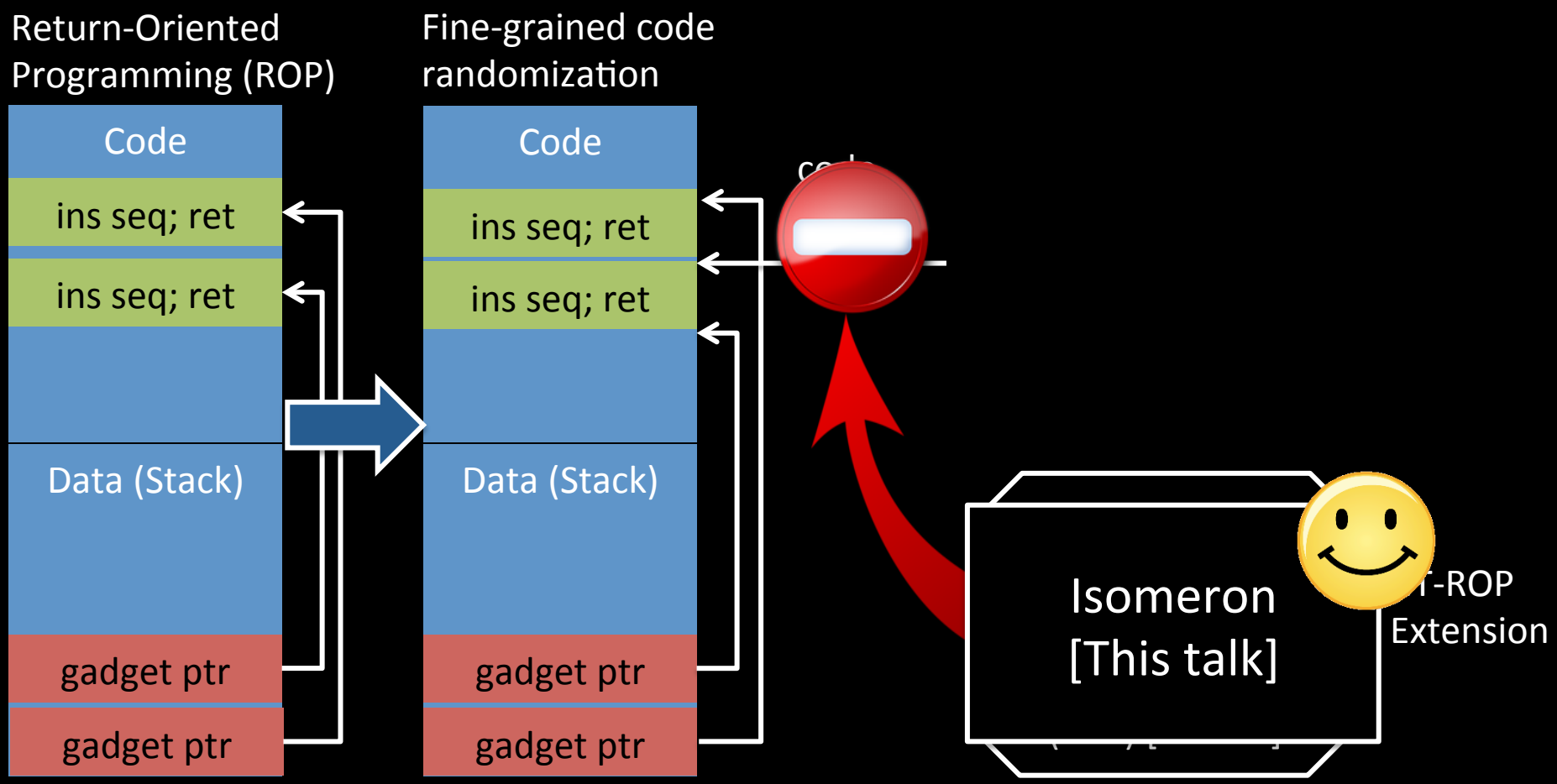
Isomeron

Code Randomization Resilient to (Just-In-Time) Return-Oriented Programming

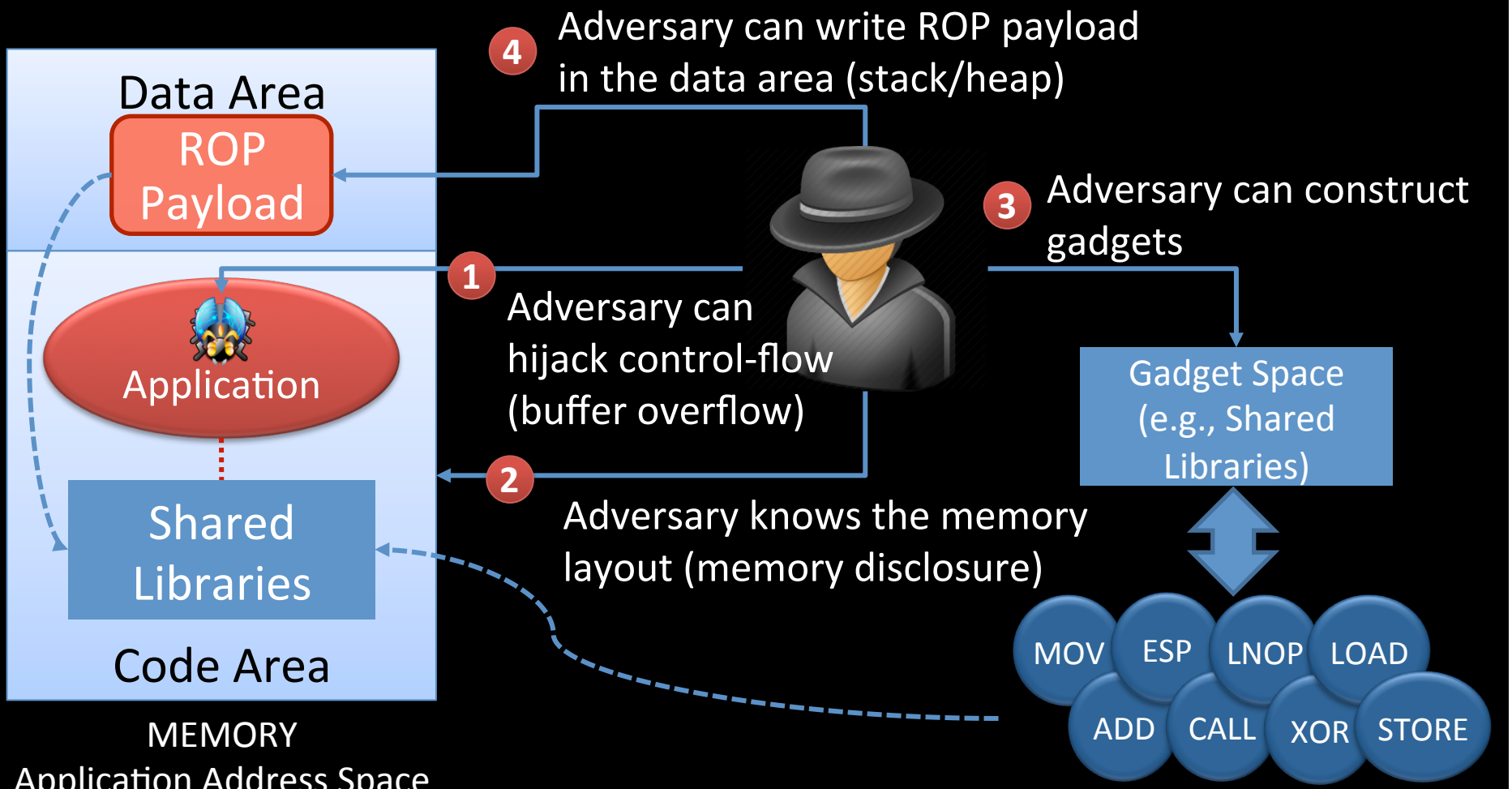
Lucas Davi, Christopher Liebchen,
Ahmad-Reza Sadeghi
CASED/Technische Universität
Darmstadt, Germany

Kevin Z. Snow, Fabian Monrose
Department of Computer Science
University of North Carolina at
Chapel Hill, USA

The Big Picture



ROP Adversary Model/Assumption



DEFENSES

(Fine-Grained) Code Randomization

Code-Randomization Approaches

- Base address permutation: Address Space Layout Randomization (ASLR)
- Function permutation: ASLP [ACSAC'06]
- Basic block permutation: STIR [CCS'12], XIFER [ASIACCS'13]
- Instruction-level randomization: IRL [S&P'12]
- In-place randomization: ORP [S&P'12]

