Thwarting Return Oriented Programming (ROP) Attacks

Theo de Raadt
OpenBSD

Control flow manipulation, oooh la la!

- Our wonderful tools of modern computation!
  - Wide sloppy granularity (pages) ... small objects in writeable memory
  - Languages / Tooling / Practice without strict-bounds
  - Oh noes memory damage!
  - Conditional logic makes decision **based upon** damage
  - Reaches control flow, which is stored in writeable memory
- Non-standard compute methodologies use the machine against our wishes

Common in 2000: Classic Buffer overflow attack

A program error permits stack damage...

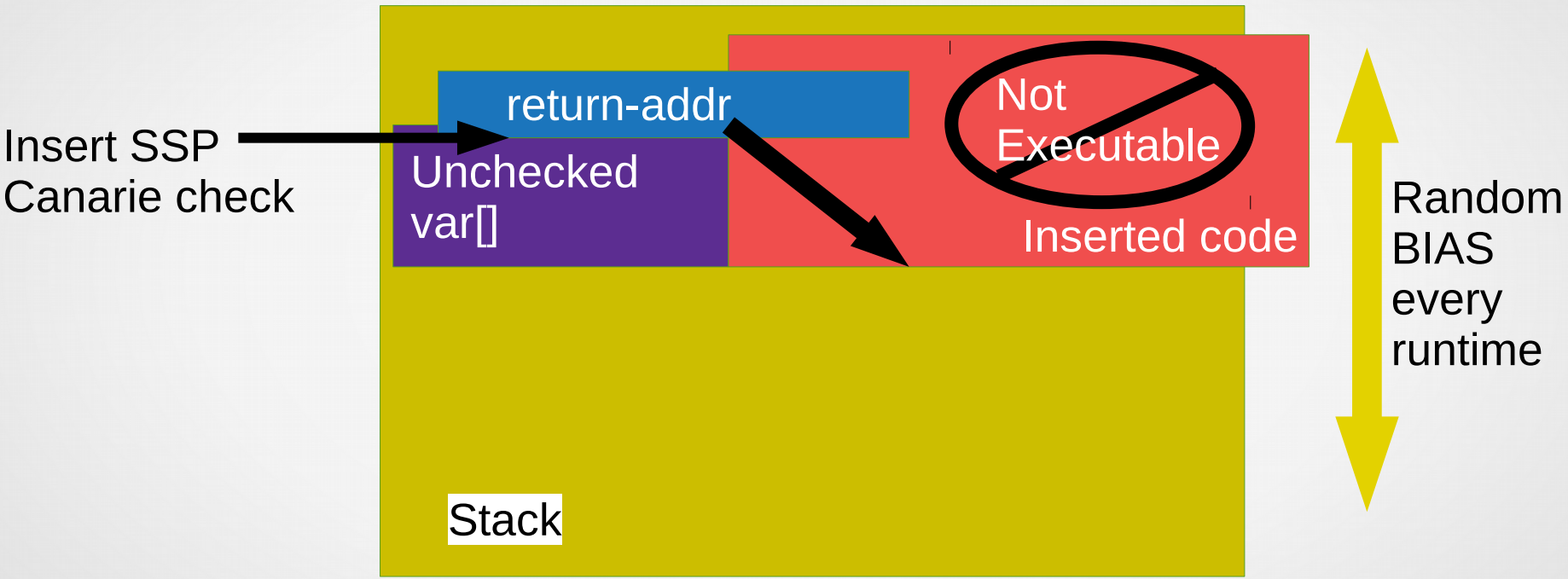
- Attackers use standard local-variable buffer stack-overflow
- Method
  - Find a mis-managed local variable buffer
  - Upload code into buffer
  - Point return address at code buffer

How it looks in memory

Mitigations for standard attack (2001-2005)

- Make stack memory non-executable (code on stack can‘t run)
- Random placement of stacks (harder to find the code offset)
- Stack protector (detect overflow before RET, and crash fast)
- Over time, practices adopted by all operating systems

