CS144 Notes: Security

- **Q: What are the goals of attackers?**
  - from machines
    * infiltration: take over machines/resources
      * defacement: replace legitimate content
    * denial of service
  - from users
    * get data
      * credit card, password, ...
    * get traffic
      * click = money
      * the real currency of our age is user's attention

- **Q: How do attackers achieve the goal?**

  Many, many different ways

  - phishing: spoof web site to look like the real one
  - pharming (DNS cache poisoning): wrong DNS resolution, for example
  - packet sniffing
  - man-in-the-middle attack
  - password brute-force attack
  - buffer overflow
  - client-state manipulation
  - cross-domain vulnerability
    * cross-site request forgery (XSRF)
    * cross-site script inclusion (XSSI)
    * cross-site scripting (XSS)
  - sql injection

  - note: some of these vulnerabilities can be controlled by "good" programming practice. more discussion later

- **Q: When we communicate over Internet, what type of "guarantee" do we want?**
- confidentiality
- message/data integrity
- authentication
- authorization

• **Q: How can we keep confidentiality of the messages?**
  - steganography: "embed" true message within harmless-looking message
    * e.g., Kathy is laughing loudly
      Osama bin laden video
      Change the lowest bit of image pixels
  
  * security by obscurity

  encryption: "scramble" message with a key, so that it wouldn't make sense to others unless they have the key
    * e.g., bitwise XOR with k

      11110000 (message) XOR 10111001 (key) -> 01001001 (ciphertext)
      01001001 (ciphertext) XOR 10111001 (key) -> 11110000 (message)
Symmetric Key Cryptography

- <explain the generalization referring to XOR example>

In general, an encryption algorithm requires:
  - \( c = F(m, k) \): encryption function (\( m \) XOR \( k \))
    * \( m \): message = plaintext. want to keep secret
    * \( c \): ciphertext. transmitted over insecure channel

  - \( m = F'(c, k) \): decryption function. inverse of \( F \) (\( c \) XOR \( k \))
    * From above, \( m = F'(F(m, k), k) \)
    * e.g., \( (m \text{ XOR } 10111001) \text{ XOR } 10111001 = m \)

- \( F(m, k), F'(m, k) \) are called "cipher"

• Q: What other property should \( F(m, k) \) have?

  Note: Ideally, one should never be able to guess \( m \) from \( c \) alone
  = ciphertext should not reveal any information about plaintext

  - Perfect secrecy
    * For all plaintext \( x \) and ciphertexts \( y \), \( \Pr[x|y] = \Pr[x] \) (a.k.a. Shannon secrecy)
    * OTP (one time pad) encryption is proven to be perfectly secret, but due to practical limitation, cannot be used directly
      * many encryption algorithms try to "mimic" OTP, e.g., RC4

• Commonly used ciphers:

  - DES (data encryption standard)
    * 64 bit block cipher
    * vulnerable to brute-force attack due to short key -> Triple DES

  - AES (advanced encryption standard)
    * 128 bit block cipher
    * 128, 192, 256 bit keys
• * adopted by NIST (national institute of standard and technology) as a replacement of DES in 2000

  – IDEA, A5 (used by GSM), Blowfish, ...

• <show AES encryption animation>

Remark:

1. addition and multiplication used for MixColumn step are slightly different from standard definition.
2. MixColumn step "mixes" values from multiple bytes. Other steps do not mix values from multiple bytes.

• Q: How can we agree on a key "secretly" over the Internet?

  – Out-of-band communication?

• Q: After A and B agreeing on secret key, how can we prevent B from impersonating A to C?

  – Q: n parties. How many keys?

• Q: Want to keep communication confidential between every party. How many keys do we need for n parties?
Asymmetric Key Cryptography

- **Basic idea**
  - two pairs of keys
    * e: encryption key
    * d: decryption key
  - \( c = F(m, e) \): encryption function
    \( m = F'(m, d) \): decryption function
  - From these, \( F'(F(m, e), d) = m \)

- **Q:** How can we keep communication secret using this mechanism?

- **Q:** How do we use this to alleviate the key agreement problem?
  - users share their "encryption" key -> public key
    * others use the public key to encrypt the message to the user
  - users keep their "decryption" key secret -> private key
    * users use their private key to decrypt message
  - no need to send the secret key over insecure channel

- **Q:** What properties should \( F, F', e \) and \( d \) satisfy to make this work?
  - "perfect secrecy" from \( F(m, e) \)
    * one cannot get \( m \) from \( c \) without \( d \)
  - one should never get \( d \) from \( e \)

- **Q:** n parties. How many keys do we need to keep all-pair confidentiality?

- **Idea first developed by Ellis, Cocks, and Williams (working for British NSA)**
  - In early 70's, but could not publish
  - First public-key cryptosystem by Diffie and Hellman in 1976
• **RSA (Rivest, Shamir and Adleman)**
  - Most widely used asymmetric key cryptography
    * other example: ECC (elliptic curve cryptography)
  
  - used by many security protocols
    * e.g., SSL, PGP, CDPD, ...
  