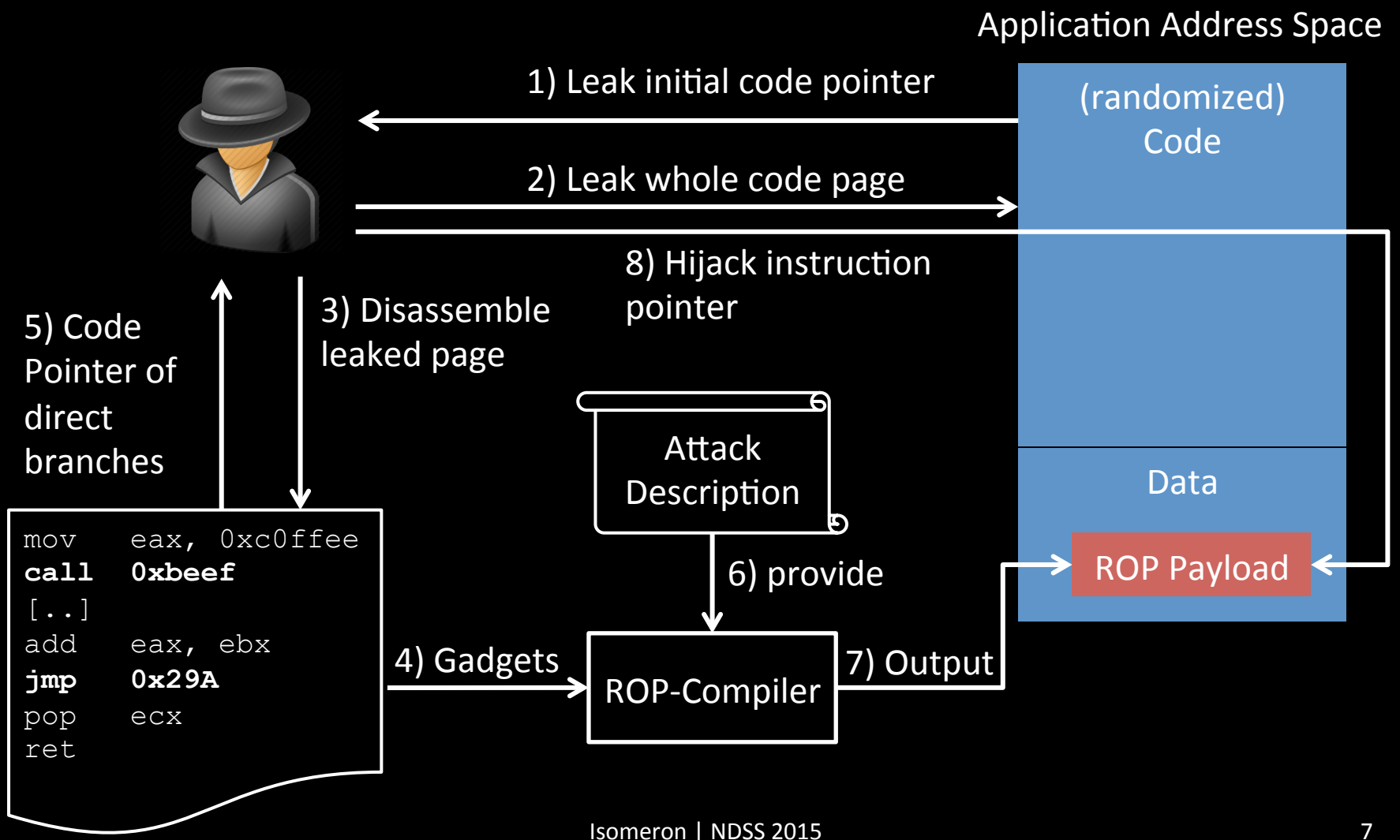
A severe attack against fine-grained ASLR

Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization

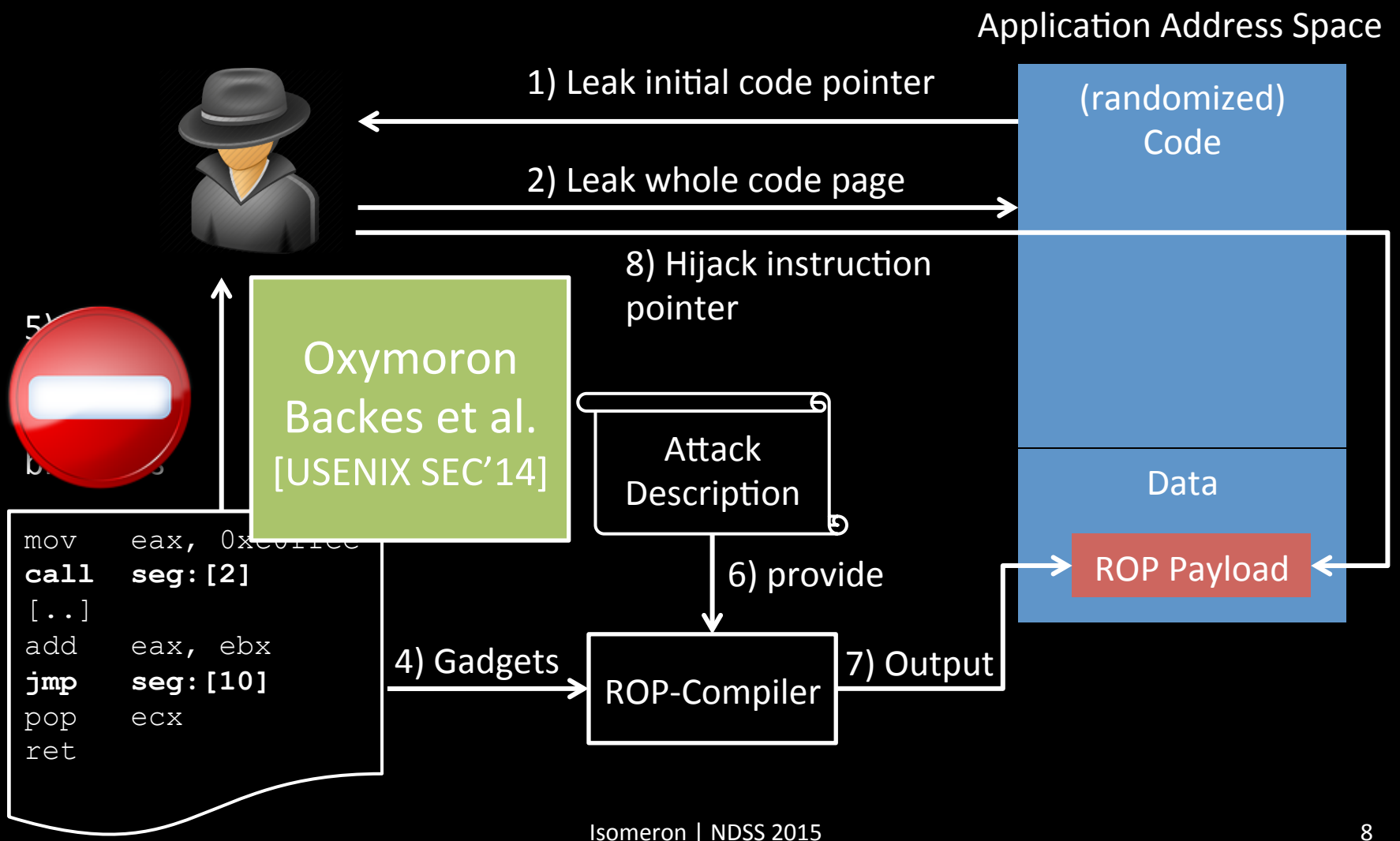
IEEE Security and Privacy 2013, and Blackhat 2013

Kevin Z. Snow, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, Fabian Monrose, Ahmad-Reza Sadeghi

Just-In-Time ROP [IEEE S&P'13]