Mitigations in action



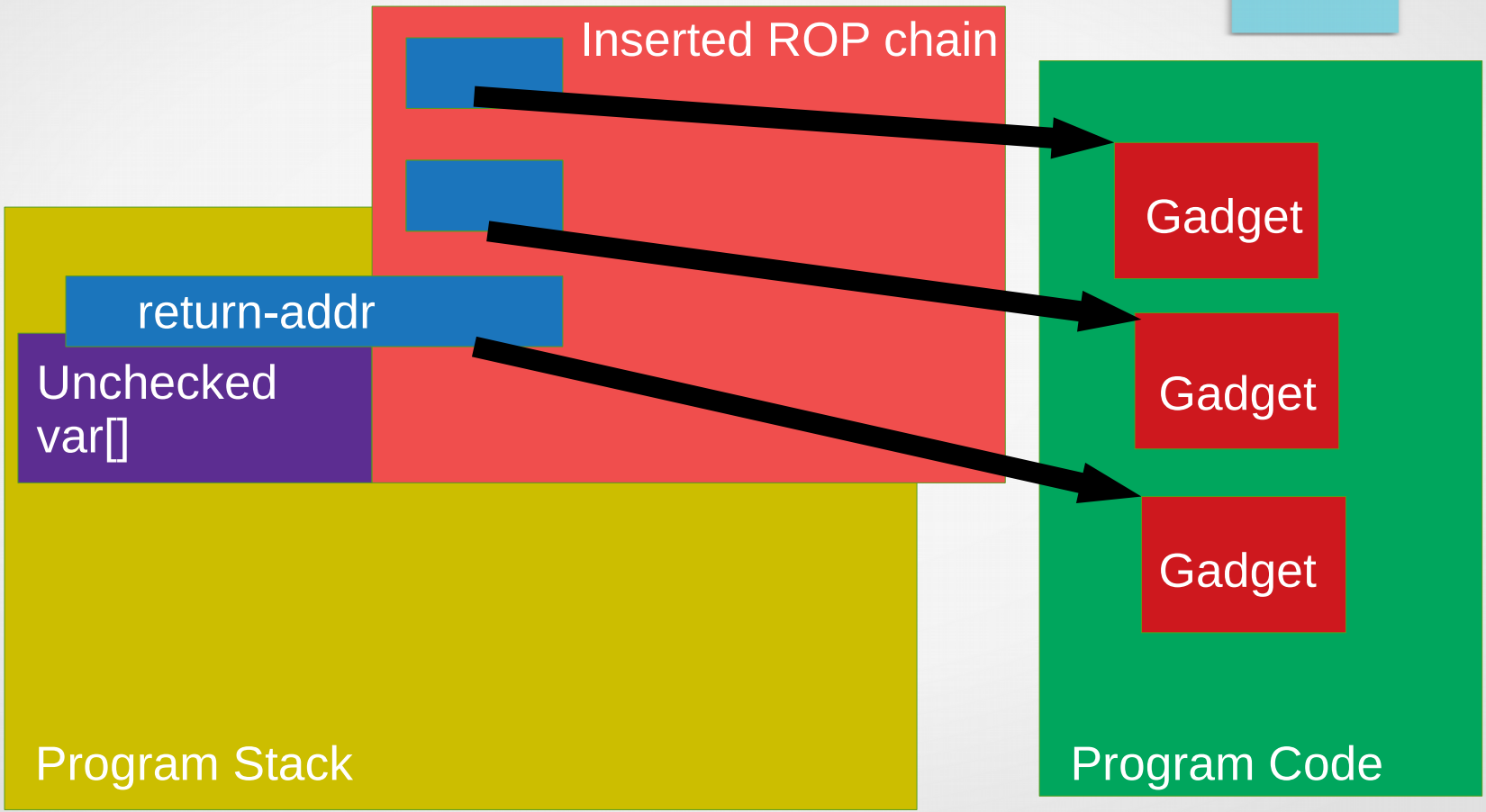
Everything solved?? NO!!!

