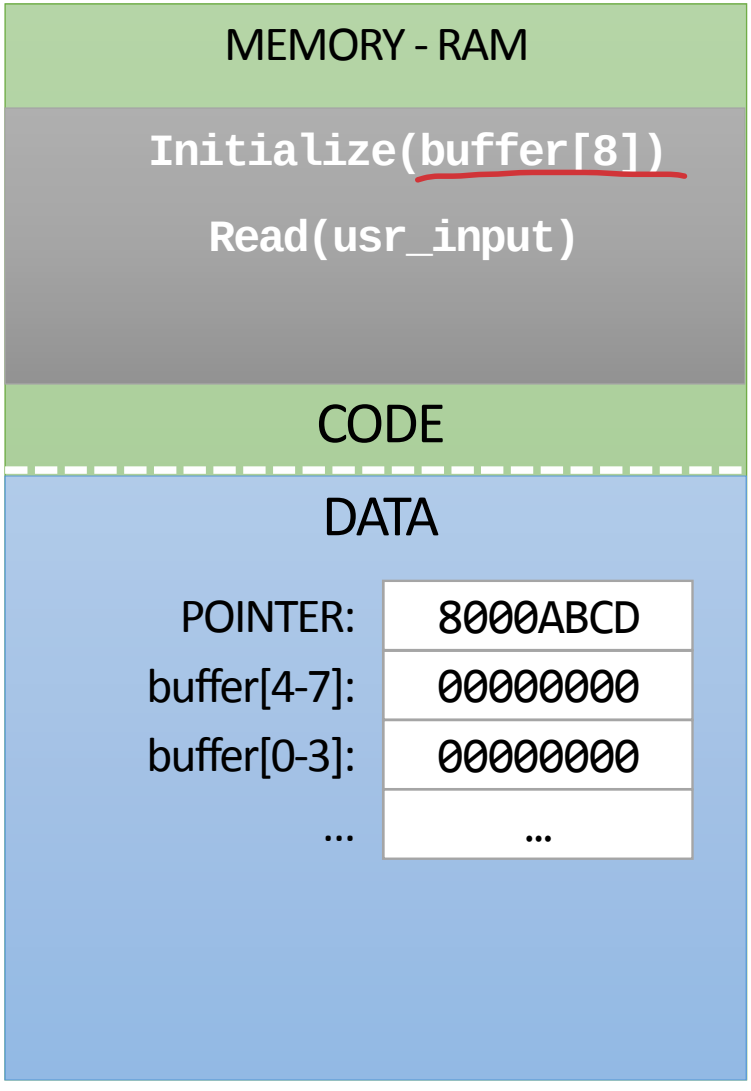
The Memory Corruption Vulnerability



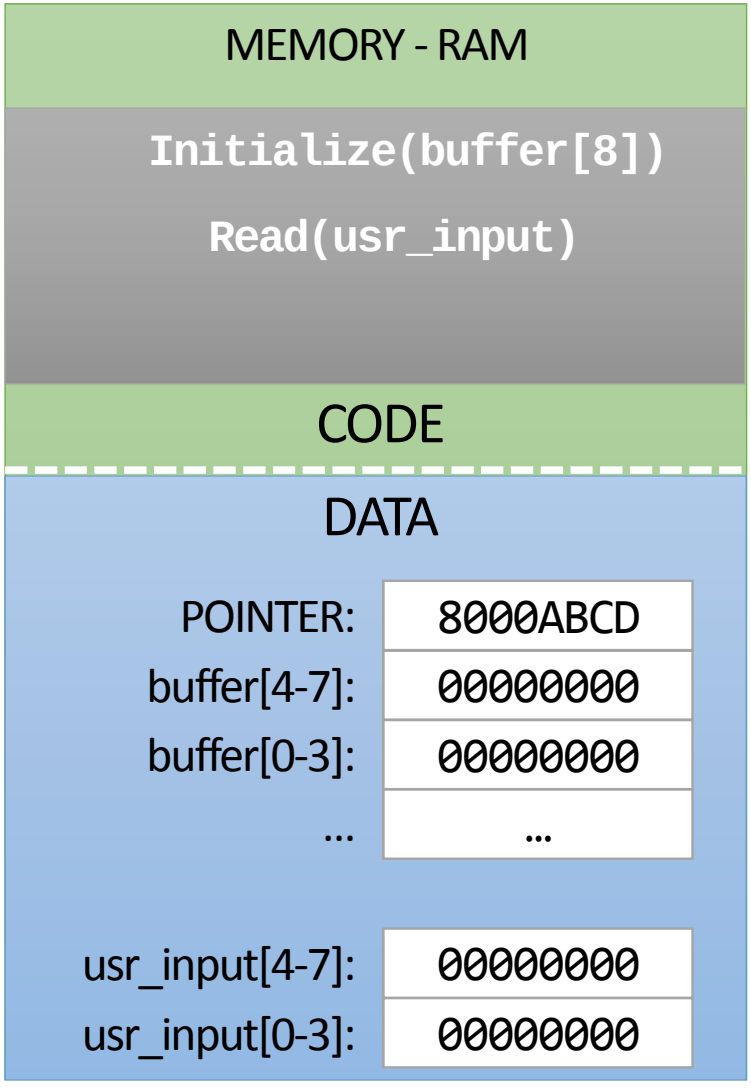
The Memory Corruption Vulnerability



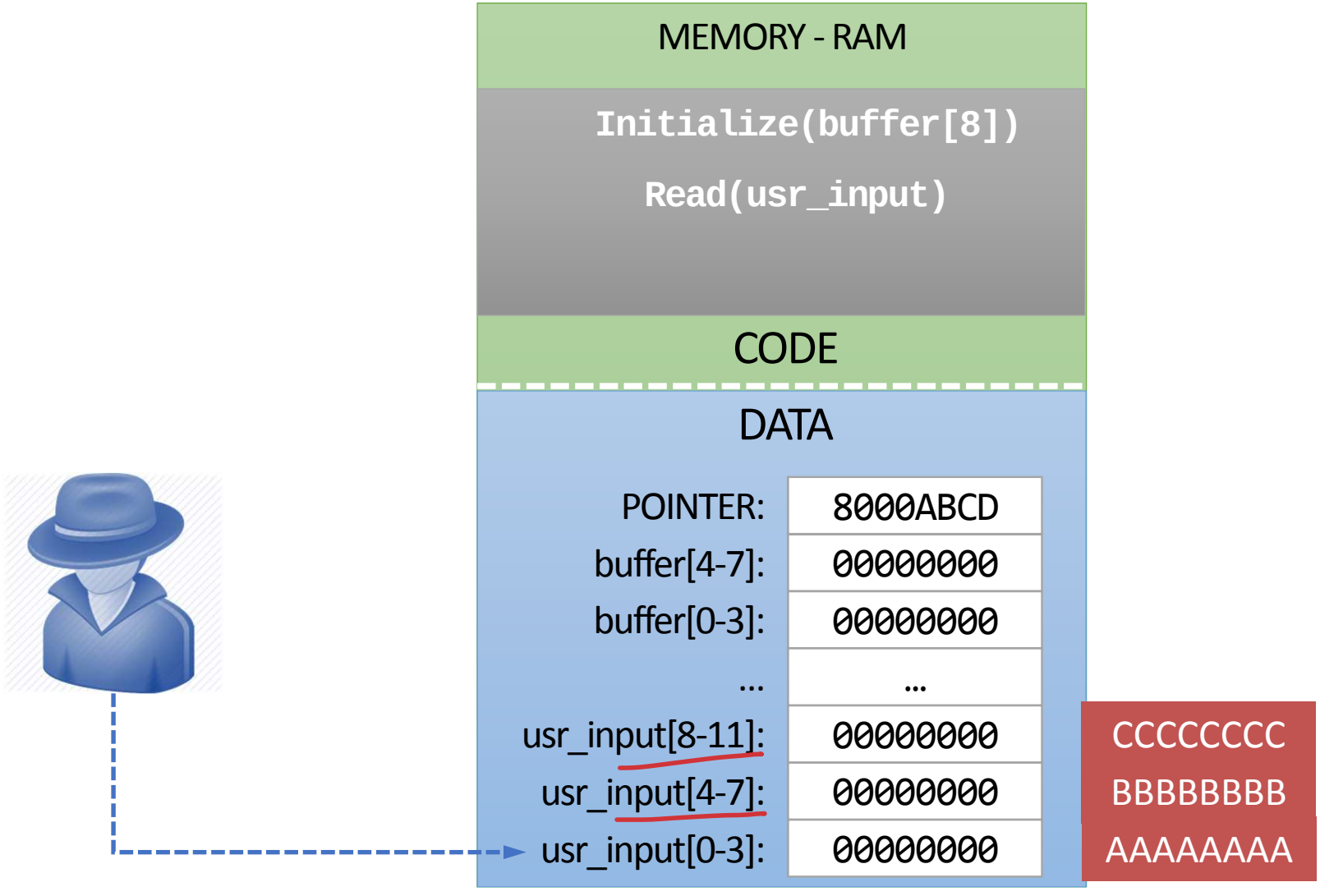
The Memory Corruption Vulnerability



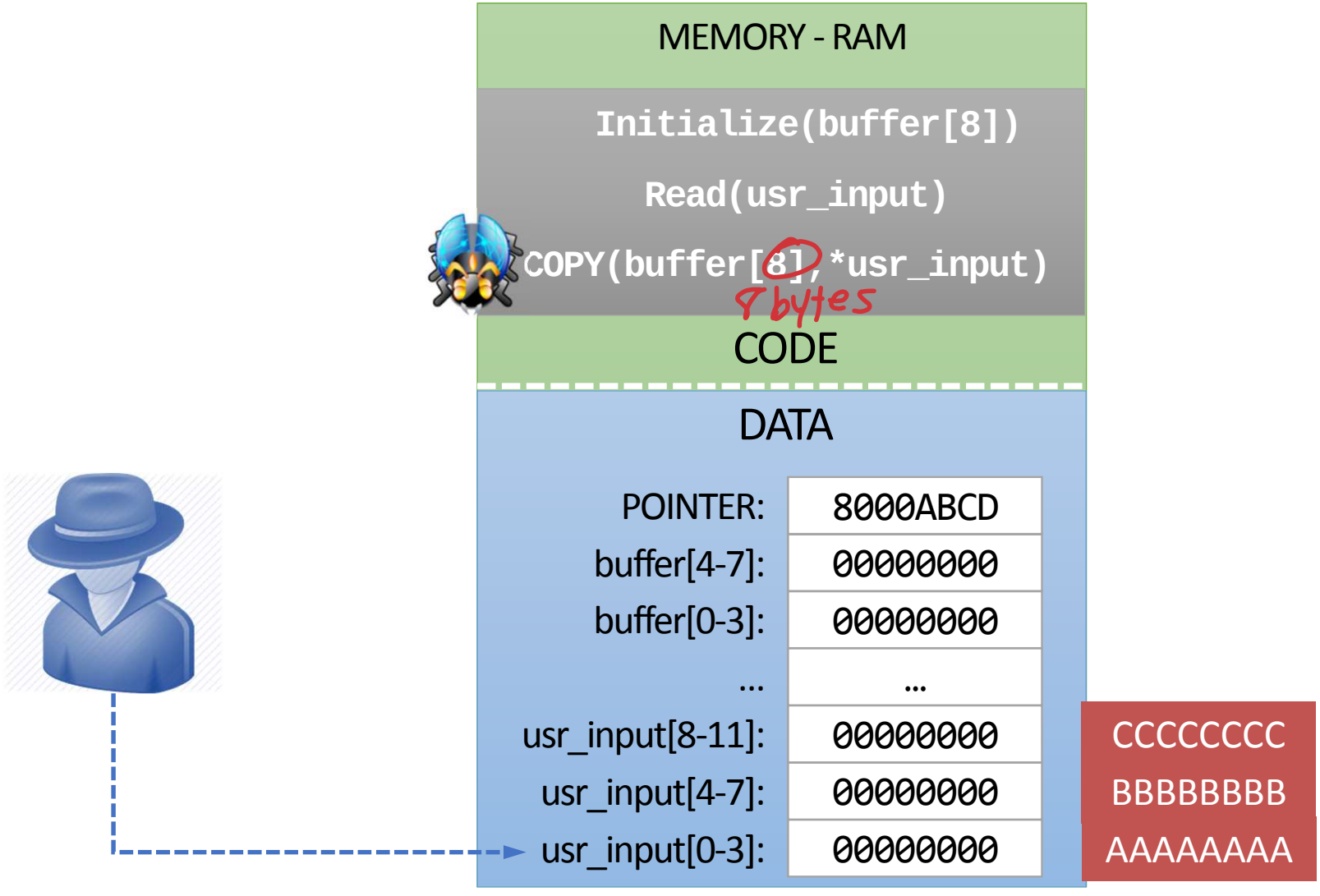
The Memory Corruption Vulnerability



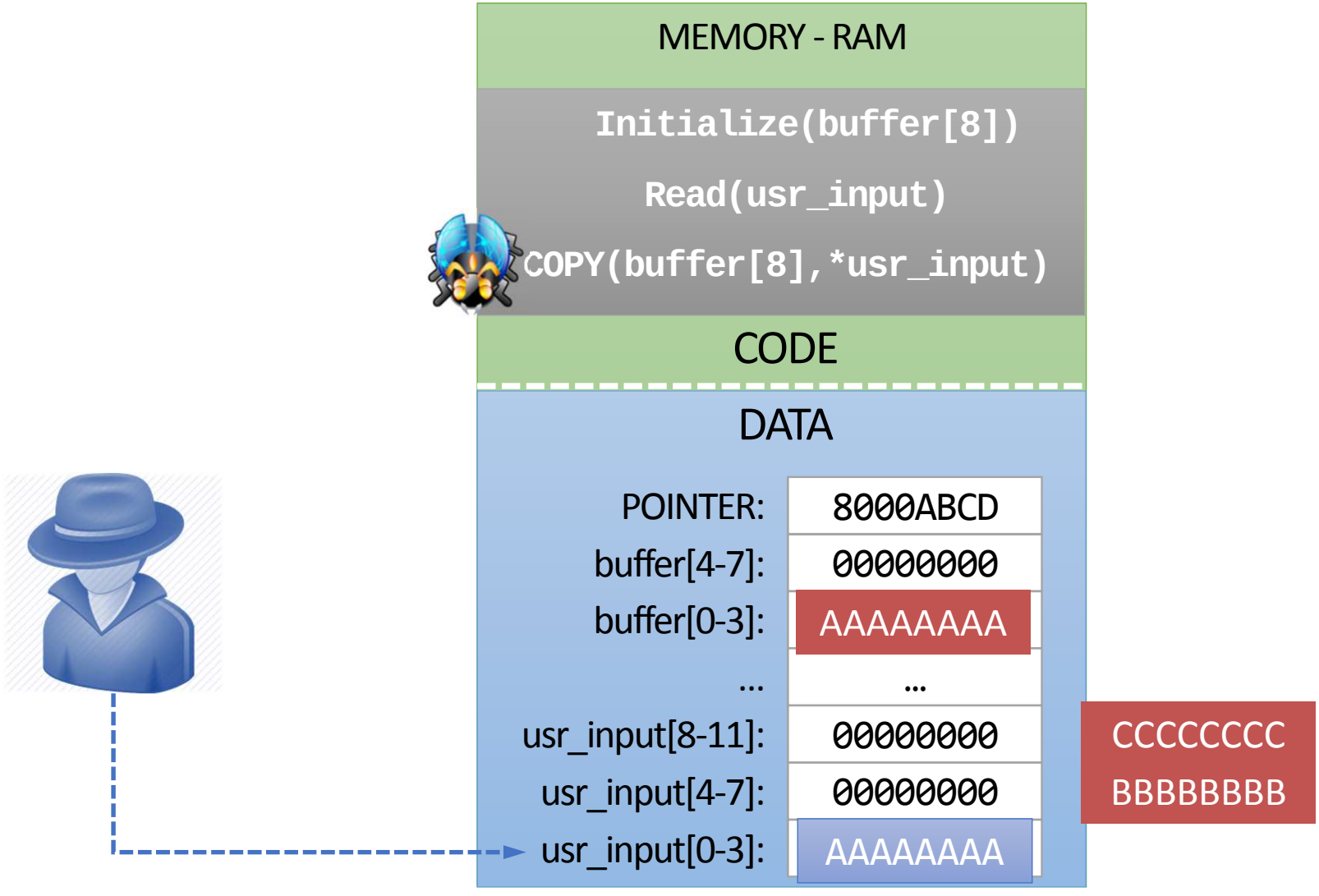
The Memory Corruption Vulnerability



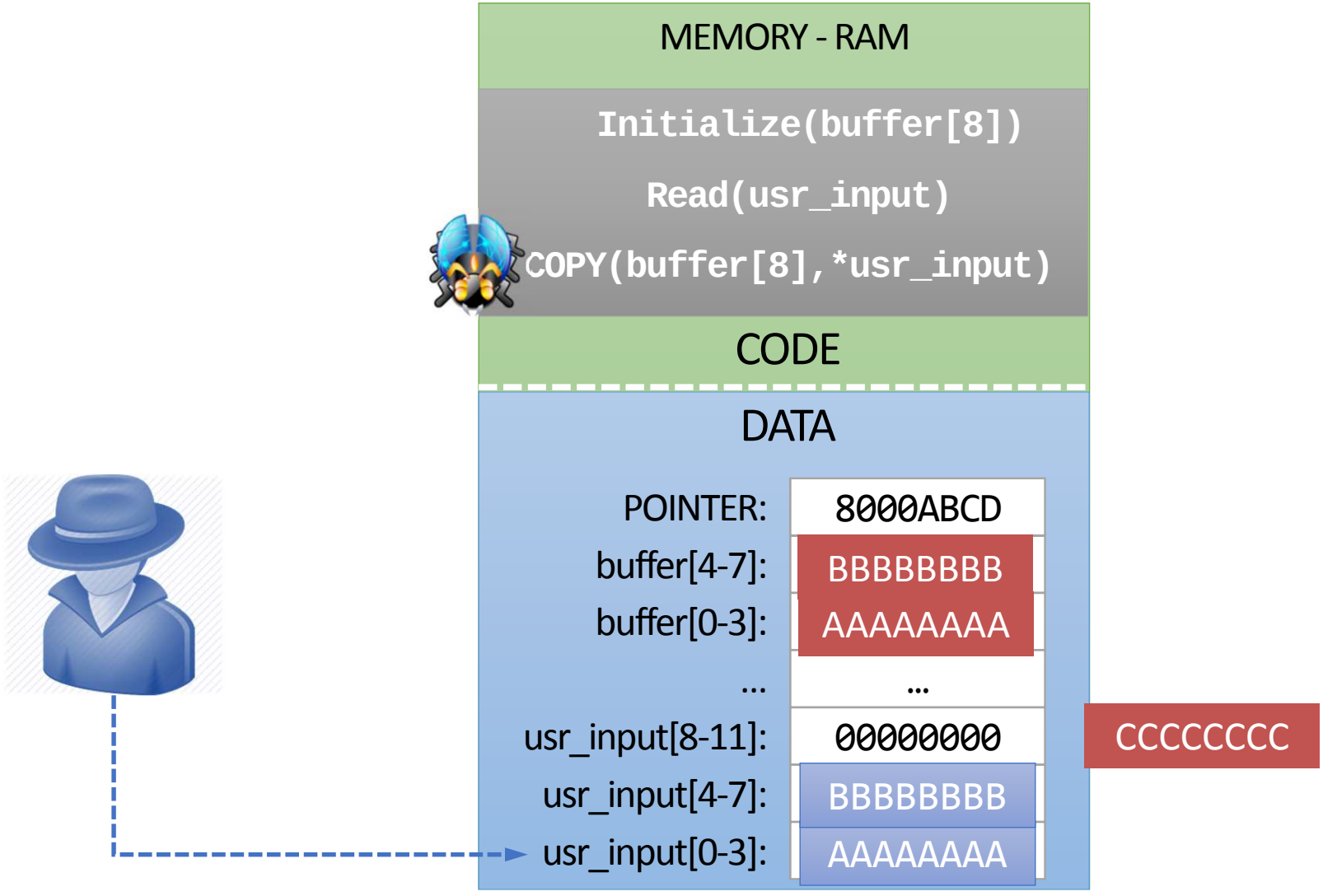
The Memory Corruption Vulnerability



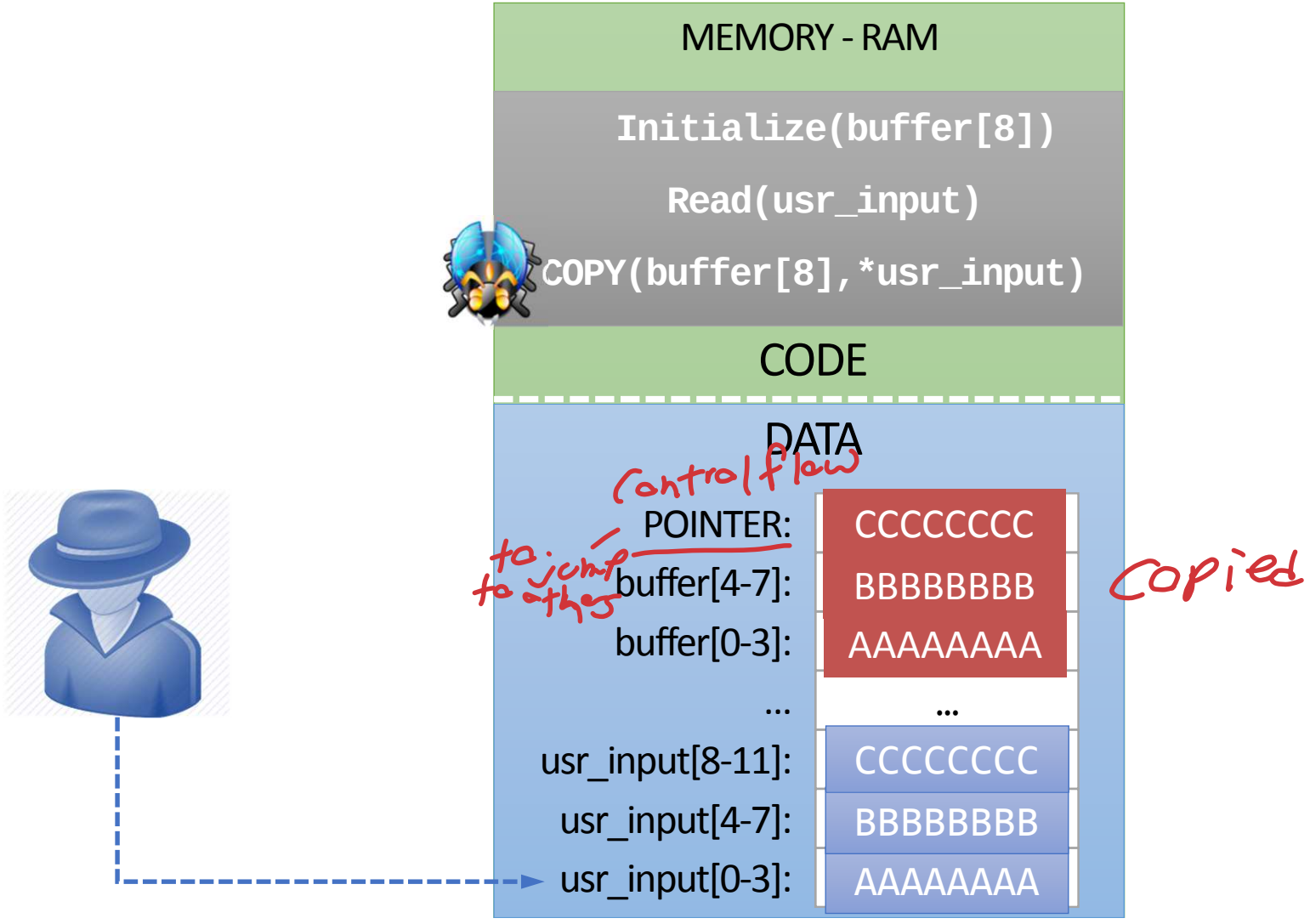
The Memory Corruption Vulnerability



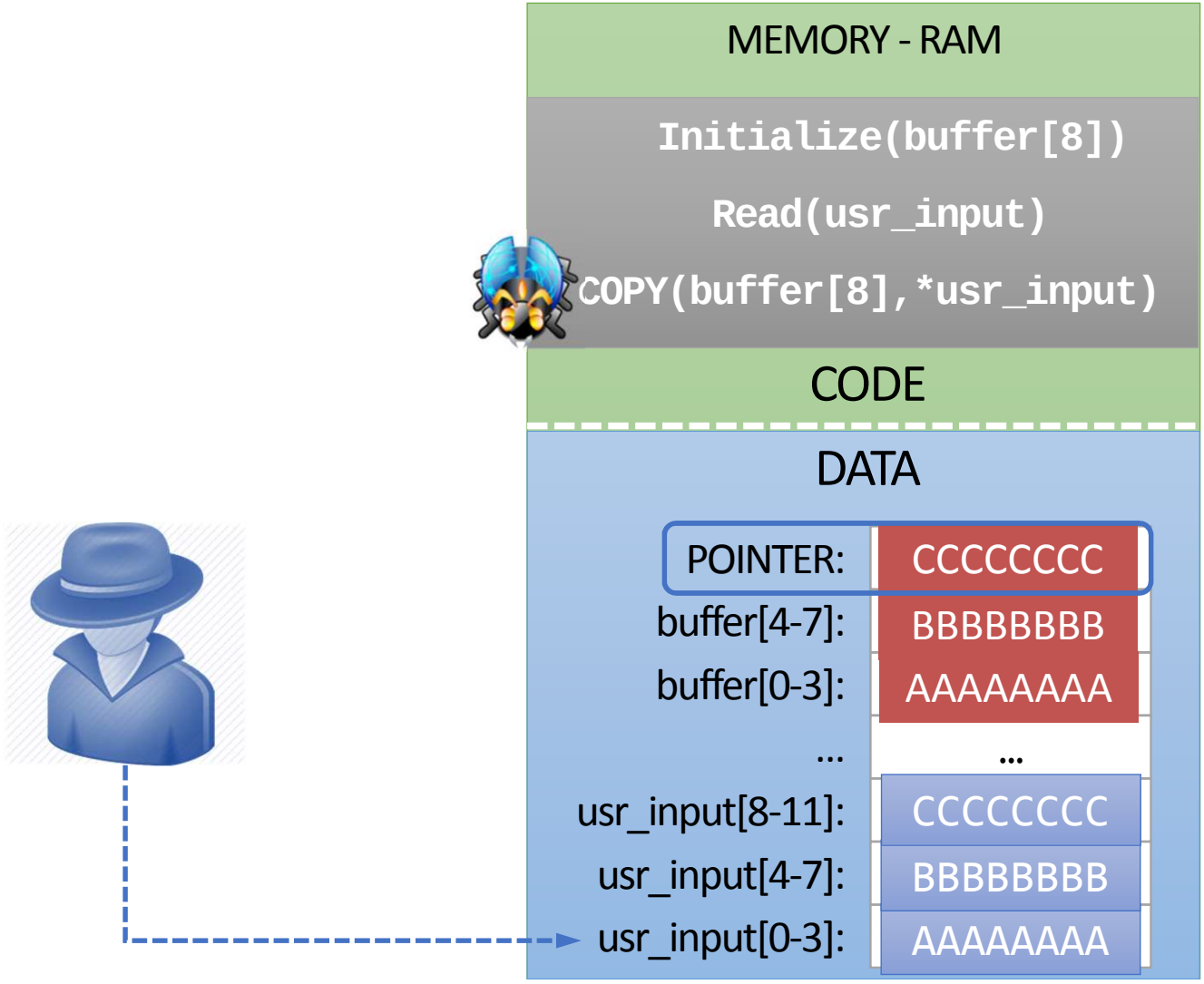
The Memory Corruption Vulnerability



The Memory Corruption Vulnerability



The Memory Corruption Vulnerability



Why It Matters

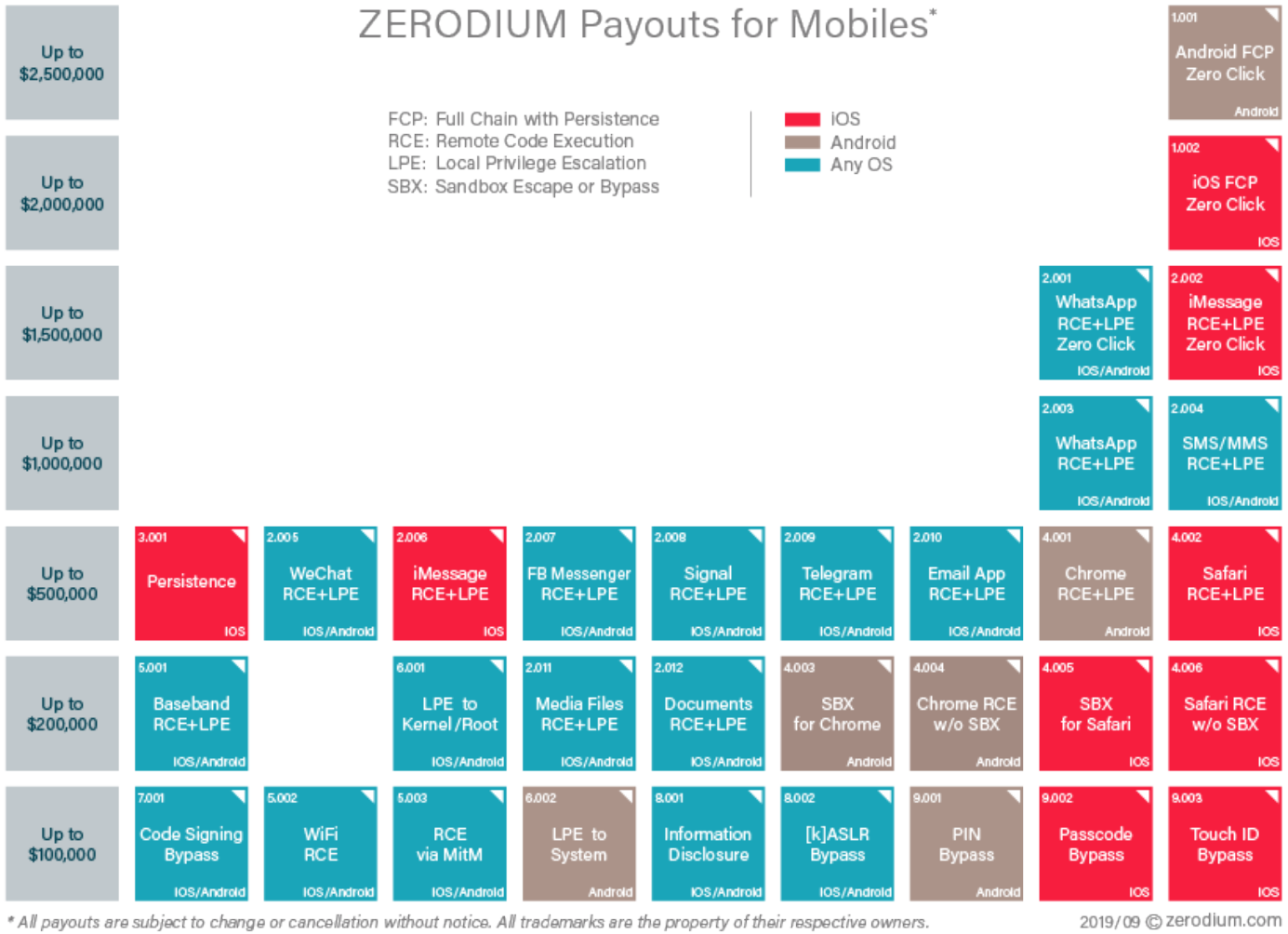
Many High Impact Attacks

attack browser

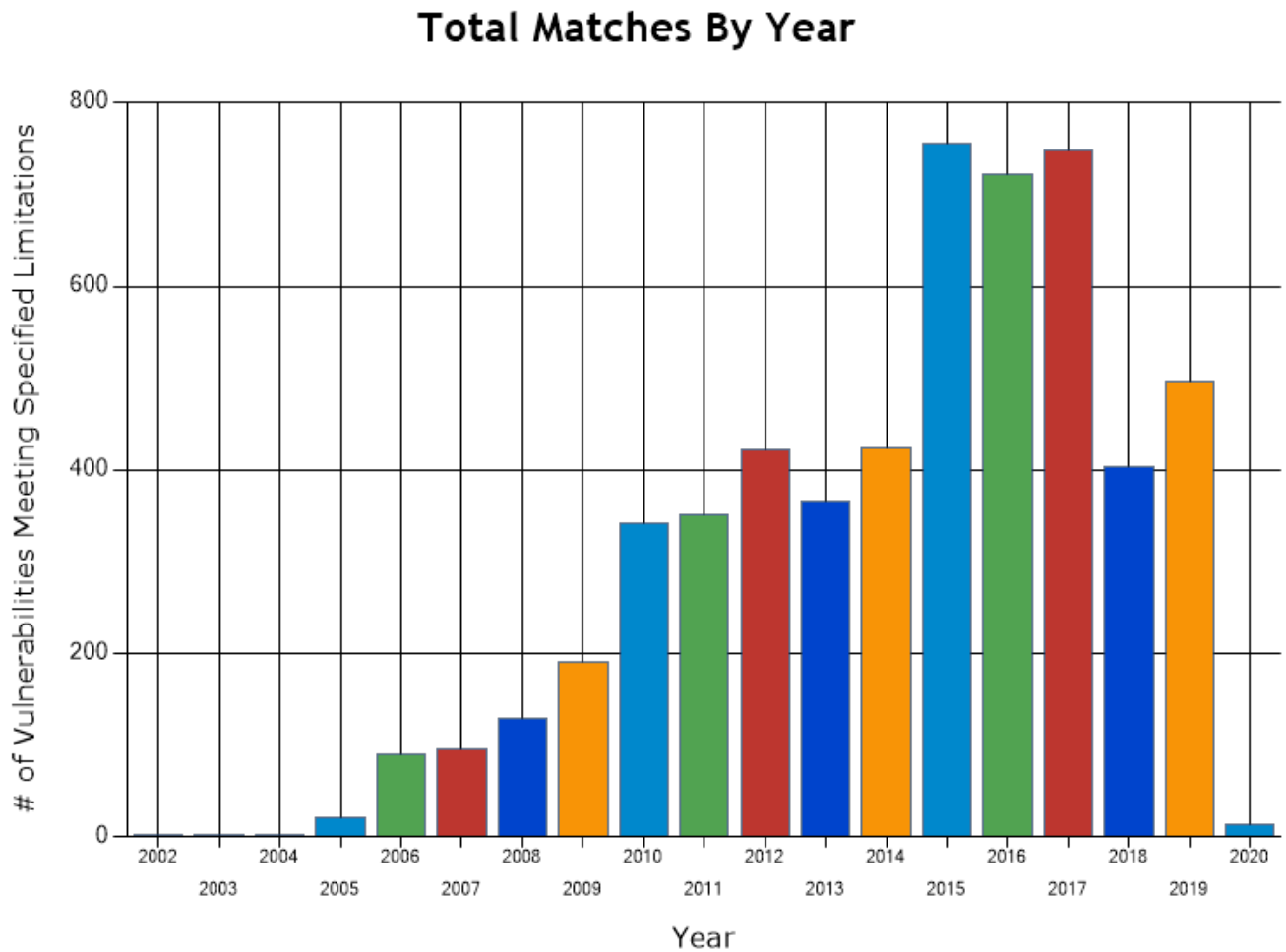
1. Web browsers repeatedly exploited in pwn2own contests
2. Zero-day issues exploited in Stuxnet/Duqu
3. iOS jailbreak

Why It Matters

Why It Matters



The Statistics of The Memory Corruption Vulnerabilities



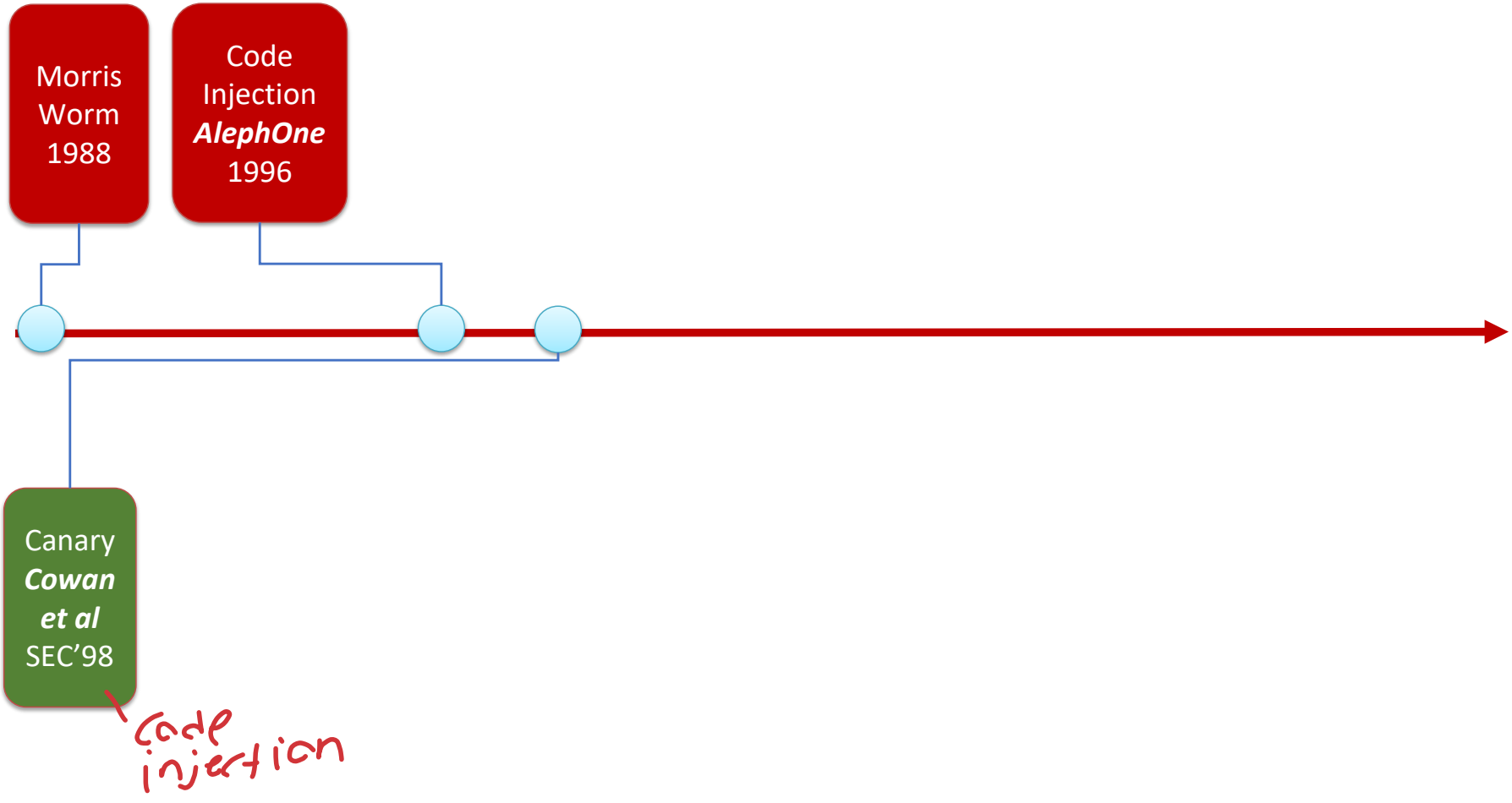
https://nvd.nist.gov/vuln/search/statistics