Defense against Just-In-Time ROP

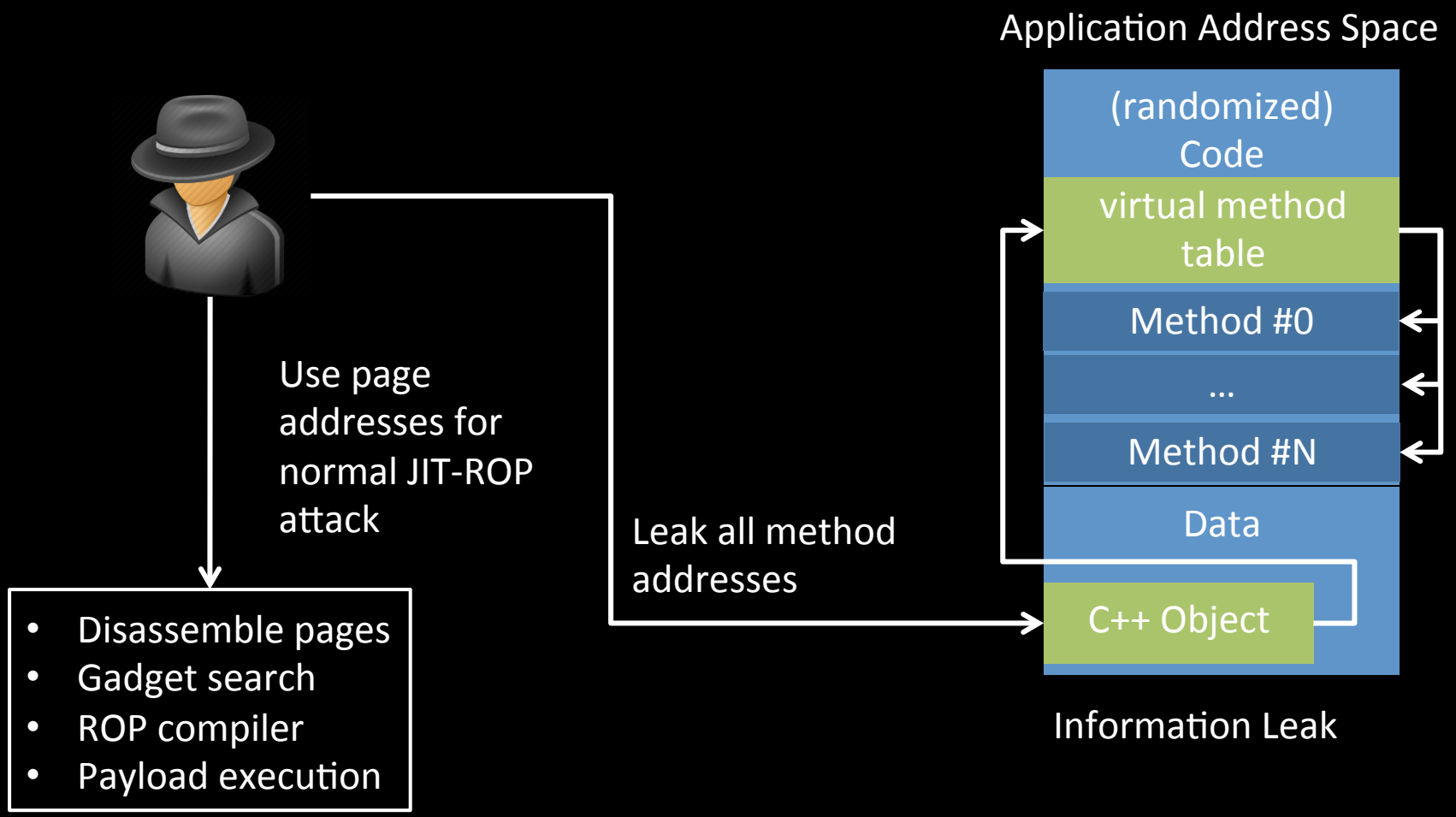


Can we bypass Oxymoron-like approaches?

Sources of Code Pointers

- Virtual method tables
- Stack frames
- Exception handling information
- Loader data
  - Import/export table
  - Global offset table

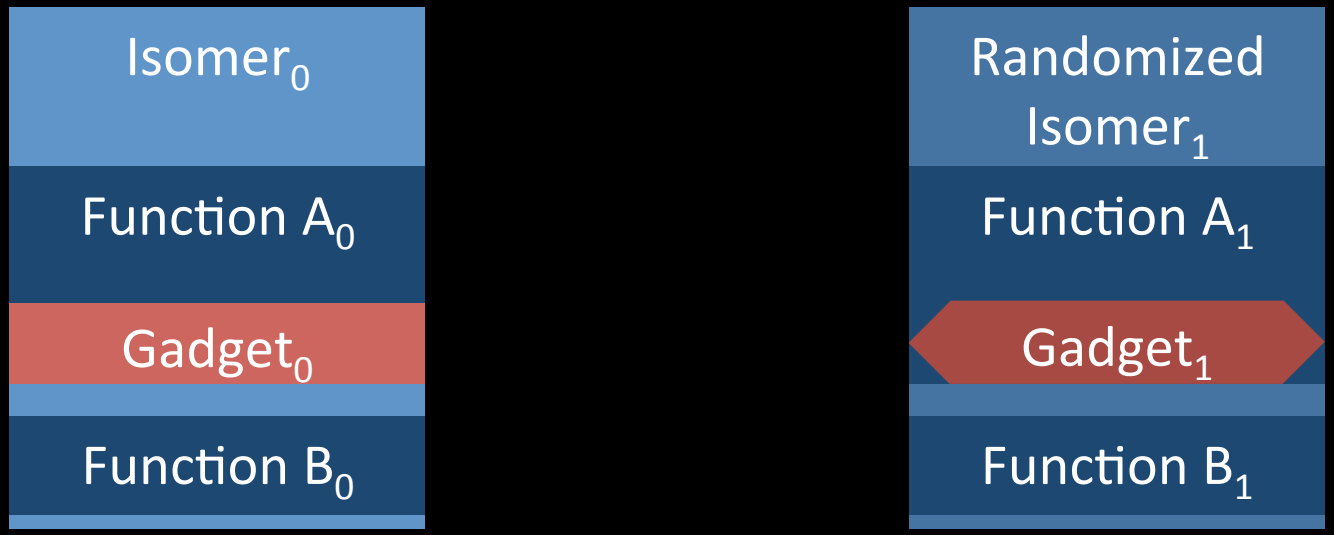
How to Bypass Oxymoron



Our Solution:
Isomeron

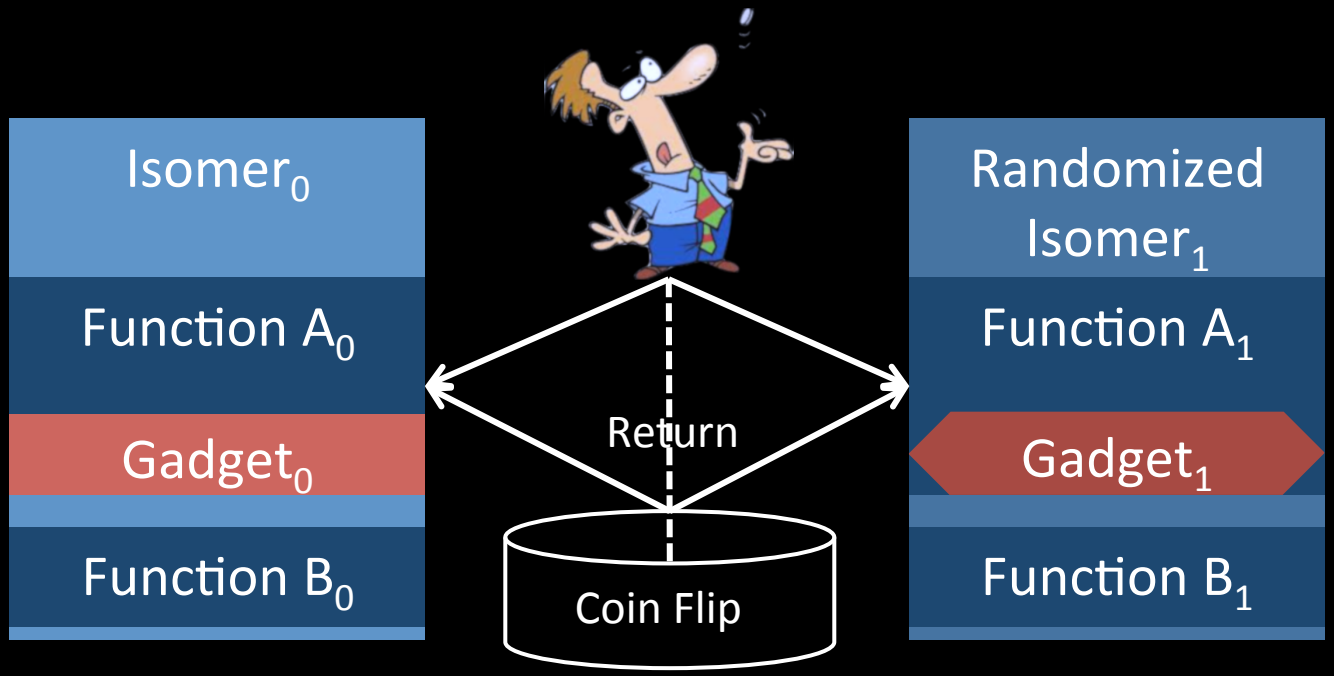
Isomeron - High-level Idea

- Create a randomized isomer (copy) of the application
  - Preserve semantics of the function
  - Affects the gadget locations

