Security and Cryptography

C Programming and Software Tools

N.C. State Department of Computer Science

Why Worry?

- There are lots of threats: viruses, worms, phishing, botnets, denial of service, hacking, etc.
- How long would it take for an unprotected, unpatched PC running an older version of Windows to be hacked?
- The cost of prevention and repair is substantial
- The number of "bad guys" successfully caught and prosecuted is low ☹

Goals of Attackers

- Crash your system, or your application, or corrupt/delete your data
- Steal your private info
- Take control of your account, or your machine

Whose Problem?

- OS writers?
- Application programmers?
- Users?
- Administrators?
- Law enforcement?

Computer Security (NIST)

- "the protection afforded to an automated information system in order to attain the applicable objectives of preserving the **integrity**, **availability** and **confidentiality** of information system resources"
- Integrity – data and system
- Availability – service is available
- Confidentiality – data and privacy

Software Engineering Security

- "Security is an emergent quality of the entire system (just like quality)" -Gary McGraw
- Software Engineering secure systems requires a broad set of practices
  - No silver bullet
  - Not just "magic crypto fairy dust"
- The process of developing secure software should incorporate knowledge of what can do wrong with tools and practices appropriate to support secure system

Challenges

- Security is hard and cross-cutting
  - Hard to consider every attack
  - Requires monitoring for attack
  - Hard to get right, especially if an afterthought
  - Value is not seen until attacked
  - More than just an algorithm – involve many parts of the system
  - May make the system less user friendly
- Stallings *Cryptography and Network Security*, 6th edition

Some Categories of Problems

Programming mistakes like…

1. Failure to validate program inputs
2. Incorrect bounds checking
3. Inadequate protection of secret info
4. False assumptions about the operating environment

Validating Inputs

- Validate all inputs; don't rely on clients having done so
- Use white listing instead of black listing
- Identify special (meta) characters and escape them consistently during input validation
- Use well-established, debugged library functions to check for (a) legal URLs (b) legal filenames/pathnames (c) legal UTF-8 strings, …
- Authenticate that the communication is from correct person

Plus…

- Be paranoid (question your assumptions)
- Stay informed of security risks
- Do thorough testing
- Always check bounds on array operations
- Minimize secrets and access to secrets
  - Cryptographic algorithms
  - Appropriate encapsulation
- Utilize tools and algorithms that can help you automatically identify security vulnerabilities and protect secrets
  - Static analysis, dynamic analyses (valgrind), etc.

Buffer Overflow

- C does not automatically do bounds checking on buffers
- E.g., the following is legal:

```
void f() {
   int a[10];
   a[20] = 3;
}
```

Often, writing outside the bounds of an array causes the program to fail

Ex.: Buffer Problem

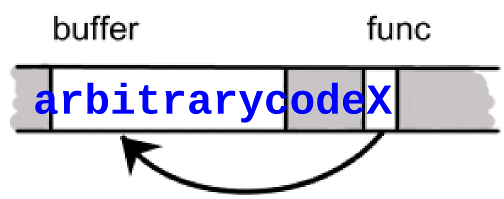
```
int main(int argc, char *argv[]) {
    char passwd_ok = 0;
    char passwd[8];
    strcpy(passwd, argv[1]);
    if (strcmp(passwd, "niklas")==0)
        passwd_ok = 1;
    if (passwd_ok) { ... }
}
```

- Layout in memory: passwd passwd_ok

- **passwd** buffer overflowed, overwriting **passwd_ok** flag
  - Any password accepted!

Another Example

```
char buffer[100];
strcpy(buffer, argv[1]);
func(buffer);
```



- Problems?
  - Overwrite function pointer
    - Execute code arbitrary code in buffer

Stack Attacks

- When a function is called…
  - parameters are pushed on stack
  - return address pushed on stack
  - called function puts local variables on the stack
- Memory layout

- Problems?
  - Return to address X which may execute arbitrary code

Risky C `<string.h>` Functions

- **`strcpy`** – use **`strncpy`** instead
- **`strcat`** – use **`strncat`** instead
- **`strcmp`** – use **`strncmp`** instead
- **`gets`** - use **`fgets`** instead, e.g.

```
char buf[BUFSIZE];
fgets(buf, BUFSIZE, stdin);
```

- More risks:
  - **`scanf, sscanf`** (use **`%20s,`** for example)
  - **`sprintf`**

Cryptography

- Art and science of secret writing
- A way of protecting communication within and between systems and stakeholders
  - Tradeoffs!

- Competing Stakeholders
  - Cryptographers – creating ciphers
  - Cryptanalysts – breaking ciphers

Encryption and Decryption

- Encryption: algorithm + key to change plaintext to ciphertext
- Decryption: algorithm + key to change ciphertext to plaintext

Caesar Cipher

- Substitution Cipher
- Symmetric Key
- Replace a letter with the letter three spots to

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- Encrypt the following: Security is important!
- Decrypt the following: SULYDFB LV, WRR!

