CSE 127 Final Review

With some slides from Nadia Heninger, Deian Stefan
Threat Modeling

- How might an attacker interact with our system?
- Which security properties do we care about in our system?
- Example: Hospital
Buffer Overflow Attacks

- Aleph One
- In a system with no defenses, what is the normal procedure for exploiting a buffer overflow?
  - What kind of patterns can you exploit?
Buffer Overflow Defenses

- Defenses:
  - Stack canaries
  - ASLR
  - W^X
  - Shadow stack (separate stack)
- How do we defeat each of these?
<table>
<thead>
<tr>
<th></th>
<th>“Normal”</th>
<th>Return-oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction pointer</td>
<td>eip</td>
<td></td>
</tr>
<tr>
<td>No-op</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>Unconditional jump</td>
<td>jmp address</td>
<td>set esp to address of gadget if some condition is met; ret</td>
</tr>
<tr>
<td>Conditional jump</td>
<td>jnz address</td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td>memory and registers</td>
<td>mostly memory</td>
</tr>
<tr>
<td>Inter-instruction (inter-gadget) register and memory interaction</td>
<td>minimal, mostly explicit; e.g., adding two registers only affects the destination register</td>
<td>can be complex; e.g., adding two registers may involve modifying many registers which impacts other gadgets</td>
</tr>
<tr>
<td></td>
<td>&quot;Normal&quot;</td>
<td>Return-oriented</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>Instruction pointer</td>
<td>eip</td>
<td>esp</td>
</tr>
<tr>
<td>No-op</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>Unconditional jump</td>
<td>jmp address</td>
<td>set esp to address of gadget if some condition is met; ret</td>
</tr>
<tr>
<td>Conditional jump</td>
<td>jnz address</td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td>memory and registers</td>
<td>mostly memory</td>
</tr>
<tr>
<td>Inter-instruction (inter-gadget) register and memory interaction</td>
<td>minimal, mostly explicit; e.g., adding two registers only affects the destination register</td>
<td>can be complex; e.g., adding two registers may involve modifying many registers which impacts other gadgets</td>
</tr>
<tr>
<td></td>
<td>“Normal”</td>
<td>Return-oriented</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Instruction pointer</td>
<td>eip</td>
<td>esp</td>
</tr>
<tr>
<td>No-op</td>
<td>nop</td>
<td>ret</td>
</tr>
<tr>
<td>Unconditional jump</td>
<td>jmp address</td>
<td>set esp to address of gadget if some condition is met; ret</td>
</tr>
<tr>
<td>Conditional jump</td>
<td>jnz address</td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td>memory and registers</td>
<td>mostly memory</td>
</tr>
<tr>
<td>Inter-instruction (inter-gadget) register and memory interaction</td>
<td>minimal, mostly explicit; e.g., adding two registers only affects the destination register</td>
<td>can be complex; e.g., adding two registers may involve modifying many registers which impacts other gadgets</td>
</tr>
<tr>
<td></td>
<td>&quot;Normal&quot;</td>
<td>Return-oriented</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Instruction pointer</td>
<td>eip</td>
<td>esp</td>
</tr>
<tr>
<td>No-op</td>
<td>nop</td>
<td>ret</td>
</tr>
<tr>
<td>Unconditional jump</td>
<td>jmp address</td>
<td>set esp to addr of gadget; ret</td>
</tr>
<tr>
<td>Conditional jump</td>
<td>jnz address</td>
<td>set esp to address of gadget if some condition is met; ret</td>
</tr>
<tr>
<td>Variables</td>
<td>memory and registers</td>
<td>mostly memory</td>
</tr>
<tr>
<td>Inter-instruction (inter-gadget) register and memory interaction</td>
<td>minimal, mostly explicit; e.g., adding two registers only affects the destination register</td>
<td>can be complex; e.g., adding two registers may involve modifying many registers which impacts other gadgets</td>
</tr>
</tbody>
</table>
Defenses

- How do we defend against ROP?
Defenses

- How do we defend against ROP?
- CFI:
  - Can be fine-grained or coarse-grained
  - Pay execution time cost
- General defense against memory attacks:
  - Type-safe/memory-safe languages
Side Channels

- Thanks to virtual memory, we can't access a victim's memory directly
- How can we do it?
- Techniques(timing):
  - Prime + Probe
  - Flush + Reload
  - Evict + Time
- Can abuse non-timing side-channels as well
  - Prime + Abort
Side Channels

- Code may leak secret value based on branches
  - Recall PA3
  - Defense?
- Spectre and Meltdown
  - Rely on cache effects of instructions that are executed, then rolled back
  - Meltdown has software patch, Spectre is unclear how to patch
Malware