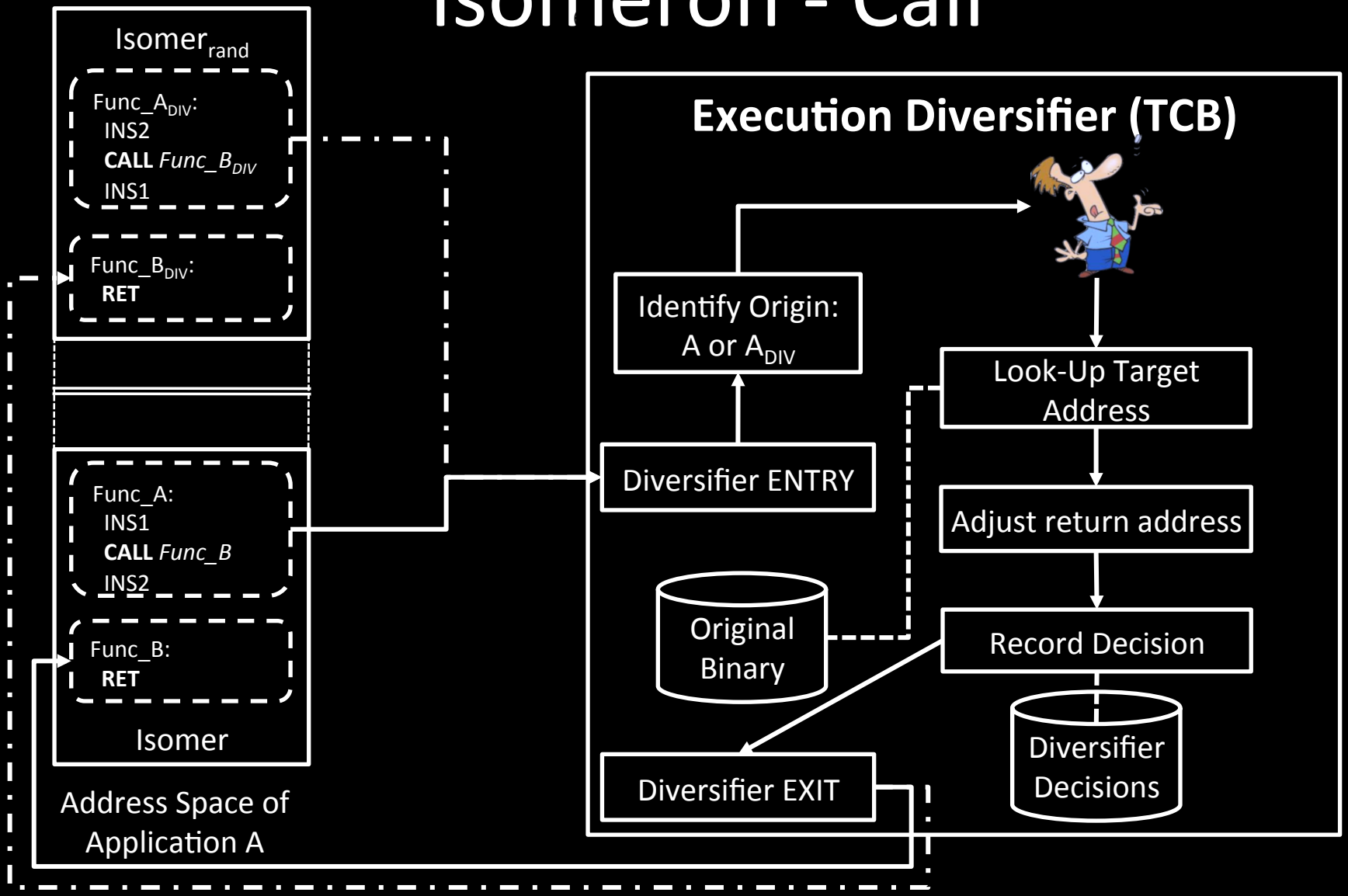


Isomeron - High-level Idea

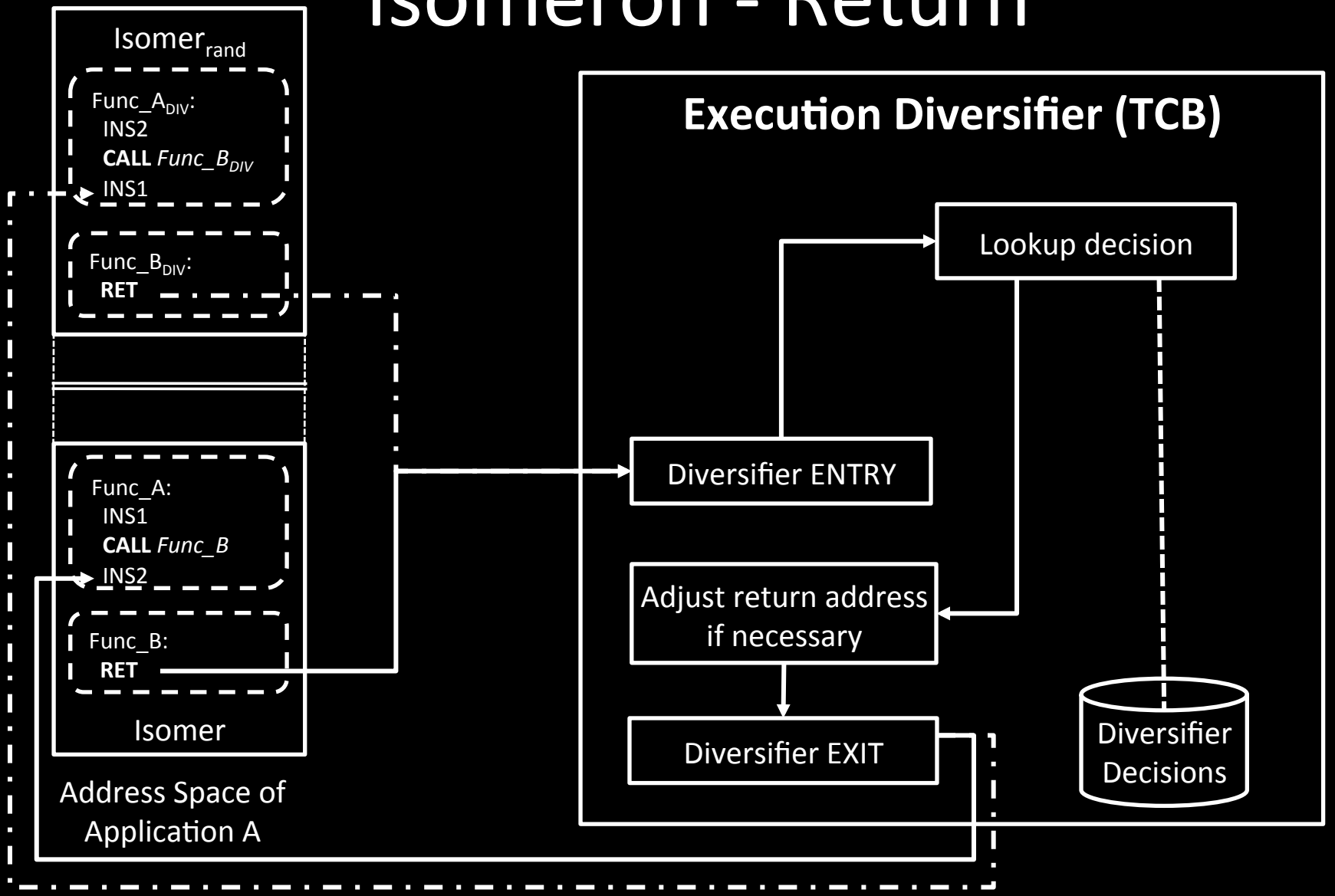
- Control-flow randomization



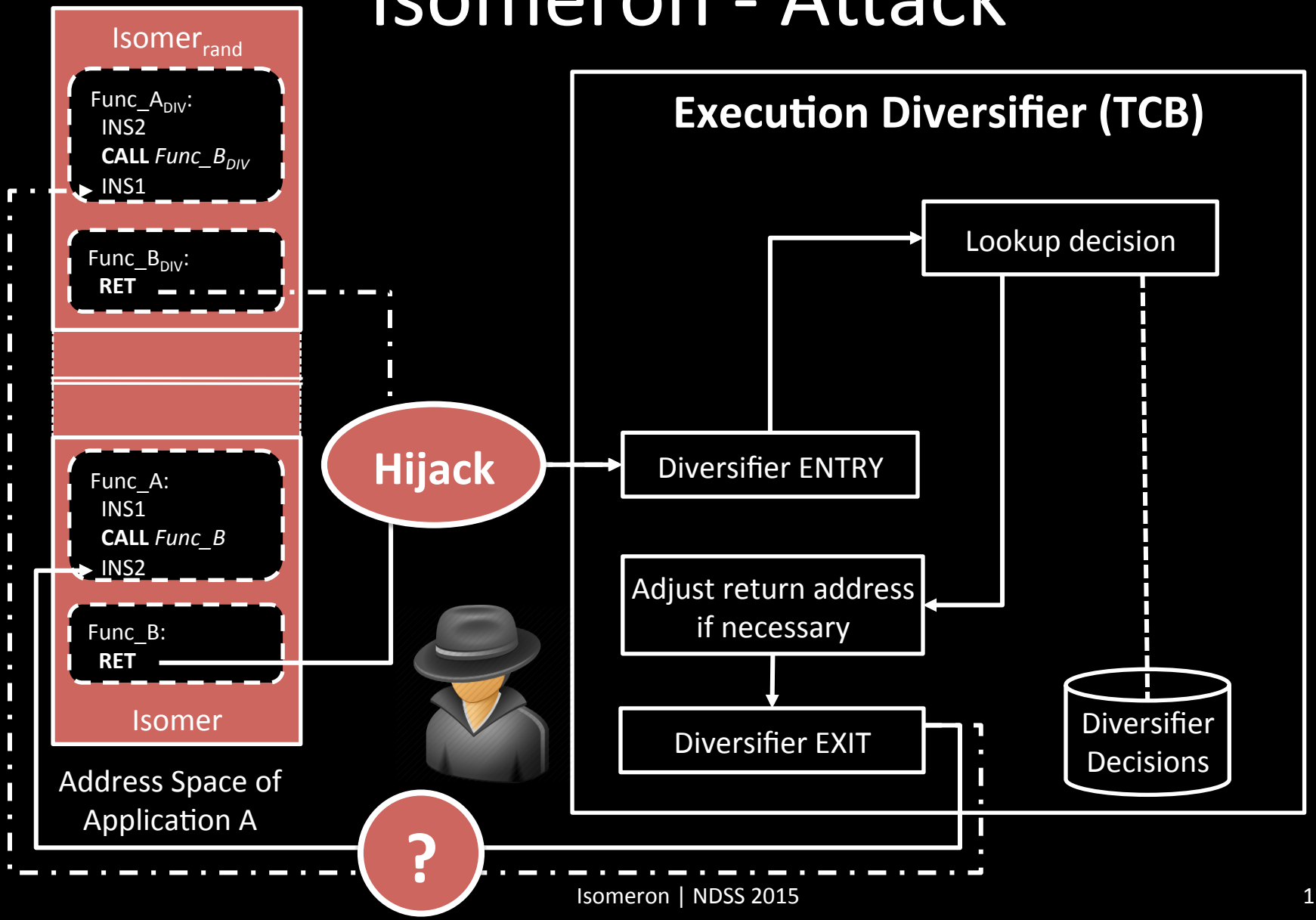
Isomeron - Call



Isomeron - Return



Isomeron - Attack



Isomeron - Security

- Conventional ROP
  - Code randomization
- (JIT) ROP
  - Code randomization and control flow randomization
- Ret-to-libc
  - Non-trivial in general
  - We restrict ret-to-libc to targets of benign indirect calls

Implementation & Challenges

- Multiple (randomized) copies
  - Custom dynamic binary instrumentation (DBI) framework
  - Existing DBI tools did not fulfill our requirements
  - Performance penalties
- Protect caller information
  - Segmentation (hardware dependent)
  - Software Fault Isolation

Current and Future Work