2008: ROP method surfaces

Once again, program error permits stack damage…

- Placement of ROP-chain – series of returns into code which already exists in the program
- Sections of code are called **gadgets**
  - Small fragments that modify machine state
  - End in a RET instruction
- Utilize gadget side-effects to implement attack

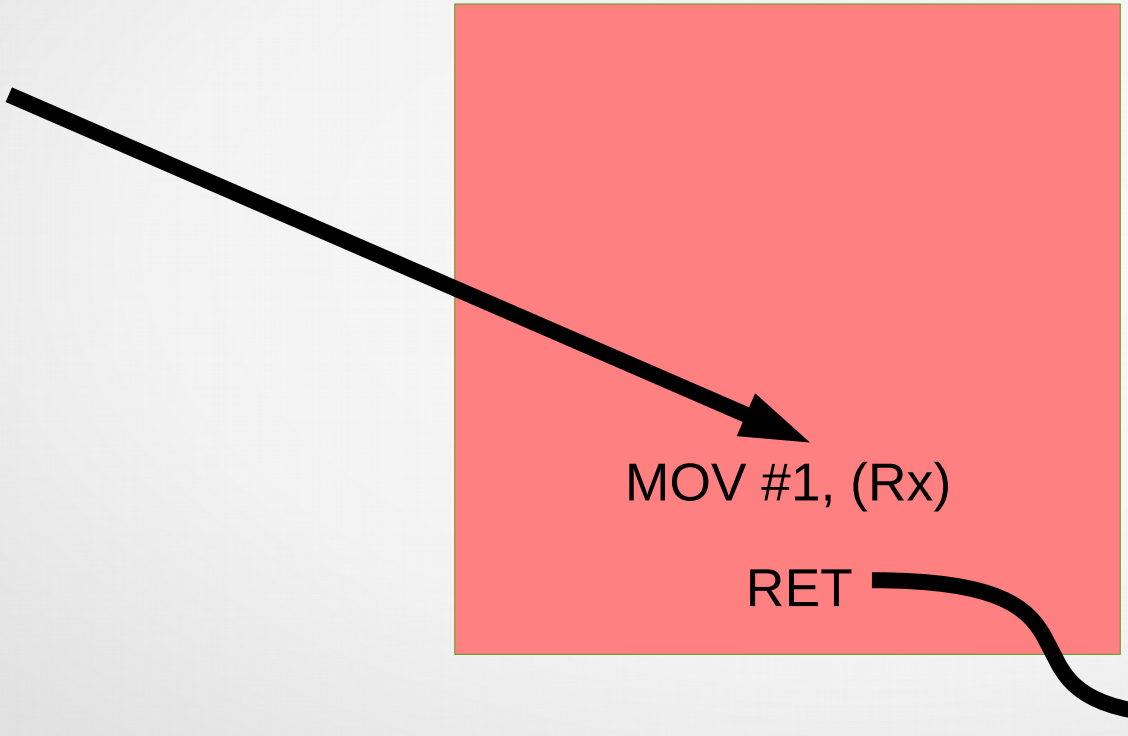
How it looks in memory



Observations made by attacker

- Discovery of gadgets
  - Gadget complexity
  - Combining artifacts – Abstract machine model
- RET instruction
  - Function tails
  - Variable-sized instruction architectures: Polymorphism, embedded 0xc3
- Shared library / PIC influences
-