Substitution Ciphers and Exploits

- Substitution ciphers replace one letter for another letter
  - Shift, random, etc.
- Exploitable since frequency of the letters is available
  - 'e' is the most frequently used letter in the English alphabet
- Can also use knowledge about frequent words
  - "the", "a", "I",

Vigenère Cipher

- Substitution and stream cipher
- Symmetric key
- Requires a key the same length as the plain text
  - Typically a repeating word
- Each letter in the key determines how to shift the alphabet for encryption/decryption of the corresponding letter in the plain text
  - Letter in the key would substitute for plaintext 'a'

Example Vigenère Cipher

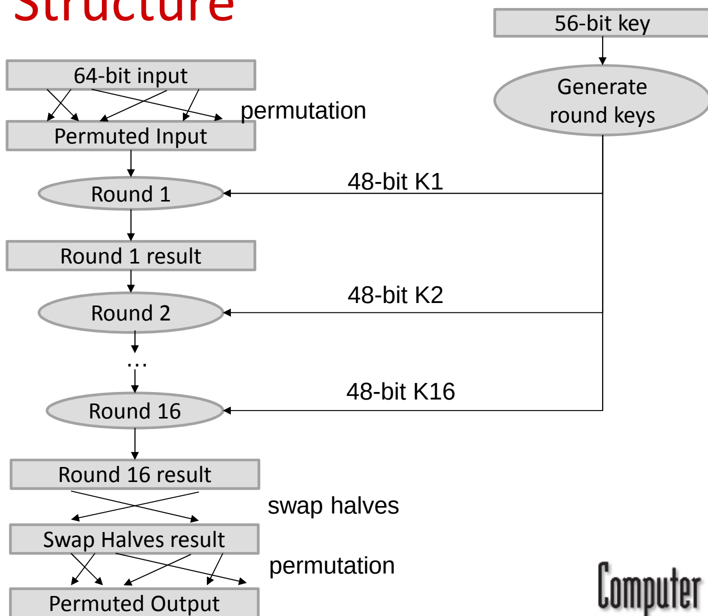
Plaintext	**S**	**E**	**C**	**U**	**R**	**I**	**T**	**Y**
Key	D	O	G	D	O	G	D	O
Ciphertext	V	S	I	X	F	O	W	M

See http://sharkysoft.com/misc/vigenere/ for trying out a Vigenère Cipher

Data Encryption Standard (DES)

- National Bureau of Standards (now NIST) in 1977
- Block cipher
  - 64-bit blocks
- Symmetric key
  - 56-bit key + 8 parity bits
  - Bits numbered 8, 16, 24, 32, 40, 48, 56, and 64 are parity bits) [assumes bits are numbered starting with 1]
- Algorithm can encrypt plaintext and decrypt ciphertext using the same key.

DES Structure



DES Algorithm

- Encryption
  - Step 1: Create 16 48-bit subkeys
  - Step 2: Encode each block
    - Initial permutation
    - Use bitwise operators to transform each half of the 64 bits
    - Repeat 16 times for each of the subkeys
- Decryption reverses the encryption algorithm
  - Subkeys are applied in reverse order

DES Exploits

- DES can be broken using a brute force attack (exhaustive key search) to identify the keys
  - With todays computing power, within hours
- Variations – increase in key size
  - Triple DES
  - Advanced Encryption Standard (AES)
  - Other block ciphers

Hashing for Authentication

- Hashing is an algorithm that transforms data
  - Difficulty to invert
  - Collision resistant
- Examples: MD4, MD5, SHA-I
- Provide the hash of information/message as an authenticator
  - The receiver can then hash the information/message to ensure that the data received is authentic

Asymmetric Ciphers

- Public-key Cryptography
  - Requires each party to have a public and a private key
  - Public key is distributed
- Confidentiality
  - Encrypt with recipient's public key
  - Recipient decrypt's with secret private key
- Authentication
  - Encrypt with sender's private key
  - Recipient authenticates message with sender's public key
- Confidentiality & Authentication
  - Sender encrypts with private key and recipient's public key
  - Recipient decrypts with private key and sender's public key

Public-Key Cryptosystem Algorithms

- RSA
- Elliptic Curve
- Diffie-Hellman
- DSS

Exploits

- Man-in-the-Middle attack
  - Diffie-Hellman lacks authentication
  - Person in the middle carries on both conversations
- RSA
  - Relies on large prime numbers
    - Knowledge of the math behind RSA can lead to exploits
  - Power/Timing attacks
    - Knowing the amount of power or how long an encryption/decryption takes can provide details about the key

Tradeoffs

- Symmetric Key Systems
  - Fast
  - Keys hard to manage and share securely
- Asymmetric Key Systems
  - Slower
  - Public keys are available and supported by infrastructure
- Cryptography algorithms are good, but only part of the solution for secure software

Software Security

- Think about security up-front
- Design and test with security in mind
- Protect your secrets and paths of communication
  - Cryptography
- Program defensively
  - Input validation
  - Check buffers and bounds
- Verification and Validation
  - Test! Think maliciously! How could you attack a system?
  - Use tools that support identifying security vulnerabilities.

References

- Dr. William Enck's CSC574 Slides
- Dr. Gary McGraw's "Building Security In Maturity Model" slides (http://www.cigital.com/presentations/bsimm10_McGraw.pdf)
- Dr. William Stallings *Cryptography and Network Security*, 6th edition slides
- Kaufman, Perlman, Speciner, *Network Security: PRIVATE Communication in a PUBLIC World*, 2nd edition