- Compiler-based randomization solutions
  - Isomeron with compiler
  - Readactor – to appear IEEE S&P'15
    - Use compiler to randomize code and hardware support to enforce real X-only memory
- CFI-based solutions
  - Counterfeit Object-oriented Programming – to appear IEEE S&P'15
    - Bypass almost all C++ CFI solutions

Thank you.

Backup

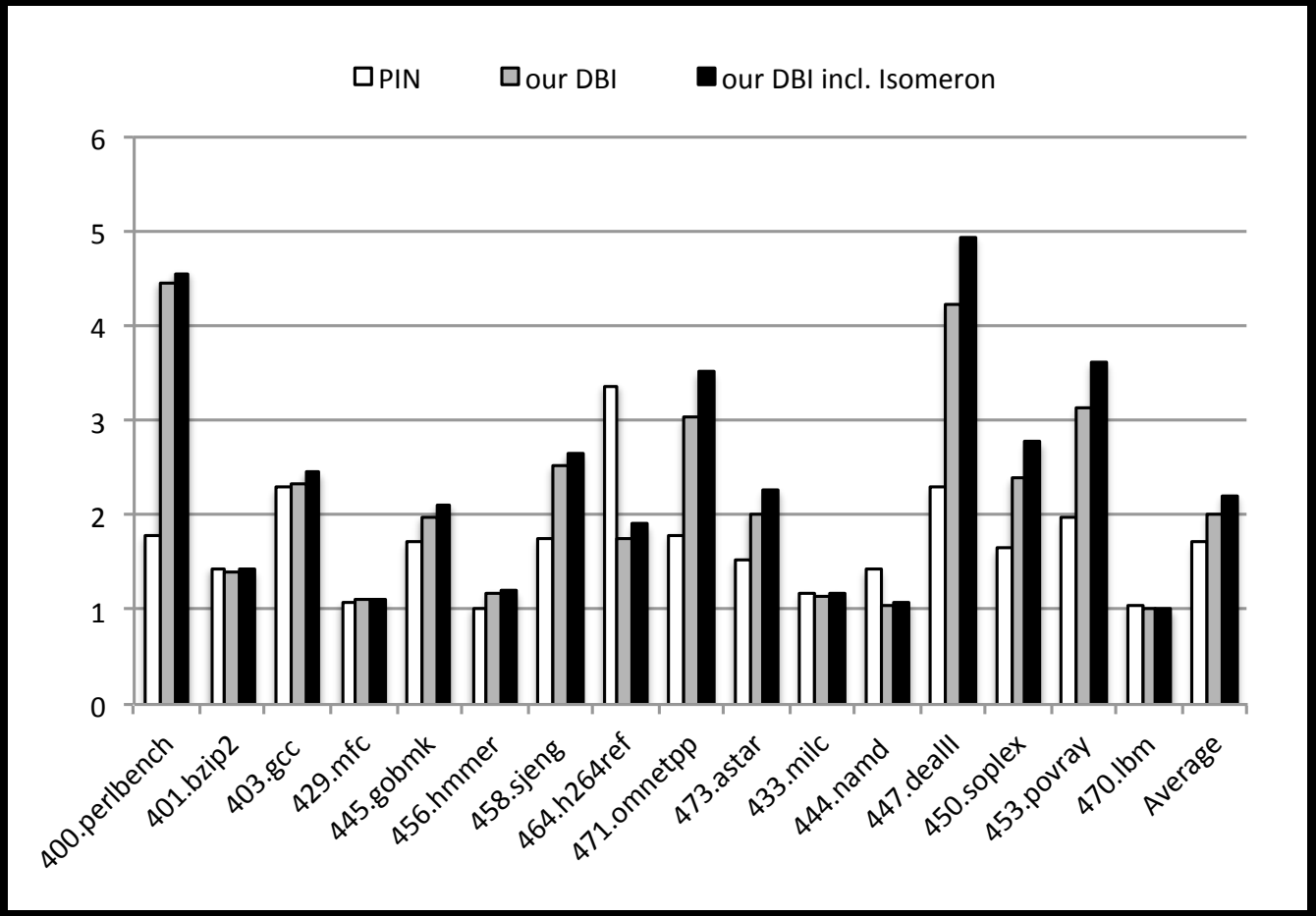
Isomeron - Security

- Special case of gadget pairs
  - intended gadgets G performs operation
  - other gadget G_{nop} performs nop
- Gadget space
  - limited to (G,G_{nop}) where G does not modify the input value
  - Examples:
    - load value from stack (stack pointer is modified)
    - load constant into register

Defenses against JIT-ROP

- Oxymoron (USENIX'14)
  - Aims at preventing JIT-ROP by obfuscating destination addresses of direct branches