The Arm Race (Attack and Defense) w/ Memory Corruptions

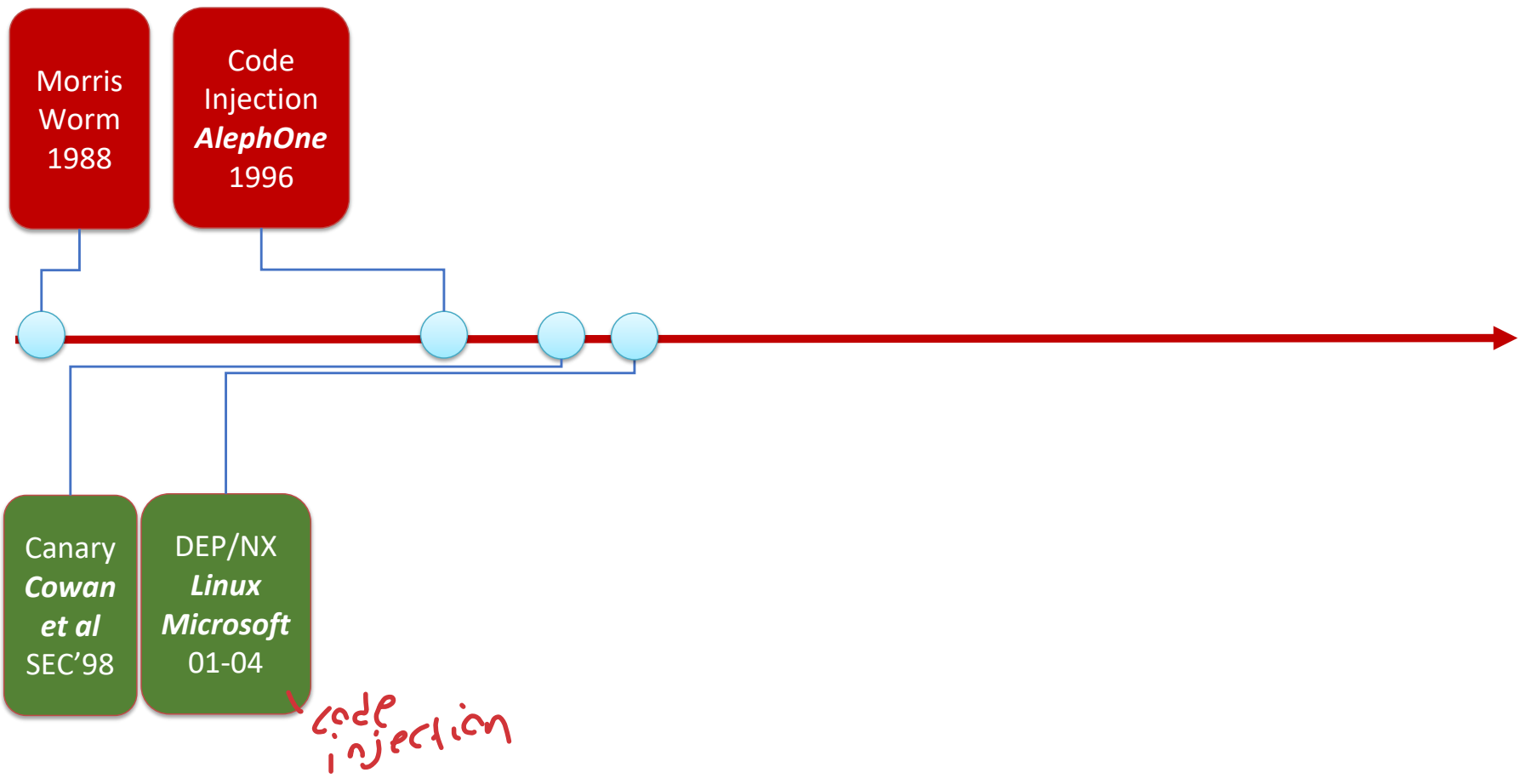
The Arm Race (Attack and Defense) w/ Memory Corruptions

The Arm Race (Attack and Defense) w/ Memory Corruptions

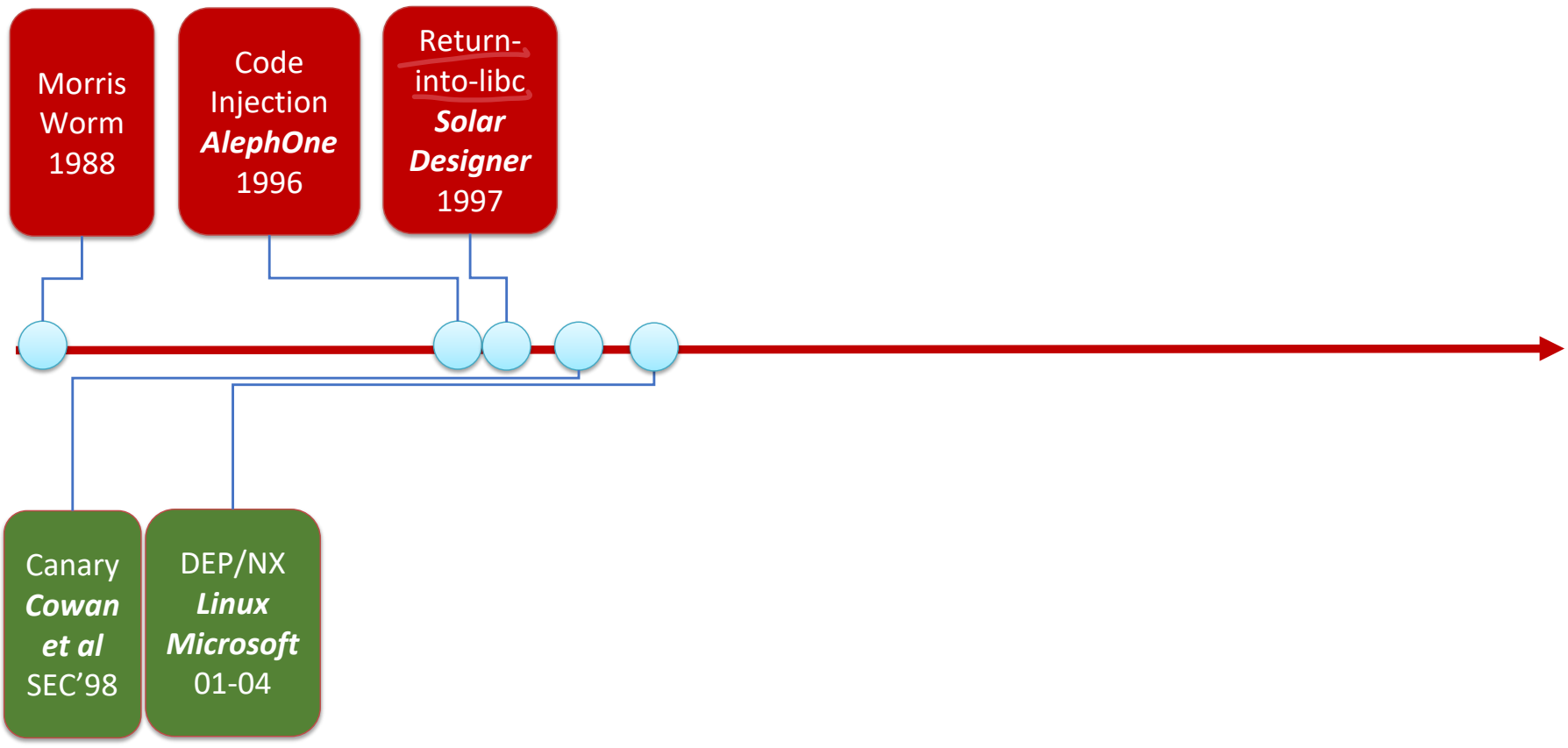
The Arm Race (Attack and Defense) w/ Memory Corruptions



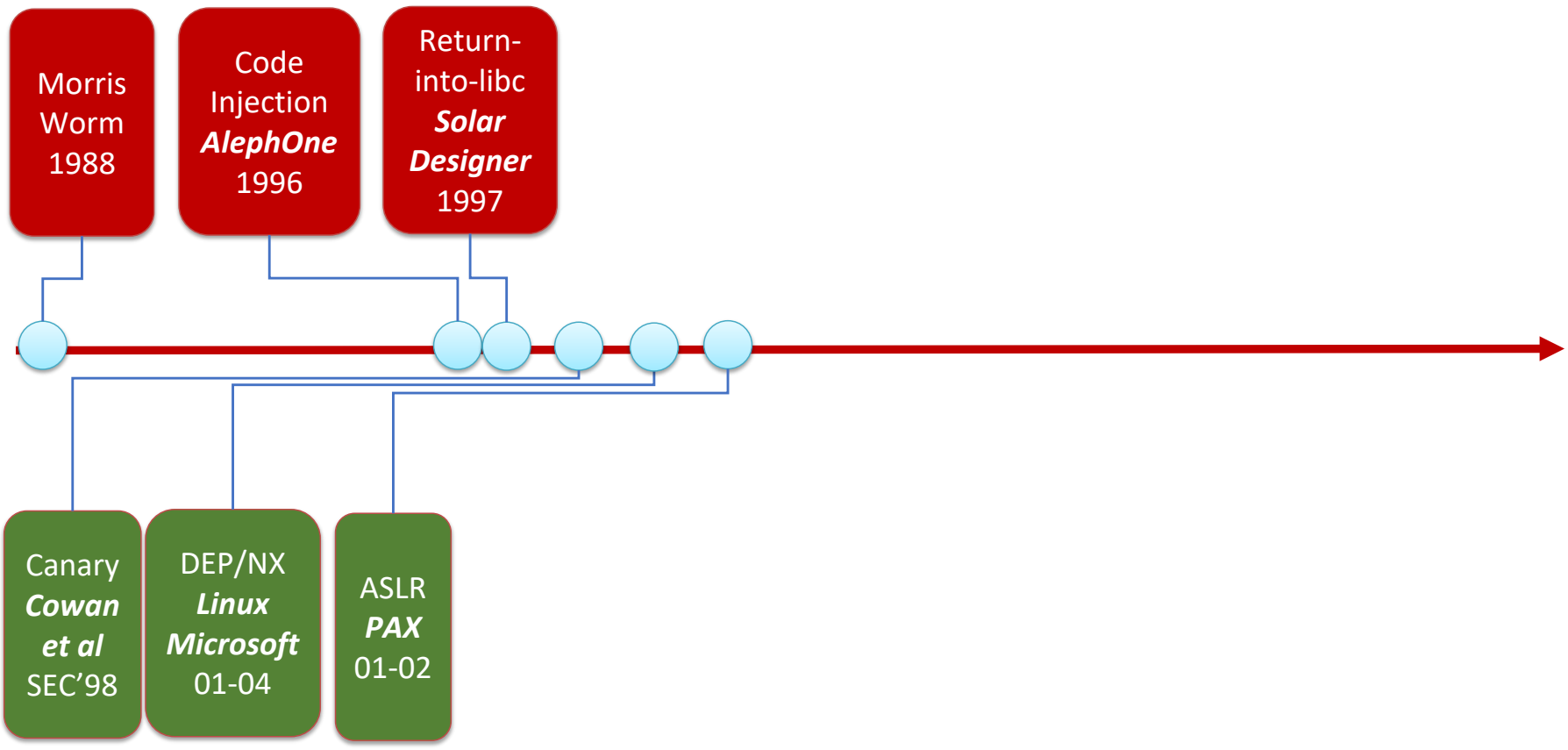
The Arm Race (Attack and Defense) w/ Memory Corruptions



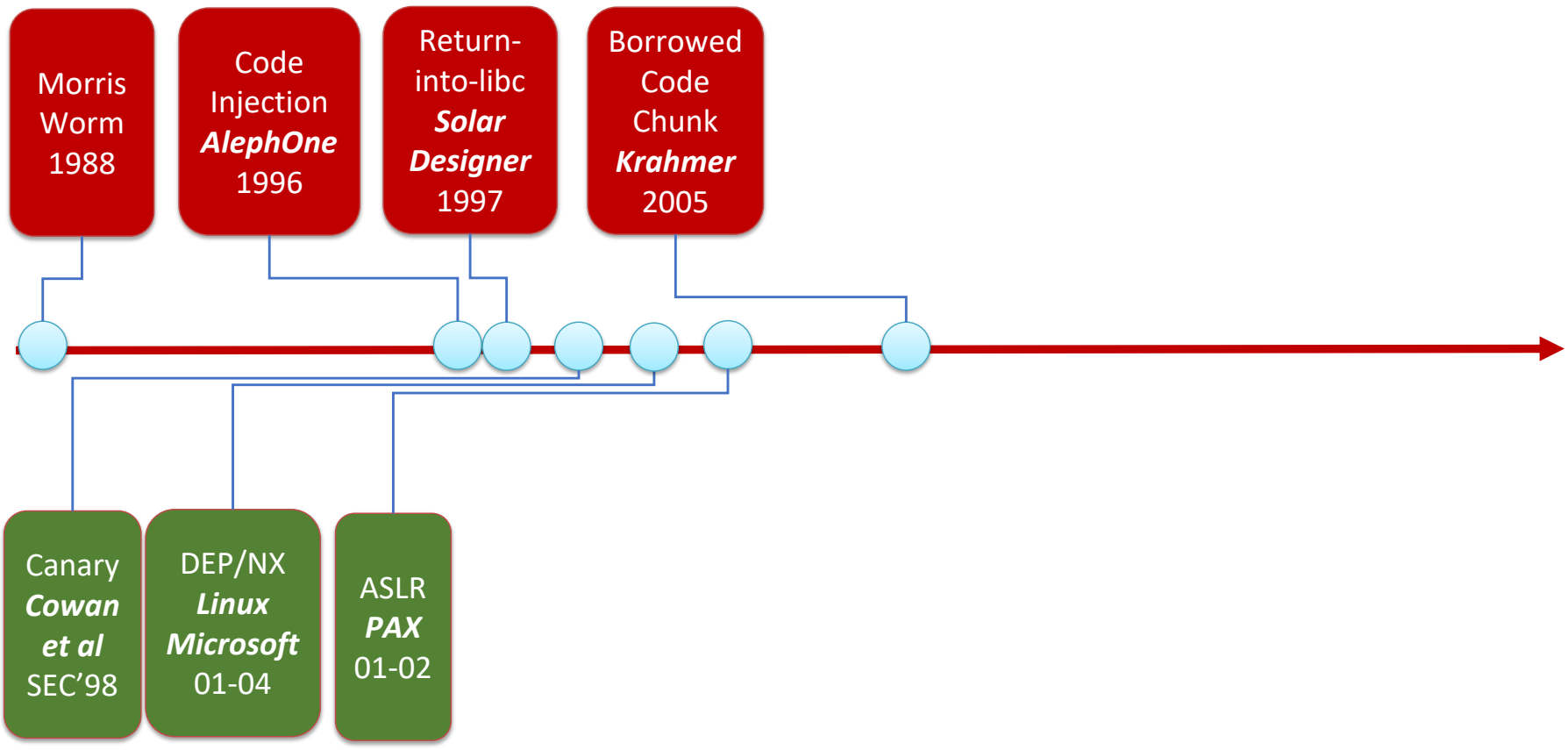
The Arm Race (Attack and Defense) w/ Memory Corruptions



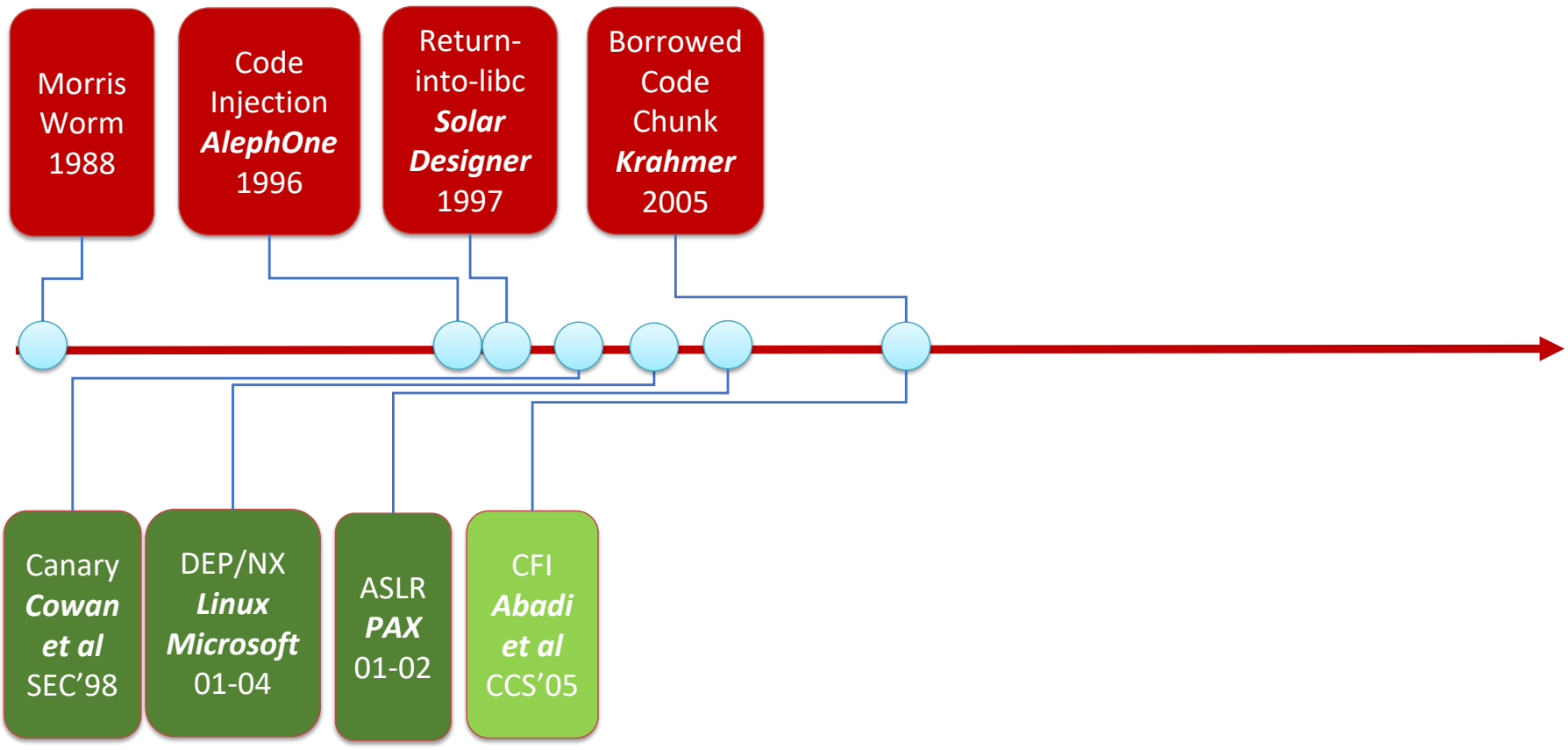
The Arm Race (Attack and Defense) w/ Memory Corruptions



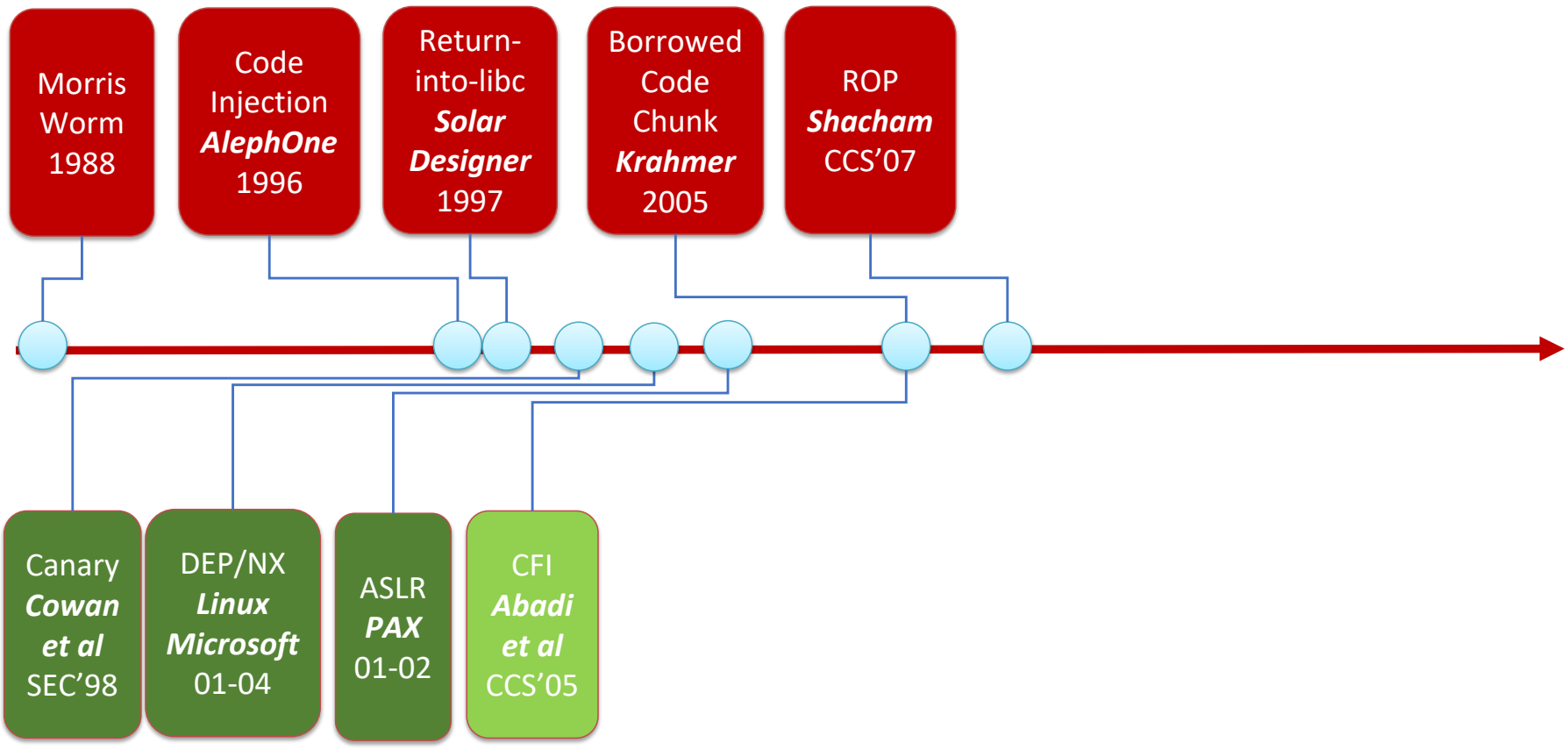
The Arm Race (Attack and Defense) w/ Memory Corruptions



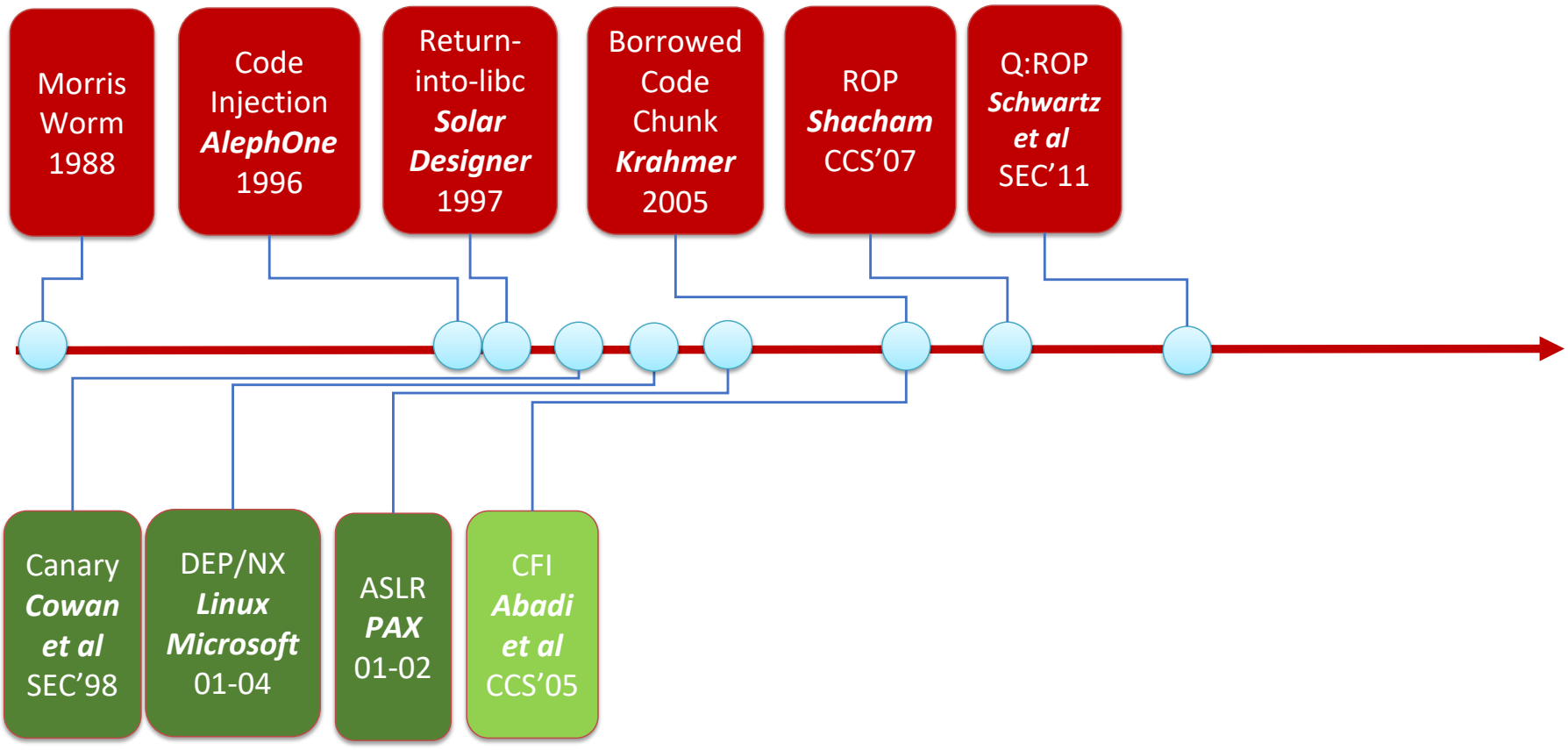
The Arm Race (Attack and Defense) w/ Memory Corruptions



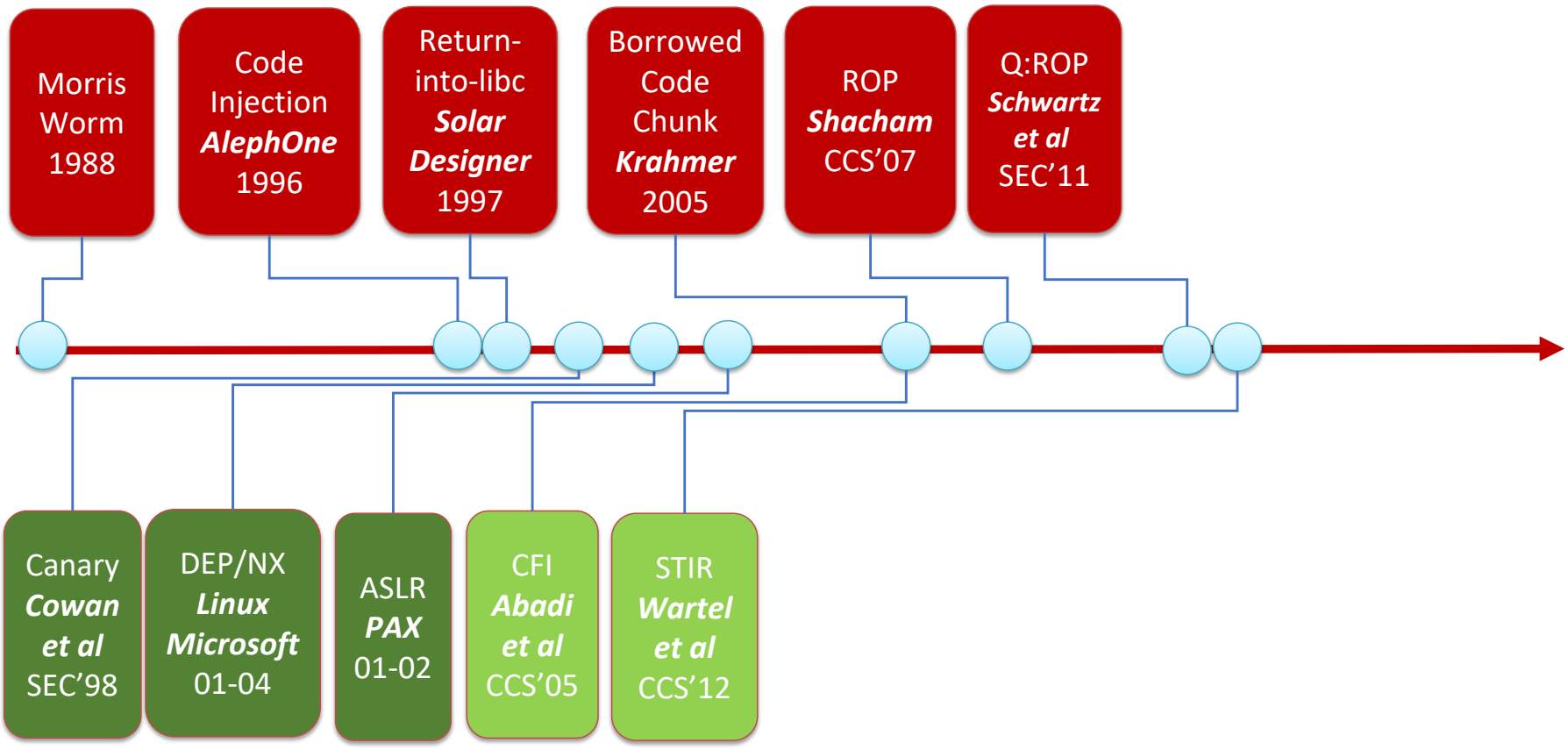
The Arm Race (Attack and Defense) w/ Memory Corruptions



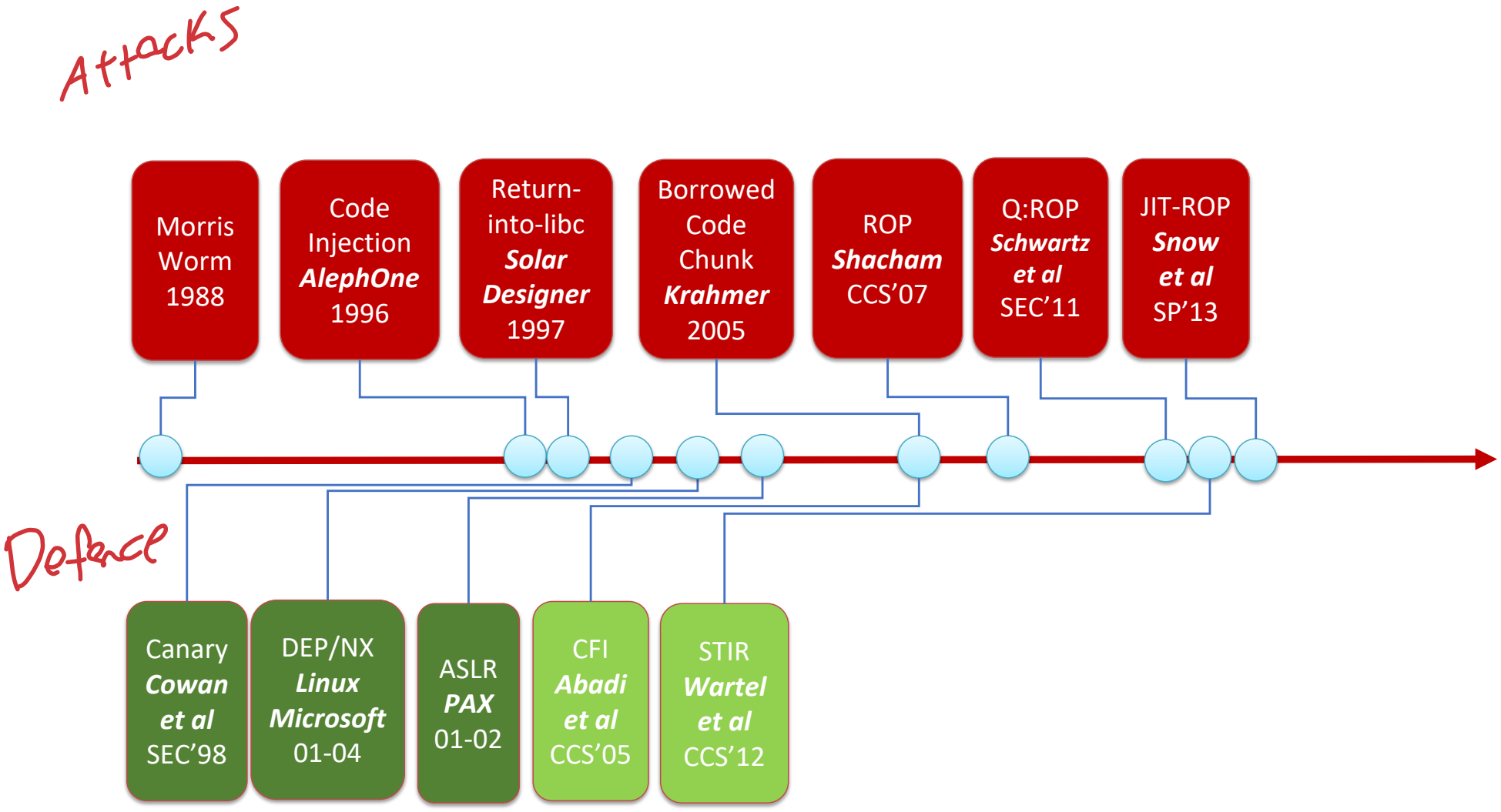
The Arm Race (Attack and Defense) w/ Memory Corruptions



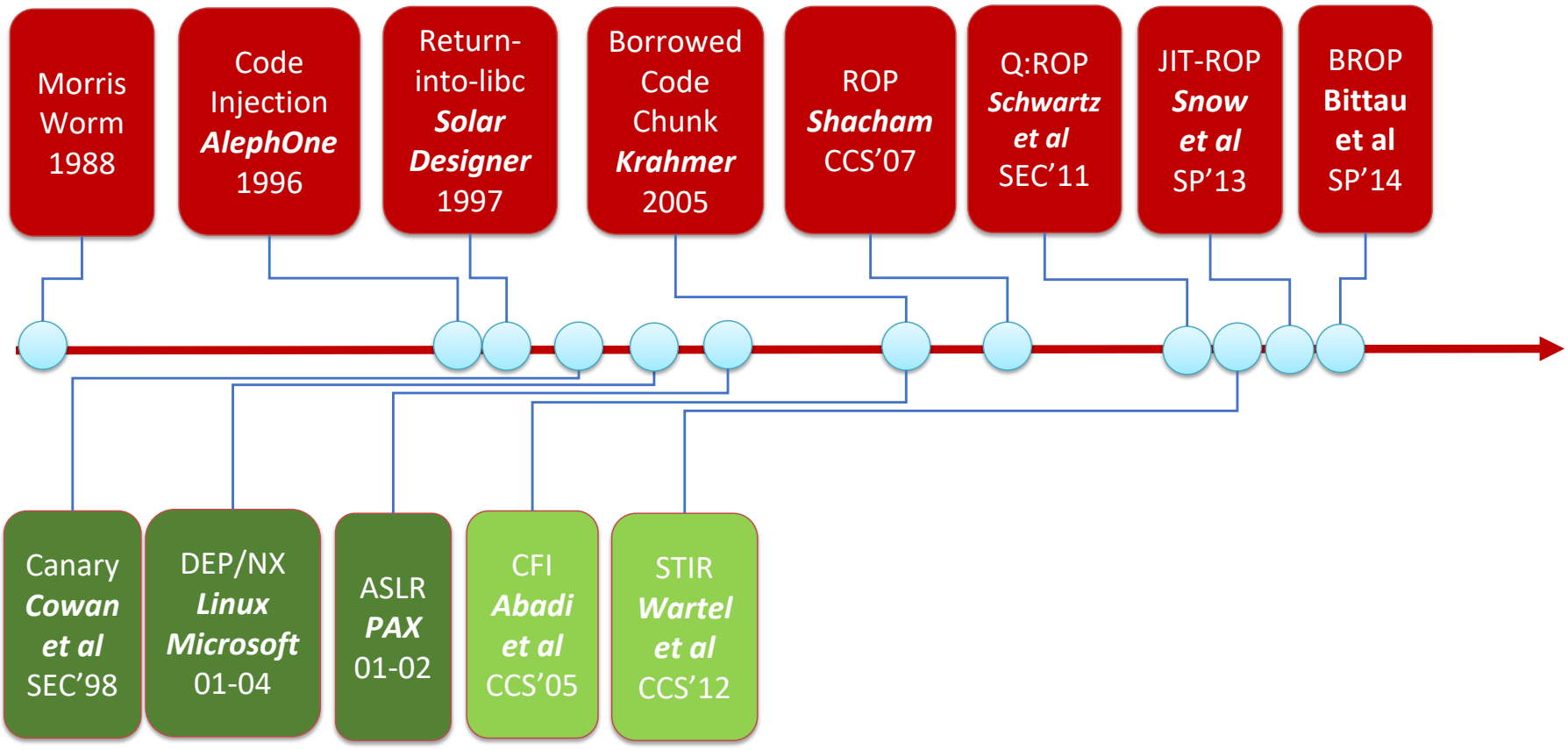
The Arm Race (Attack and Defense) w/ Memory Corruptions