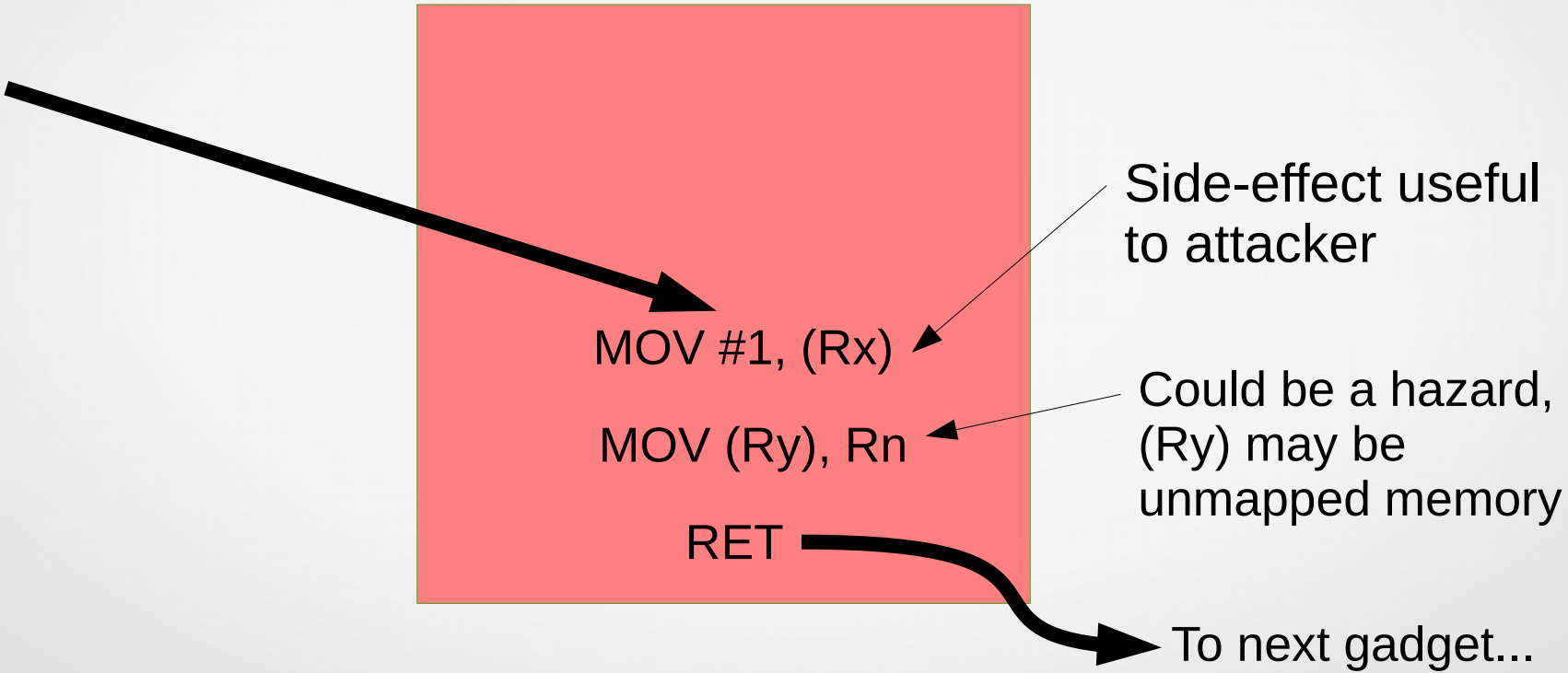
Simple Gadget



Imagine this is the side effect attacker wants

Single side-effect

More complex Gadget



x86: Gadgets hiding inside Instructions

```
ffffffff8100411c: 0f 82 64 02 00 00   jb ffffffff81004386 <intel_psr_disable+0486>
ffffffff81004122: 48 69 c3 d0 07 00   imul $0x7d0,%rbx,%rax
```

¨poly-RET¨

Look what hiding inside!

```
ffffffff81004121: 00 48 69            add byte ptr [rax + 0x69], cl
ffffffff81004124: c3                  ret
```

To solve this we would need to eliminate the byte sequences 0xc2, 0xc3, 0xca, 0xcb inside any instruction – including constant loading sequences, relative addresses, etc etc!!!!

Observations made by defender

- Reduction of usable sequence+RET would help
  - Canarie-checks before RET
  - Some Poly RET instructions can be eliminated
- Attackers like to read code for discovery
  - Remove readability?
- Complex gadgets are fragile
  - Reduce existance of simple ones, forcing use of complex ones
  - Search for ways to increase fragility further

RetGuard4

- Todd Mortimer working on a replacement for stack protector

- Non-polymorphic check in epilogue before RET
- Uses a per-function random cookie: **.openbsd.randomdata**

- Ensures standard end-of-function RET is not a gadget

RetGuard4

Function prologue:

new Localvar = retaddr ^ perFNrandomcookie

Function epilogue:

```
if  (retaddr ^ perFNrandomcookie != Localvar)
      TRAP
RET
```

X-only instruction space

- Mike Larkin has started work on making code-segments X-only
  - Kernel first, maybe userland later
- Code becomes not-readable

- Attackers will have less opportunity to read in the .text segment
- Gadgets which accidentally inspect code regions will crash

- Now possible because clang compiler doesn‘t produce data islands (switch tables, etc etc)

JIT ROP – Stack pivots

- W^X progressed to minimizing R, W, X permissions on all objects
- New: MAP_STACK option to mmap()
- Now kernel knows what memory is a stack
- Upon kernel entry, check if stack-pointer points to stack memory
  - If not, kill program
- Concerns: pthread stacks, sigaltstack