Isomeron - Performance



SPEC 2006

Unfortunately randomization can be bypassed

Just-In-Time ROP - Oakloand'13

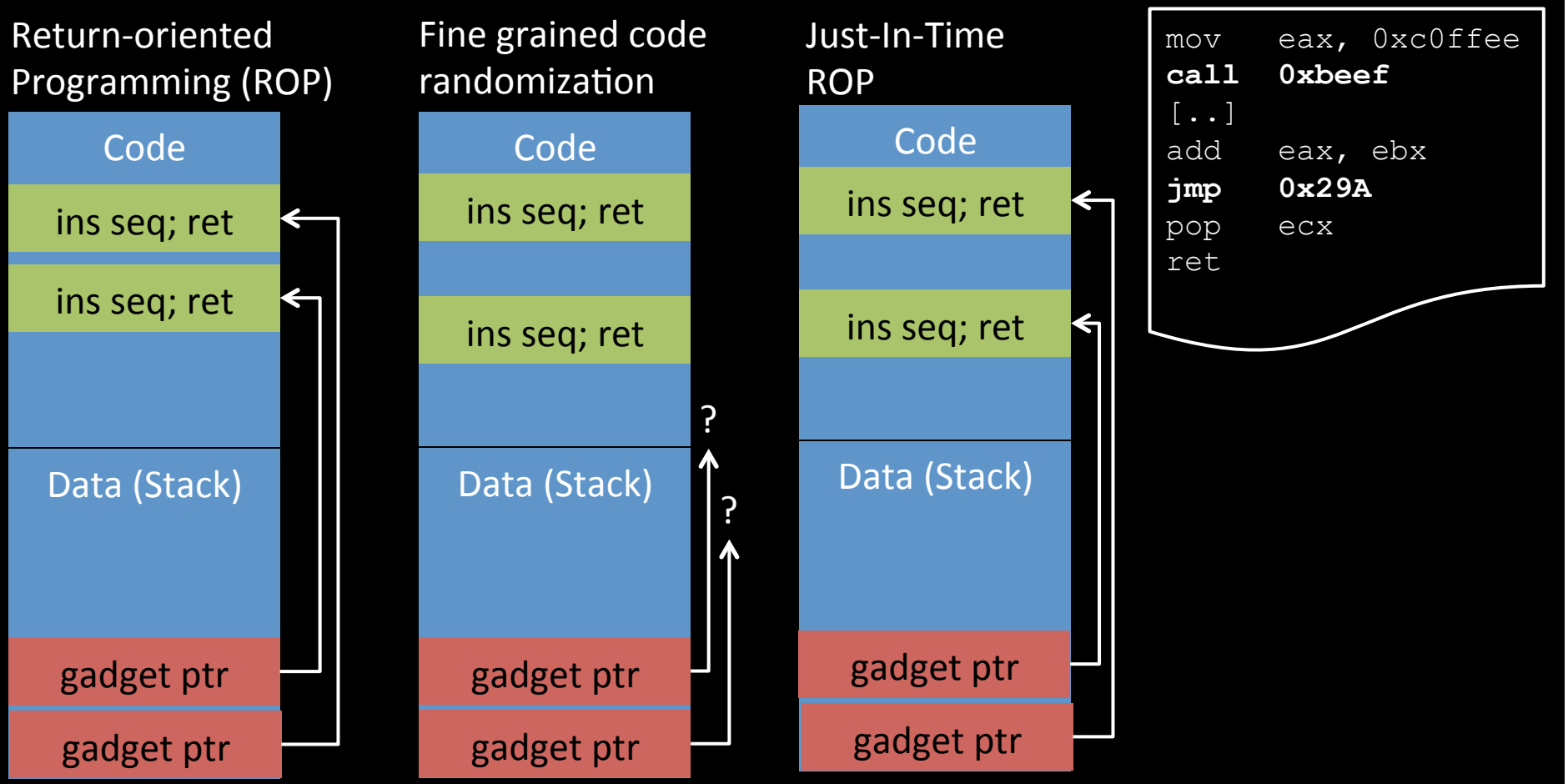
Kevin Z. Snow, Fabian Monrose

Department of Computer Science
University of North Carolina at
Chapel Hill, USA

Lucas Davi, Alexandra Dmitrienko,
Christopher Liebchen, Ahmad-
Reza Sadeghi

CASED/Technische Universität
Darmstadt, Germany

The Big Picture

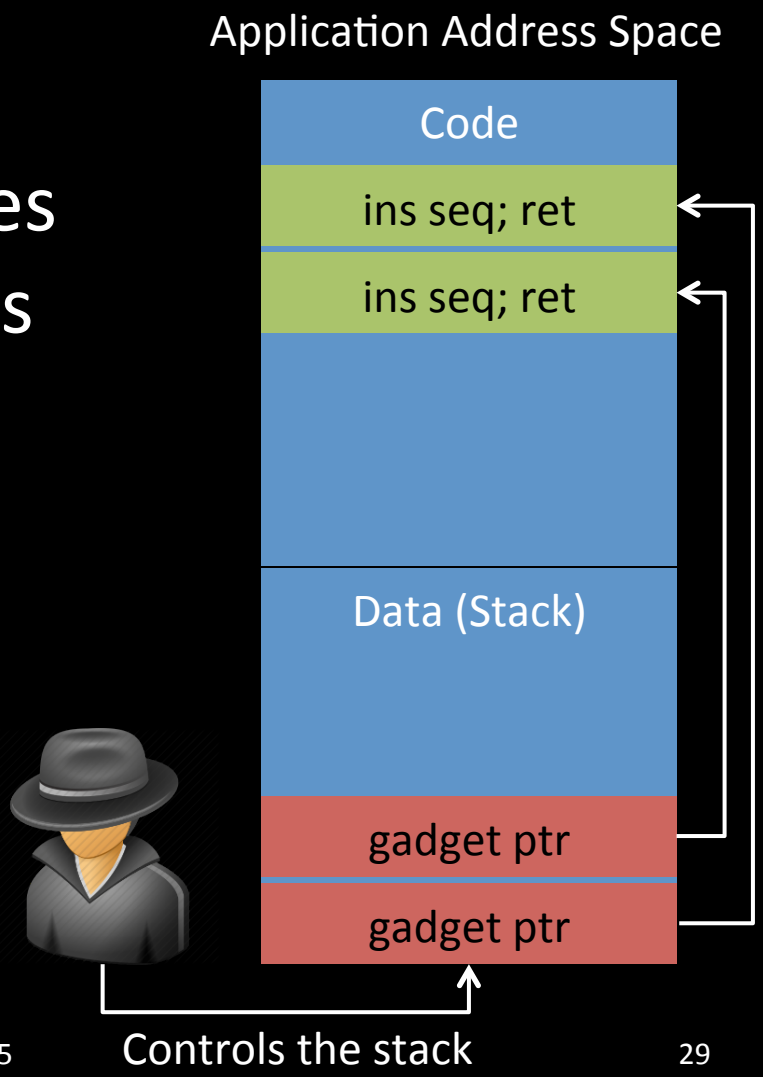


Motivation

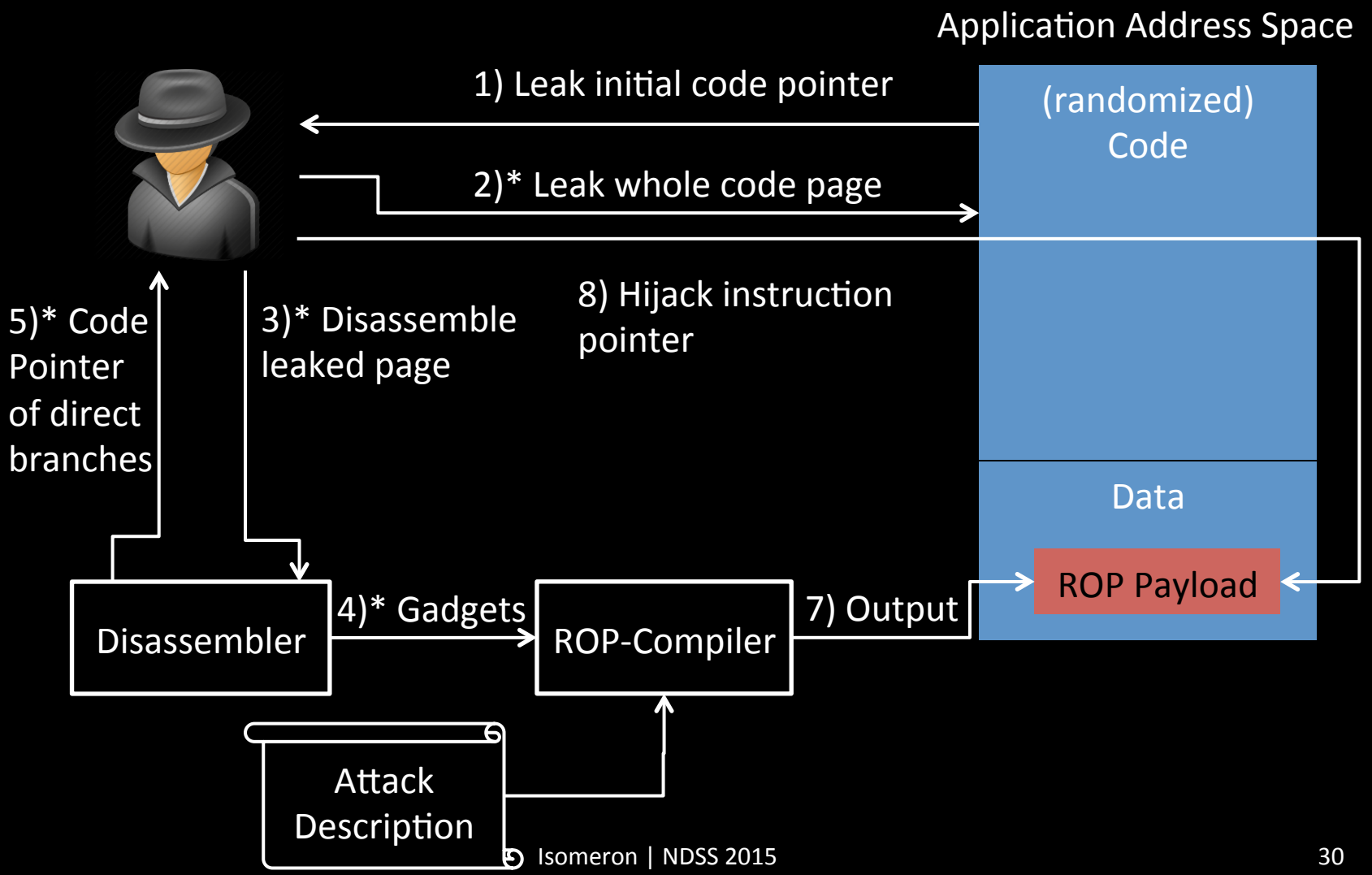
- Software suffers from security vulnerabilities, no end in sight
- Software complexity is increasing
  - Advanced devices
  - Many developers involved
- Complex software exposes large attack surface
- **Currently runtime attacks are still a crucial threat**