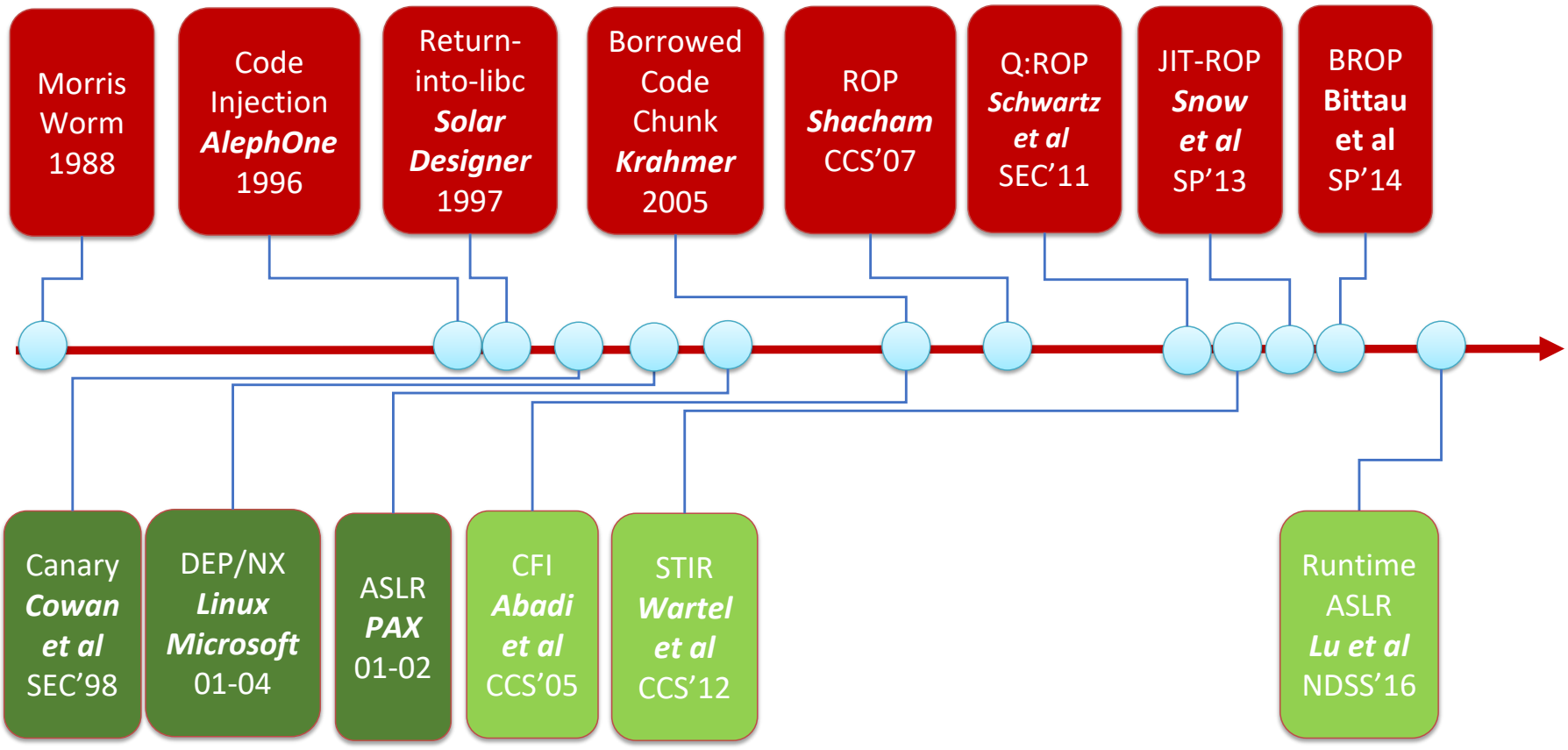
The Arm Race (Attack and Defense) w/ Memory Corruptions



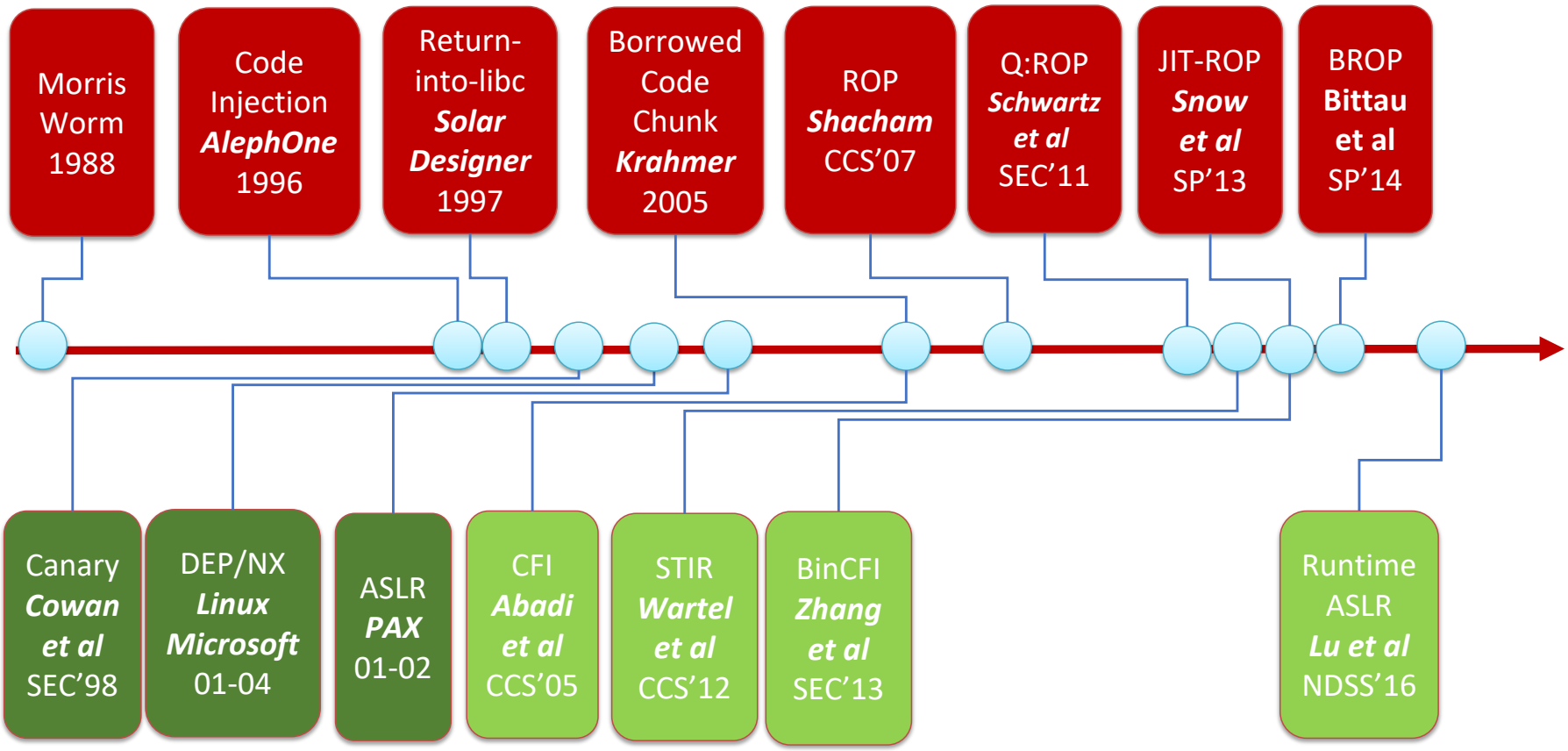
The Arm Race (Attack and Defense) w/ Memory Corruptions



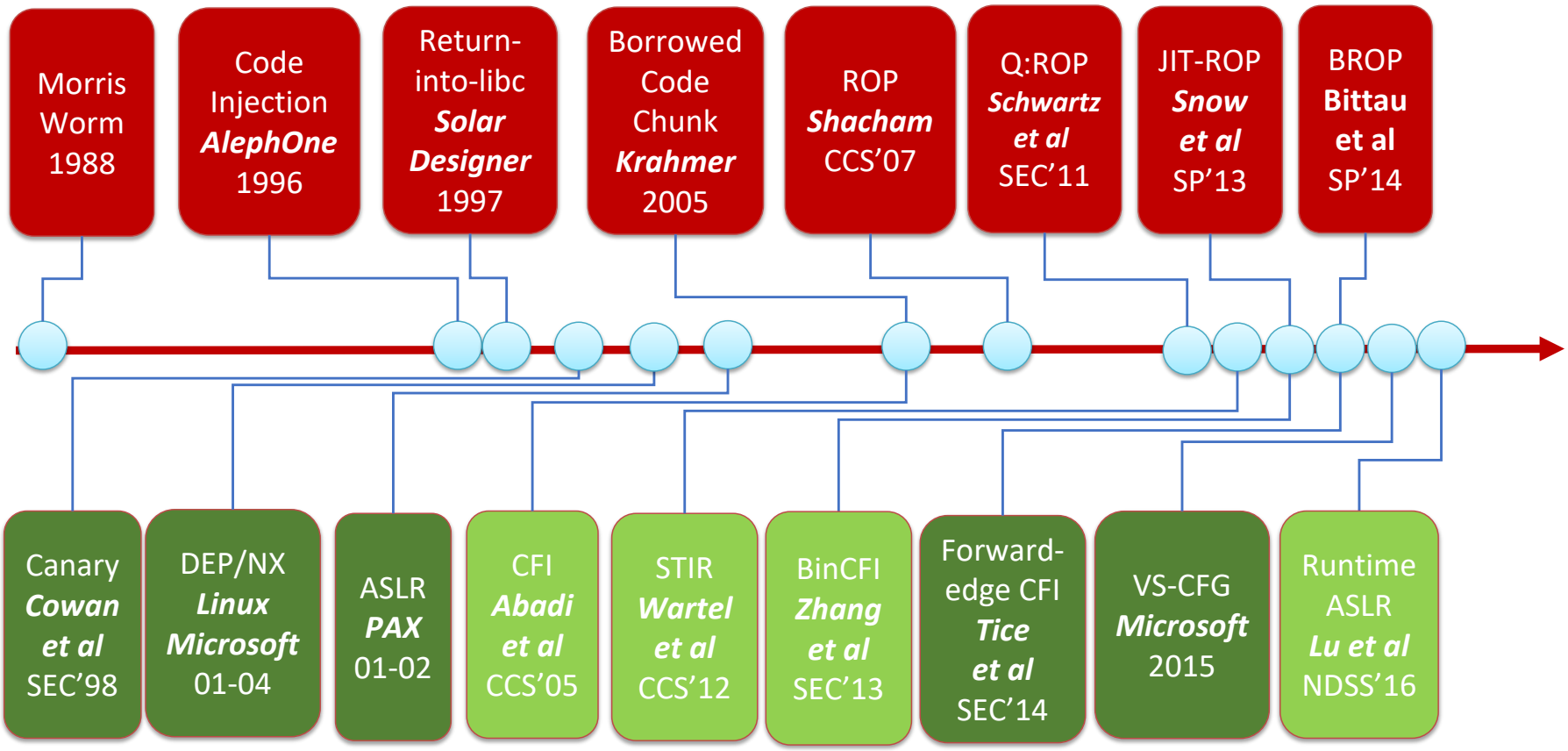
The Arm Race (Attack and Defense) w/ Memory Corruptions



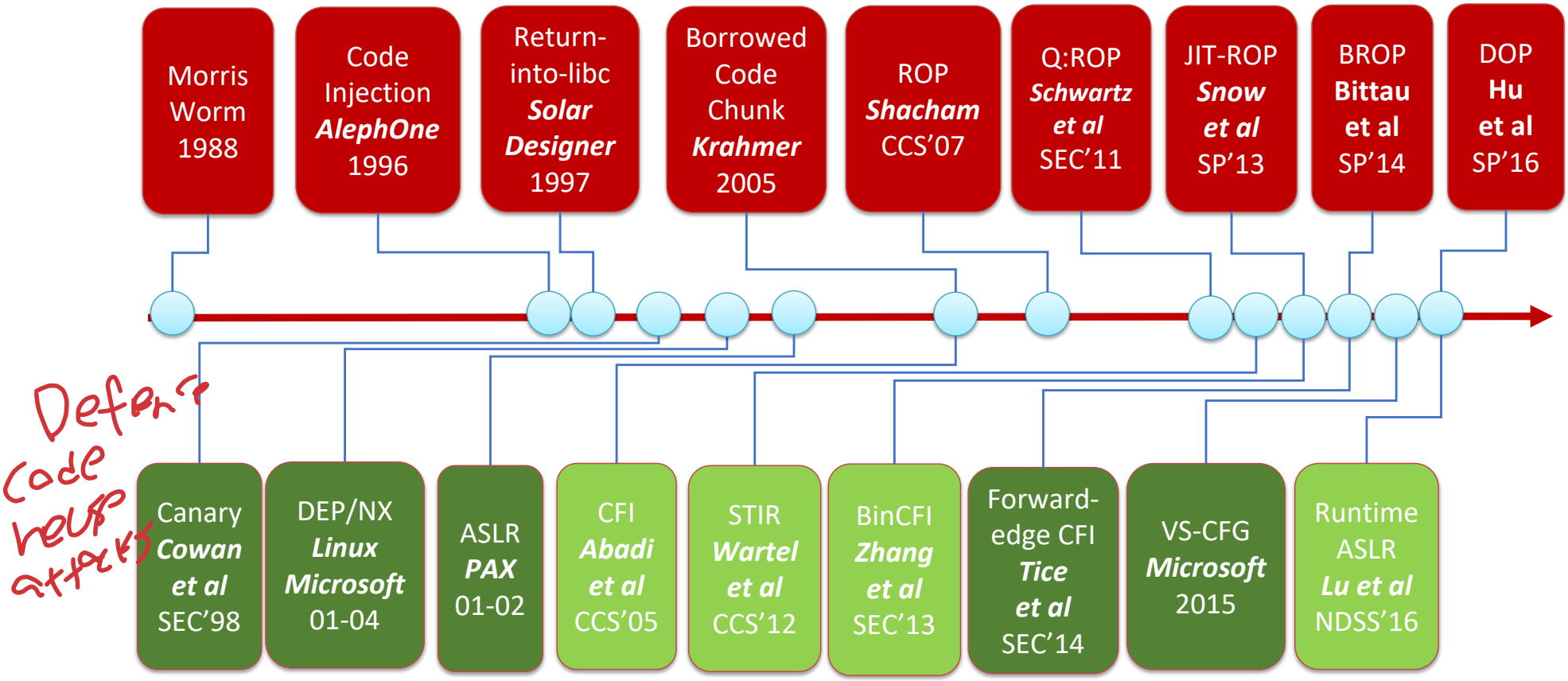
The Arm Race (Attack and Defense) w/ Memory Corruptions



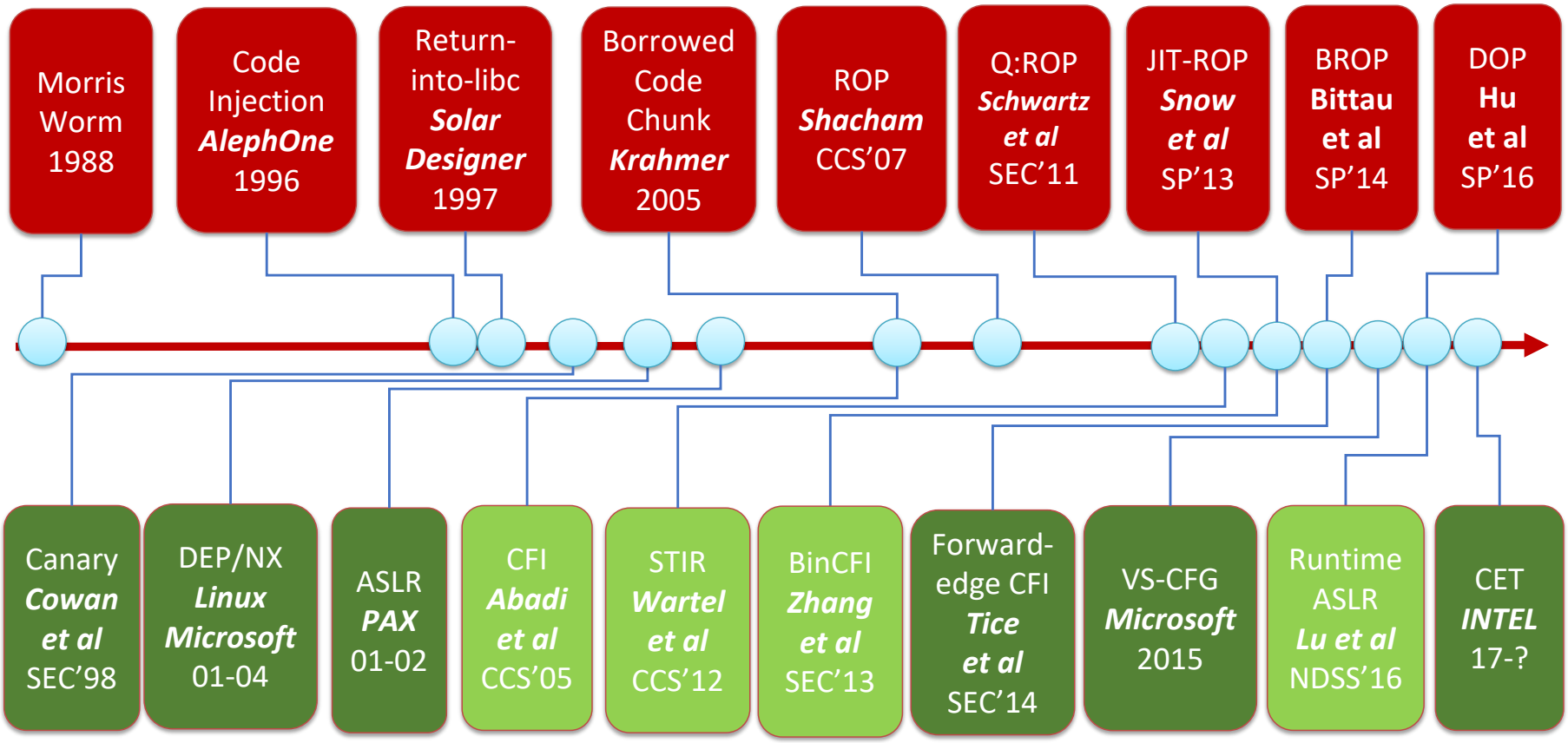
The Arm Race (Attack and Defense) w/ Memory Corruptions



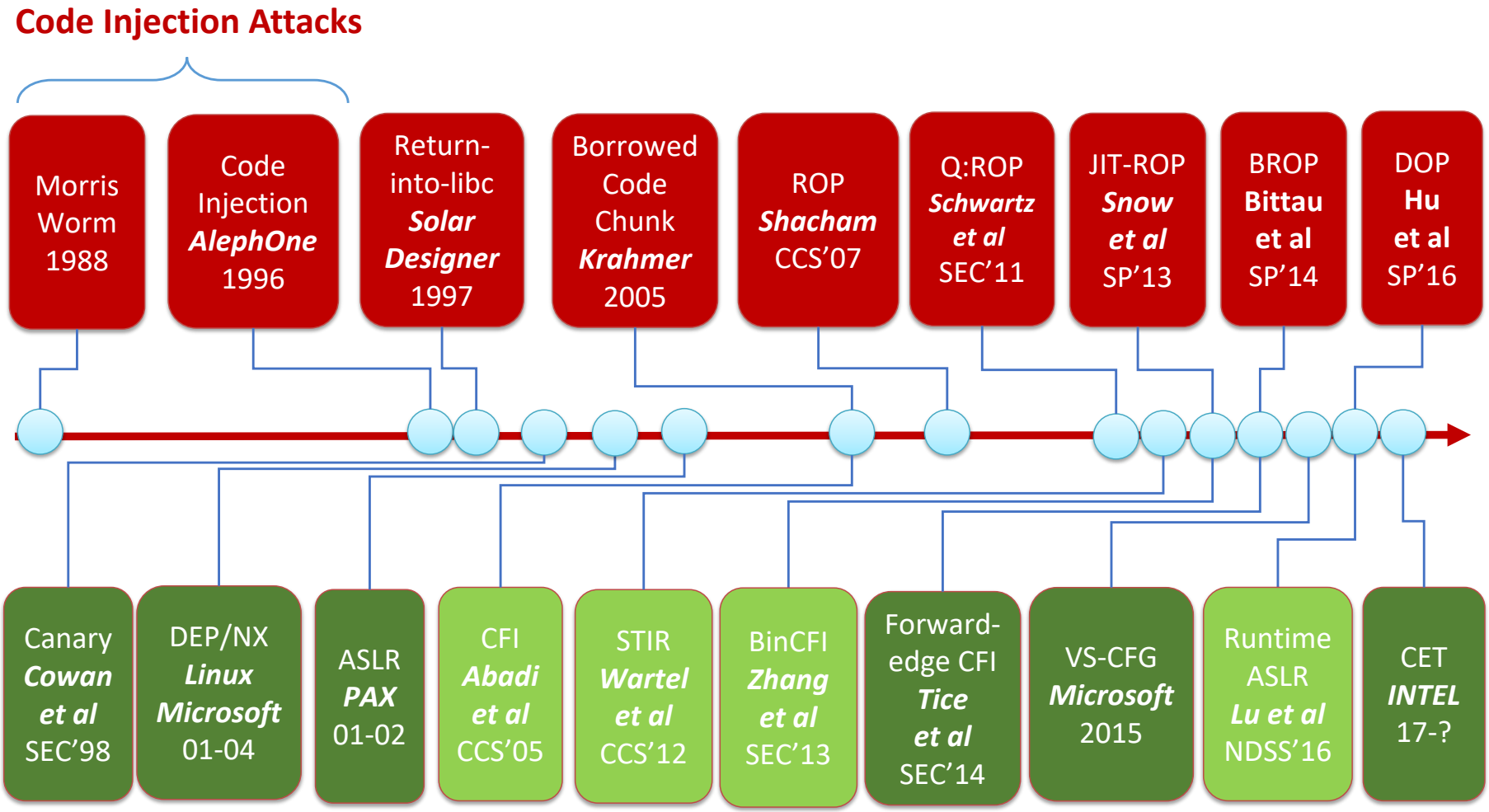
The Arm Race (Attack and Defense) w/ Memory Corruptions



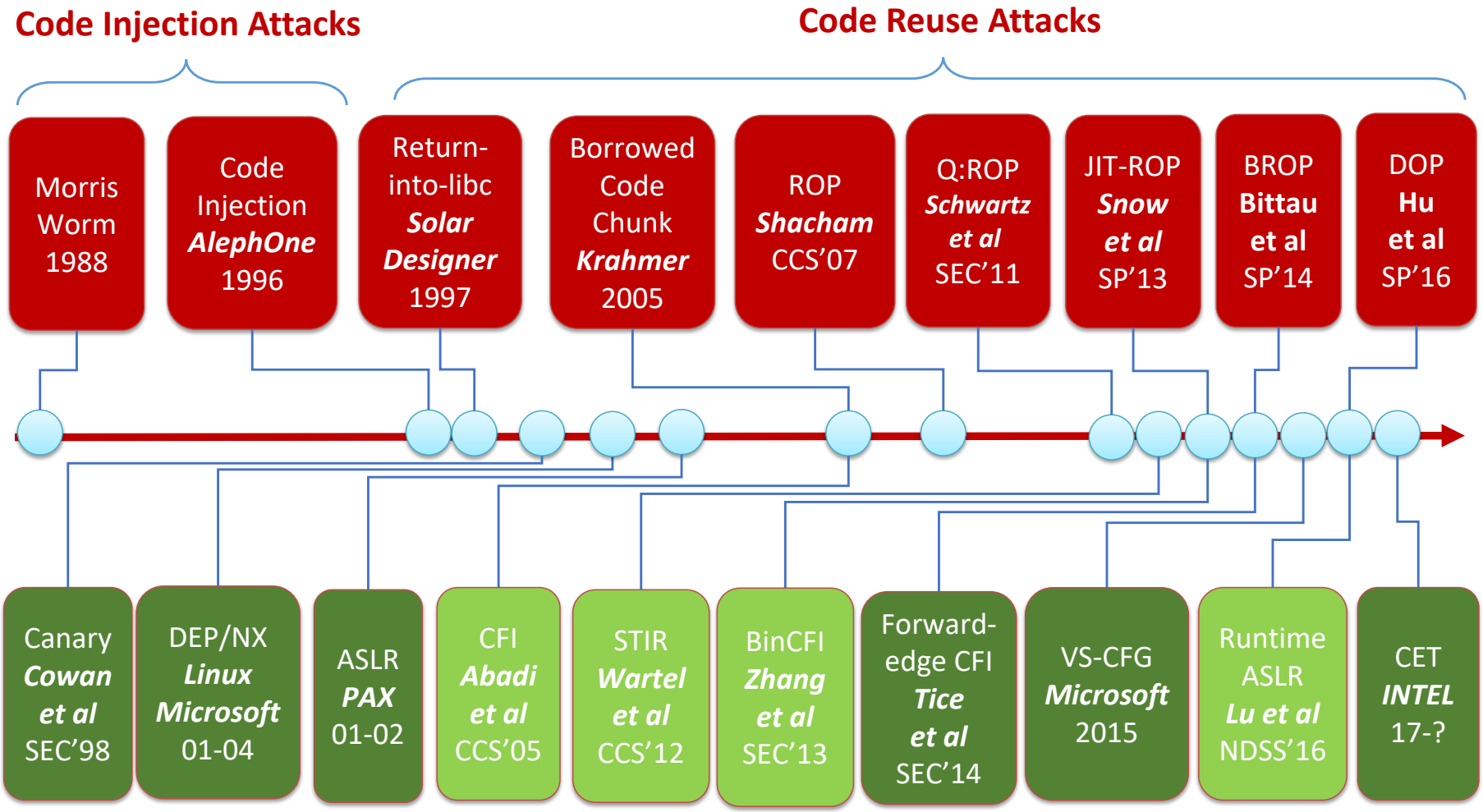
The Arm Race (Attack and Defense) w/ Memory Corruptions



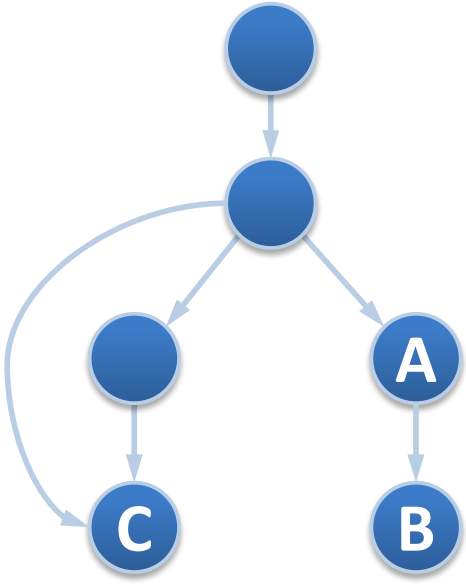
The Arm Race (Attack and Defense) w/ Memory Corruptions



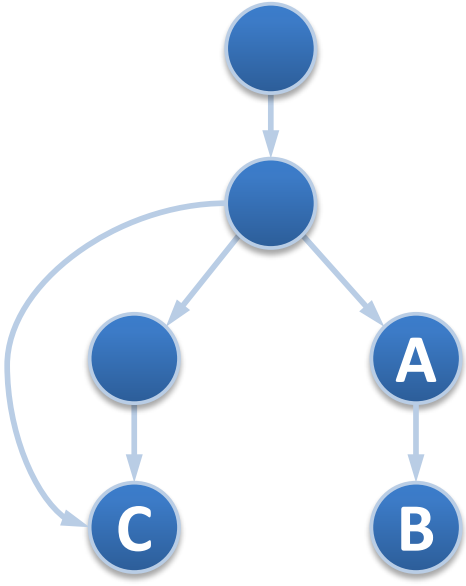
The Arm Race (Attack and Defense) w/ Memory Corruptions



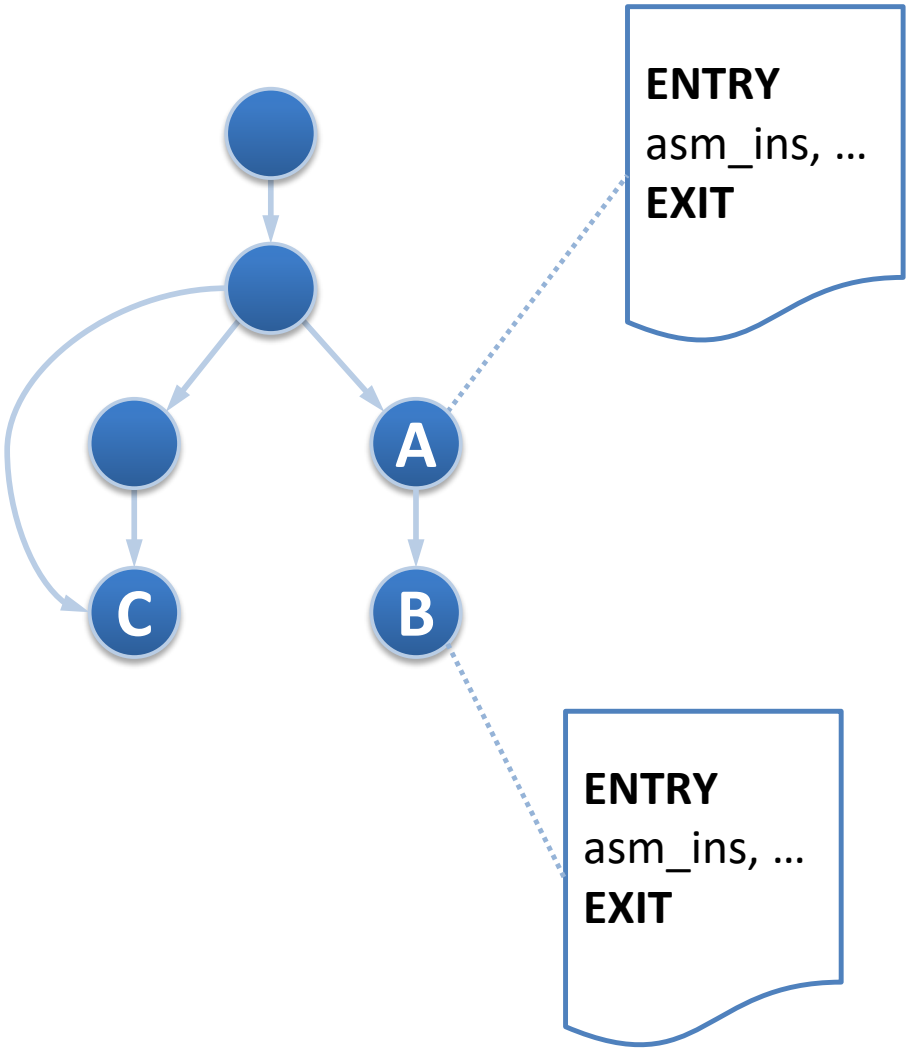
General Principle of **Code Injection Attacks**



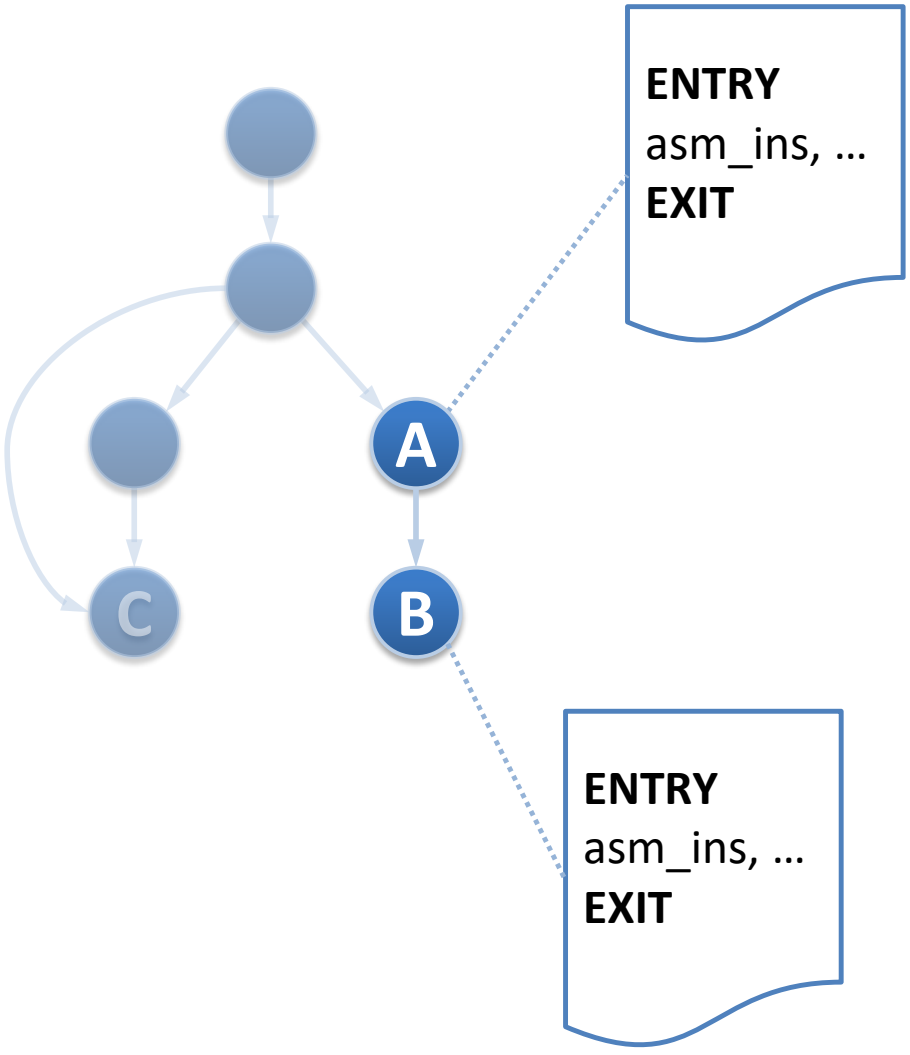
General Principle of **Code Injection Attacks**



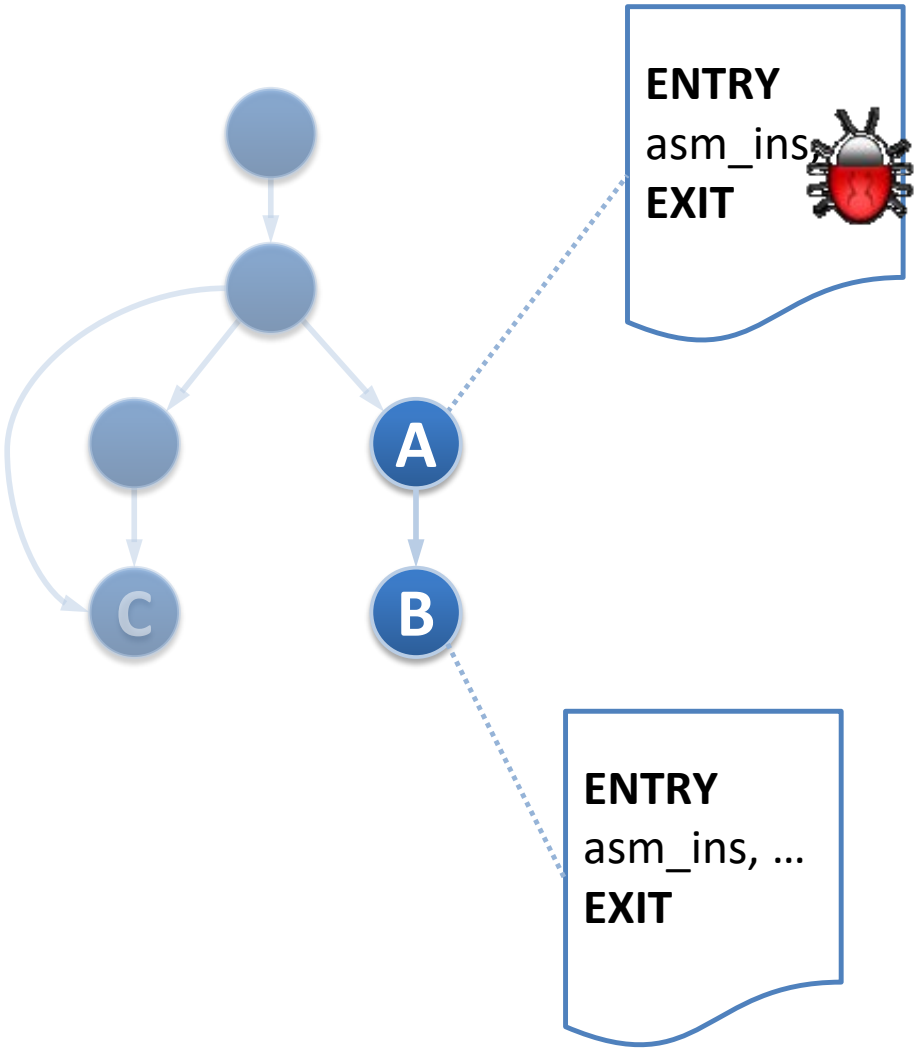
General Principle of **Code Injection Attacks**



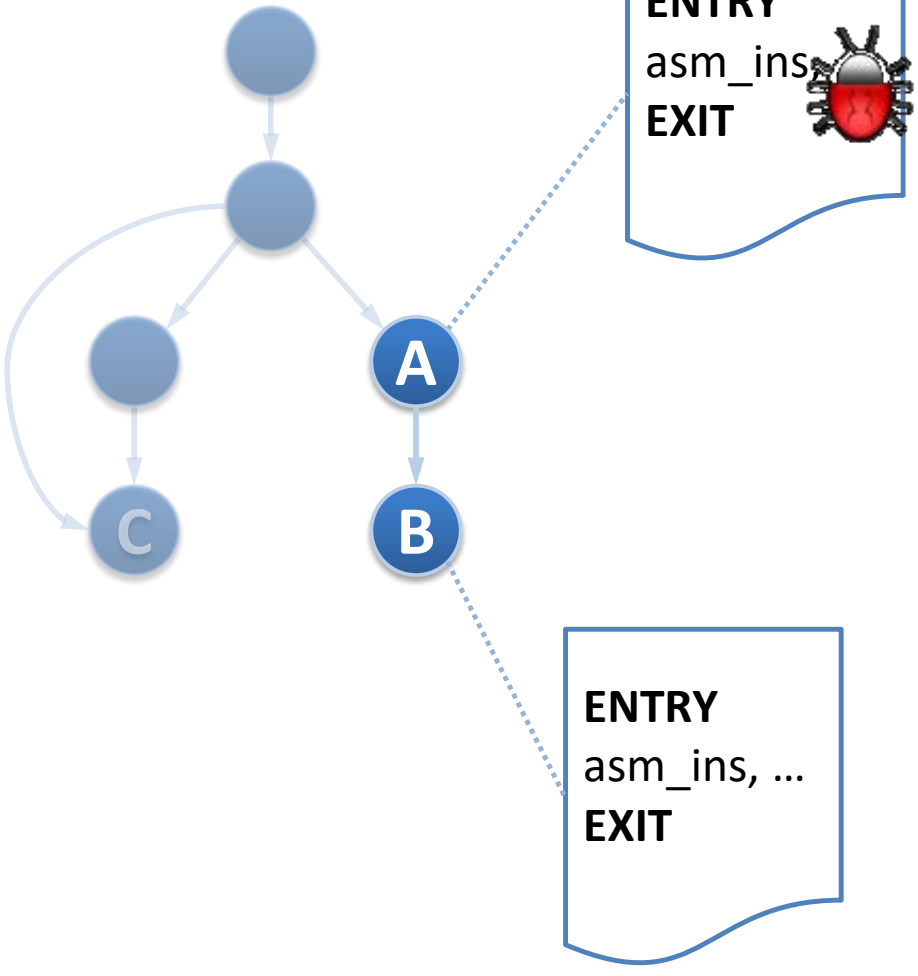
General Principle of **Code Injection Attacks**



