Outline
- Exploiting other vulnerabilities
- Announcements intermission
- W-X (DEP)
- Return-oriented programming (ROP)

Non-control data overwrite
- Overwrite other security-sensitive data
- No change to program control flow
- Set user ID to 0, set permissions to all, etc.

Heap meta-data
- Boundary tags similar to doubly-linked list
- Overwritten on heap overflow
- Arbitrary write triggered on `free`
- Simple version stopped by sanity checks

Heap meta-data

Use after free
- Write to new object overwrites old, or vice-versa
- Key issue is what heap object is reused for
- Influence by controlling other heap operations
- Unsafe because it makes modifications the program normally would not
  - Including different types

Integer overflows
- Easiest to use: overflow in small (8-, 16-bit) value, or only overflowed value used
- 2GB write in 100 byte buffer
  - Find some other way to make it stop
- Arbitrary single overwrite
  - Use math to figure out overflowing value

Null pointer dereference
- Add offset to make a predictable pointer
  - On Windows, interesting address start low
- Allocate data on the zero page
  - Most common in user-space to kernel attacks
  - Read more dangerous than a write
### Format string attack

- **Attacker-controlled format:** little interpreter
- Step one: add extra integer specifiers, dump stack
  - Already useful for information disclosure

### Format string attack layout

```
+-----------------------------------+    +-----------------------------------+
| caller frame                      | ←→ | print frame                        |
+-----------------------------------+    +-----------------------------------+
    | copy of %rcx                      | ←→ | copy of %rdx |  argument pointer |
    | copy of %r8x                      | ←→ | copy of %rsi |
    | copy of %rdi |  lower padding |  upper padding |
```

### Format string attack: overwrite

- `%n` specifier: store number of chars written so far to pointer arg
- Advance format arg pointer to other attacker-controlled data
- Control number of chars written with padding

### Outline

- Exploiting other vulnerabilities
- Announcements intermission
- **W ⊕ X (DEP)**
- Return-oriented programming (ROP)

### Problem Set 1 available tonight

- Covers threat modeling and low-level attacks
  - You should be able to start working tomorrow, next Tuesday will have further relevant threat-modeling content
  - Due next Friday 2/9, online by 11:59pm
    - Submission will be Canvas or Gradescope-via-Canvas
  - Remember, individual and no AI help on problem sets

### Basic idea

- Traditional shellcode must go in a memory area that is writable, so the shellcode can be inserted executable, so the shellcode can be executed
- But benign code usually does not need this combination
- W xor X, really \(\neg(W \land X)\)
Non-writable code, $X \rightarrow \neg W$

- E.g., read-only .text section
- Has been standard for a while, especially on Unix
- Lets OS efficiently share code with multiple program instances

Non-executable data, $W \rightarrow \neg X$

- Prohibit execution of static data, stack, heap
- Not a problem for most programs
  - Incompatible with some GCC features no one uses
  - Non-executable stack opt-in on Linux, but now near-universal
- Windows synonym for $W \oplus X$ is “Data Execution Prevention” (DEP)

Implementing $W \oplus X$

- Page protection implemented by CPU
  - Some architectures (e.g. SPARC) long supported $W \oplus X$
  - x86 historically did not
    - One bit controls both read and execute
    - Partial stop-gap “code segment limit”
  - Eventual obvious solution: add new bit
    - NX (AMD), XD (Intel), XN (ARM)

One important exception

- Remaining important use of self-modifying code: just-in-time (JIT) compilers
  - E.g., all modern JavaScript engines
- Allow code to re-enable execution per-block
  - mprotect, VirtualProtect
- Now a favorite target of attackers

Counterattack: code reuse

- Attacker can’t execute new code
- So, take advantage of instructions already in binary
- There are usually a lot of them
- And no need to obey original structure

Classic return-to-libc (1997)

- Overwrite stack with copies of:
  - Pointer to libc’s `system` function
  - Pointer to “/bin/sh” string (also in libc)
- The `system` function is especially convenient
- Distinctive feature: return to entry point

Chained return-to-libc

- Shellcode often wants a sequence of actions, e.g.
  - Restore privileges
  - Allow execution of memory area
  - Overwrite system file, etc.
- Can put multiple fake frames on the stack
  - Basic idea present in 1997, further refinements

Outline

- Exploiting other vulnerabilities
- Announcements intermission
- $W \oplus X$ (DEP)
- Return-oriented programming (ROP)
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

ret2pop (Nergal, Müller)
- Take advantage of shellcode pointer already present on stack
- Rewrite intervening stack to treat the shellcode pointer like a return address
  - A long sequence of chained returns, one pop

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

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
<table>
<thead>
<tr>
<th>Building instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>String together gadgets into manageable units of functionality</td>
</tr>
<tr>
<td>Examples:</td>
</tr>
<tr>
<td>- Loads and stores</td>
</tr>
<tr>
<td>- Arithmetic</td>
</tr>
<tr>
<td>- Unconditional jumps</td>
</tr>
<tr>
<td>Must work around limitations of available gadgets</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardest case: conditional branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing jCC instructions not useful</td>
</tr>
<tr>
<td>But carry flag CF is</td>
</tr>
<tr>
<td>Three steps:</td>
</tr>
<tr>
<td>1. Do operation that sets CF</td>
</tr>
<tr>
<td>2. Transfer CF to general-purpose register</td>
</tr>
<tr>
<td>3. Add variable amount to %esp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Further advances in ROP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can also use other indirect jumps, overlapping not required</td>
</tr>
<tr>
<td>Automation in gadget finding and compilers</td>
</tr>
<tr>
<td>In practice: minimal ROP code to allow transfer to other shellcode</td>
</tr>
</tbody>
</table>