- Keep in mind your threat model
  - How will your defenses differ if the attacker is some script kiddie vs the Mossad?
- How does malware spread?
  - Stuxnet
  - WannaCry
  - Morris worm
  - Samy worm
  - SolarWinds
Web Attacks and Defenses

- Attacker model
- SOP
  - Confidentiality and integrity of legitimate sites
  - Exists separately for DOM, message passing, cookies, CSS/fonts, network access
Web Attacks and Defenses

- Can a website inspect the contents of an iframe? What about an image?
- Cookie SOP: subtle difference in how an origin is defined
- What does SOP not defend against?
  - Can leak data!
  - Can make requests to other domains - CSRF!
- Defenses
  - SameSite cookies
  - CSRF tokens
Network attacks and defenses

- OSI 7-layer stack
  - Which layers do we actually care about?
Network attacks

- OSI 7-layer stack
  - Which layers do we actually care about?
- Physical/link layer threats
  - Wire tapping, packet injection
- Network layer threats
  - Network scanning, DOS attacks
  - Connection hijacking/RST injection
  - TCP connection spoofing
  - DNS poisoning
Network Defenses

- Perimeter defense
  - Firewalls - types?
  - NAT
  - Proxies
  - NIDS
- Host-based detection
- Log analysis
- Vulnerability scanning/Honeypots
Symmetric Crypto

- Premise: Alice and Bob share a key
- Types of attacks?
- Requirements:
  - Correctness: $D_k (E_k (m)) = m$
  - Security against chosen property
- One-time pad
  - Is it secure?
  - Problems?
Symmetric Crypto

- Stream ciphers: Extend small key using PRG
  - Can we reuse the key?
- Block ciphers
  - Fixed size input/output, string them together for arbitrary-length messages
  - Which is the correct choice?
- Block cipher gotchas
  - ECB mode
  - CBC/CTR
Hash Functions

- AES provides secrecy, but what if we want authentication?
  - What type of attack does this protect against?
- Hash functions
  - Which ones do you use?
- MACs
  - What type of MAC should you use?
  - What can go wrong? Is $H(k || m)$ a secure instruction?
- In what order should you apply encryption and MAC?
Public-key Crypto

- What if Alice and Bob don’t share a secret?
- Public-key crypto relies on problems that are hard one way, but not another.
- Diffie-Hellman Key Exchange!
  - What does DH rely on?
  - Is this secure against MITM?
- RSA
  - What does RSA rely on?
  - Is textbook RSA secure?
    - No! Homomorphic under multiplication
- Use ECDSA
Public-key Crypto

- Solution to forging: Digital Signatures
  - Can use RSA as well
  - Runs into same problems as RSA encryption
  - Bleichenbacher attack
  - Again, use ECDSA
Putting it all together: TLS

- Creation of a secure channel between client and server
  - What does secure mean?
- Know the TLS handshake procedure
- How do we know who we're talking to?
- Why is TLS secure (ideally?)
- How to attack TLS?
  - Downgrade attacks
  - Passive eavesdropper
TLS continued

- Why TLS 1.3?
  - Remove backwards compatibility
    - Source of attacks
  - Hide metadata
  - Remove RSA

- Have some knowledge of history behind TLS, crypto wars

---

**TLS 1.3**

TLS 1.3 encrypts the handshake immediately after doing a Diffie-Hellman key exchange.

client hello: client random, DH key exchange

server hello: server random, DH key exchange

Encrypted certificate

Encrypted signature of handshake

server finished

KDF(pms, random) → \( k_m, k_m, k_e \)

client finished

Enc\(_k_e\) (request)

KDF(pms, random) → \( k_m, k_m, k_e \)
Authentication

- Methods to authenticate, problems with each
  - Passwords
    - How do we protect passwords?
  - Security tokens
  - Biometrics
- Use multi-factor
Privacy/Anonymity

- PGP
  - What are the goals of PGP? Did PGP accomplish its goals?
- End-to-end encryption
  - Platforms, goals?
- Tor
  - Anonymous browsing
- Browser privacy enhancement
Advanced threats

● Juniper and Dual EC
● RSA S-boxes
● Flame malware
● SolarWinds
● Should you be worried about these?
  ○ Probably not, not a whole lot you can do
Law and Policy

- Relevant laws: DMCA, ECPA, CFAA
  - Very short and sweet version: you probably break the law if you didn't ask permission, but unclear if you asked permission for something and did something else
  - Government can track your communications and seize your data, but usually only with warrants (with exceptions)
- Responsible disclosure
  - Be a good citizen, work with companies so that you announce after vulnerability fixed
  - Bug bounty programs
General Tips

- Slides are your best friend
  - We want to see how well you learned the security mindset along with the topics in class
  - For each concept, ask yourself why?
- Review your methods for PAs (your old writeups may help here)
- Know your crypto
- Get sleep! Sleep improves your performance on exam