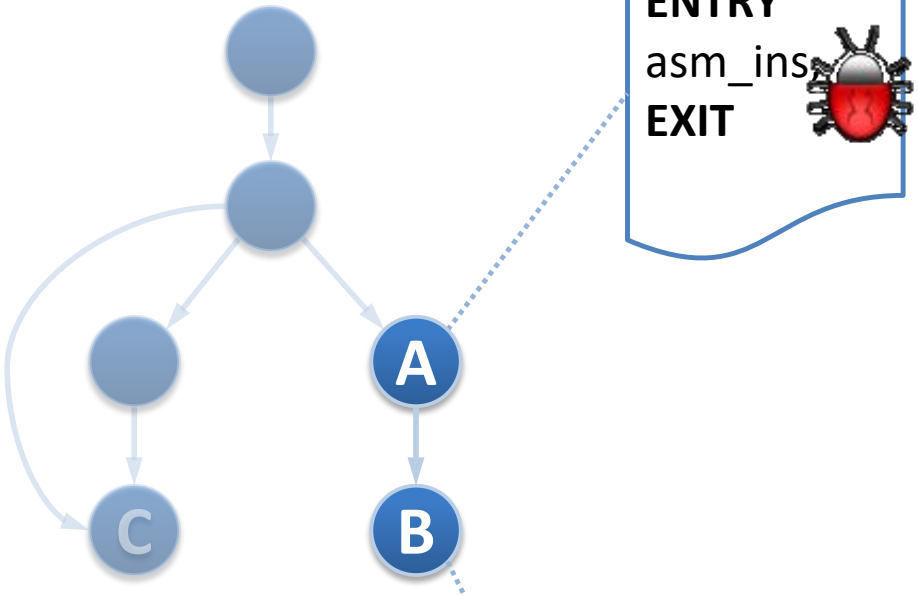
General Principle of **Code Injection Attacks**



General Principle of **Code Injection Attacks**



General Principle of **Code Injection Attacks**

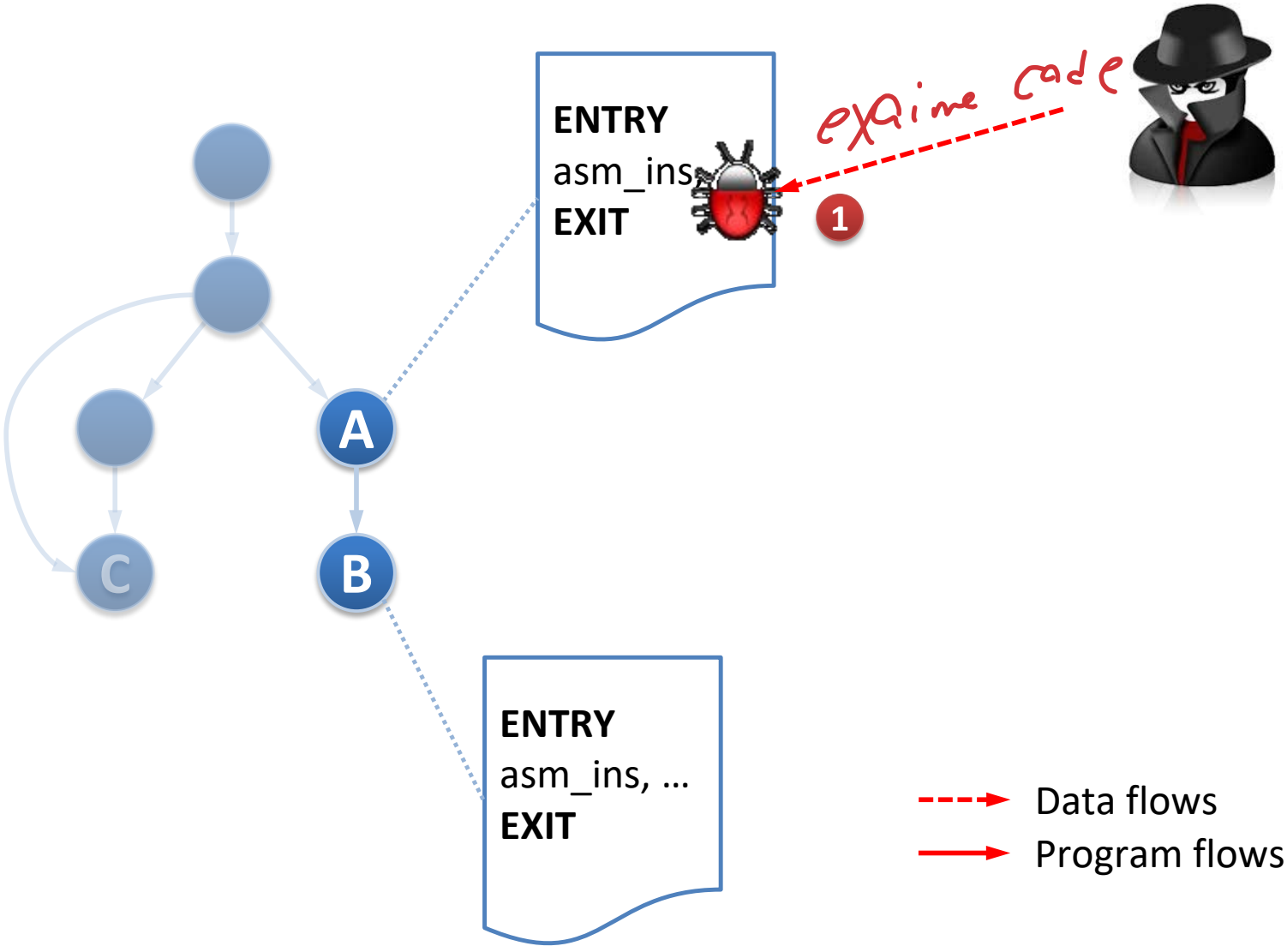


ENTRY
asm_ins
EXIT

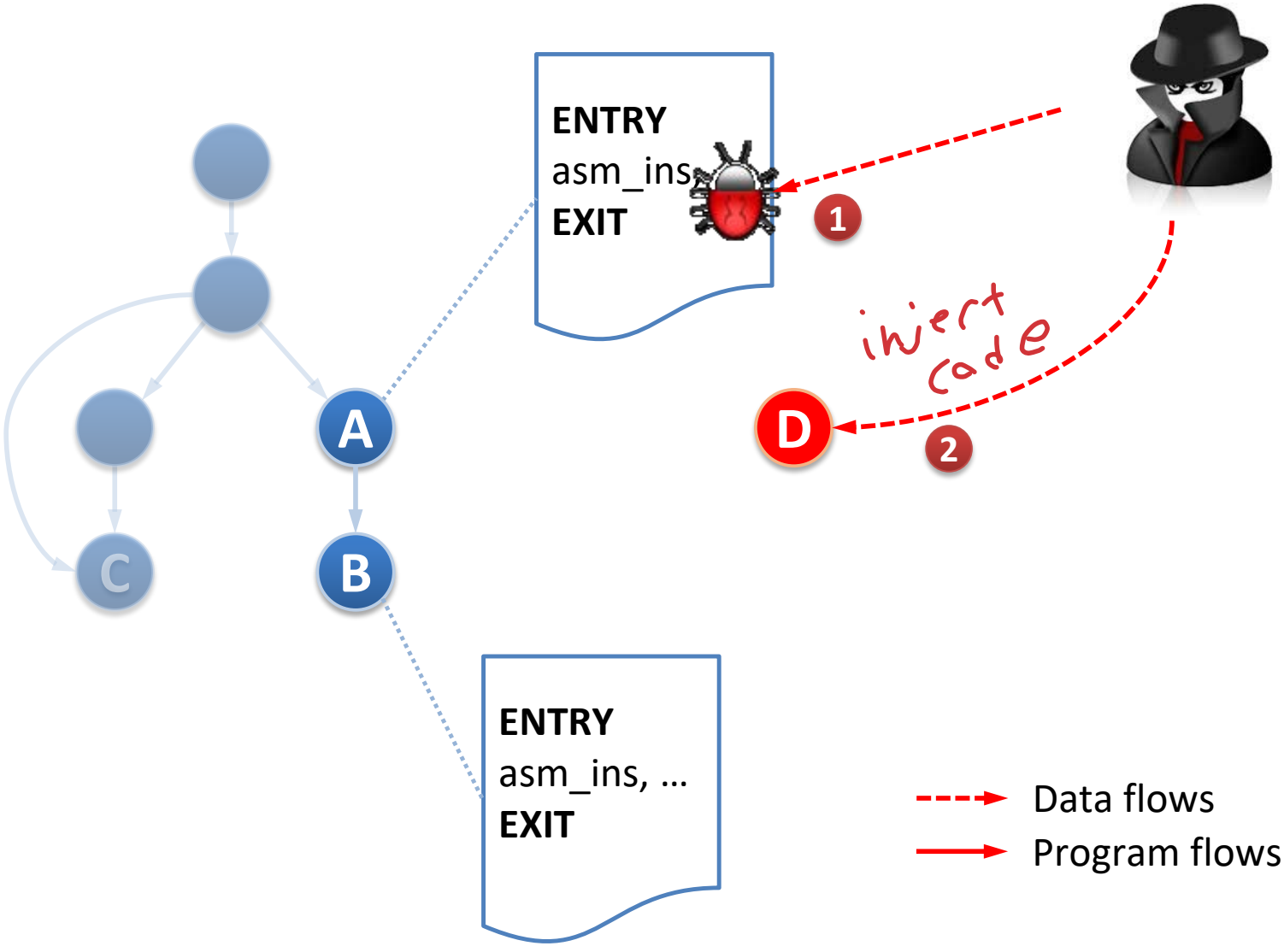
ENTRY
asm_ins, ...
EXIT

Data flows
Program flows

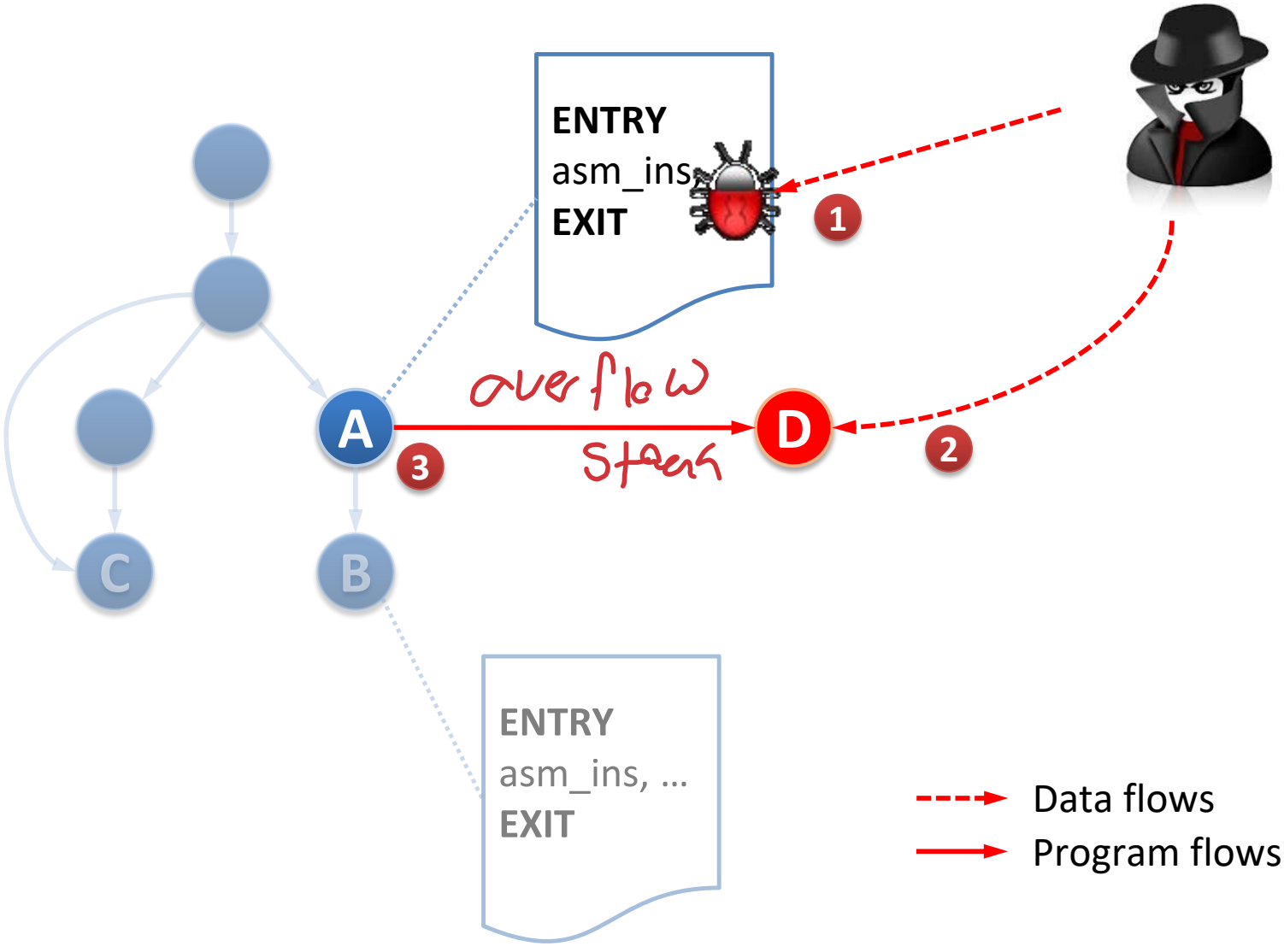
General Principle of **Code Injection Attacks**



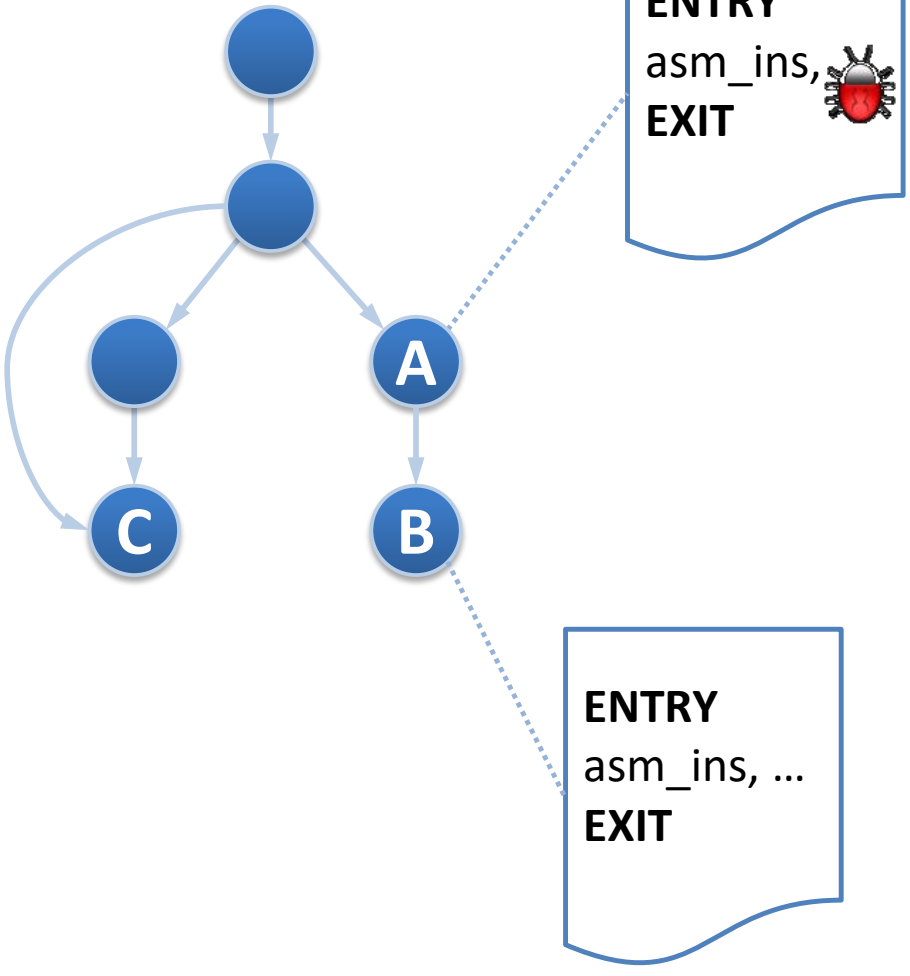
General Principle of **Code Injection Attacks**



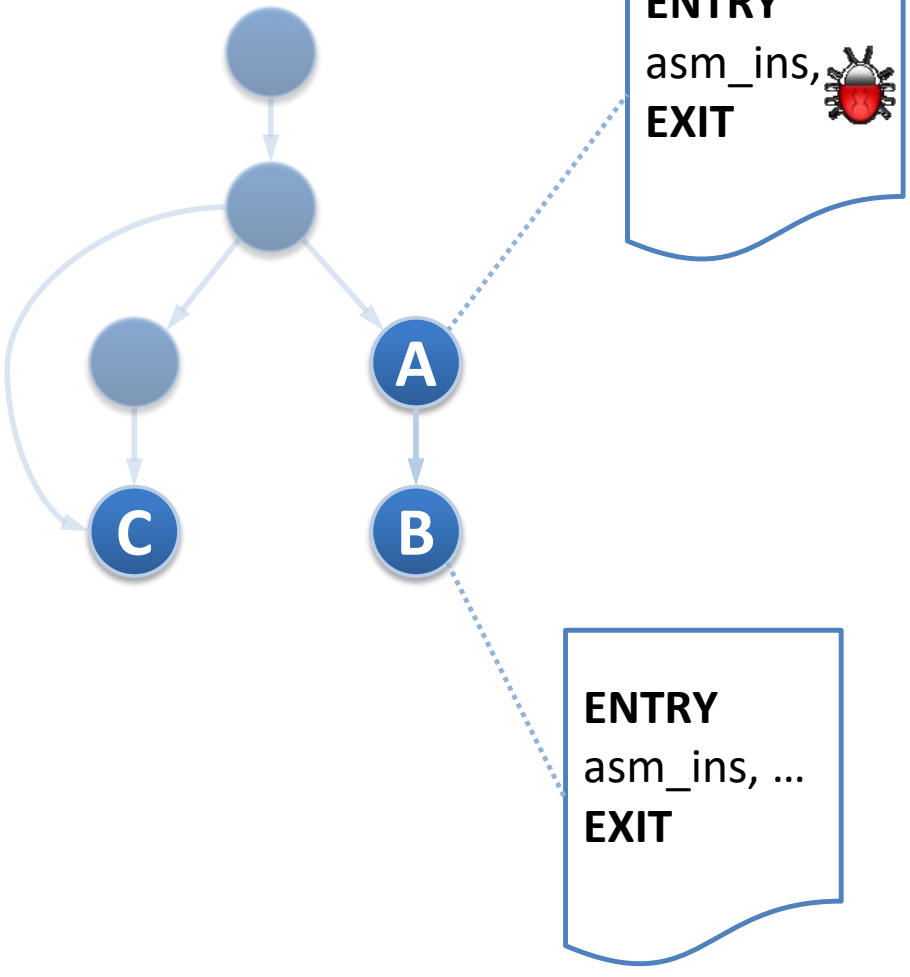
General Principle of **Code Injection Attacks**



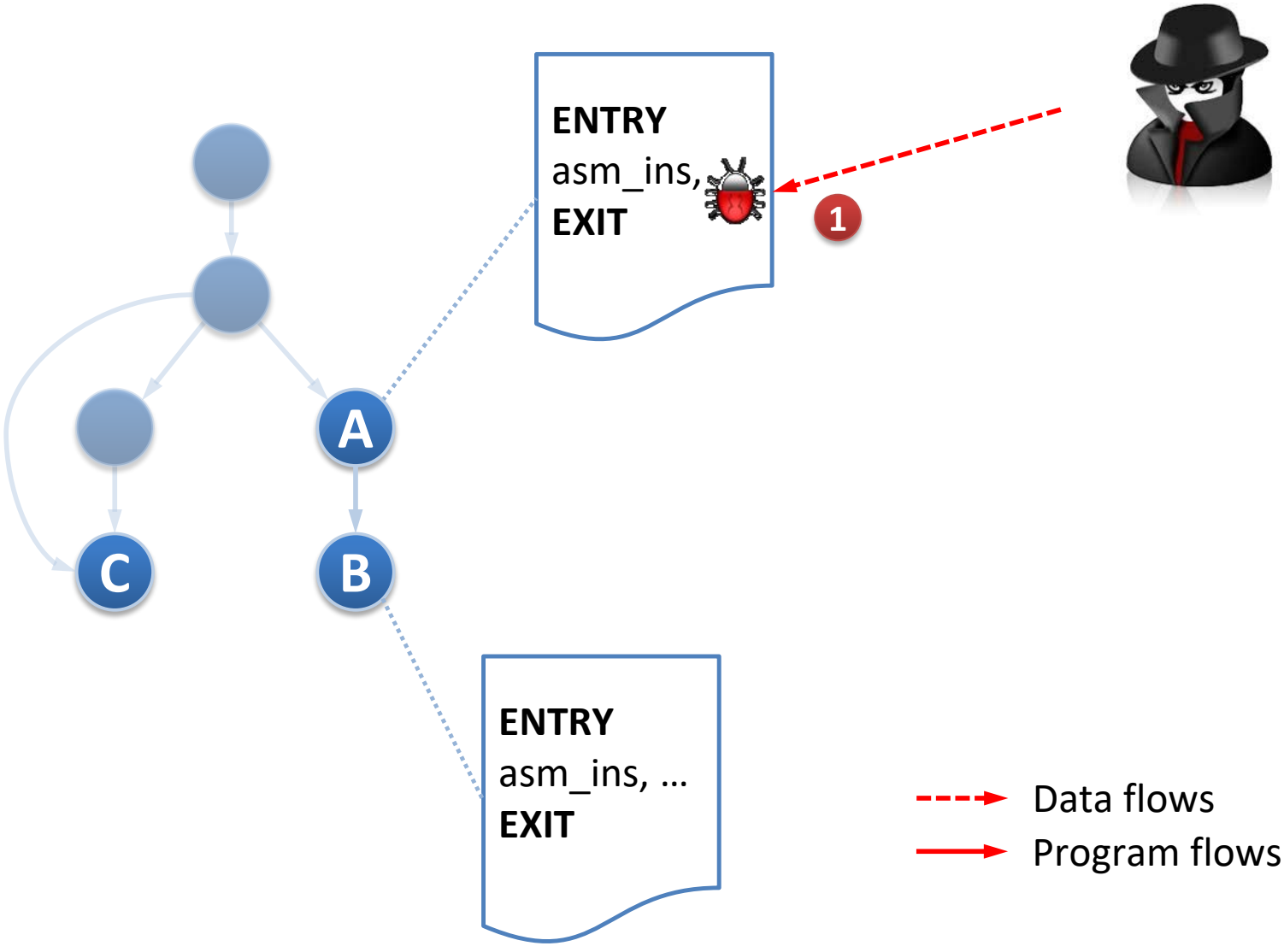
General Principle of **Code Reuse Attacks**



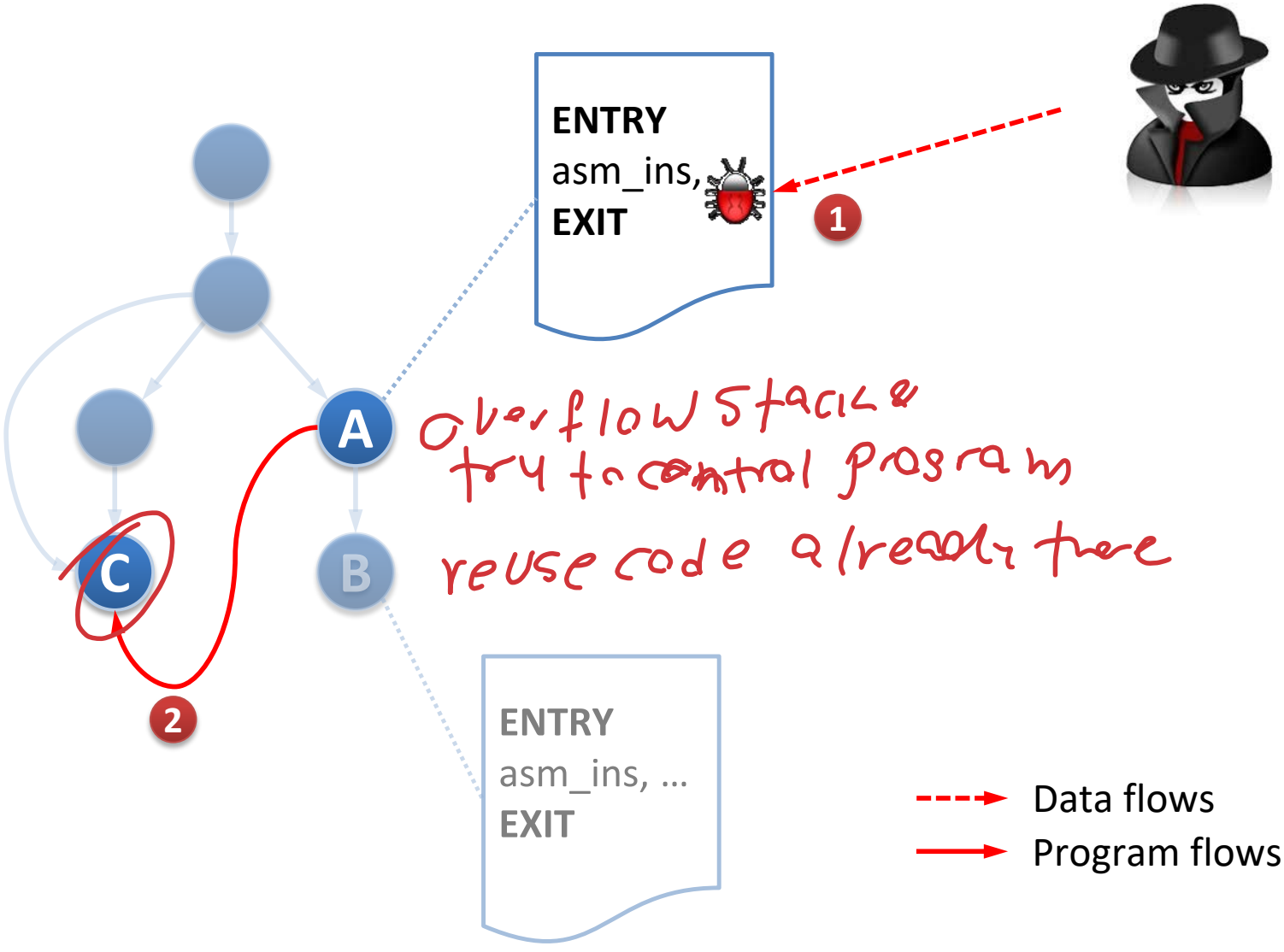
General Principle of **Code Reuse Attacks**



General Principle of **Code Reuse Attacks**



General Principle of **Code Reuse Attacks**



Memory Corruption → Control Flow Hijack

overwritten pointers

inject code here

buffer

computation + ***control***

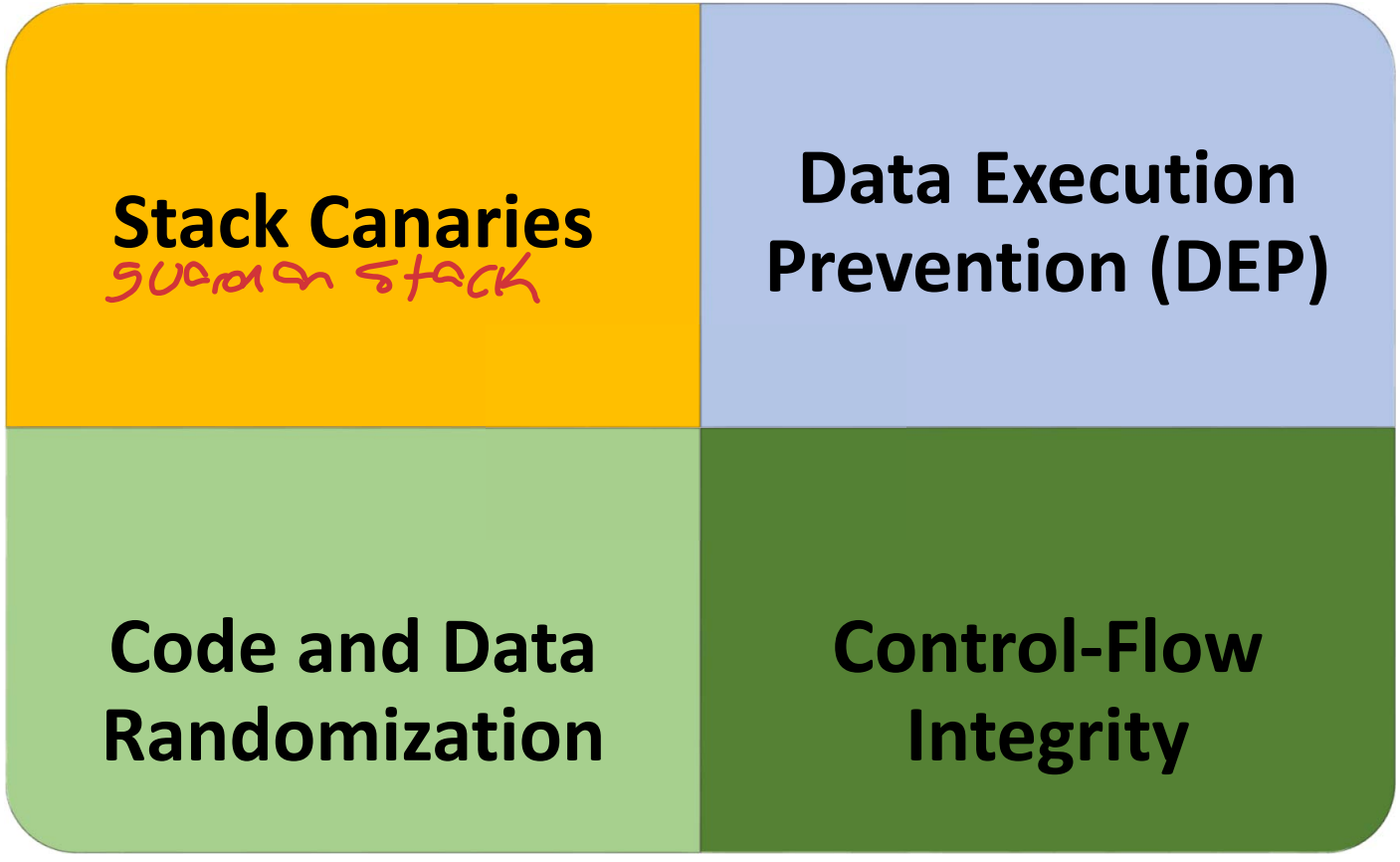
steps

1. Leveraging memory corruption **vulnerabilities**
2. Constructing special input (**exploit**):
   - Performing certain **computation**
   - Gaining the **control** of the program

Same principle, different mechanism

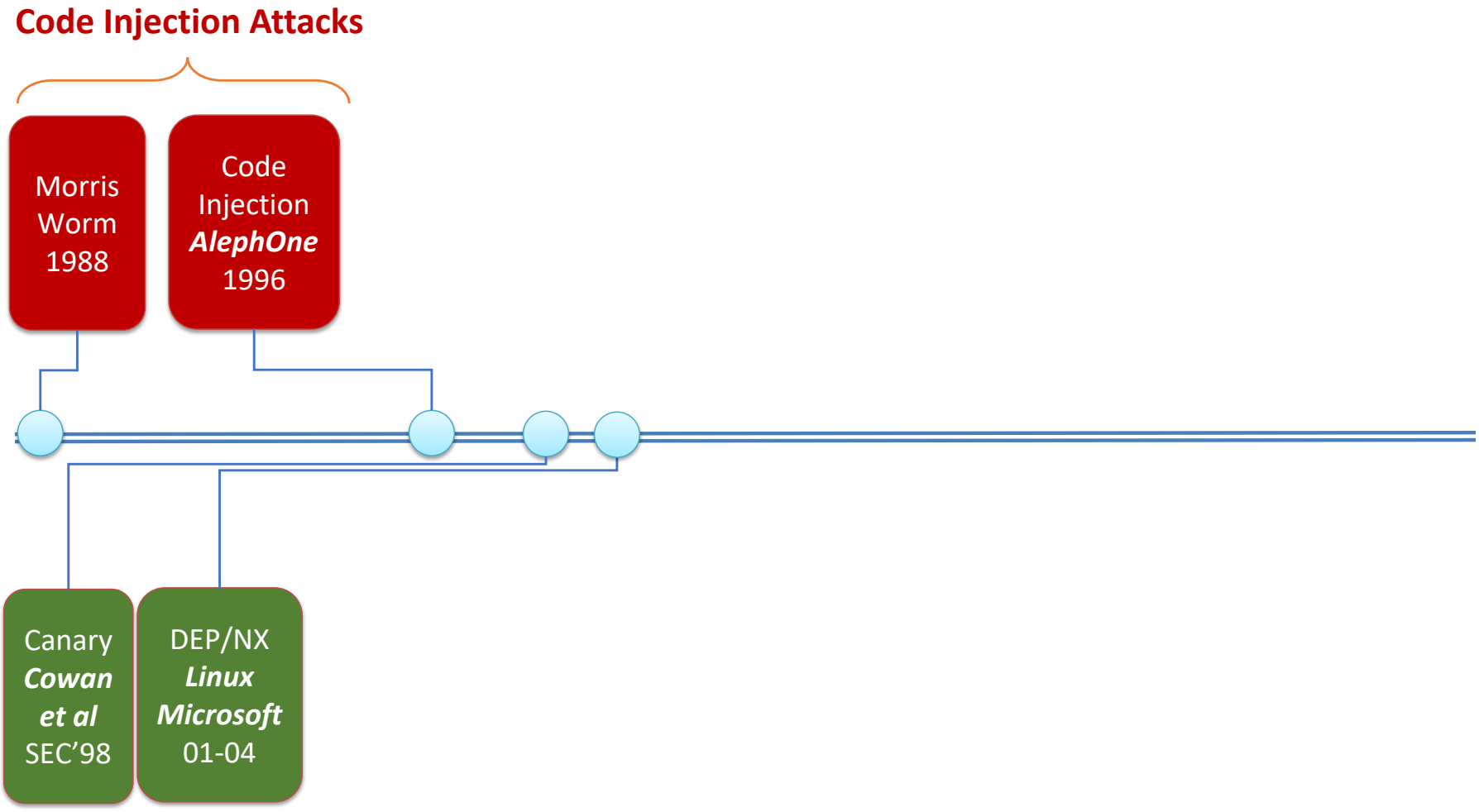
1. Code injection
2. Code reuse

The Run-time Defenses Against Memory Corruptions



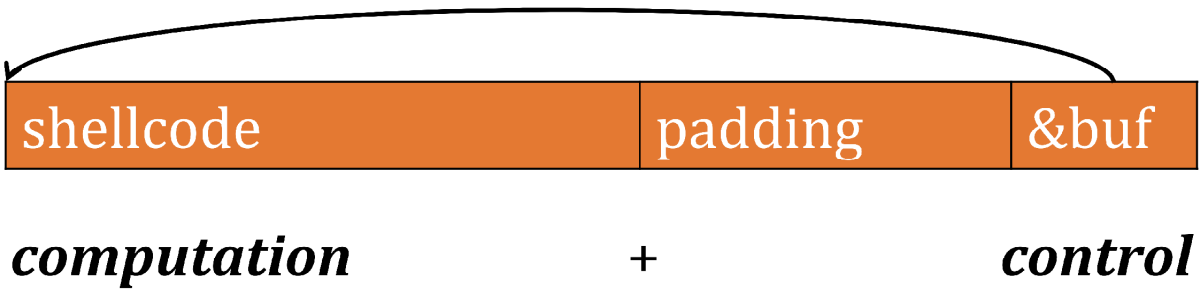
Defense Against Code Injection

Defense Against Code Injection

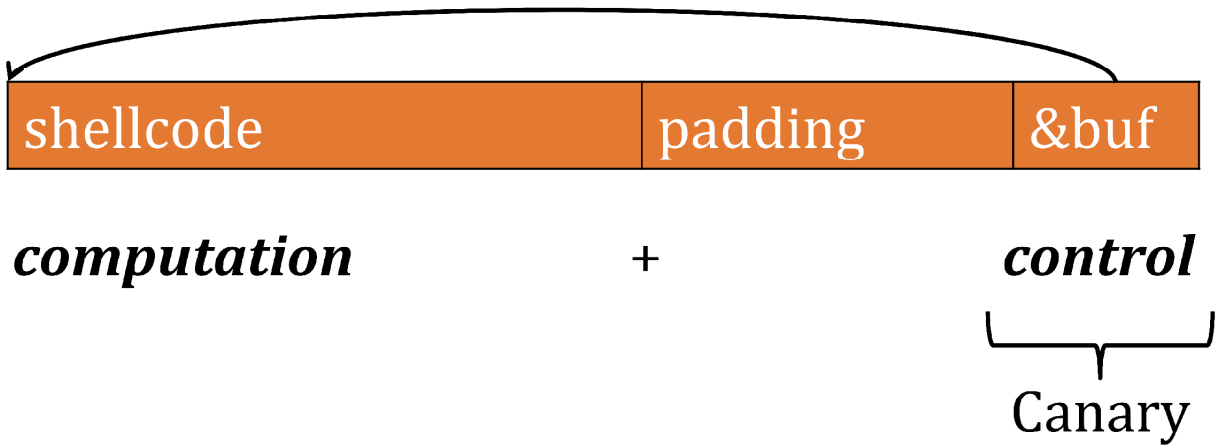


How Canary Defending Against Control Flow Hijacks

overflow buffer



How Canary Defending Against Control Flow Hijacks



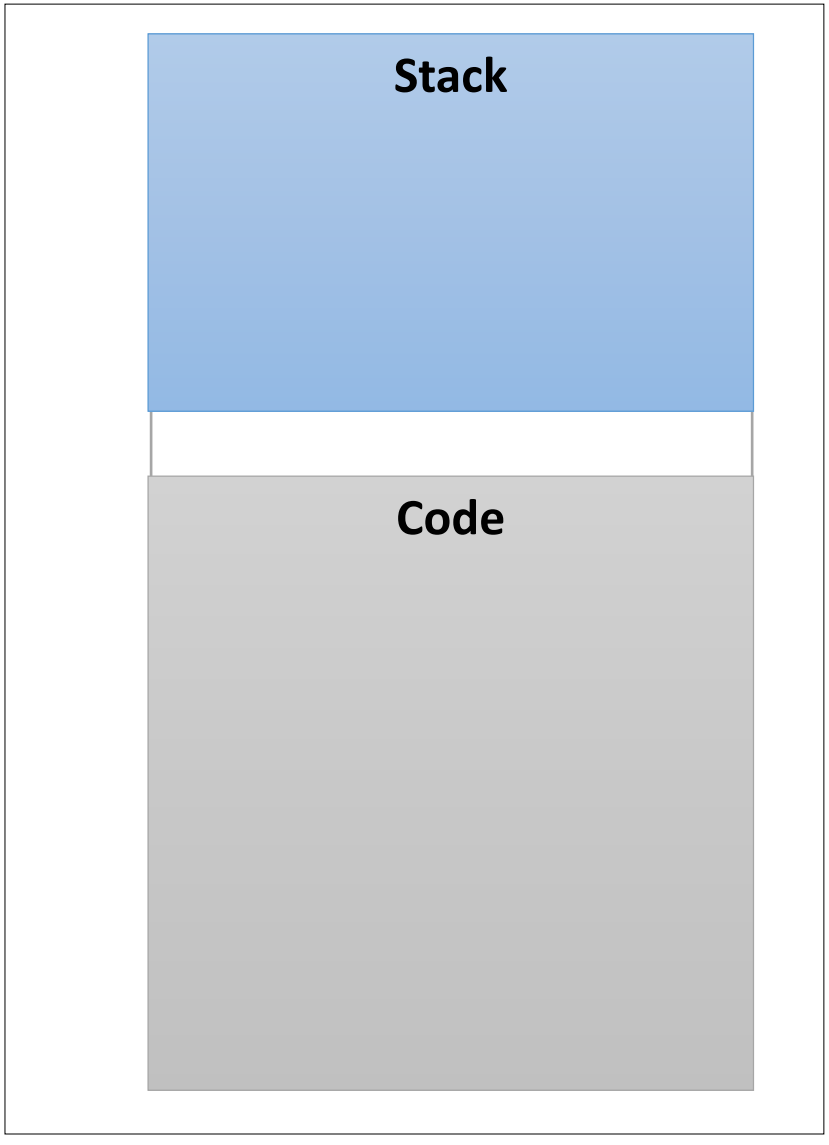
Canary Defense

Wikipedia: "the historic practice of using canaries in coal mines, since they would be affected by toxic gases earlier than the miners, thus providing a biological warning system."

