CSci 4271W
Development of Secure Software Systems
Day 8: ROP and More Threat Modeling

Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline

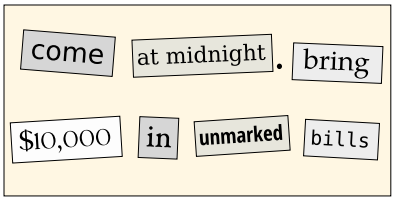
Return-oriented programming (ROP), cont'd

ROP shellcoding exercise

More perspectives on threat modeling

Attacks and shellcode lab followup

Pop culture analogy: ransom note trope



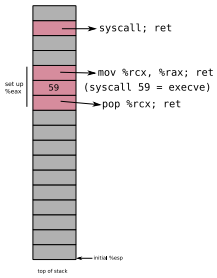
Basic new idea

- Treat the stack like a new instruction set
- "Opcodes" are pointers to existing code
- Generalizes return-to-libc with more programmability
- Academic introduction and source of name: Hovav Shacham, ACM CCS 2007

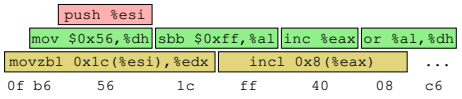
Gadgets

- Basic code unit in ROP
- Any existing instruction sequence that ends in a return
- Found by (possibly automated) search

Another partial example



Overlapping x86 instructions



- Variable length instructions can start at any byte
- Usually only one intended stream

Where gadgets come from

- Possibilities:
  - Entirely intended instructions
  - Entirely unaligned bytes
  - Fall through from unaligned to intended
- Standard x86 return is only one byte, 0xc3

Building instructions

- String together gadgets into manageable units of functionality
- Examples:
  - Loads and stores
  - Arithmetic
  - Unconditional jumps
- Must work around limitations of available gadgets

Hardest case: conditional branch

- Existing jCC instructions not useful
- But carry flag CF is
- Three steps:
  1. Do operation that sets CF
  2. Transfer CF to general-purpose register
  3. Add variable amount to `%esp`

Further advances in ROP

- Can also use other indirect jumps, overlapping not required
- Automation in gadget finding and compilers
- In practice: minimal ROP code to allow transfer to other shellcode

Outline

Return-oriented programming (ROP), cont'd

ROP shellcoding exercise

More perspectives on threat modeling

Attacks and shellcode lab followup

Setup

- Key motivation for ROP is to disable $W \oplus X$
- Can be done with a single syscall, similar to `execve` shellcode
- Your exercise for today: put together such shellcode from a limited gadget set
- Puzzle/planning aspect: order to avoid overwriting

Outline

Return-oriented programming (ROP), cont'd

ROP shellcoding exercise

More perspectives on threat modeling

Attacks and shellcode lab followup

Software-oriented modeling

- This is what we've concentrated on until now
  - And it will still be the biggest focus
- Think about attacks based on where they show up in the software
- Benefit: easy to connect to software-level mitigations and fixes

Asset-oriented modeling

- Think about threats based on what assets are targeted / must be protected
- Useful from two perspectives:
  - Predict attacker behavior based on goals
  - Prioritize defense based on potential losses
- Can put other modeling in context, but doesn't directly give you threats

Kinds of assets

- Three overlapping categories:
  - Things attackers want for themselves
  - Things you want to protect
  - Stepping stones to the above

Attacker-oriented modeling

- Think about threats based on the attacker carrying them out
  - Predict attacker behavior based on characteristics
  - Prioritize defense based on likelihood of attack
- Limitation: it can be hard to understand attacker motivations and strategies
  - Be careful about negative claims

Kinds of attackers (Intel TARA)

- Competitor
- Data miner
- Radical activist
- Cyber vandal
- Sensationalist
- Civil activist
- Terrorist
- Anarchist
- Irrational individual
- Gov't cyber warrior
- Corrupt gov't official
- Legal adversary

Kinds of attackers (cont'd)

- Internal spy
- Government spy
- Thief
- Vendor
  - Disgruntled employee
- Reckless employee
- Information partner

Outline

Return-oriented programming (ROP), cont'd

ROP shellcoding exercise

More perspectives on threat modeling

Attacks and shellcode lab followup

Reminder: what is shellcode

- Machine code that does the attacker's desired behavior
- Just a few instructions, not a complete program
- Usually represented as sequence of bytes in hex

Reminder: basic attack sequence

- Make the program do an unsafe memory operation
- Use control to manipulate contol-flow choice
  - E.g.: return address, function pointer
- Make the target of control be shellcode

Overflow example hands-on

- Steps of overflow-from-file example

Side-effects example

- A second example with a new wrinkle