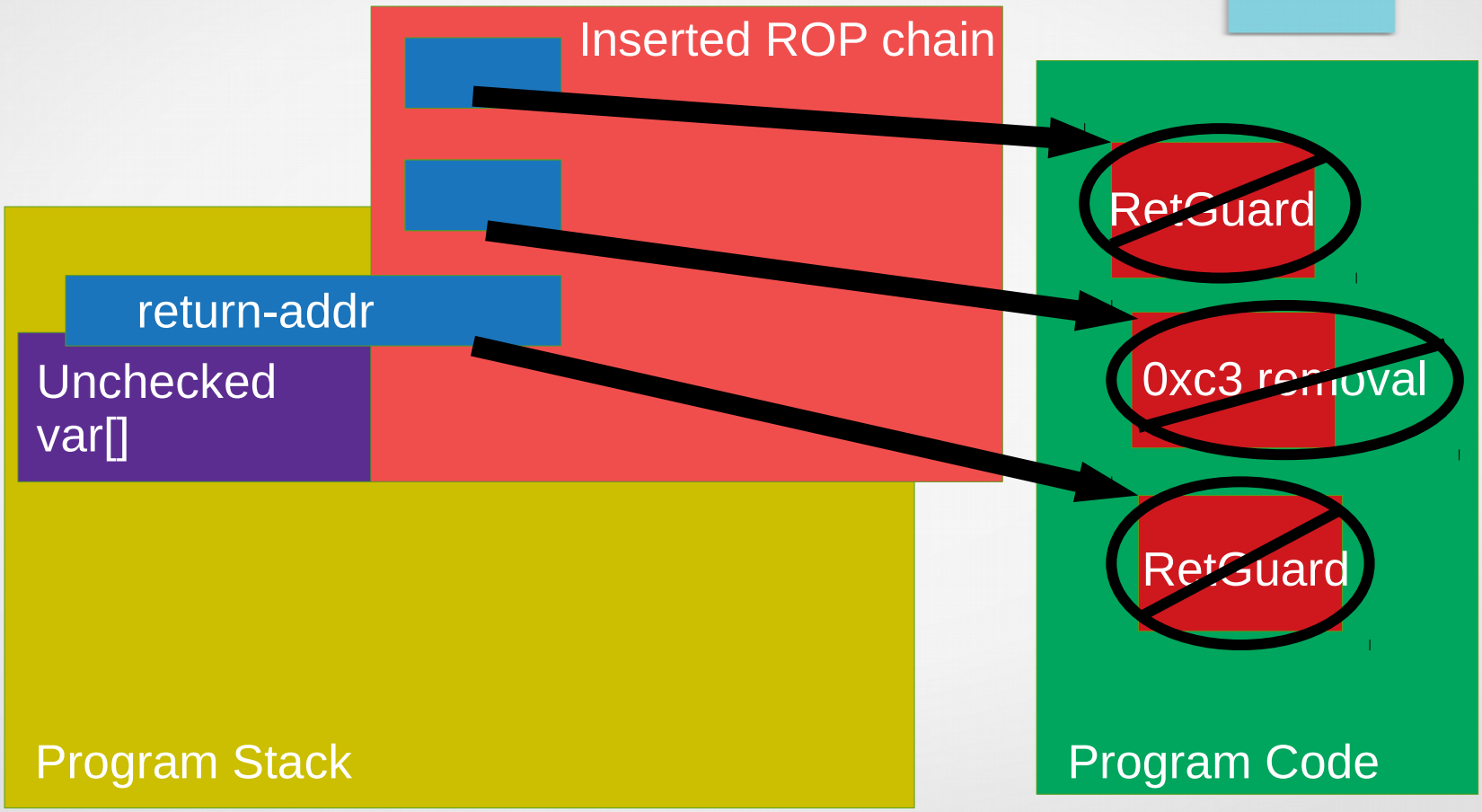
- JIT attacks often do stack-pivots onto payload in heap/data

x86: 0xc2/0xc3/0xcb reduction

- Many components to this problem
  - compiler output, assembler output, link-time
  - Instructions which must be avoided
- Ideas, but no substantial work started

Attackers depend on a rich gadget portfolio. Let‘s starve them.

Maybe we can get to this?



Everything solved?? NO!!

- None of these are complete solutions for ROP methodology
- Together, we hope they increase resistance
- Best we can do without throwing entire hardware/software ecosystem away
- Question time: Go ahead, ask about RUST...