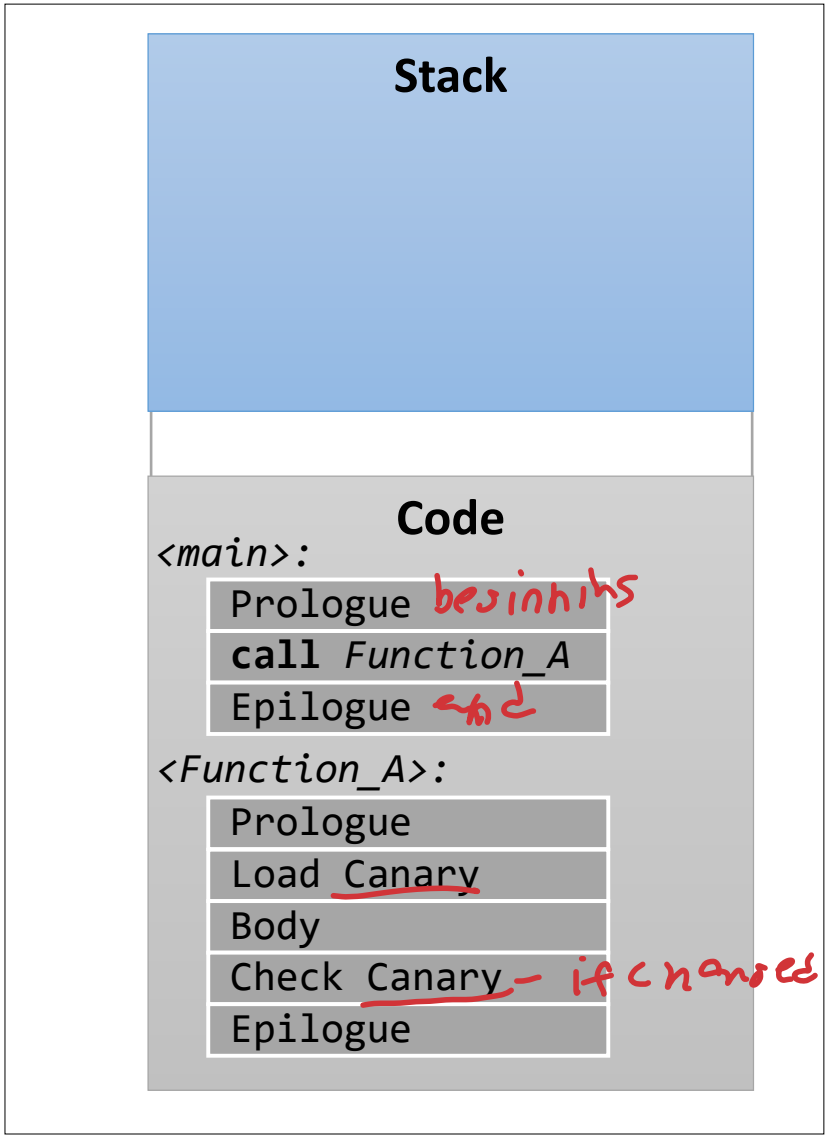
Canary / Stack Cookies

http://en.wikipedia.org/wiki/File:Domestic_Canary_-_Serinus_canaria.jpg

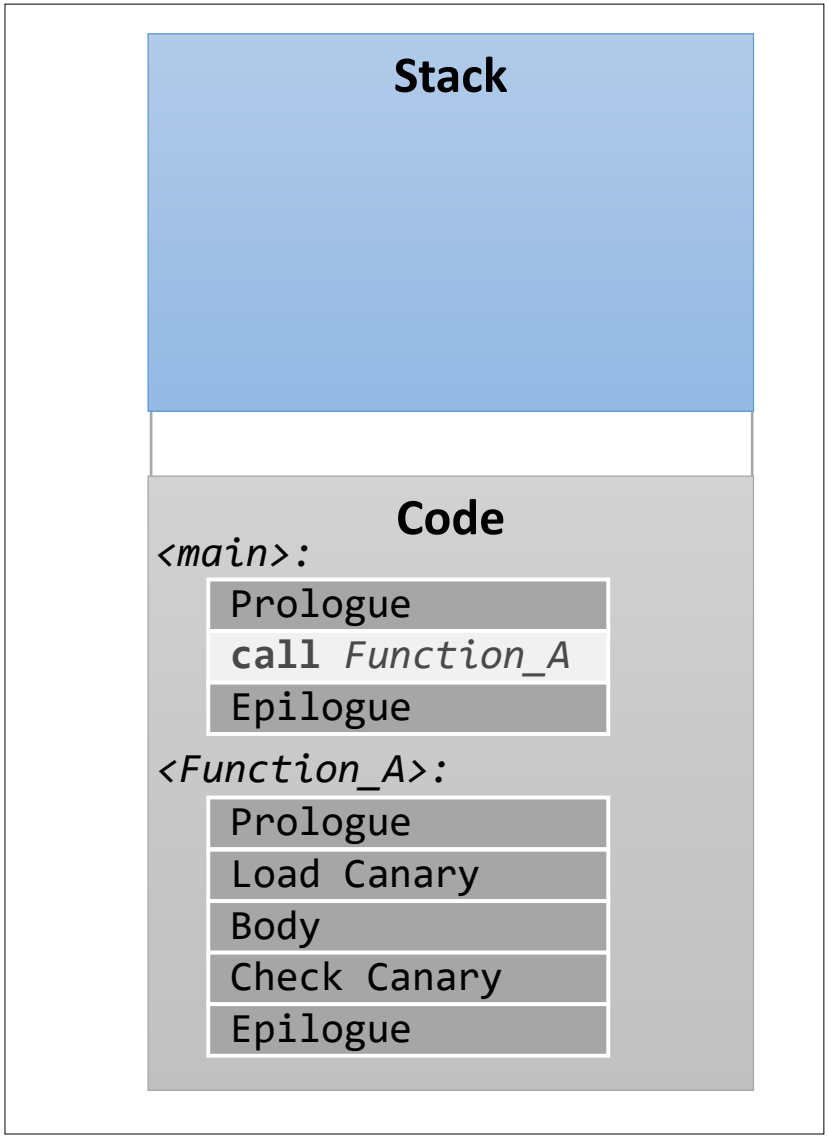
Canary Defense



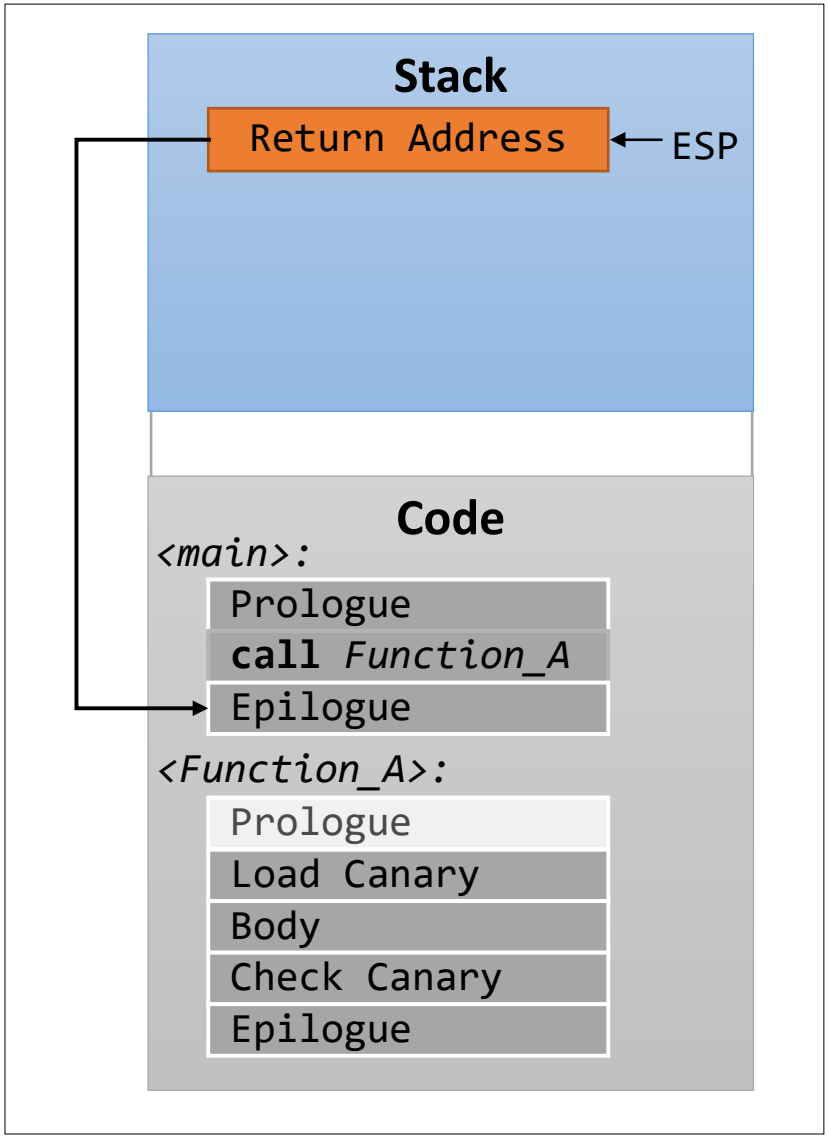
Canary Defense



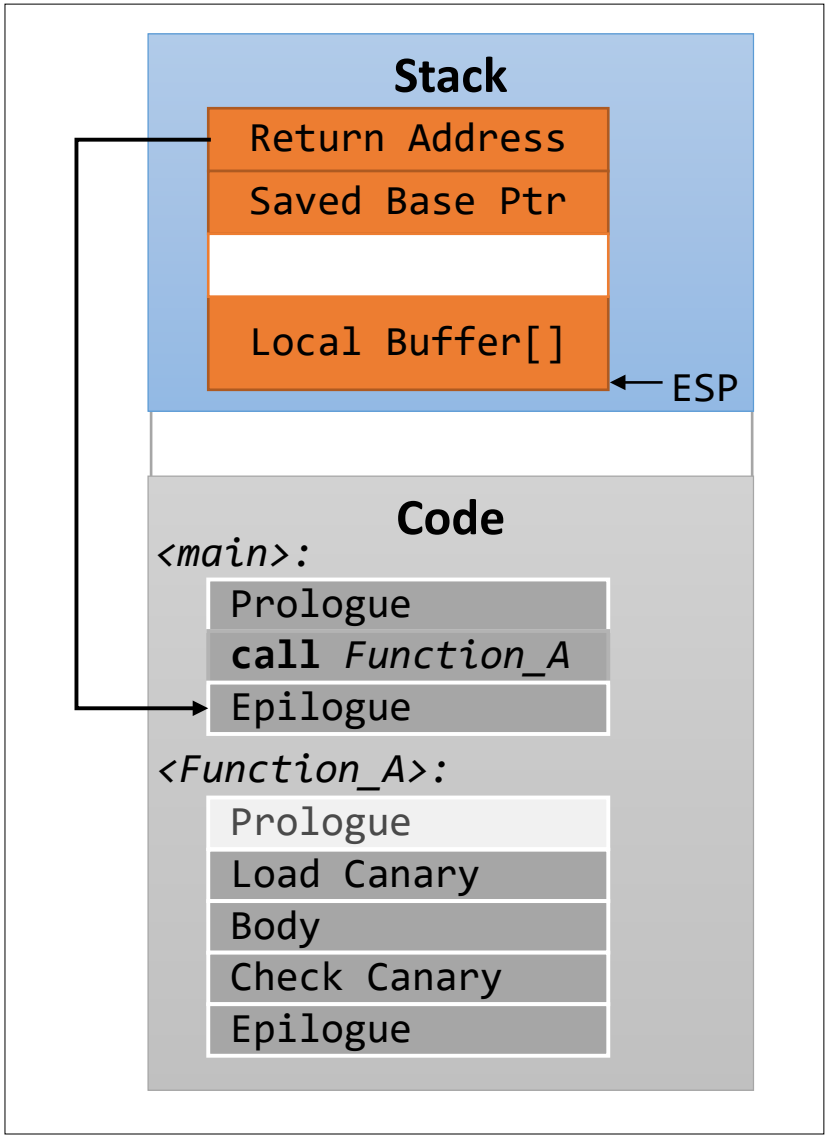
Canary Defense



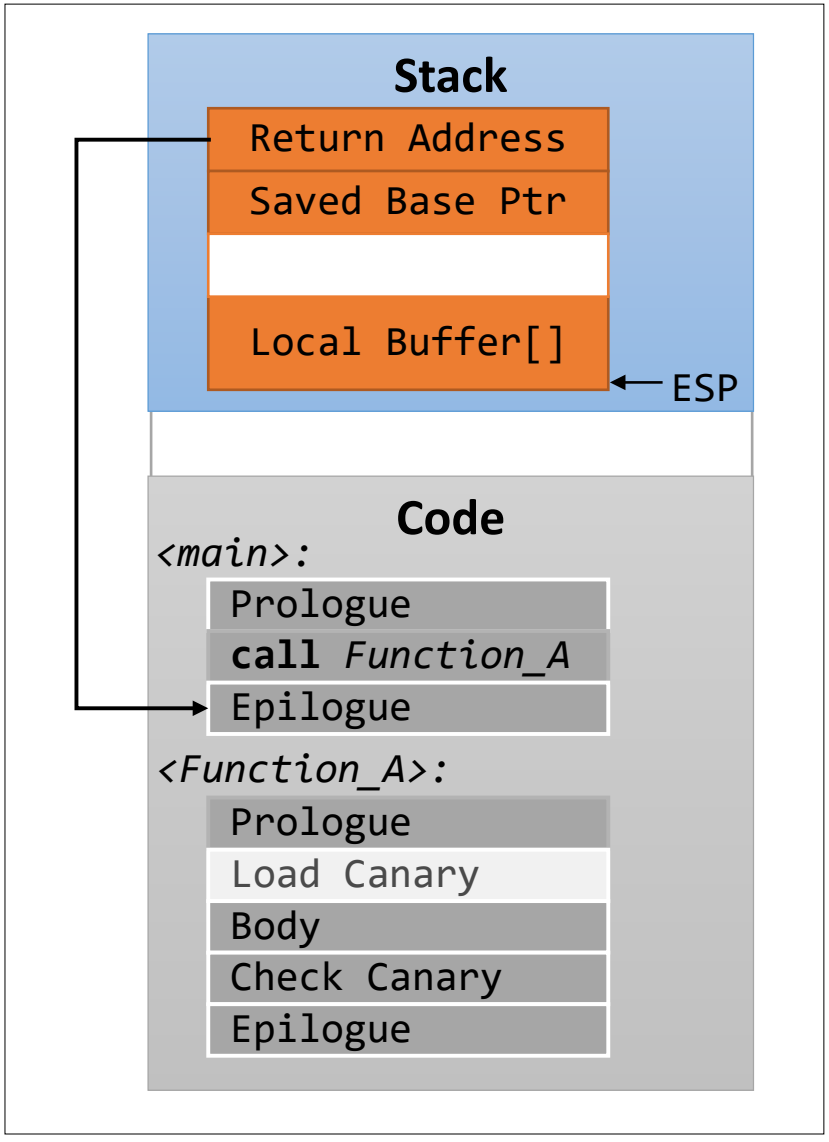
Canary Defense



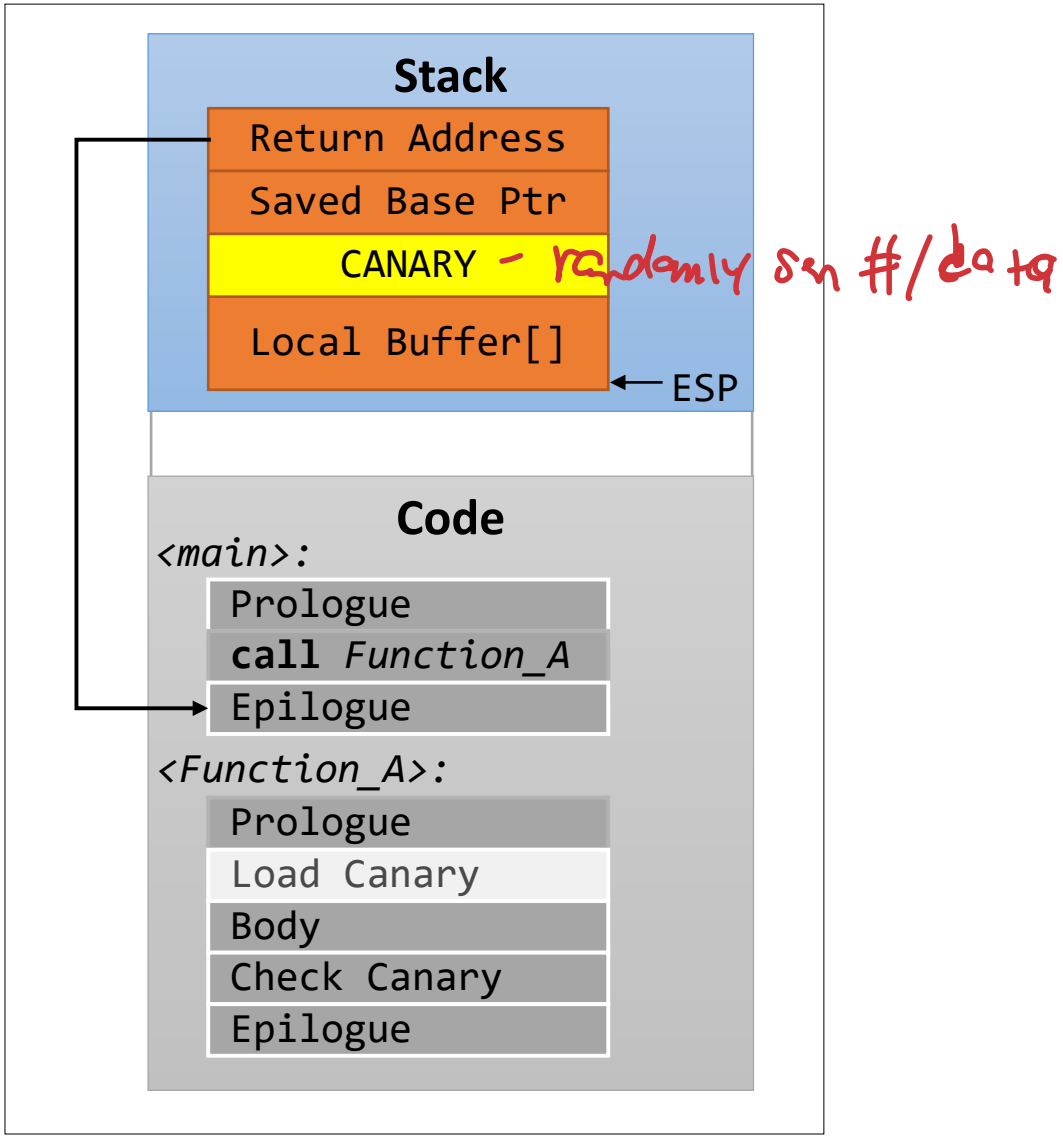
Canary Defense



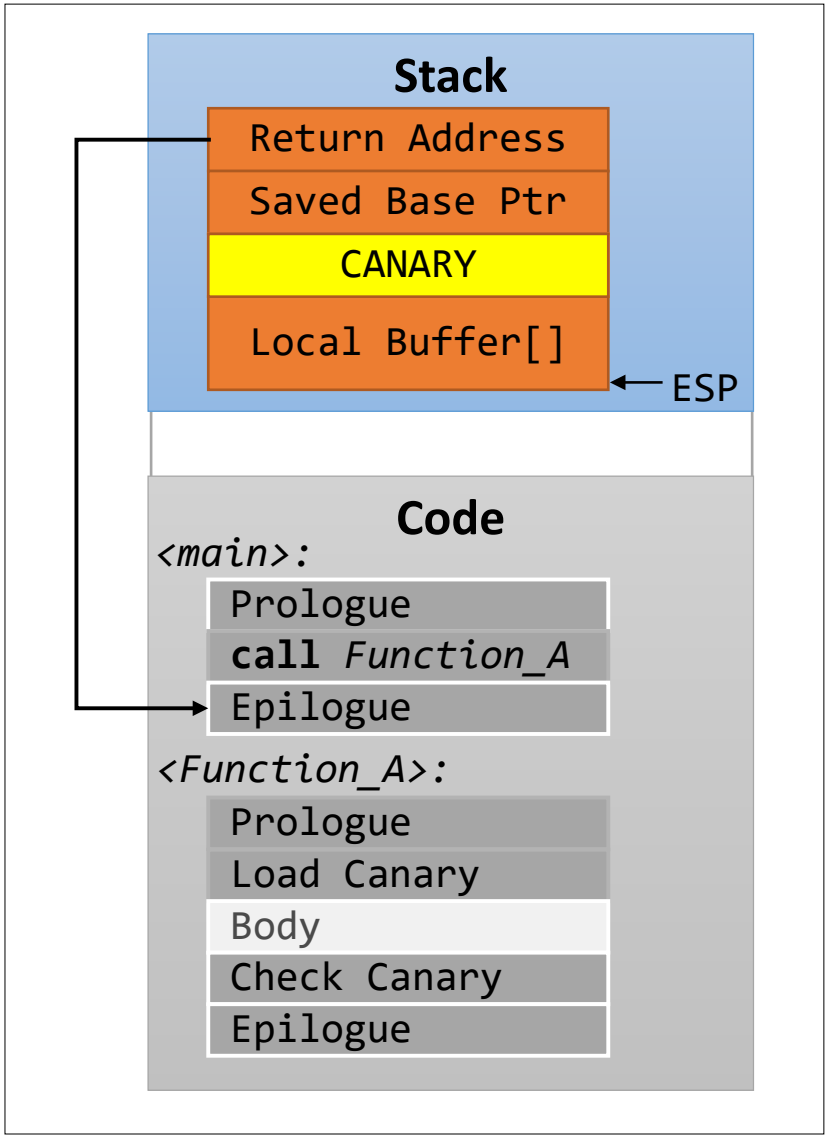
Canary Defense



Canary Defense

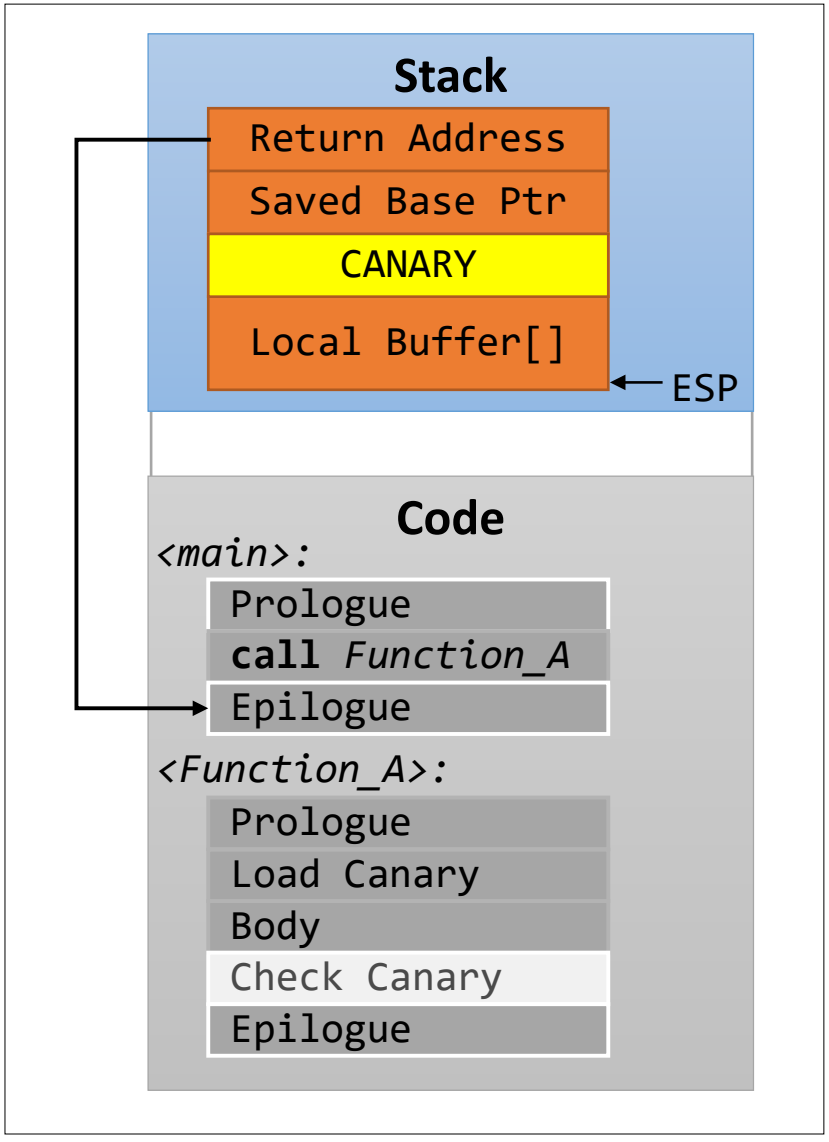


Canary Defense

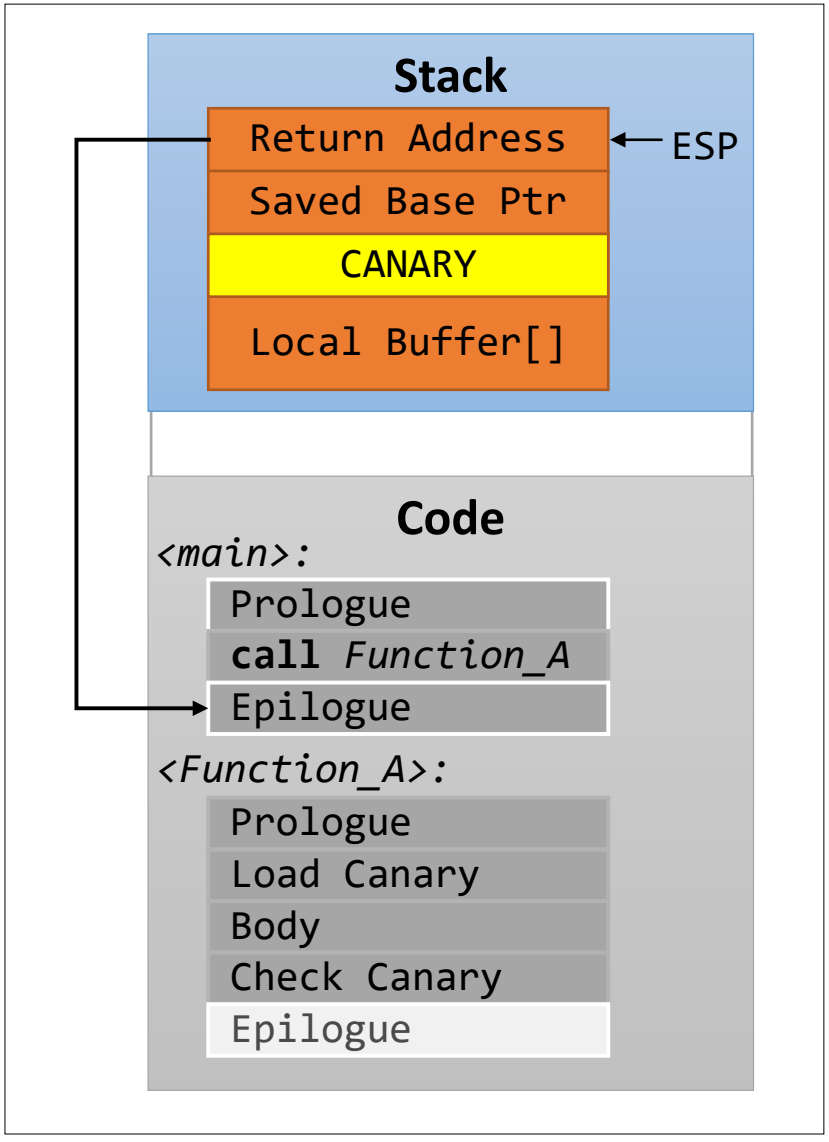


B/C randomly generated if the attacker override the generated data the program will know

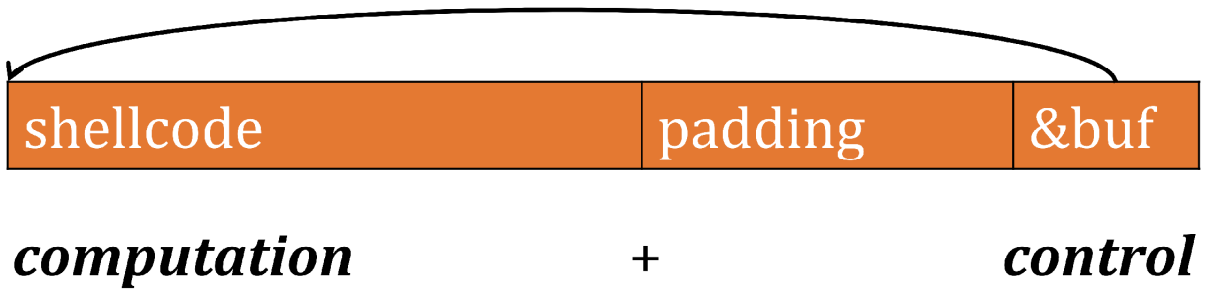
Canary Defense



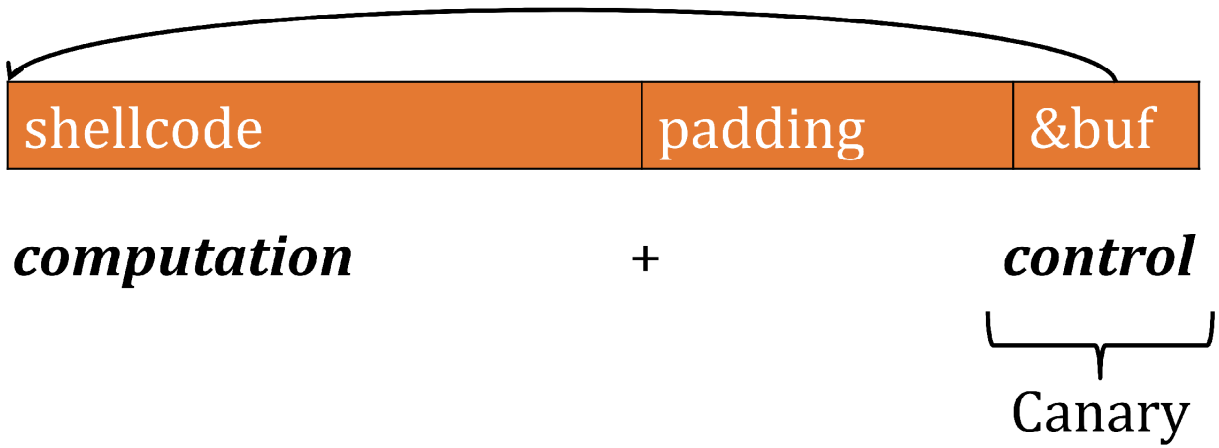
Canary Defense



How to defending against control flow hijacks



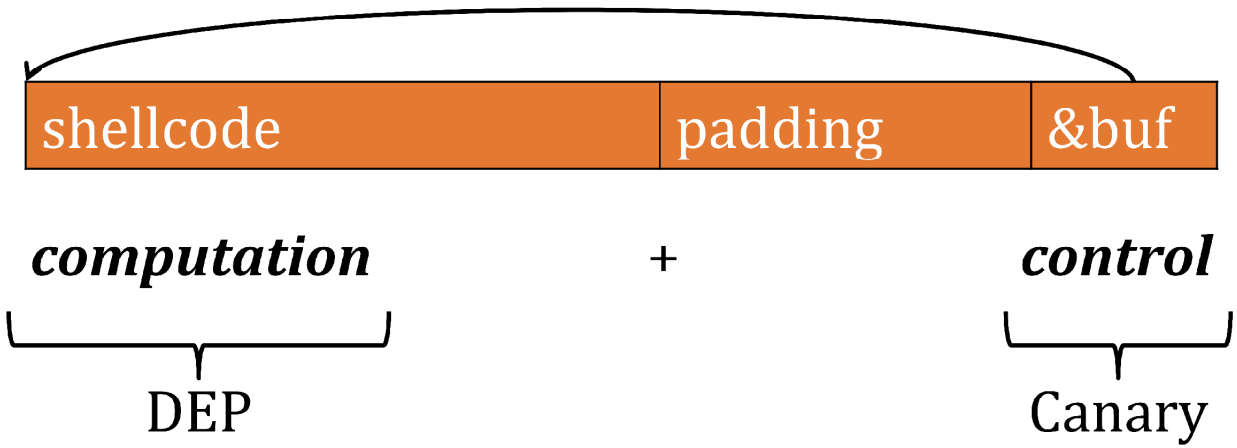
How to defending against control flow hijacks



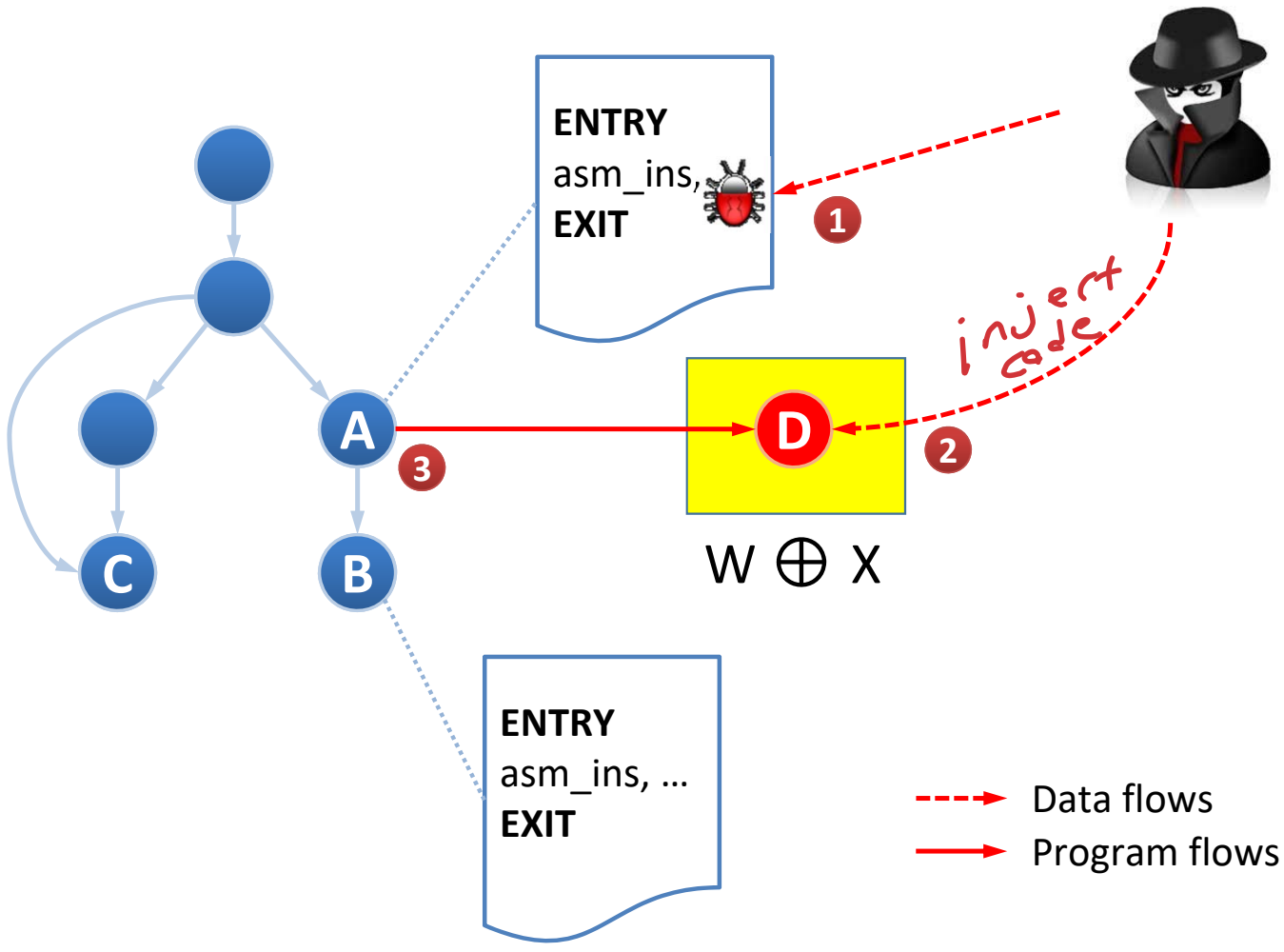
How to defending against control flow hijacks

