Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Ada Lerner, John Manferdelli, John Mitchell, Franziska Roesner, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Admin

- Lab 1:
  - Checkpoint Friday
  - Come to quiz section this week!
Password Checker

• Functional requirements
  – PwdCheck(RealPwd, CandidatePwd) should:
    • Return TRUE if RealPwd matches CandidatePwd
    • Return FALSE otherwise
  – RealPwd and CandidatePwd are both 8 characters long

• Implementation (like TENEX system)

```
PwdCheck(RealPwd, CandidatePwd)  // both 8 chars
  for i = 1 to 8 do
    if (RealPwd[i] != CandidatePwd[i]) then
      return FALSE
  return TRUE
```

• Clearly meets functional description
Attacker Model

```
PwdCheck(RealPwd, CandidatePwd)  // both 8 chars
for i = 1 to 8 do
  if (RealPwd[i] != CandidatePwd[i]) then
    return FALSE
return TRUE
```

- Attacker can guess CandidatePwds through some standard interface
- Naive: Try all $256^8 = 18,446,744,073,709,551,616$ possibilities
- Better: Time how long it takes to reject a CandidatePasswd. Then try all possibilities for first character, then second, then third, ....
  - Total tries: $256*8 = 2048$
Timing Attacks

• Assume there are no “typical” bugs in the software
  – No buffer overflow bugs
  – No format string vulnerabilities
  – Good choice of randomness
  – Good design

• The software may still be vulnerable to timing attacks
  – Software exhibits input-dependent timings

• Complex and hard to fully protect against
Other Examples

• Plenty of other examples of timings attacks
  – AES cache misses
    • AES is the “Advanced Encryption Standard”
    • It is used in SSH, SSL, IPsec, PGP, ...
  – RSA exponentiation time
    • RSA is a famous public-key encryption scheme
    • It’s also used in many cryptographic protocols and products
  – Recently: Spectre and Meltdown
Software Security: So what do we do?
Fuzz Testing

• Generate “random” inputs to program
  – Sometimes conforming to input structures (file formats, etc.)

• See if program crashes
  – If crashes, found a bug
  – Bug may be exploitable

• Surprisingly effective

• Now standard part of development lifecycle
General Principles

• Check inputs
General Principles

• Check inputs
• Check all return values
• Least privilege
• Securely clear memory (passwords, keys, etc.)
• Failsafe defaults
• Defense in depth
  – Also: prevent, detect, respond

• NOT: security through obscurity
General Principles

• Reduce size of trusted computing base (TCB)
• Simplicity, modularity
  – But: Be careful at interface boundaries!
• Minimize attack surface
• Use vetted component
• Security by design
  – But: tension between security and other goals
• Open design? Open source? Closed source?
  – Different perspectives
Vulnerability Analysis and Disclosure

• What do you do if you’ve found a security problem in a real system? E.g.,
  – A commercial website?
  – UW grade database?
  – Boeing 787?
  – TSA procedures?
Vulnerability Analysis and Disclosure

• Suppose companies A, B, and C all have a vulnerability, but have not made the existence of that vulnerability public
  – Company A has a software update prepared and ready to go that, once shipped, will fix the vulnerability; but B and C are still working on developing a patch for the vulnerability
  – Company A learns that attackers are exploiting this vulnerability in the wild
  – Should Company A release their patch, even if doing so means that the vulnerability now becomes public and other actors can start exploiting Companies B and C?
  – Should Company A wait until Companies B and C have patches?
CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography [Intro]

Autumn 2018

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Ada Lerner, John Manferdelli, John Mitchell, Franziska Roesner, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Cryptography and Security

• Art and science of protecting our information.
  – Keeping it confidential, if we want privacy.
  – Protecting its integrity, if we want to avoid forgeries.

Images from Wikipedia and Barnes & Noble
Some Thoughts About Cryptography

• Cryptography only one small piece of a larger system
• Must protect entire system
  – Physical security
  – Operating system security
  – Network security
  – Users
  – Cryptography (following slides)

• Recall the weakest link

• Famous quote: “Those who think that cryptography can solve their problems don’t understand cryptography and don’t understand their problems.”
Improved Security, Increased Risk

• RFIDs in car keys:
  – RFIDs in car keys make it harder to hotwire a car
  – Result: Car jackings increased
Improved Security

- RFIDs in car keys:
  - RFIDs in car keys
  - Result: Car jackings increased

**A Crypto Nerd’s Imagination:**

His laptop’s encrypted. Let’s build a million-dollar cluster to crack it.

No good! It’s 4096-bit RSA!

Blast! Our evil plan is foiled!

**What Would Actually Happen:**

His laptop’s encrypted. Drug him and hit him with this $5 wrench until he tells us the password.

Got it.
Kerckhoff’s Principle

• Security of a cryptographic object should depend only on the secrecy of the secret (private) key.

• Security should not depend on the secrecy of the algorithm itself.
Ingredient: Randomness

• Many applications (especially security ones) require randomness

• Explicit uses:
  – Generate secret cryptographic keys
  – Generate random initialization vectors for encryption

• Other “non-obvious” uses:
  – Generate passwords for new users
  – Shuffle the order of votes (in an electronic voting machine)
  – Shuffle cards (for an online gambling site)
C’s rand() Function

• C has a built-in random function: `rand()`

```c
unsigned long int next = 1;
/* rand: return pseudo-random integer on 0..32767 */
int rand(void) {
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}
/* srand: set seed for rand() */
void srand(unsigned int seed) {
    next = seed;
}
```

• Problem: don’t use `rand()` for security-critical applications!
  – Given a few sample outputs, you can predict subsequent ones