  - algorithm
    * pick two *random* prime numbers p and q.
    * pick $e < (p-1)(q-1)$
      • does not have to be random
      • popular choice $e = 2^{16} + 1 = 65537$ or others, like 3, 5, 35, ...
    * find $d < (p-1)(q-1)$ such that "$de \mod (p-1)(q-1) = 1$"
      • using extended-euclid algorithm in $\log((p-1)(q-1))$ time

  - two theorems
    1. there exists such unique d if e is a "coprime" to $(p-1)(q-1)
       i.e. e does not share any factor with $(p-1)(q-1)$
    2. assuming $n = pq$, $m = m^{ed} \mod n$

  - usage
    n, e: public key
    n, d: private key

    $F(m, e): c = m^e \mod n$
    $F'(c, d): m = c^d \mod n$

  - now three things to verify
    1. $F'( F(m, e), d) == m$?
    2. can we derive $m$ from $c = m^e \mod n$?
    3. can we derive $d$ from $de \mod (p-1)(q-1) = 1$?

  - Q: Is $F'( F(m, e), d) == m$?

  - Q: Can we compute $m$ from $c = m^e \mod n$?
* RSA problem.

- **Q:** can we compute d by solving \( de \mod (p-1)(q-1) = 1 \)?

* Q: Isn't it easy to get p and q from \( n = pq \)?

* large-number factorization problem

- Note: Security of RSA depends on the difficulty of factorization and RSA problems.

- Note: asymmetric cryptography is 1000x slower than symmetric cryptography
Application of Asymmetric Key Cryptography

Recap: authentication, authorization, confidentiality, message integrity

- **Q: How can we keep message "confidential"?**
  - Performance and complexity issue

- **Q: How can we "authenticate" the other party?**
  - Main idea: \( F( F'(m, d), e) = m \)
    - e.g., RSA \( m = (m^e)^d = (m^d)^e \)
  - Challenge: generate random value \( r \) and send \( c = F'(r, d) \)
    Response: send back \( F(c, e) = r \)

- **Q: How can we check the message integrity?**
  - Q: How can we make sure others did not temper with checksum?
  - signature
    - secret key encrypted checksum of the text
    - others can ensure the authenticity of message by decrypting it using public key of the author

- **Q: How do we know the public key for A *really* belongs to A?**
  - CA (certificate authority)
    - guarantees that the public key really belongs to the person
      - out of band identity check
    - issues "certificate" to each person
      - "text" (XXXX is the public key of A) signed by CA's secret key
      - others can "trust" the public key if they trust CA
SSL (https)
  - very high level description

* when contacted by client, server presents its signed certificate
  "XXX is the public key of amazon.com. This certificate is valid until
  ../../...."
* client "authenticates" server through challenge/response using the public
  key
* client/server agrees on exchange symmetric key using public key
  encryption
* client/server communicate securely through symmetric-key encryption

  - real protocol is much more complicated

* mutual authentication
* handshake of encryption algorithm
* make sure freshness of conversation
* ...

  - enabling SSL on tomcat
  * uncomment the "SSL HTTP/1.1 Connector" entry in
    $CATALINA_HOME/conf/server.xml
Key Management

- **Q: How to generate keys?**
  - user selection vs random-number generator
  - random-number generator + encryption by user password
  
  Note:
  * the need for perfect random number generator
  * the need for "safe" key storage

- **Q: What if a key/password is stolen?**
  - Multi-factor authentication
    * to minimize possibility of compromised keys, systems authenticate users based on combinations of
      * what you have (e.g., physical key, id card)
      * what you know (e.g., password)
      * who you are (e.g., fingerprint)
    * 2-factor authentication
  - smartcard
    * temper-resistant
    * stores password (or digital certificate)
    * some performs on-board RSA encryption/decryption to avoid revealing the password to the reader
  - OTP (one time password) card: e.g. SecurID by RSA security
    * a physical card flashing a new security code, say, every minute
      * temper resistant
    * keys are generated from current time + "seed key"
      * the server knows the security code generation algorithm
      * the need for time synchronization
    * user provides the security code to the server for logging in
      * often requires additional PIN from the user
    * prevents password replay attack
      * a physical device such as your laptop and smart phone
Common vulnerabilities

- common vulnerabilities to discuss
  - buffer overflow
  - client state manipulation
  - SQL/command injection
  - cross-site scripting
  - cross-site request forgery

**** buffer overflow ****

```c
int main() {
    if (login()) {
        start_session();
    }
    return 0;
}

int login() {
    char passwd[10];
    gets(passwd);
    return (strcmp(passwd, "mypasswd") == 0);
}

int start_session() {
    ...
}
```

- Q: main() -> login() -> start_session().
  How does the system remember where to return inside a function call?

  - structure of stack after function call

    main() -> login()
stacks typically grow bottom up

• What will happen if the user-input is longer than 10 characters?

<illustrate the possibility by drawing the information in the stack>

  - by making a local variable "overflow", a malicious user may jump to any part of the program
  - "attack string": carefully constructed user input for attack

• Java, or c++ string: "mostly" safe from buffer overflow attack
  - most java run time actively check incorrect address, buffer overflow, array bound checking.
  - c++ stl string class also actively checks overflow.
  - do not use c str functions: gets, strcpy, strcat, sprintf, ...