We can stop people injecting code into mem

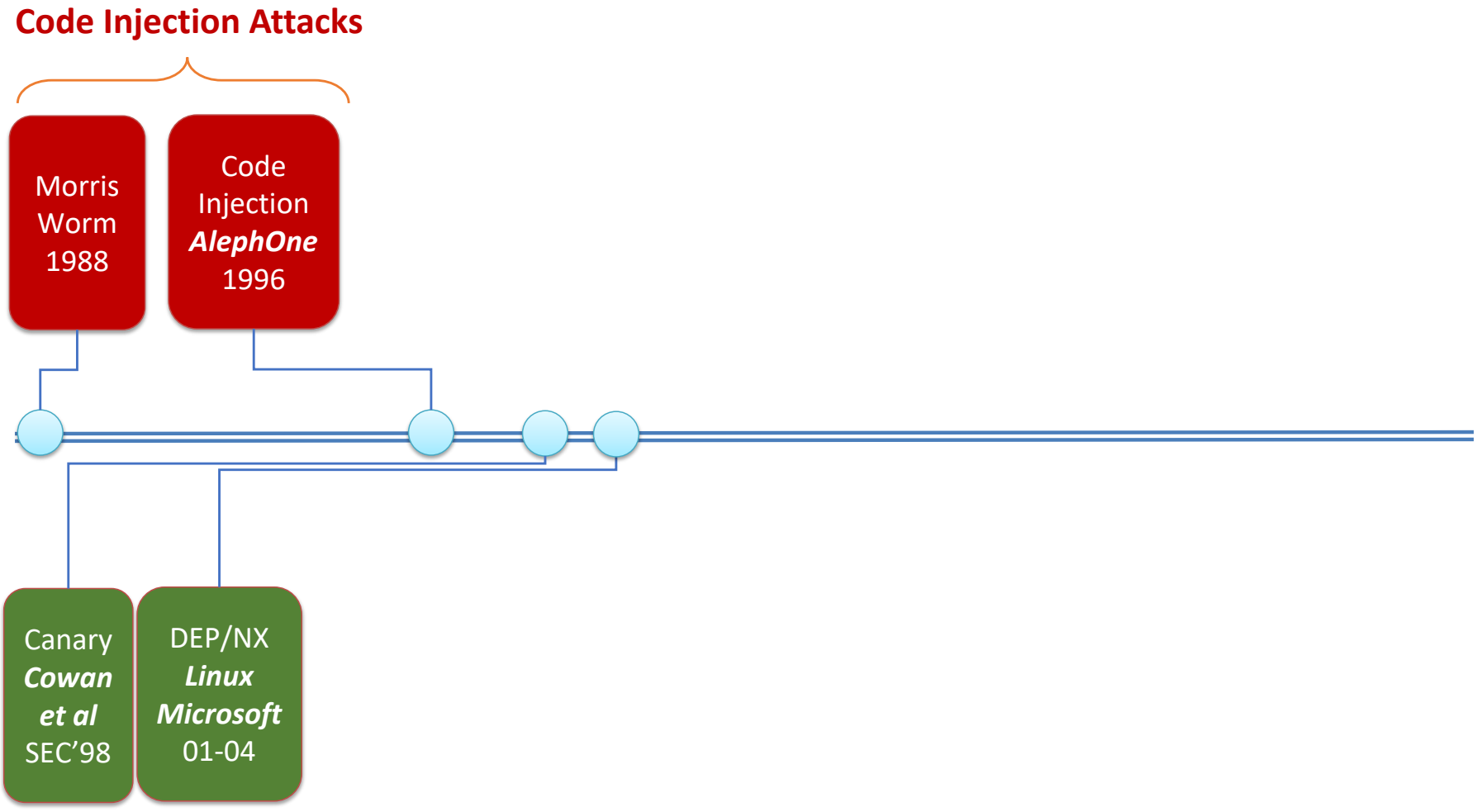
Data execution Proveation



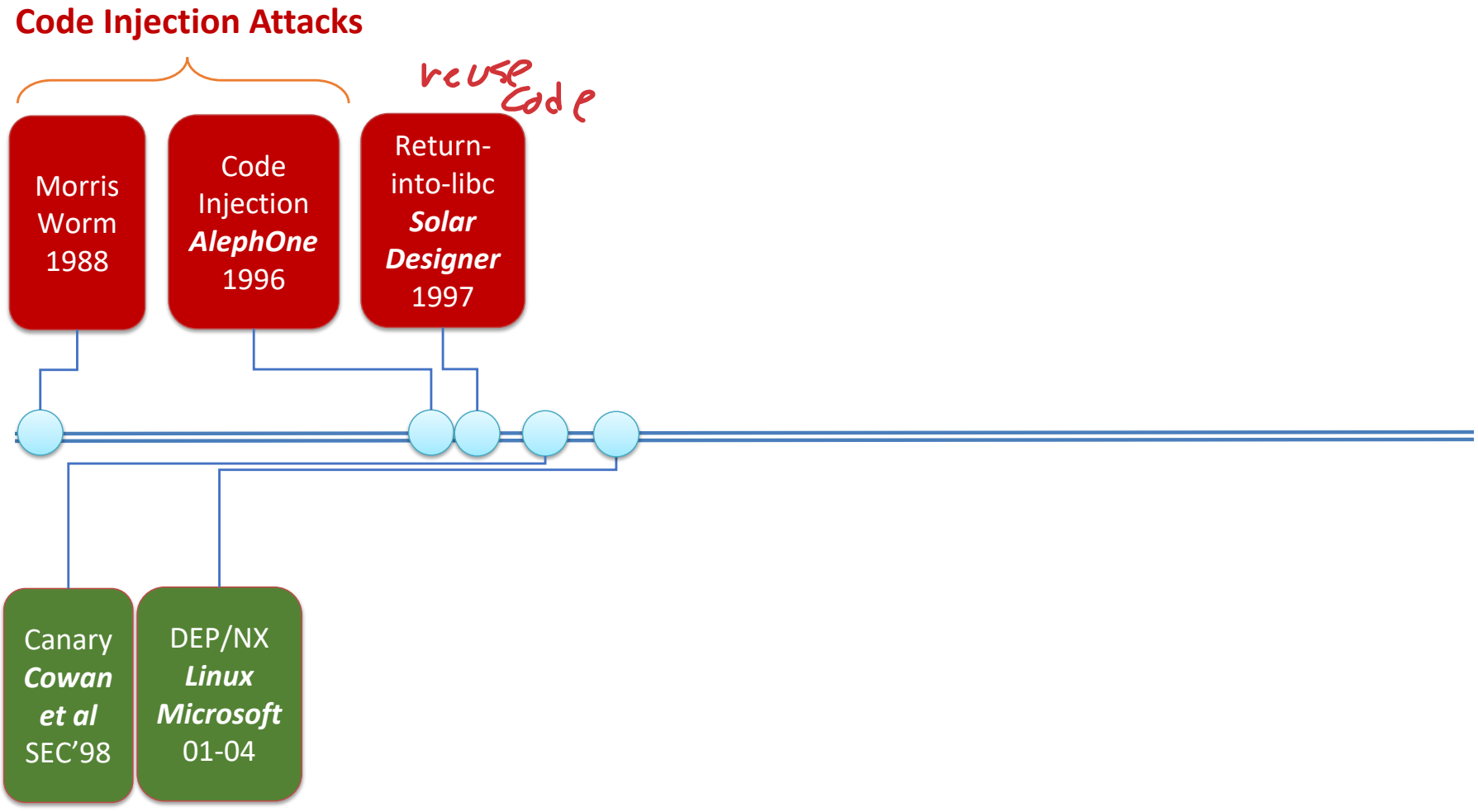
Data Execution Prevention



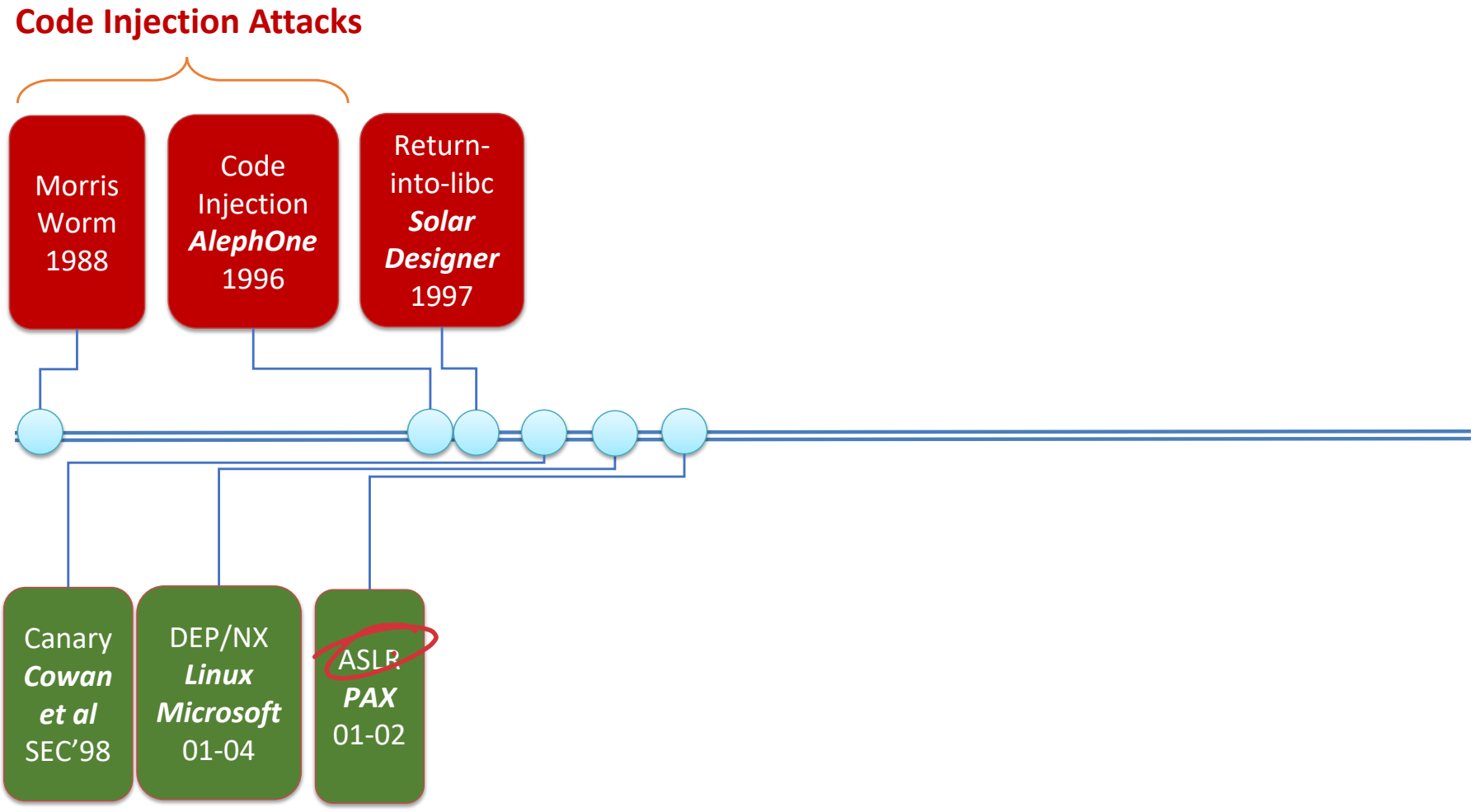
Defense Using Randomization



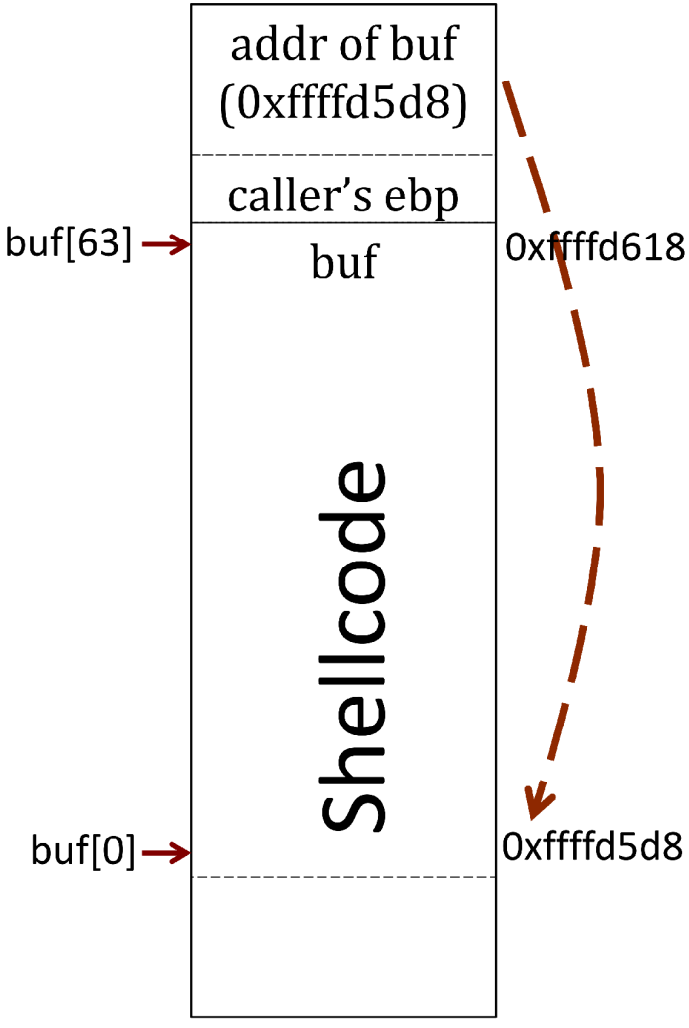
Defense Using Randomization



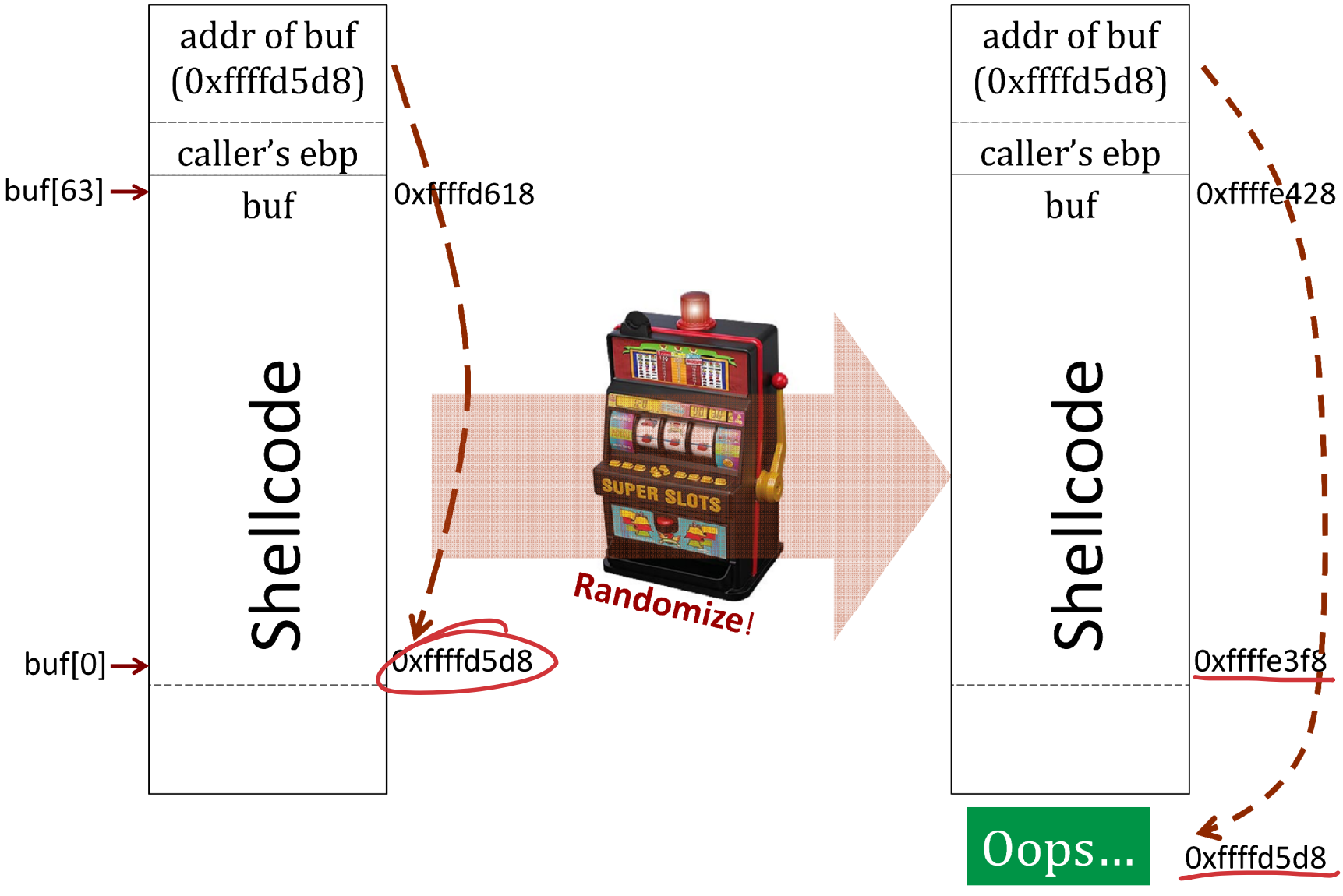
Defense Using Randomization



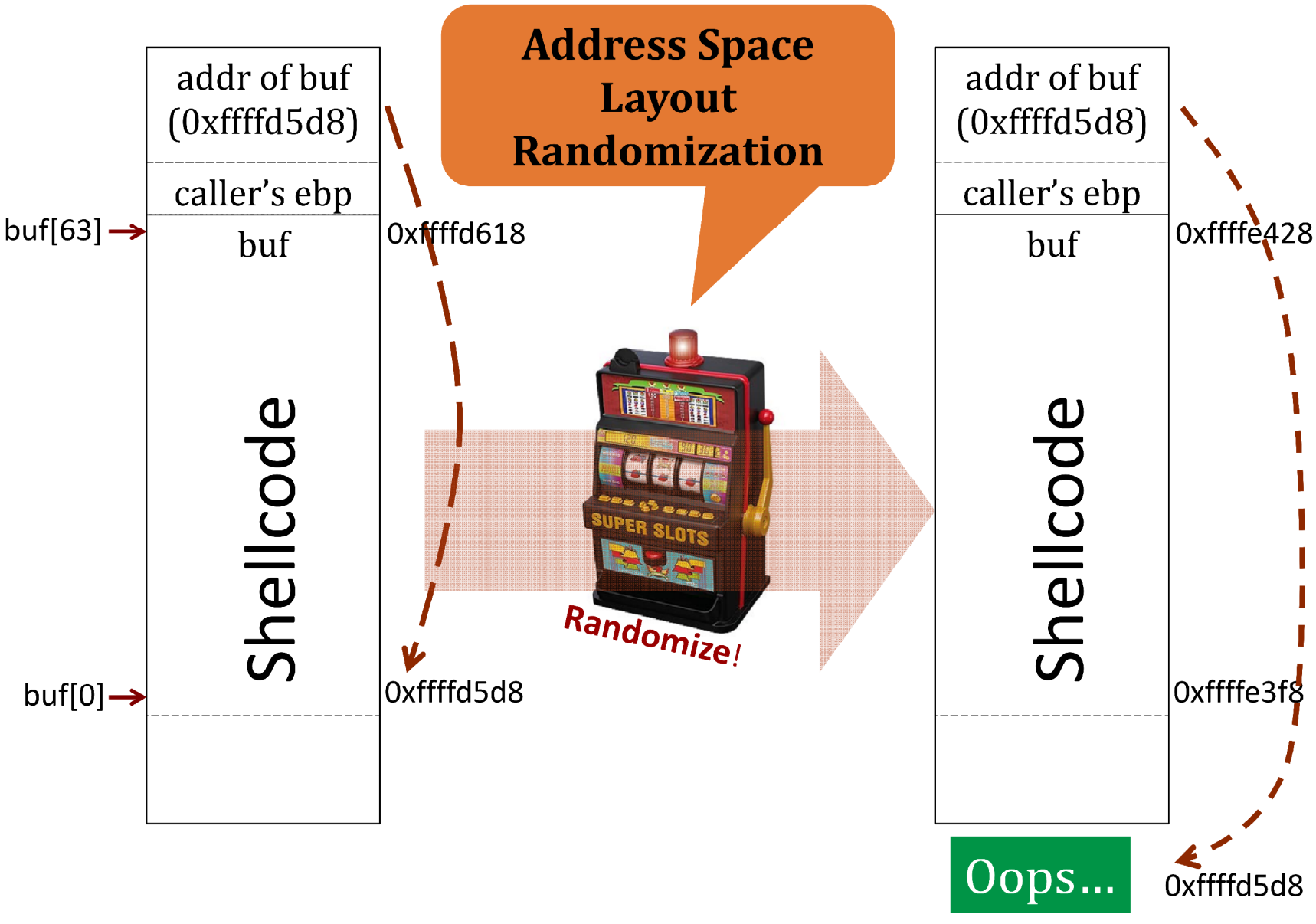
Address Space Layout Randomization (ASLR)



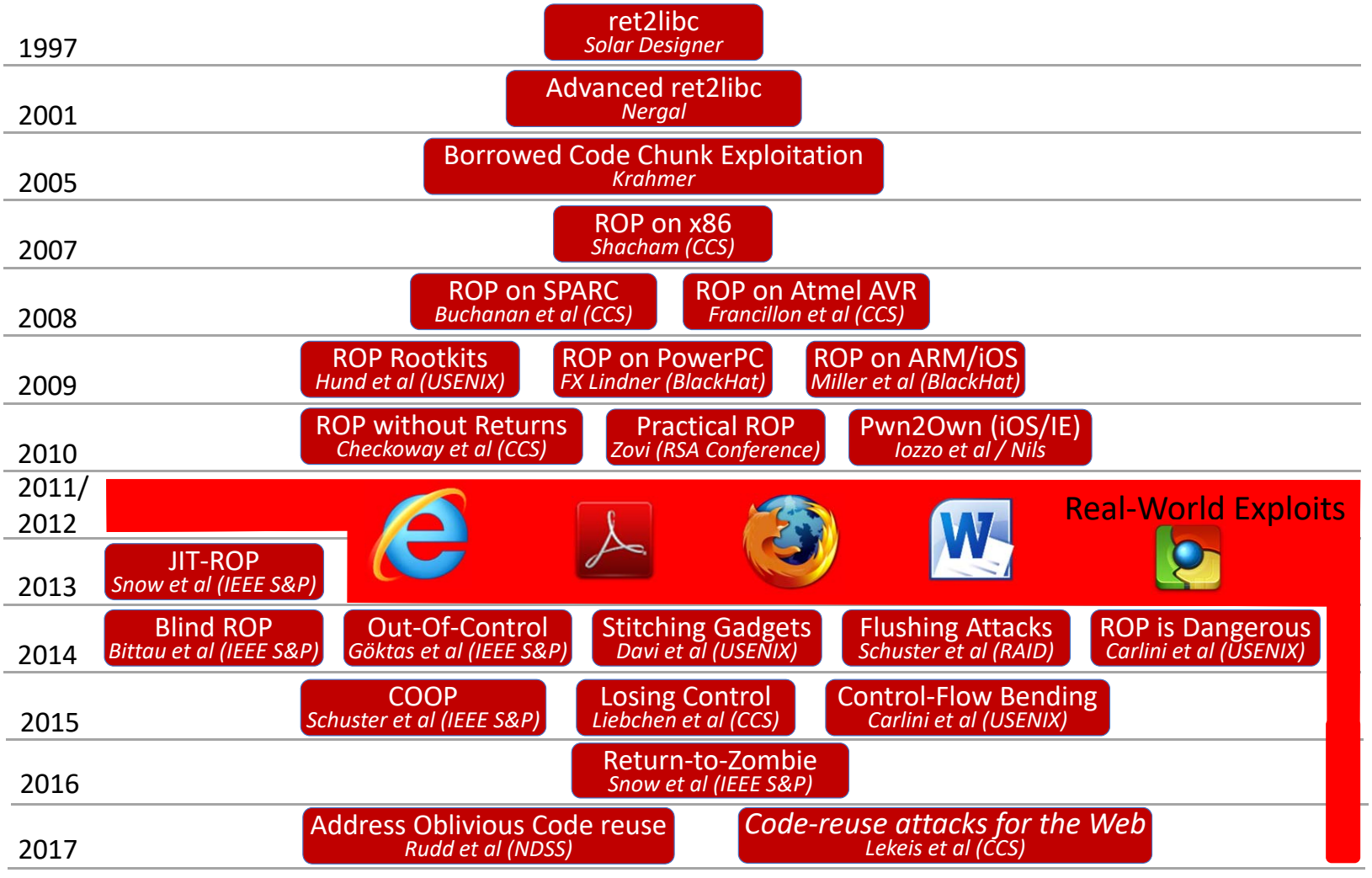
Address Space Layout Randomization (ASLR)



Address Space Layout Randomization (ASLR)



Recent Efforts on Code Reuse Attack



Return Oriented Programming (ROP)

The New Yo

Saturday, January 6, 2007

Daily Blog Tips awarded th

Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called "Reviews and Predictions". Among the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that

Ren
follo
imp

The
that
rela
the

Return Oriented Programming (ROP)

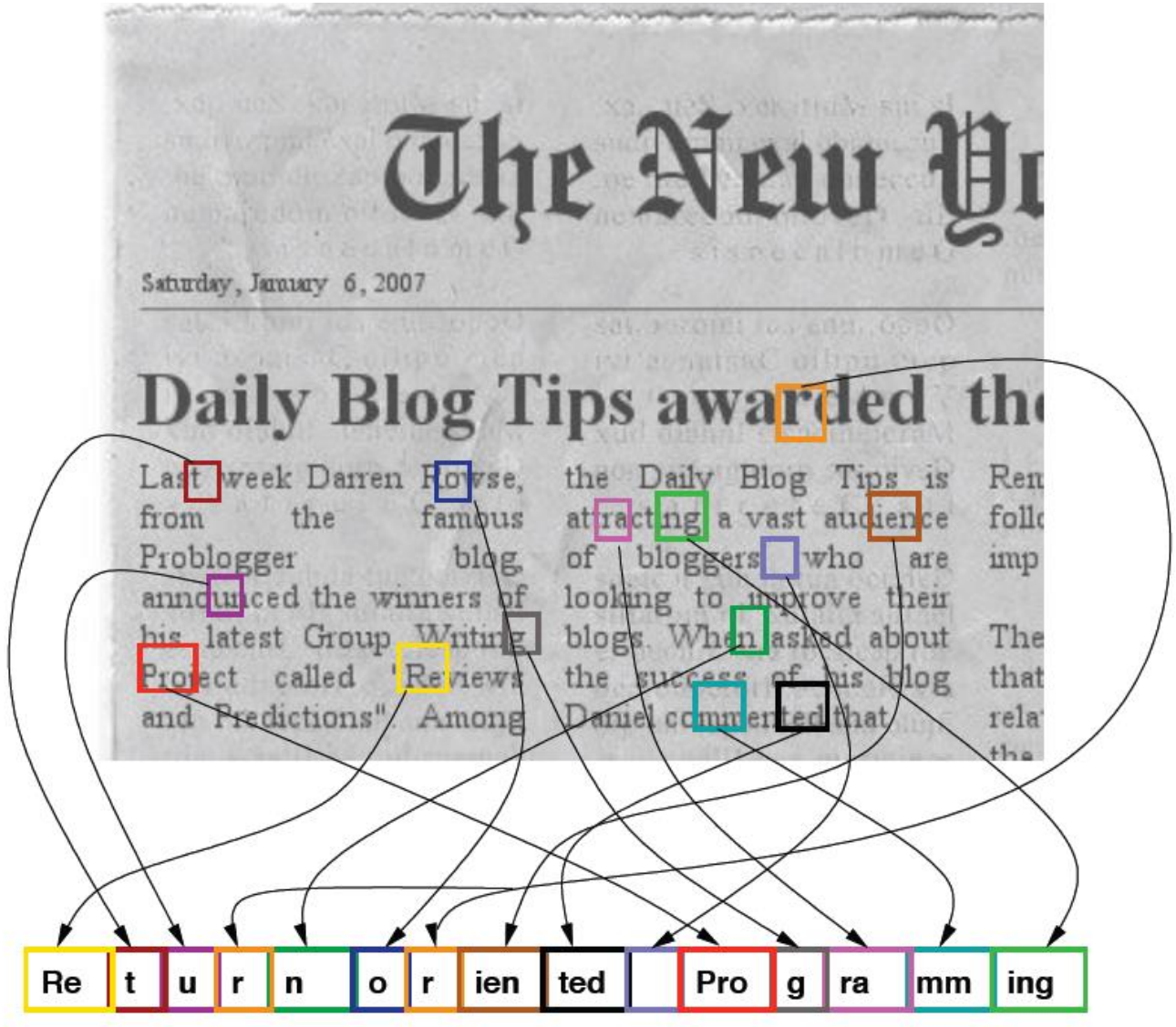
The New Yo

Saturday, January 6, 2007

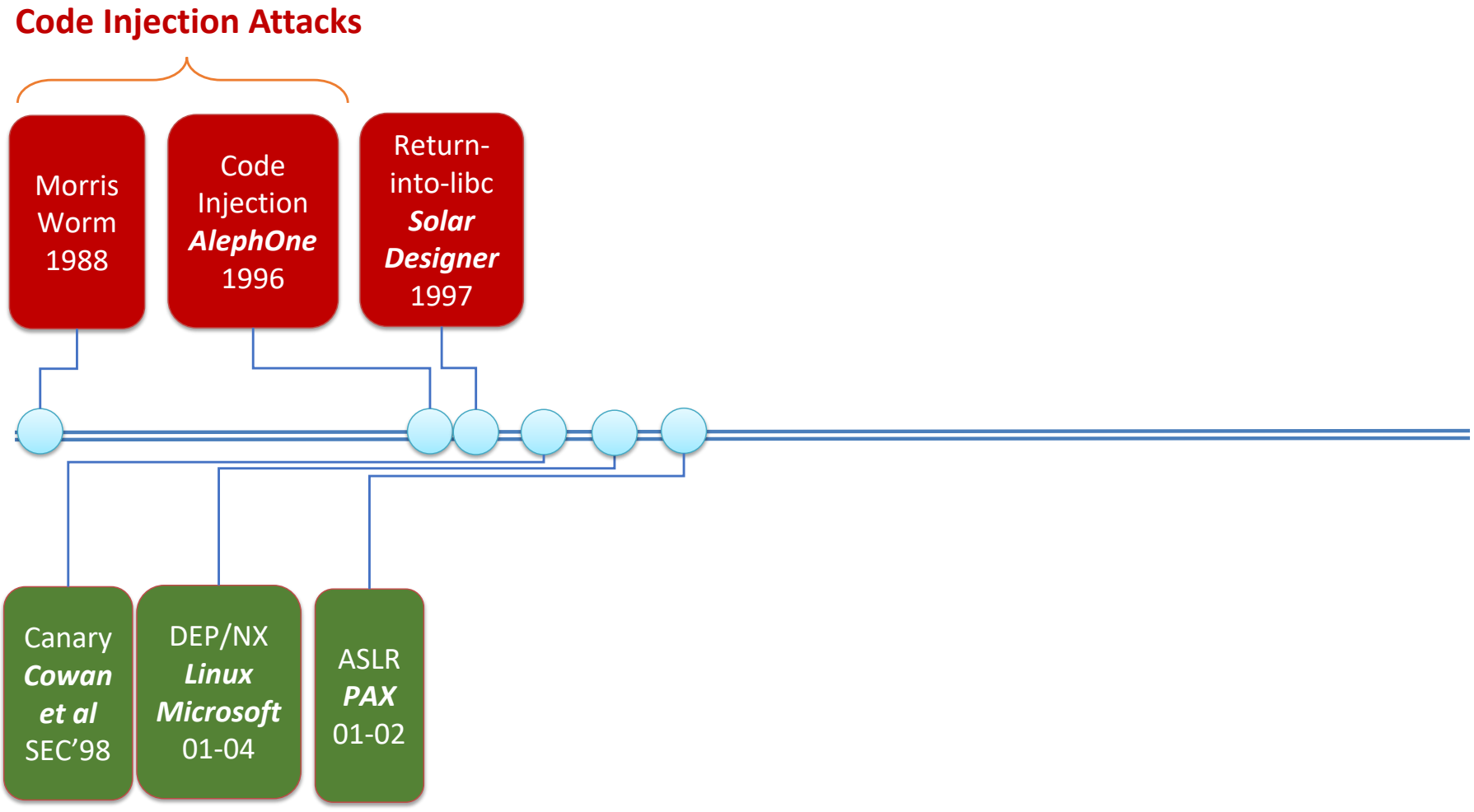
Daily Blog Tips awarded the

Last week Darren Rowse, from the famous Problogger blog, announced the winners of his latest Group Writing Project called 'Reviews and Predictions". Among the Daily Blog Tips is attracting a vast audience of bloggers who are looking to improve their blogs. When asked about the success of his blog Daniel commented that

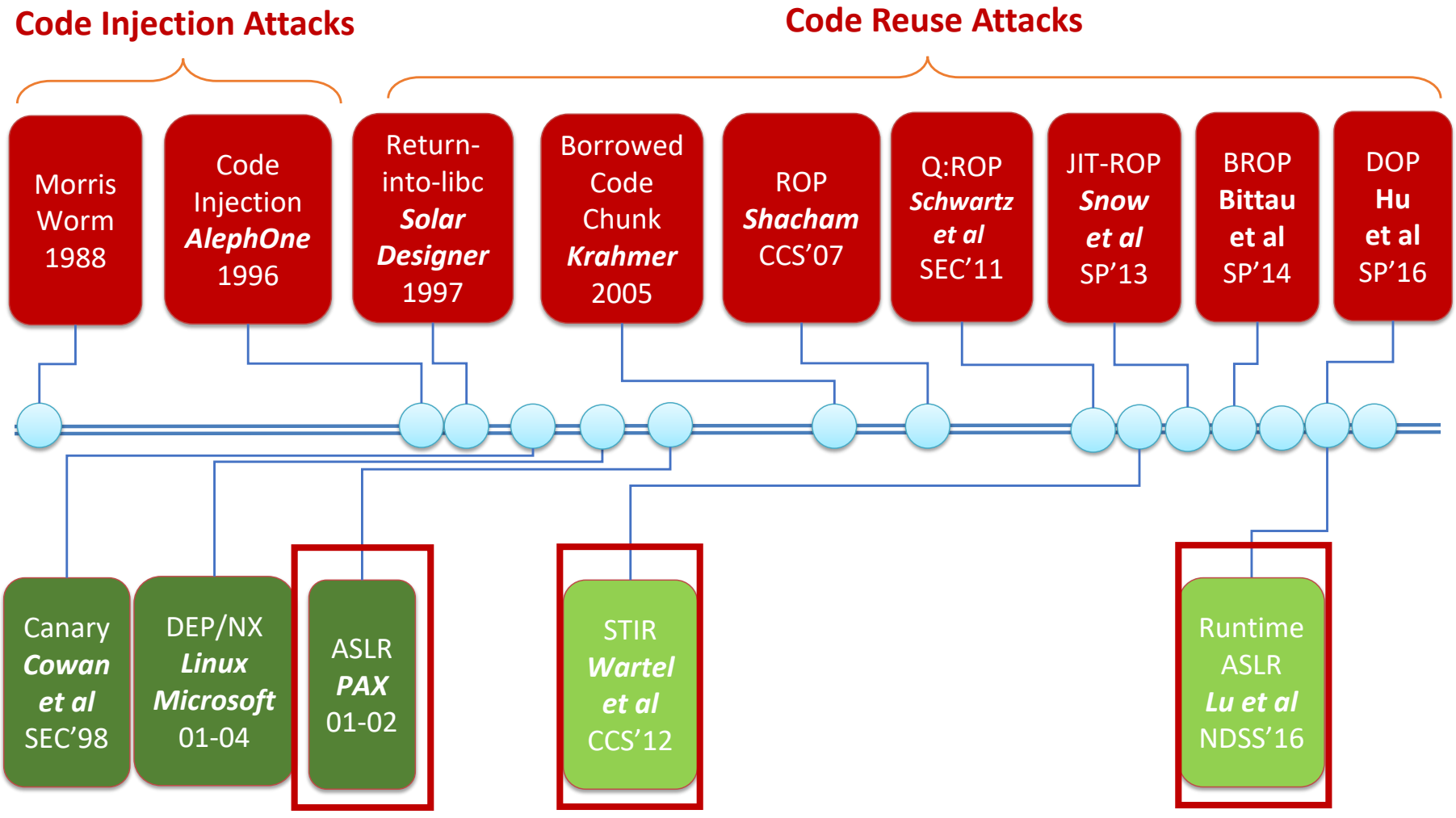
Return Oriented Programming (ROP)