Return-oriented Programming

- Code-reuse attack
- Short instruction sequences ending in indirect branches
- Turing-complete
- Applicable to many architectures



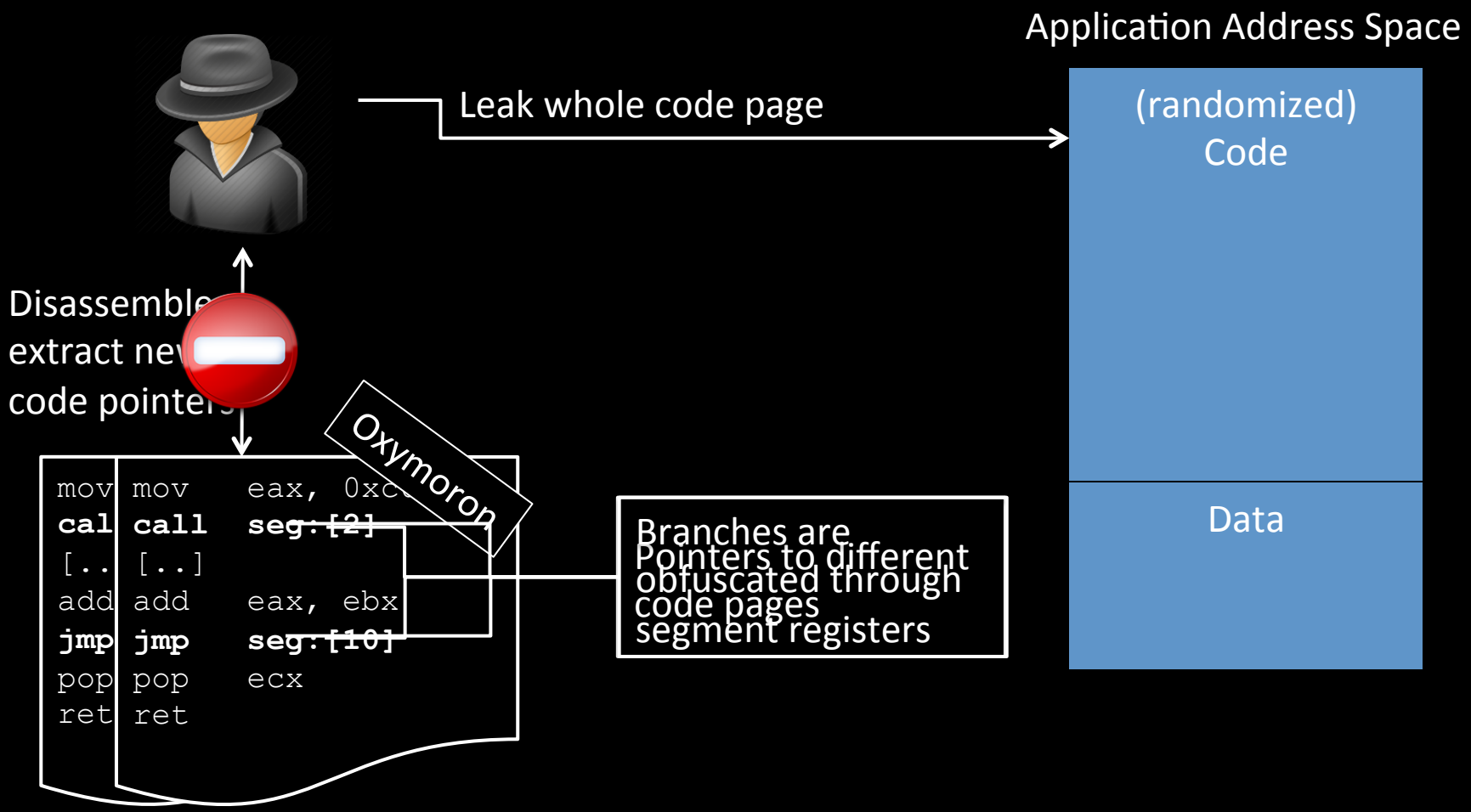
Just-In-Time ROP [IEEE S&P'13]



Oxymoron [USENIX Sec'14]

- Goal
  - Prevent conventional ROP by applying page-based randomization
  - Prevent JIT-ROP from disclosing pages by obfuscating destination addresses of direct branches
  - Allow code sharing despite randomization
- Approach
  - Addresses of direct branches are substituted through indirect branches
  - These indirect branches use segmentation registers
  - Destination of direct branches are maintained in a separate table allocated at a random address in memory

Oxymoron [USENIX Sec'14]



Isomeron

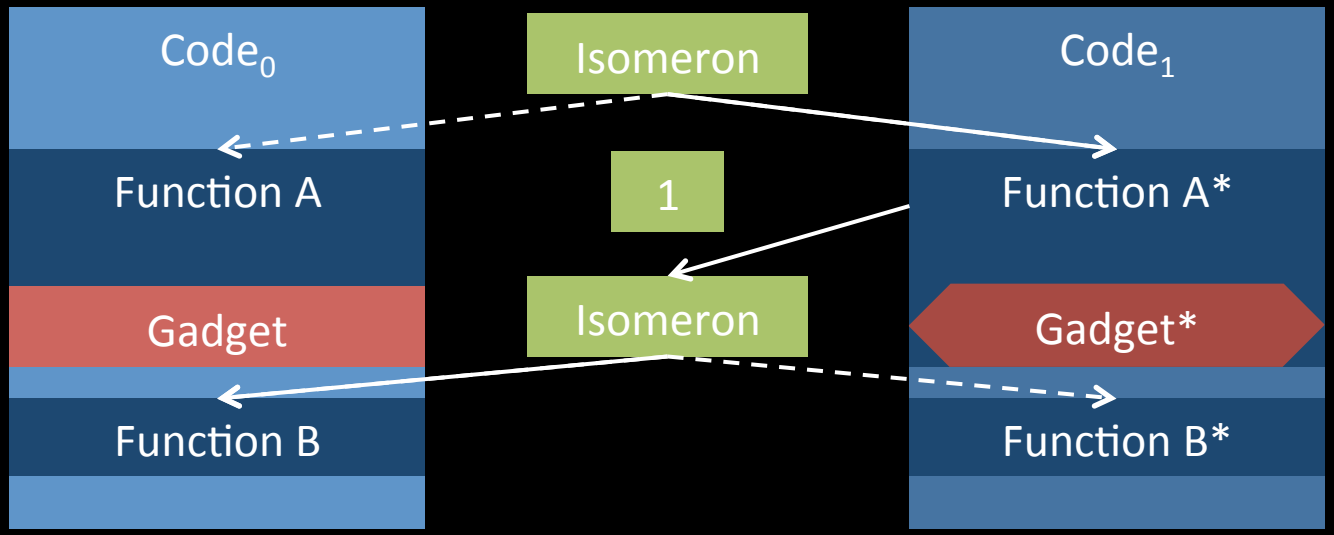
- Create a randomized copy of the application
- Ensure that gadgets at the same offset have different semantics
- Switch randomly between both copies at every function call
- Ensure that returns always arrive the original caller