• Q: Any general solution?

  - stackguard: inserts random "canary" before return addr and checks corruption before return.
    * not a complete protection against buffer overflow.
    * /GS flag for ms c++ compiler
    * -fstack-protector-all for some gcc

  - Also, never trust user input!!

**** Client state manipulation ****

<example>
<form ...>
  <input type="hidden" name="price" value="5.50">
  ...
</form>

• Q: what is the problem?
Note: the same goes for information stored in Cookie
Note: again, never trust user's input!!

- **Q: How do we avoid the problem?**
  - authoritative state stays at the server
    * store values at the server and send session id only
      * **session id:** random number generated by the server
        Note: to avoid stolen session id attack
        * pick a random session id from a large pool
        * use "client ip" for session id generation
        * make session id short lived

  - signed-state sent to client
    * verify whether the parameters from client matches the signature
      Note:
      * sign every parameter. e.g., signature on price only
      * make the signature short lived. e.g., price fluctuation over time

- **Q: Pro/con of each approach?**

**** SQL/command injection attack ****

- **Q: Is there any problem with the following code?**

  "SELECT name, price FROM Product WHERE prod_id = " + user_input + ";"

  - Q: What if user_input = "1002 OR TRUE"?

  - Q: What if user_input = "0; SELECT * from CreditCard"?

  - CardSystems lost 263,000 card numbers through SQL injection vulnerability and was acquired by another company

- **Q: Any problem?**
system("cp file1.dat $user_input");

• Protection:
  – Never trust user input!!!! reject unless it is absolutely safe
  – For SQL: prepared statements and bind variables

    <example>
    PreparedStatement s =
        db.prepareStatement("SELECT * from Product WHERE id = ?");
    s.setInt(1, Integer.parseInt(user_input));
    ResultSet rs = s.executeQuery();

    Note: invalid input cannot make it into the SQL statement filtered out during parsing
      * similar support for other languages
  – Java Runtime.exec(command_string) executes the first word in the string as the command and the rest as the parameters.
    * Not as vulnerable as C/C++/php/...
  – "taint" propagation support in Perl/Ruby
    * user supplied strings are marked "tainted"
    * if tainted string is used inside sensitive commands (SQL, shell,...) system generates error
    * tainted string needs to be explicitly "untainted" by programmer
  – to minimize the damage in case of successful attack
    * give only the necessary privilege to your application
    * encrypt sensitive data in dbms

**** cross site scripting (XSS) ****

    <example>
    $user_name$'s profile

• Q: Any problem?
- Q: What will happen if $user_name is "<script>hack()</script>"?

- Note: if page includes user input, users may execute *any* script

• Q: How to prevent it?

- Q: not allow any html tag?

* At the minimum, escape &, <, >, ", '

- Q: What if html tags should be allowed (like html email)?

- Q: What about <img src=$user_url>?
  $user_url can be "javascript:attack-code;"

* Note: Be very careful with html attributes, scripts, URLs to other sites

Note:
- General protection against all XSS attack is VERY difficult
- Importance of white listing as opposed to black listing
- Both input validation and output sanitization

**** cross site request forgery (XSRF) ****

• review of HTTP cookie

  - arbitrary name/value pair set by the server and stored by client
  - server -> client
    * Set-Cookie: foo=bar; path=/; domain=cs188.edu; expires=Mon, 09-Dec ...

  • path and domain specify when to return the cookie
  • if expiration date is set in the future, the cookie becomes "permanent" and is stored in the hard drive
- if unspecified, the cookie becomes transient (= session cookie)
- to erase a cookie:
  1. change the expiration date to a past time
  2. set the value to null

- client -> server
  * Cookie: foo=bar

- often used to track a user login session
  * cookies are "valid" during a web browser session

- Q: Can a malicious page "access" cookie from another site?
  * same domain policy
    - basic security mechanism to protect data from malicious web site
    - a script can access documents and cookies that are from the same "domain" (= site)
    - cookies are sent back only to the same domain

<example>  http://evilsite.com
  The user has visited http://victim.com and has not logged out

  <form ... name="hack" action="http://victim.com">
    <input type="hidden" name="newpassword" value="hacked">
    ...
  </form>

  <script>
    document.hack.submit()
  </script>

- Q: What will happen? Will http://victim.com reject the request?

- Note:
  * XSRF allows attacker only to "write" to the server
  * due to same-domain policy, "read" from the server is not possible

- Q: How to prevent it?
* S1: Check Referrer header?

  • Note: Referrer header may be missing for legitimate reasons

* S2: Ask user password for every request?

* Q: Any other way?

  - "action token"

  * basic idea: make sure requests from our pages include a "signature" that a malicious page cannot get

* procedure
  1. generate "action token":
     a. action token: secret-key signed signature of session id
     b. we assume session id is random, unique per session, short lived and hard to guess
  2. embed the action token as a hidden field of the form

    - for every request
      1. compute the action token of the request
      2. take action only if it matches with the session id

* Q: can a malicious page obtain the action token from our page?

* Q: any other way to obtain the action token?