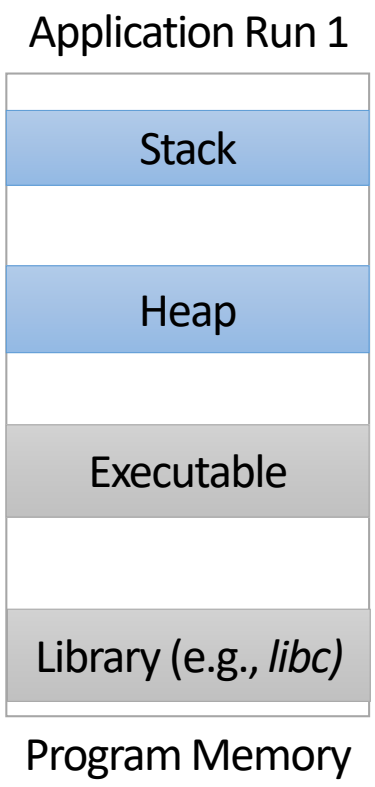
More Defenses Using Randomization



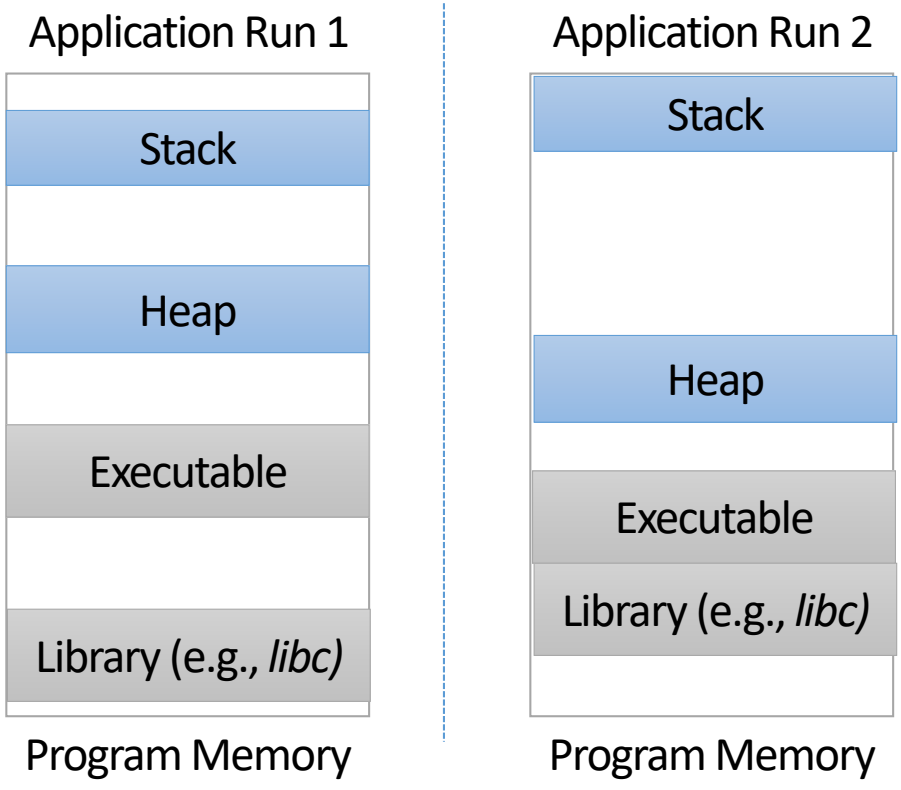
More Defenses Using Randomization



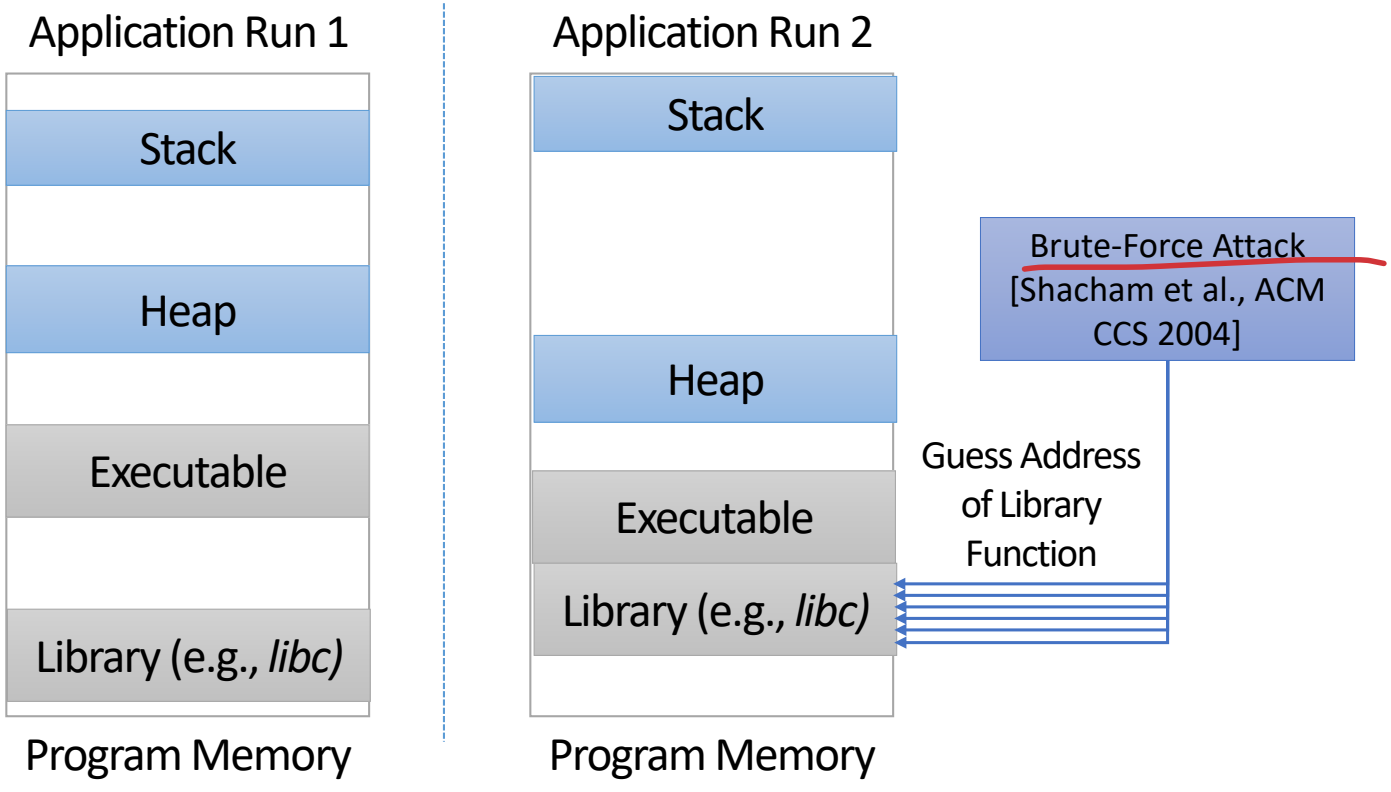
From Coarse-Grained to More Fine-Grained ASLR



From Coarse-Grained to More Fine-Grained ASLR

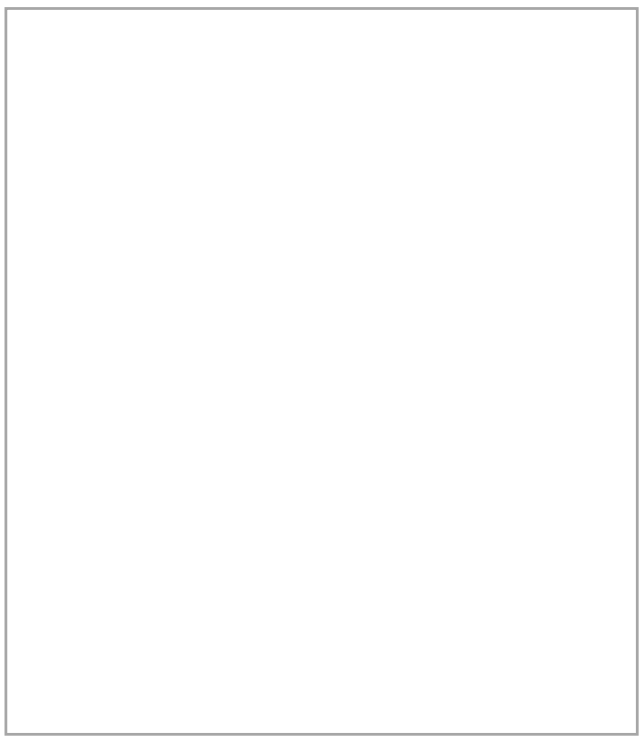


From Coarse-Grained to More Fine-Grained ASLR

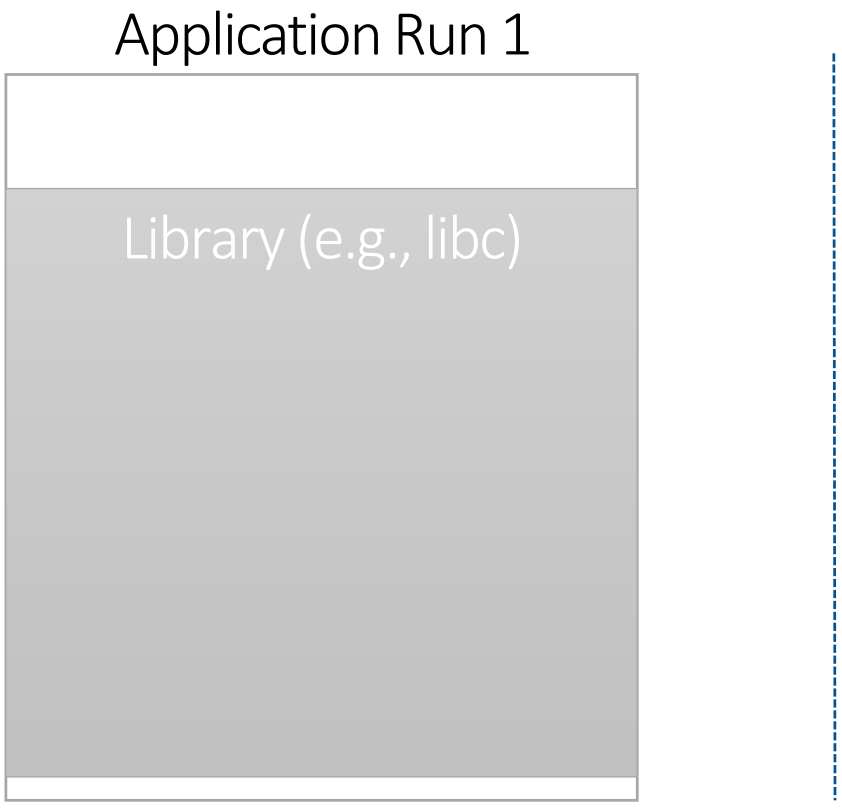


From Coarse-Grained to More Fine-Grained ASLR

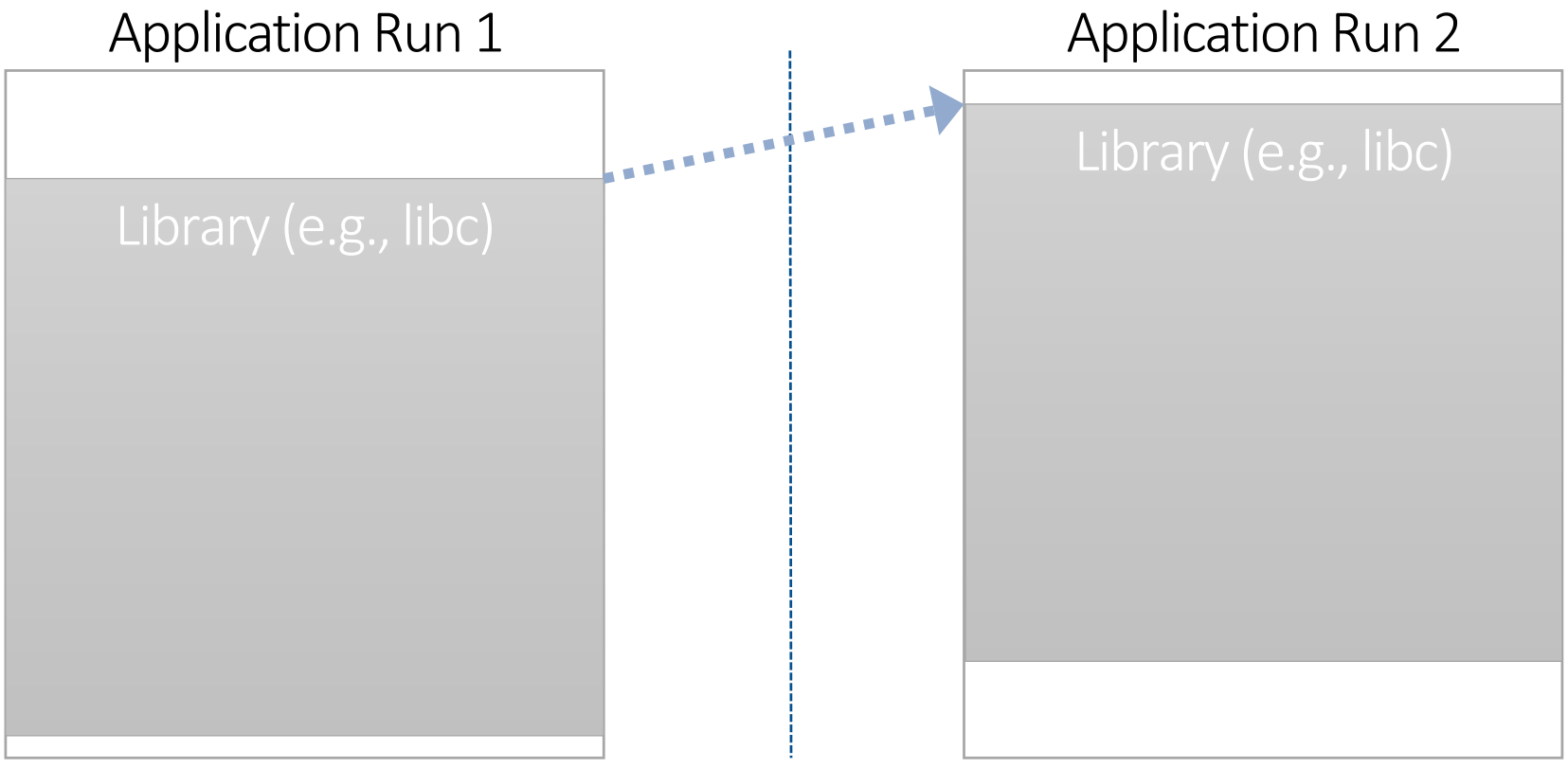
Application Run 1



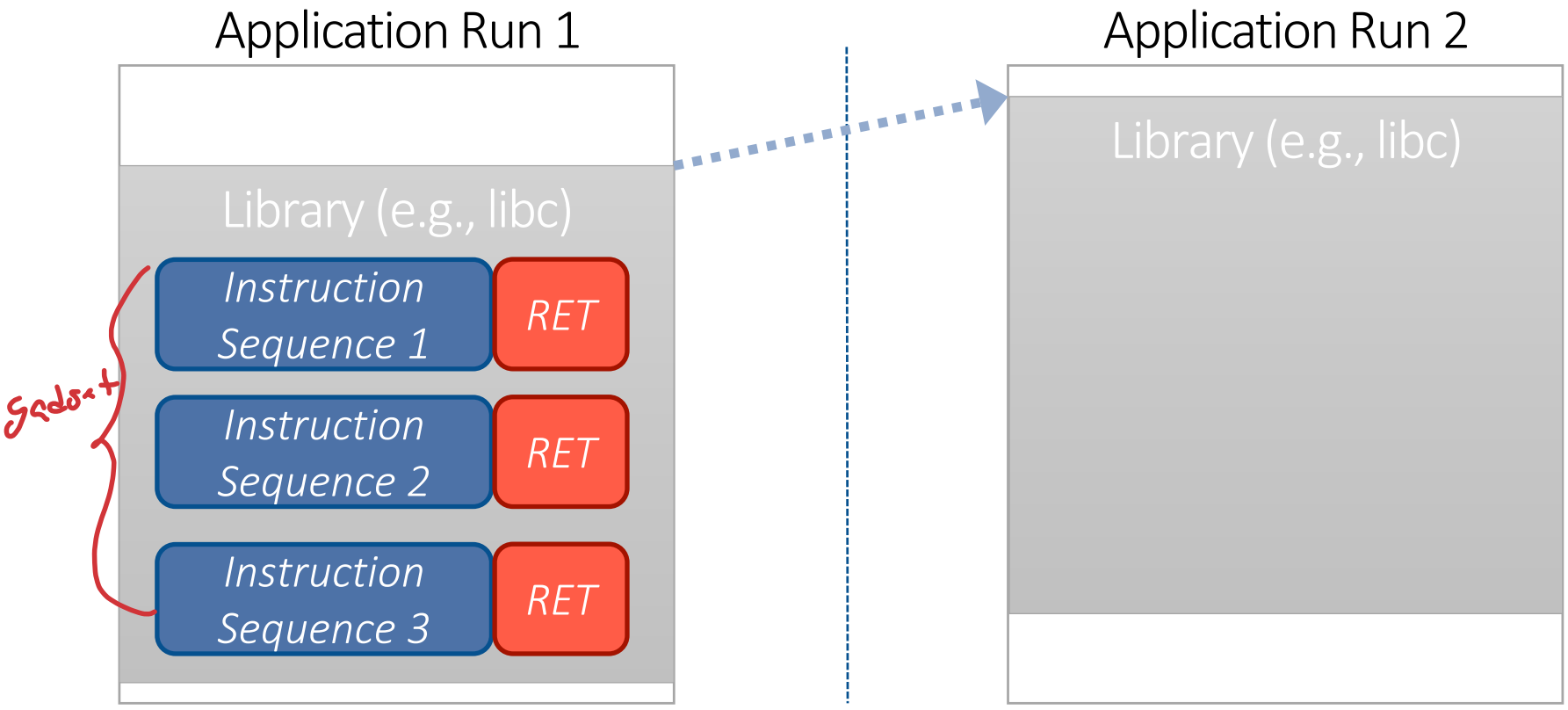
From Coarse-Grained to More Fine-Grained ASLR



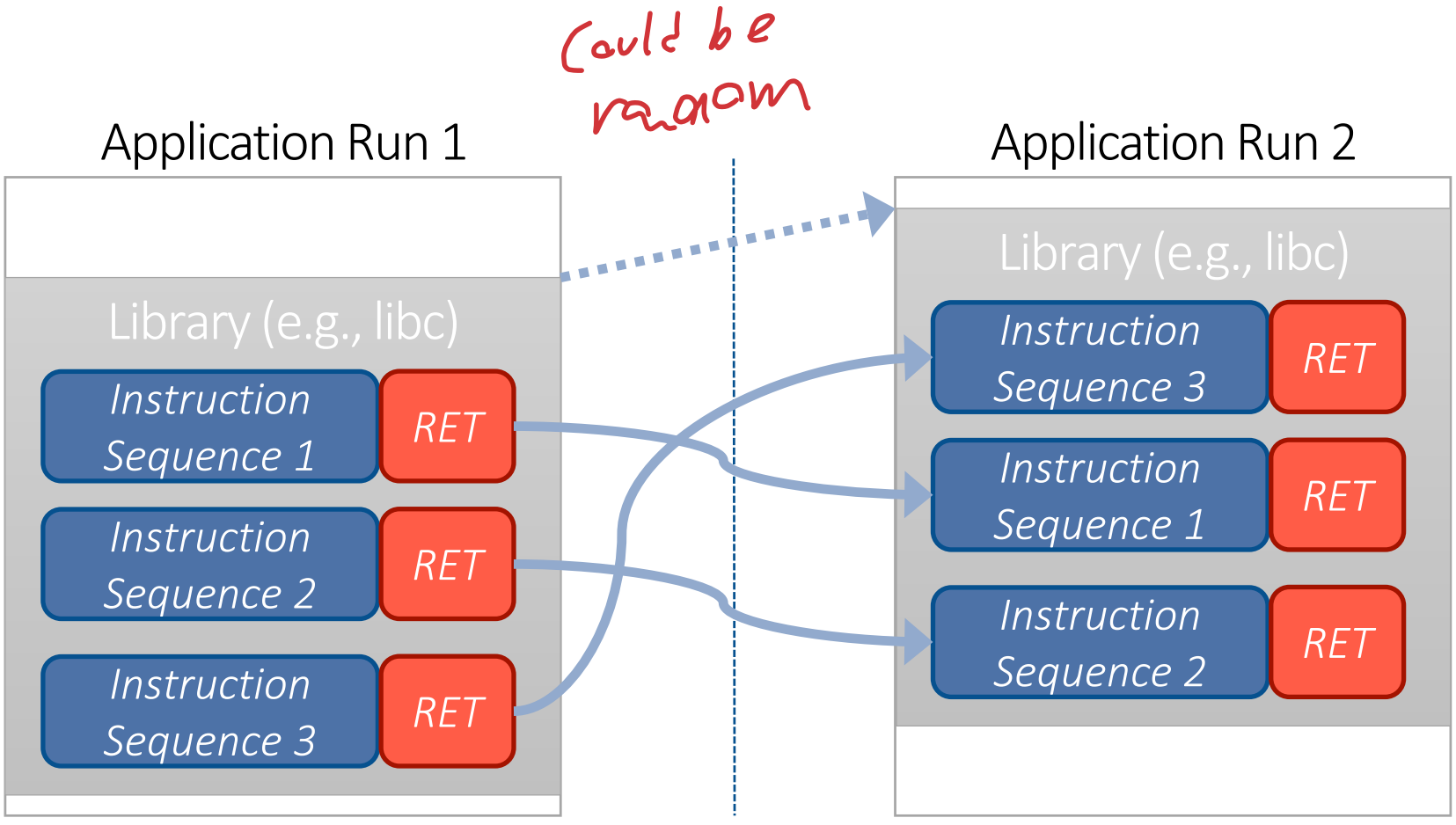
From Coarse-Grained to More Fine-Grained ASLR



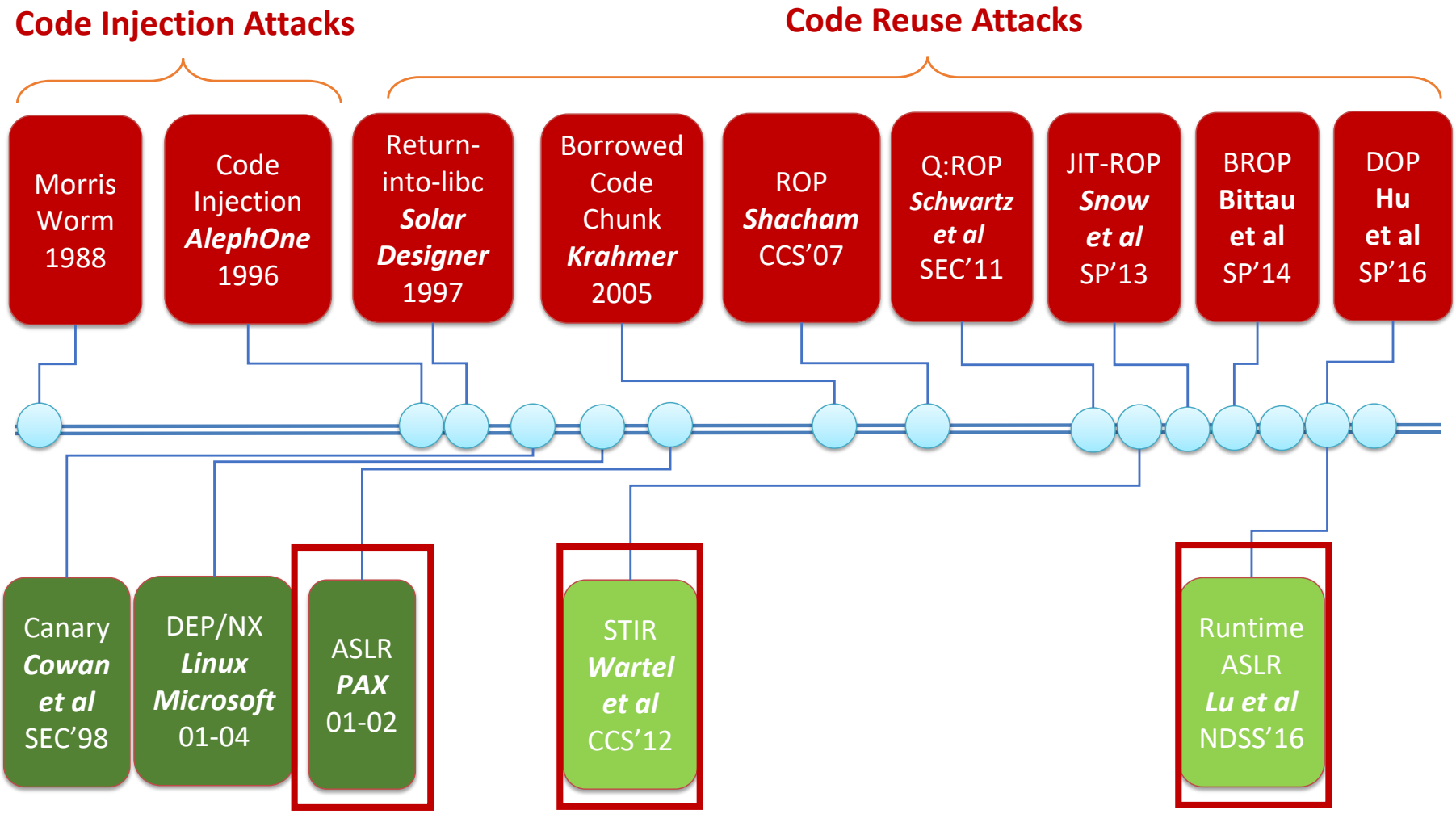
From Coarse-Grained to More Fine-Grained ASLR



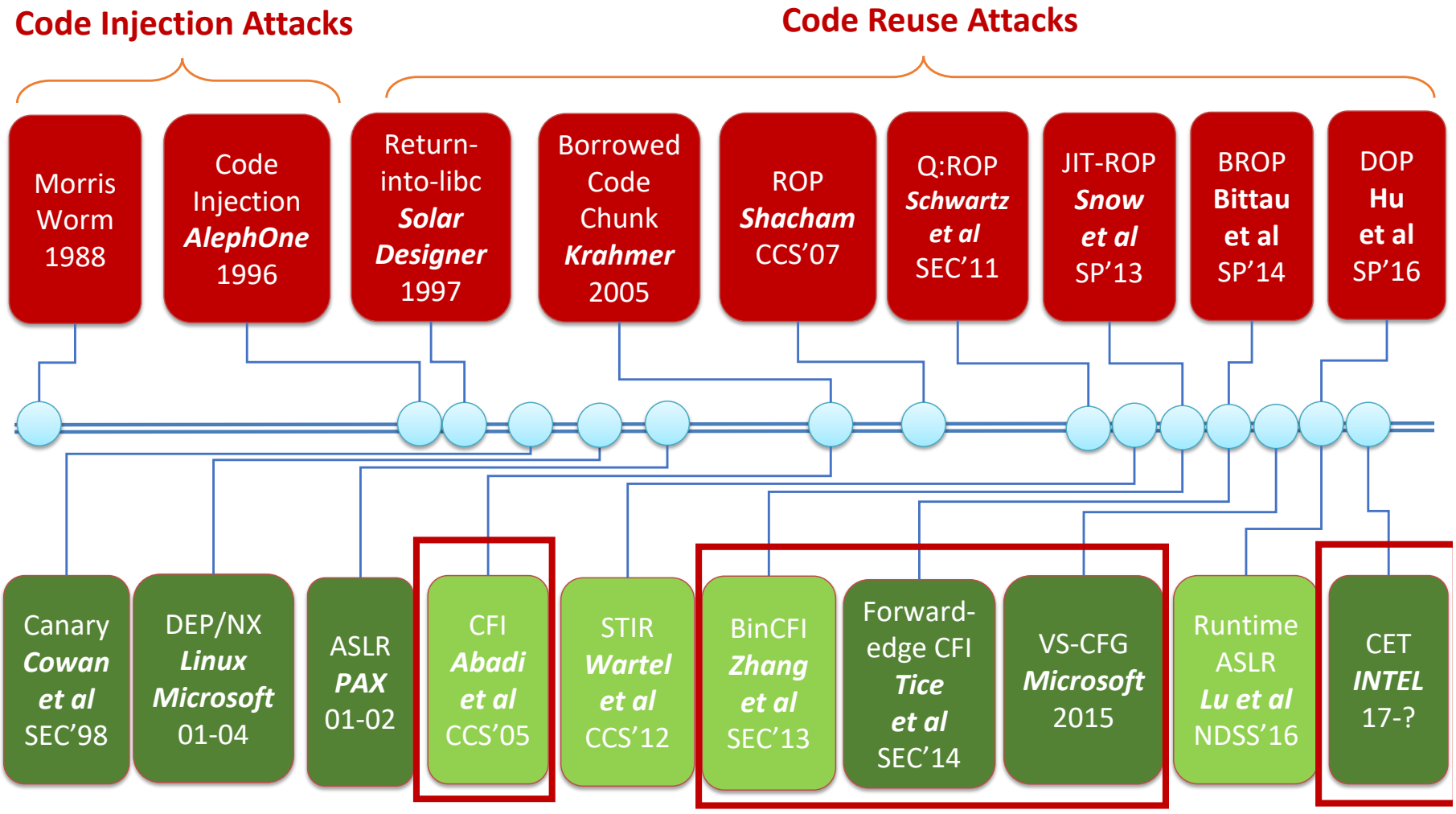
From Coarse-Grained to More Fine-Grained ASLR



Defense Using Control Flow Integrity



Defense Using Control Flow Integrity



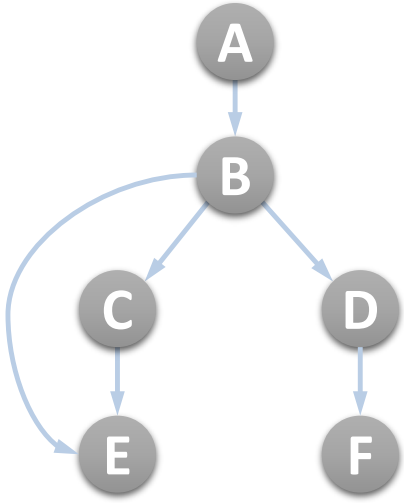
ASLR and CFI

(Fine-grained) Code Randomization

Control-Flow Integrity (CFI)

ASLR and CFI

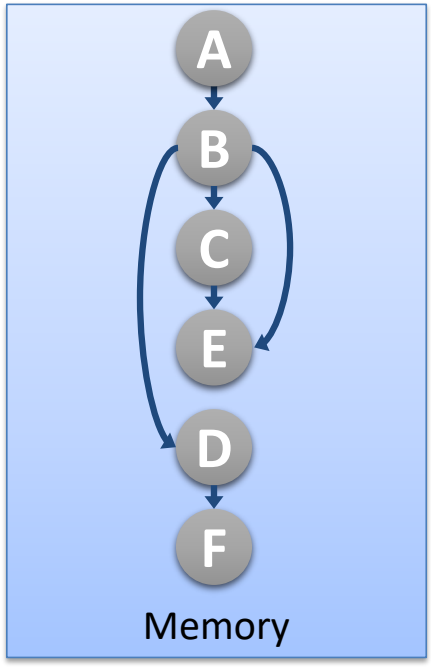
(Fine-grained) Code Randomization



Control-Flow Integrity (CFI)

ASLR and CFI

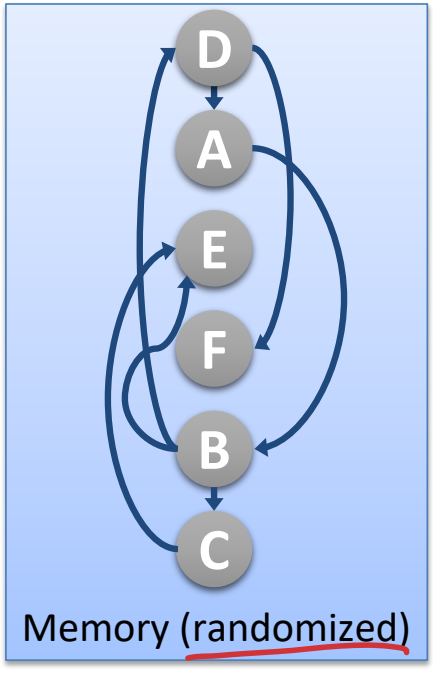
(Fine-grained) Code Randomization



Control-Flow Integrity (CFI)

ASLR and CFI

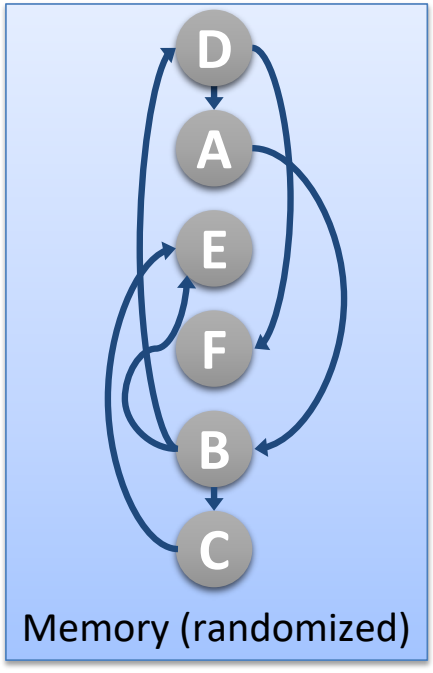
(Fine-grained) Code Randomization



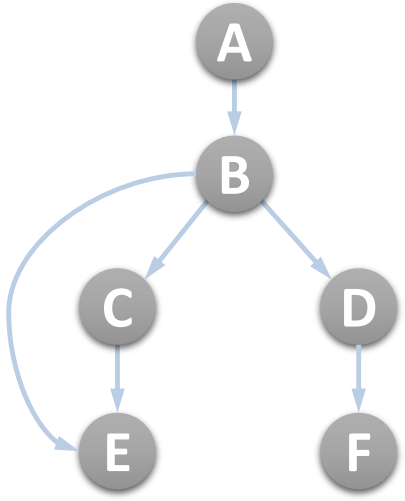
Control-Flow Integrity (CFI)

ASLR and CFI

(Fine-grained) Code Randomization

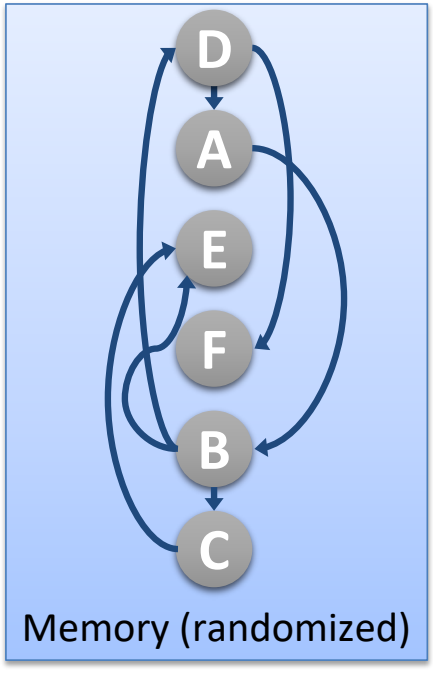


Control-Flow Integrity (CFI)

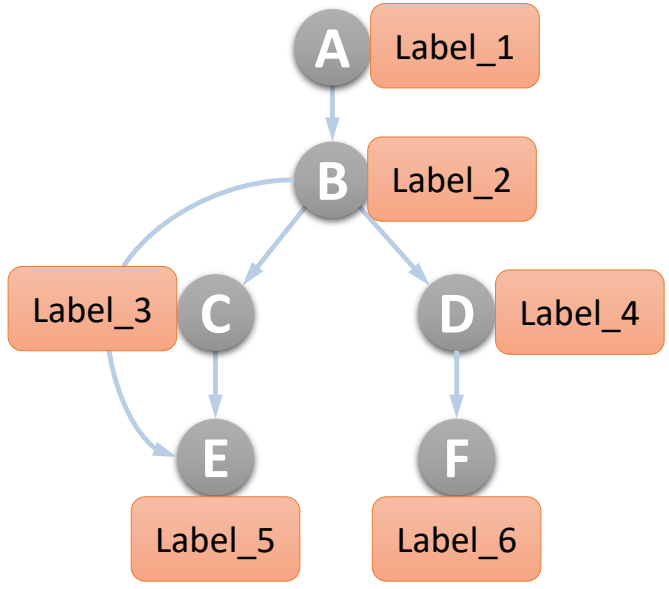


ASLR and CFI

(Fine-grained) Code Randomization

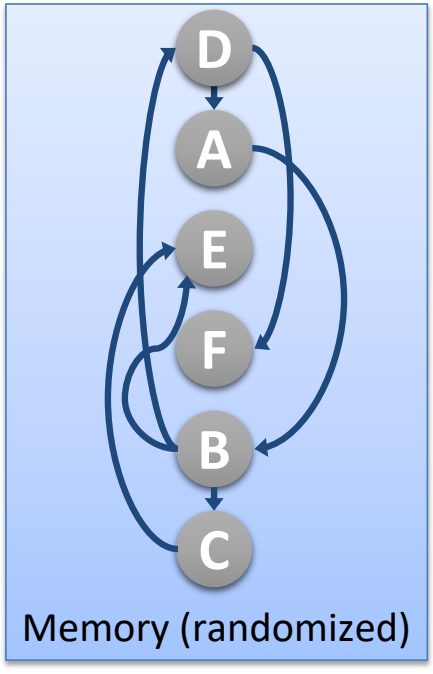


Control-Flow Integrity (CFI)

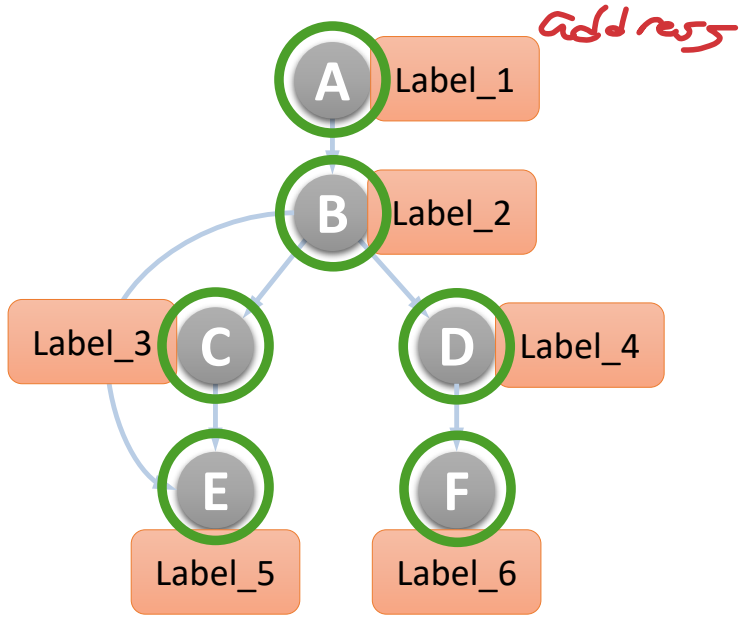


ASLR and CFI

(Fine-grained) Code Randomization

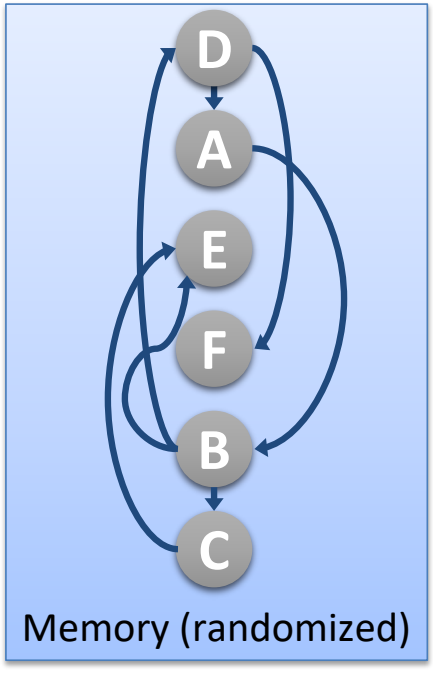


Control-Flow Integrity (CFI)