Isomeron – High level

- Create a randomized copy of the application
- Ensure that gadgets at the same offset have different semantics
- Switch randomly between both copies at every function call
- Ensure that returns always arrive the original caller

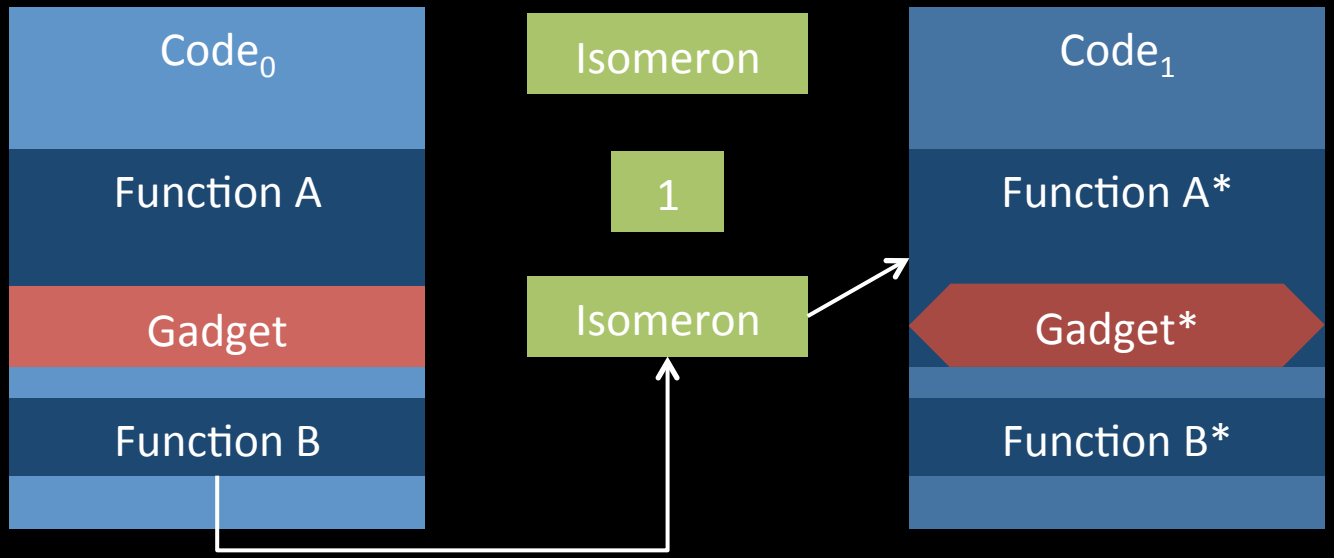
Isomeron

- Switch randomly between both copies at every function call and save call origin



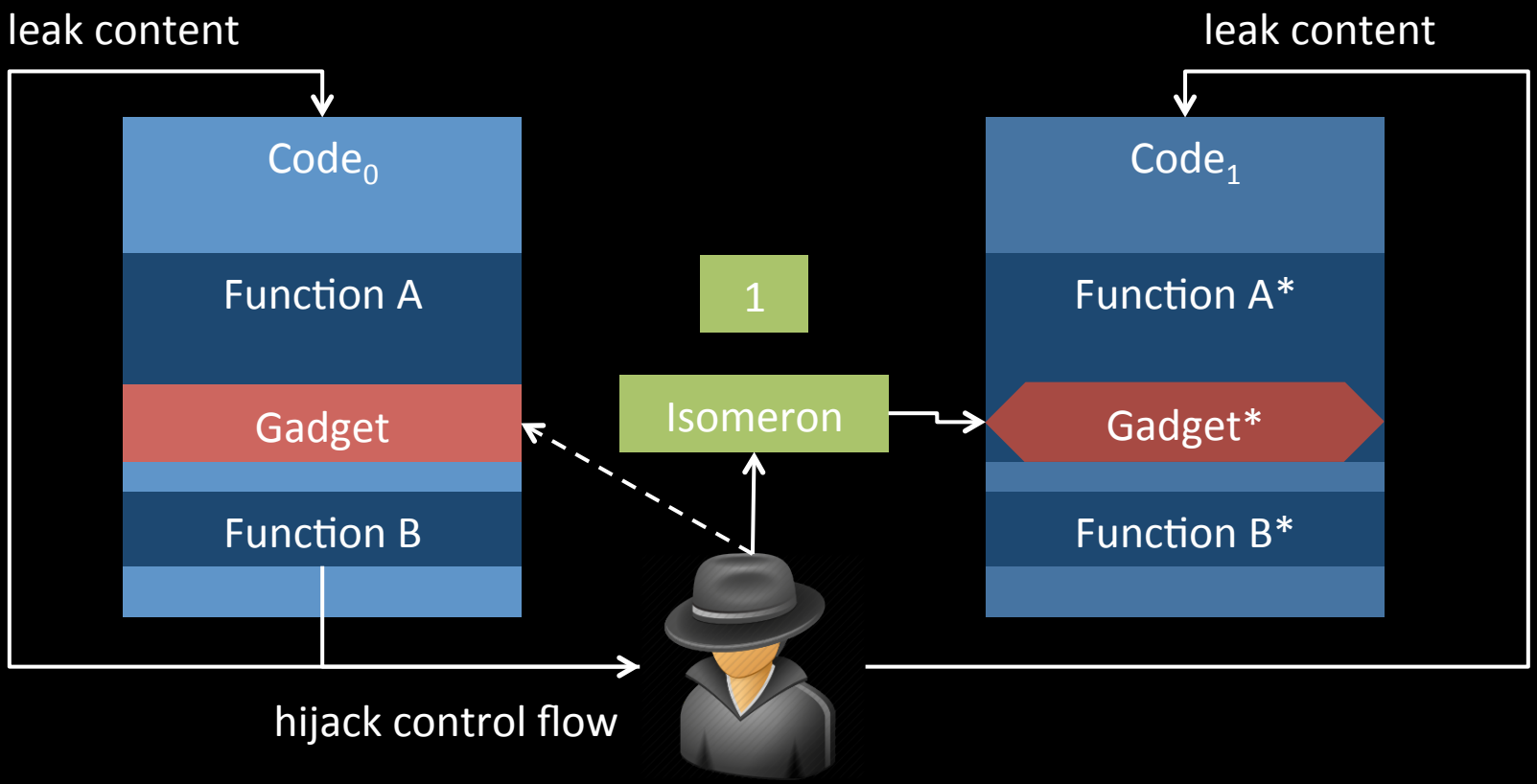
Isomeron

- Ensure that returns always arrive the original caller



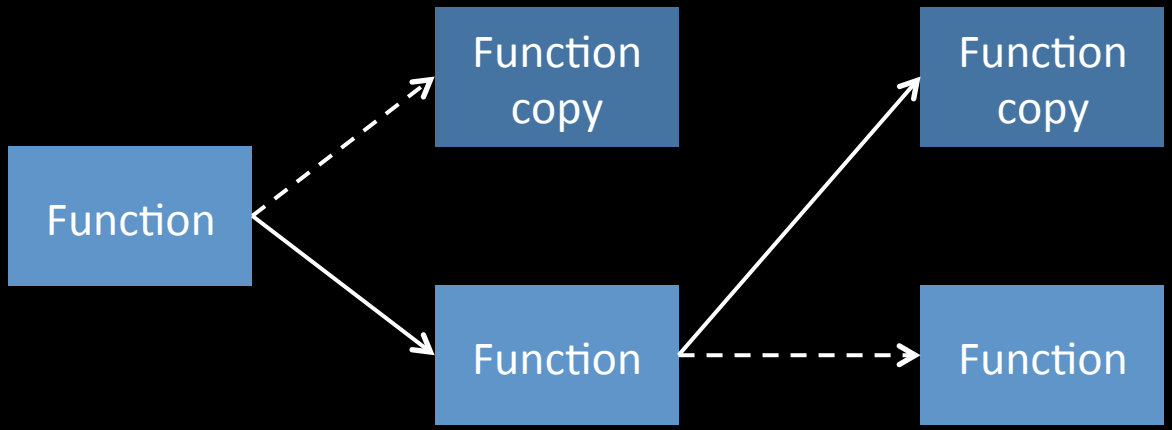
Isomeron

- Attacker is forced to guess the call origin to execute the intended gadget



Isomeron

- Switch randomly between both copies at every function call
- Ensure that returns always arrive the original caller



Isomeron

Attack flow with Isomeron

1. Leak addresses of both copies
2. Select a gadget (#1)
3. Call vulnerable function
4. Attack fails because execution is redirected to the original Gadget #0 which contains a different gadget at the same relative offset

3. Randomize each call
   - P(Func) = 0.5
   - P(Func‘) = 0.5
4. Return to orignal caller (unknown to the attacker!)