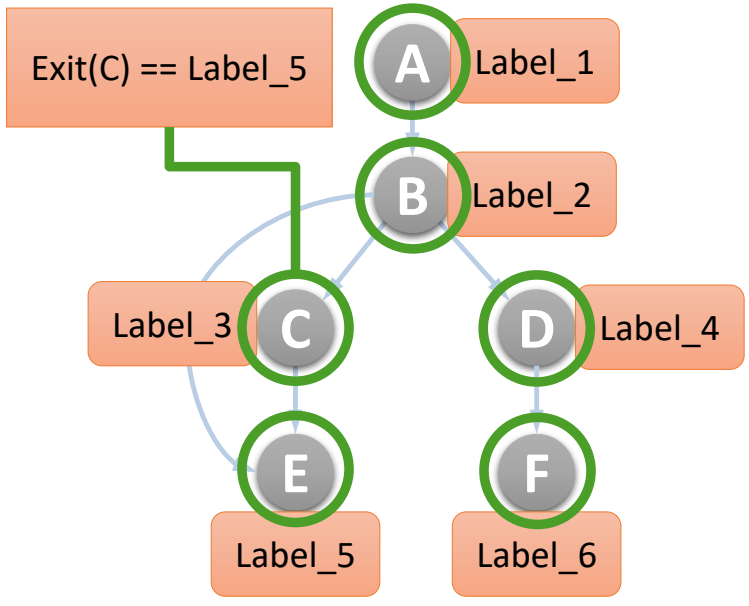
ASLR and CFI

(Fine-grained) Code Randomization

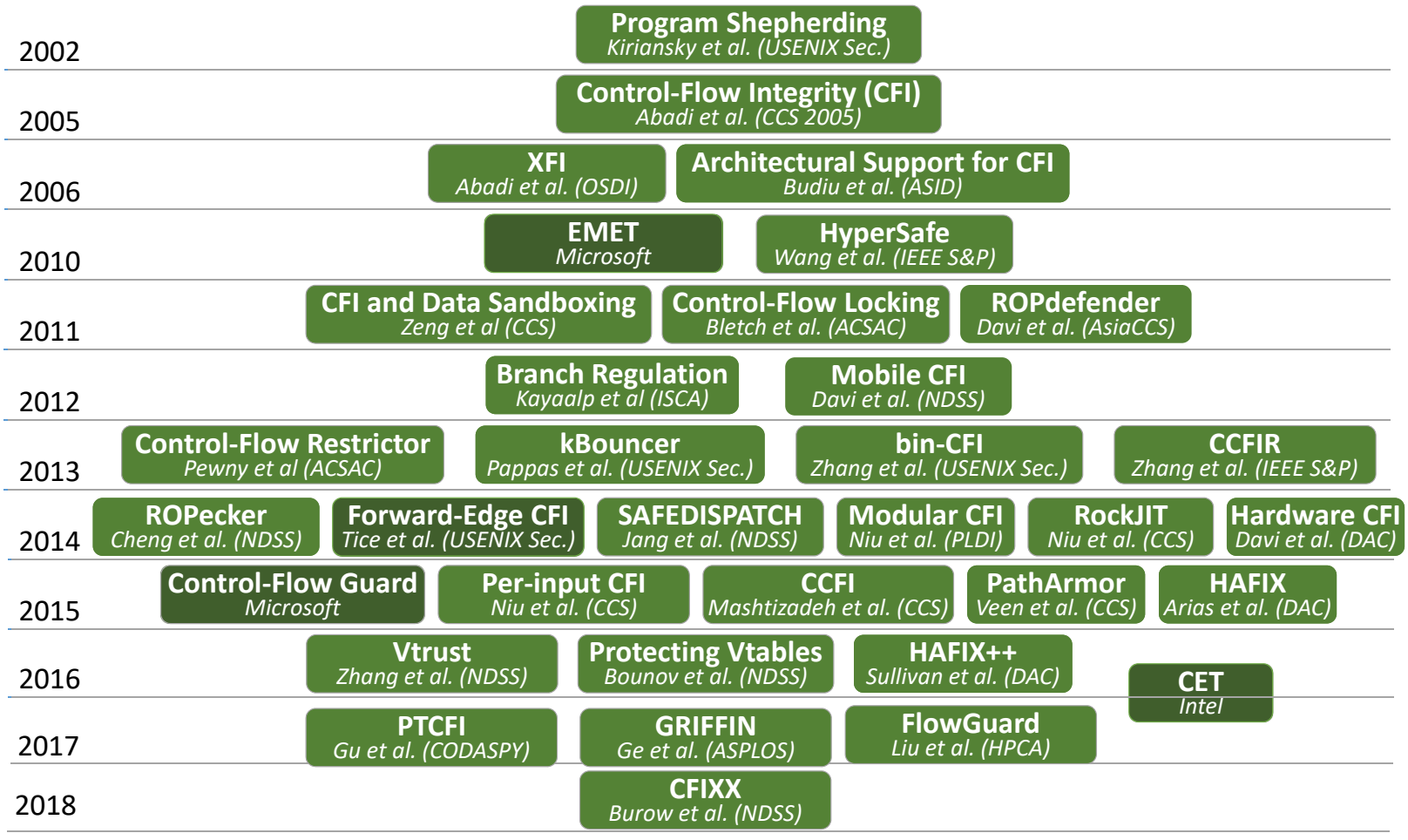


Control-Flow Integrity (CFI)

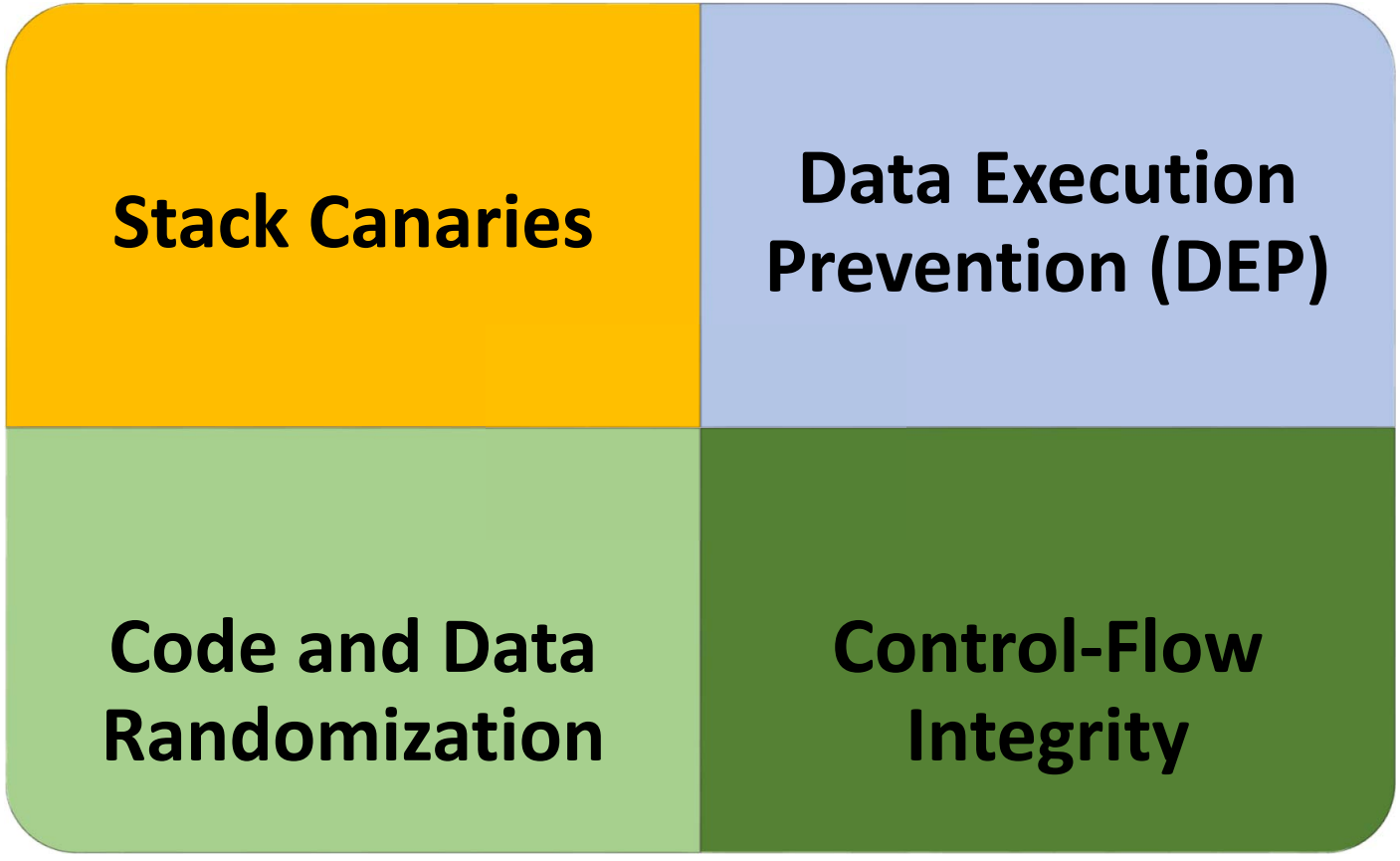
integrity checked



Recent Efforts on CFI Research

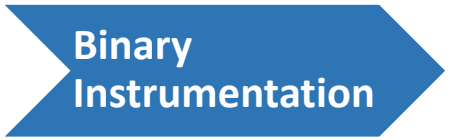


The Practical Run-time Defenses Against Memory Corruptions

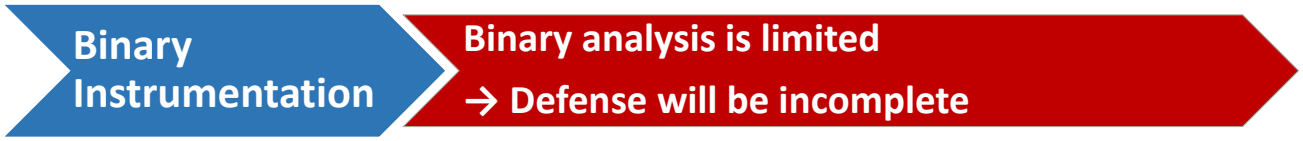


How to Implement the Defense Mechanisms (or Software in General)

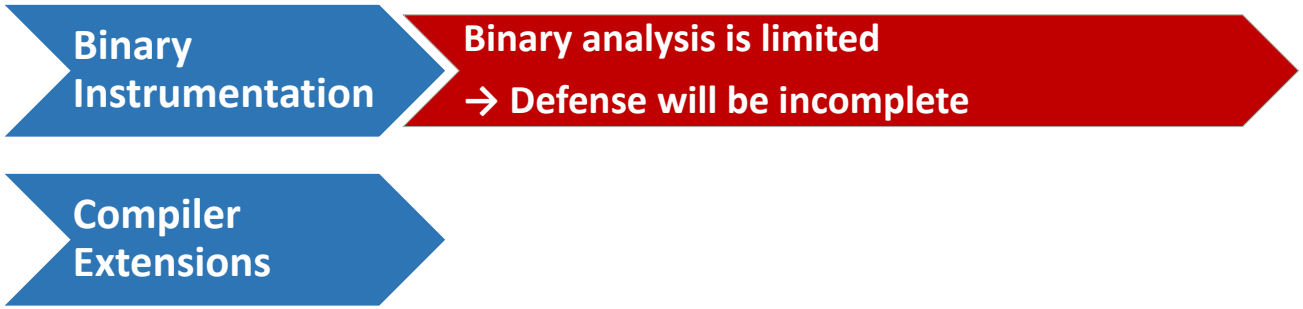
How to Implement the Defense Mechanisms (or Software in General)



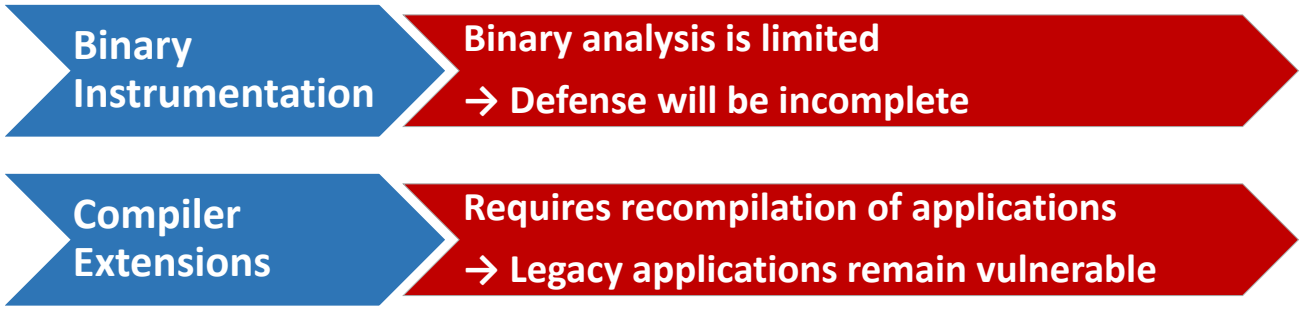
How to Implement the Defense Mechanisms (or Software in General)



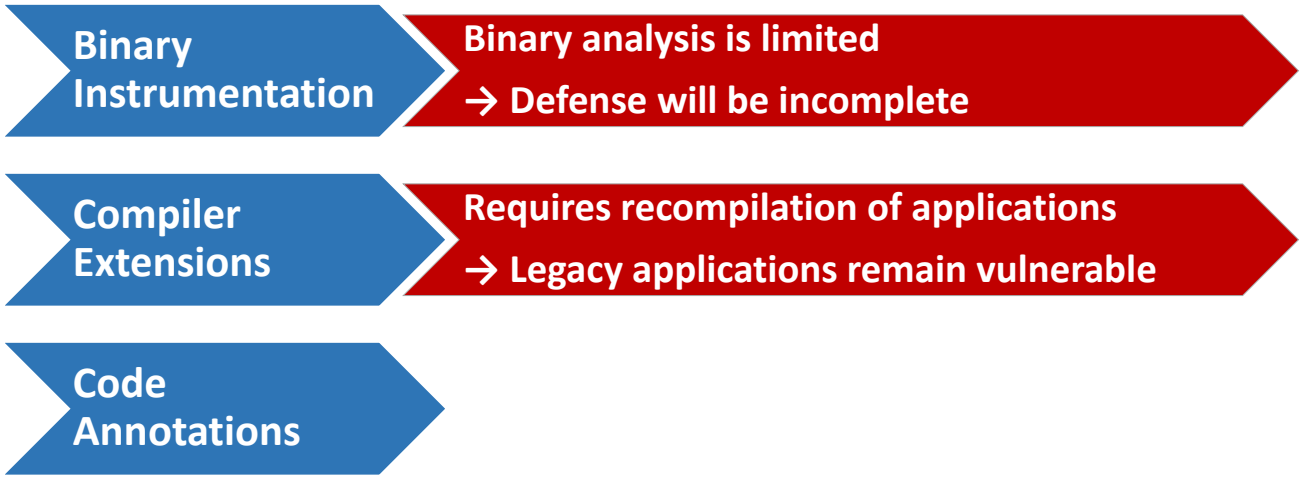
How to Implement the Defense Mechanisms (or Software in General)



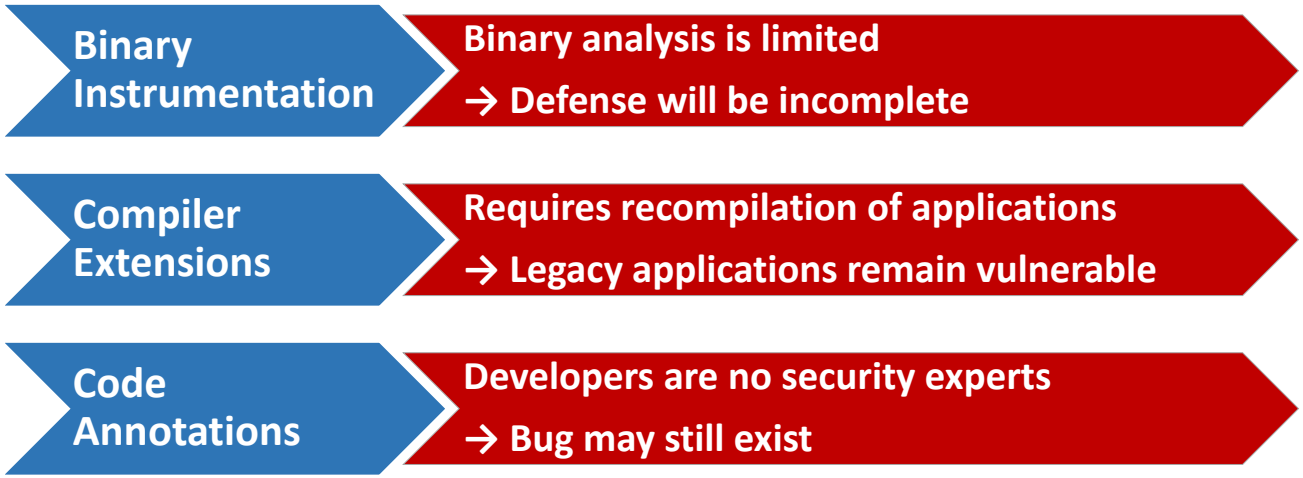
How to Implement the Defense Mechanisms (or Software in General)



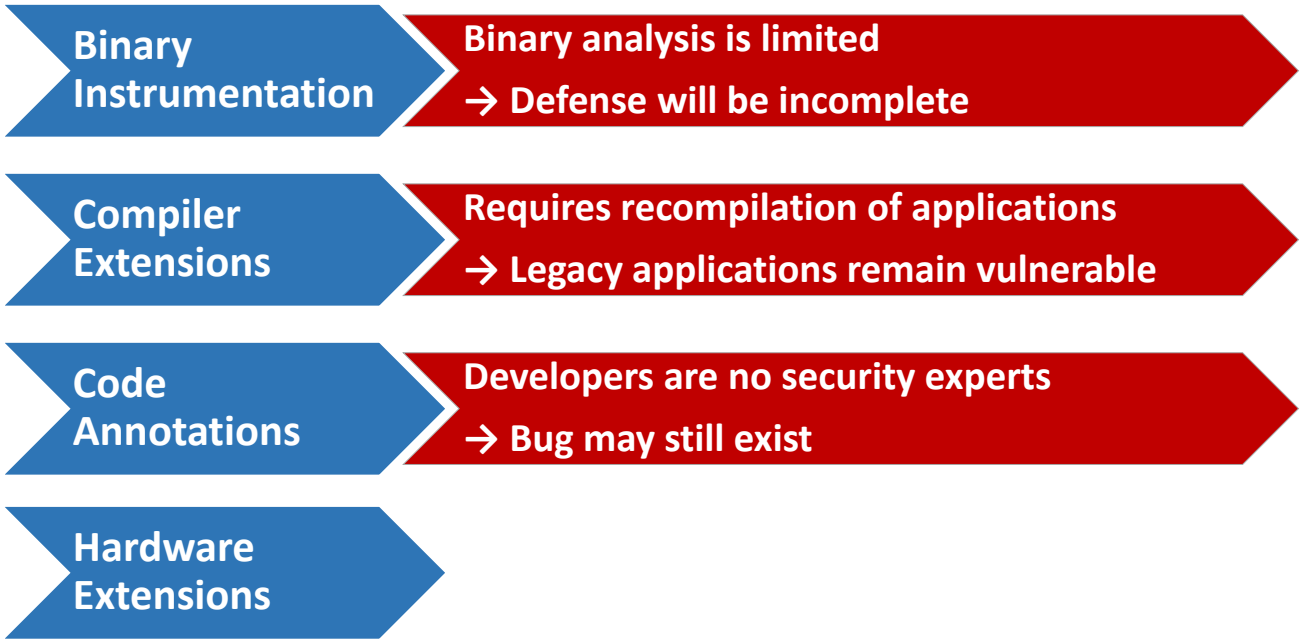
How to Implement the Defense Mechanisms (or Software in General)



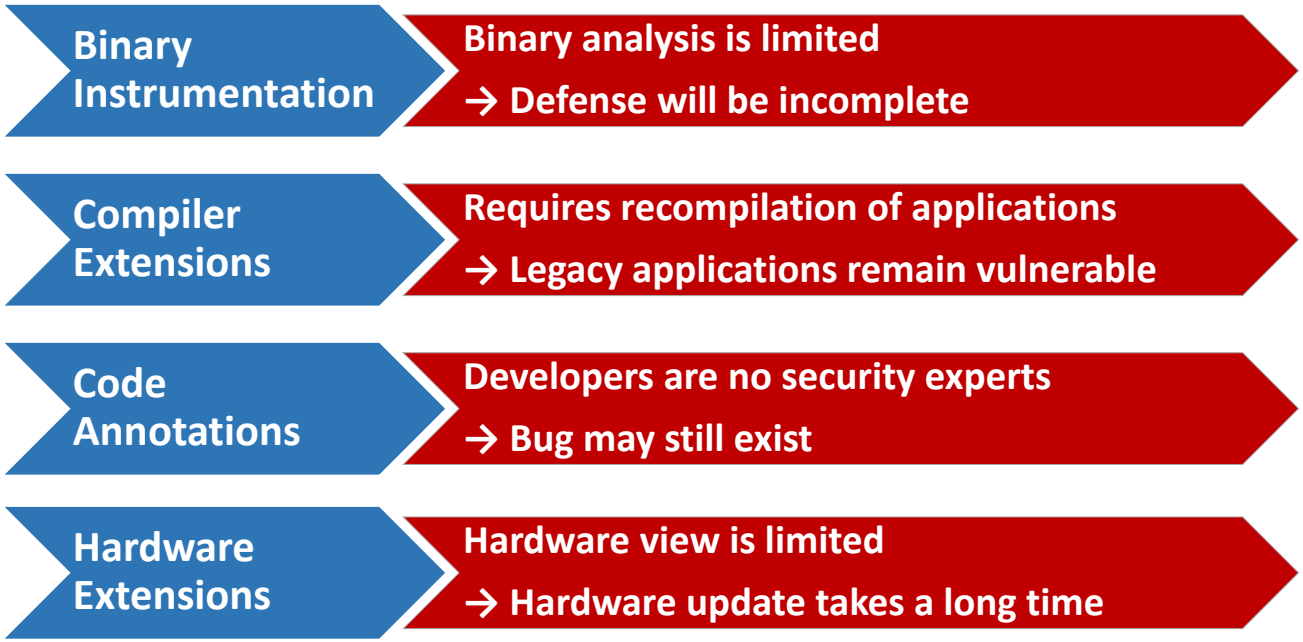
How to Implement the Defense Mechanisms (or Software in General)



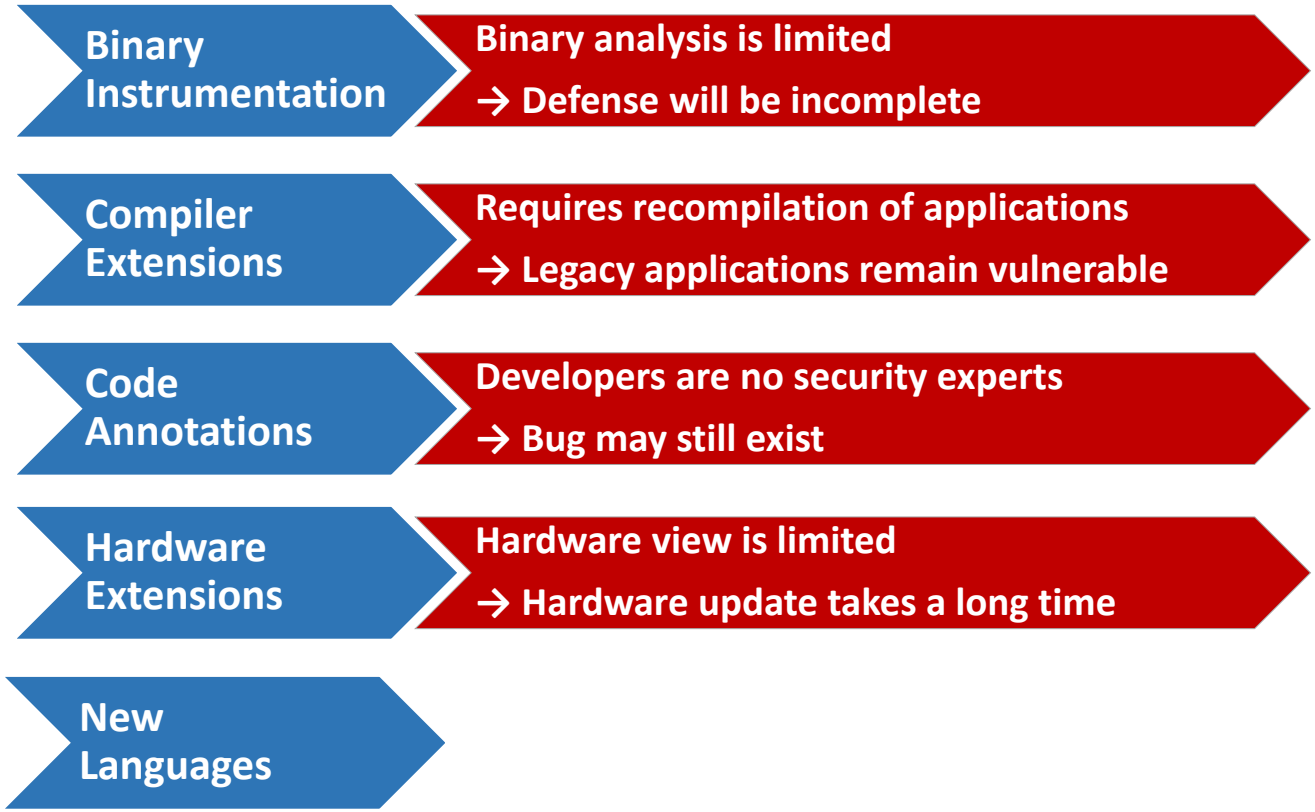
How to Implement the Defense Mechanisms (or Software in General)



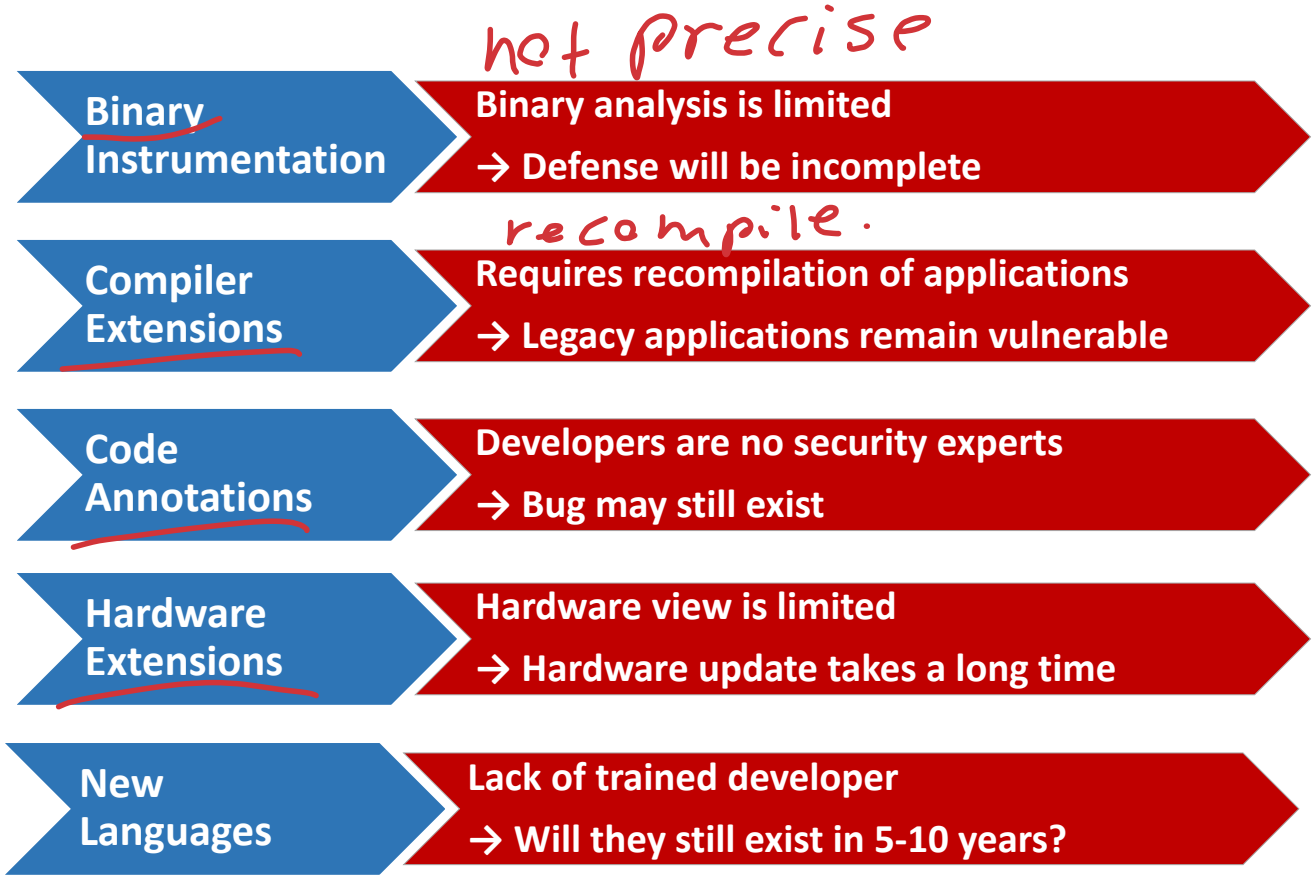
How to Implement the Defense Mechanisms (or Software in General)



How to Implement the Defense Mechanisms (or Software in General)



How to Implement the Defense Mechanisms (or Software in General)



Other Approaches: Proactive Vulnerability Identification

Fuzzing

1. Identifying vulnerabilities before attackers
   - Blackbox (dumb) fuzzing
   - Generational aka grammar-based fuzzing
   - Whitebox fuzzing with SAGE
     - Looking at symbolic execution of the code
   - Evolutionary fuzzing with afl
     - Grey-box, observing execution of the (instrumented) code

Other Approaches: Proactive Vulnerability Identification

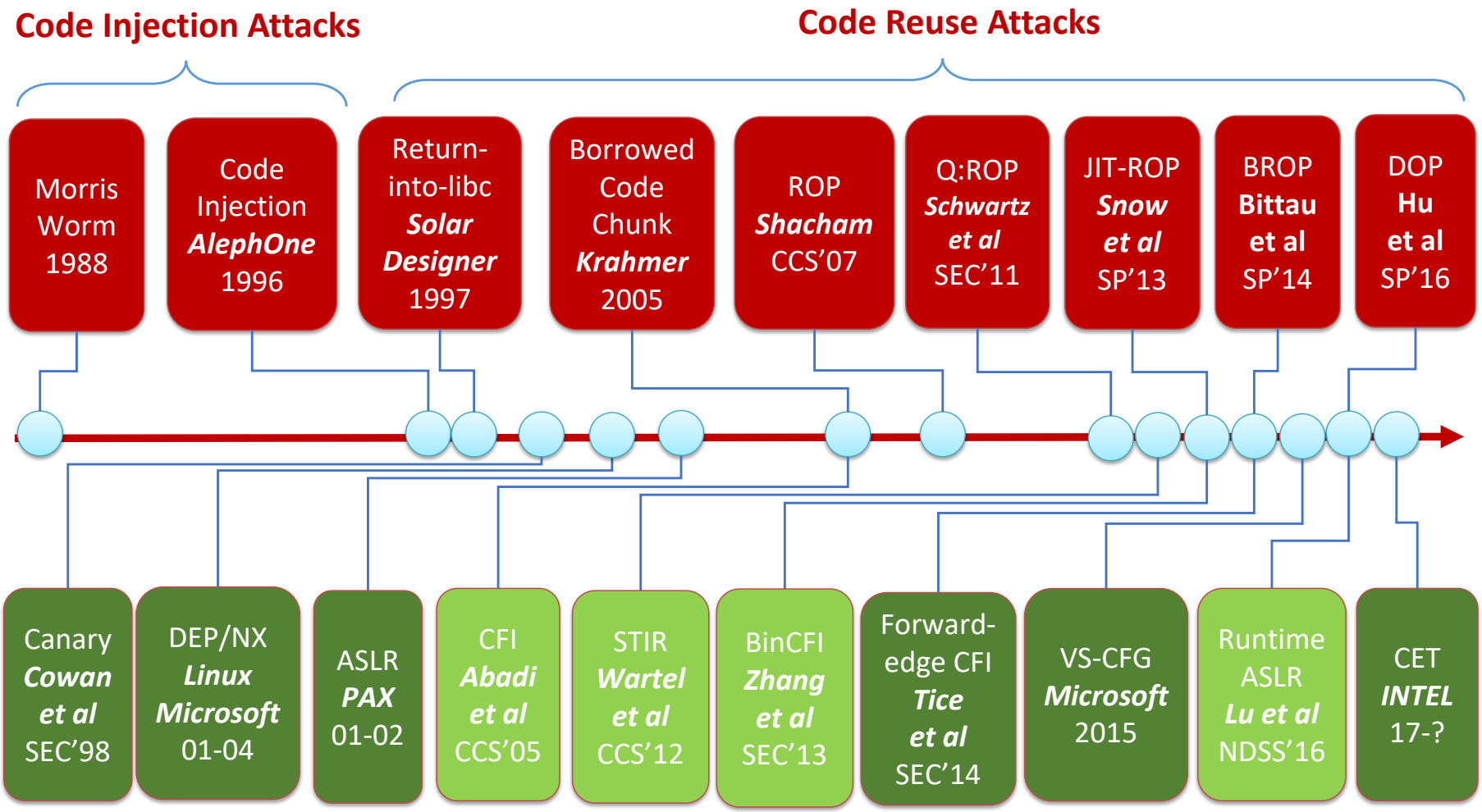
Fuzzing

1. Identifying vulnerabilities before attackers
   - Blackbox (dumb) fuzzing
   - Generational aka grammar-based fuzzing
   - Whitebox fuzzing with SAGE
     - Looking at symbolic execution of the code
   - Evolutionary fuzzing with afl
     - Grey-box, observing execution of the (instrumented) code

AFL (American Fuzzy Lop)

1. Support software w/
   - Source code (using a compiler flag)
     - C/C++/Object-C
     - Hand-written assembly
   - Binary (executed in an emulator)
2. Identified many 0-day vulnerabilities
3. Highly practical

The Arm Race Between Offense and Defense



Practical Defenses

When Developing New Software

1. Turnning on compiler flag
   - Canary (/gs, -fstack-protector)
   - CFI (/cfg, -fsanitize=cfi-icall)
   - CET (gcc 8.0)
   - ASLR (-pie -fPIE)
2. Using type safe language
   - Rust

Practical Defenses

When Developing New Software

1. Turnning on compiler flag
   - Canary (/gs, -fstack-protector)
   - CFI (/cfg, -fsanitize=cfi-icall)
   - CET (gcc 8.0)
   - ASLR (-pie -fPIE)
2. Using type safe language
   - Rust

When Deploying Old Software

1. Turnning on kernel DEP
2. Turnning on ASLR
   - Load-time
   - Microsoft EMET
3. Keeping software patched
4. Penetration testing
   - Using fuzzing

Software Security

Software is Not Secure

1. Memory unsafe code (written in C/C++, asm, ...)
2. Prevalent software defects
   - Stack/Heap smashing
   - Format string bugs
   - Pointer errors

Defenses

1. Stack/Heap Canaries
2. ASLR, DEP, CFI
3. RELRO, BIND_NOW
4. BPF_SECCOMP, FORTIFY_SRC

Software Security

Software is Not Secure

1. Memory unsafe code (written in C/C++, asm, ...)
2. Prevalent software defects
   - Stack/Heap smashing
   - Format string bugs
   - Pointer errors

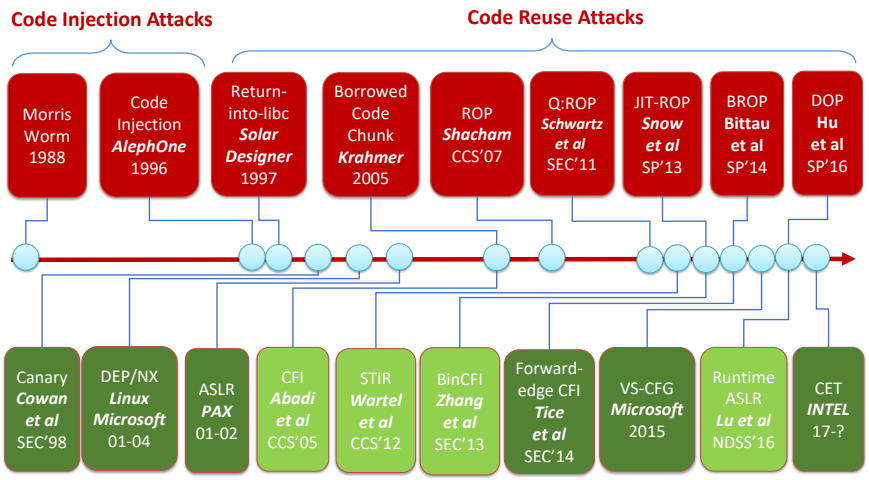
Defenses

1. Stack/Heap Canaries
2. ASLR, DEP, CFI
3. RELRO, BIND_NOW
4. BPF_SECCOMP, FORTIFY_SRC

Offenses

1. Code injection
2. Code Reuse
   - Return-to-libc (ret2libc)
   - Return-oriented prog. (ROP)
   - Just-In-Time ROP (JIT-ROP)

Thank You



1

schen@auburn.edu
schuan.github.io

[1] Instructor appreciates the help from Prof. Zhiqiang Lin.