SECURITY

DR LINA BARAKAT

Department of Informatics

King's College London

Email: lina.barakat@kcl.ac.uk

Bush House (N) 6.04

Office Hours: Wednesday 13pm- 15pm

Chapters 15 of "Operating Systems Concepts" Silberschatz, Galvin, Gagne
Plus "A Classification of SQL Injection Attacks and Counter measures" Halfond, Viegas, Orso

Aims and Objectives

- To discuss security threats and attacks
- To discuss SQL Injection attacks and how to prevent them
- To explain the fundamentals of encryption, authentication, and hashing

Passwords

- Encrypt to avoid having to keep secret
  - But keep secret anyway (i.e. Unix uses superuser-only readably file /etc/shadow)
  - Use algorithm easy to compute but difficult to invert
  - Only encrypted password stored, never decrypted
  - Add **salt** to avoid the same password being encrypted to same value
- One-time passwords
  - Use a function based on a seed to compute a password, both user and computer
  - Hardware device / calculator / key fob to generate the password
  - Changes very frequently
- Biometrics: some physical attribute (fingerprint, hand scan)
- Multi-factor authentication: need two or more factors for authentication, i.e. USB dongle, biometric measure, and password

Program and System Threats

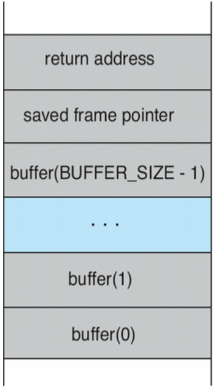
- **Trojan Horse**
  - A program that secretly performs some maliciousness in addition to its visible actions.
  - Classic Example: login emulator.
- **Trap Door**
  - A designer or a programmer deliberately inserts a security hole that they can use later.
  - Could be included in a compiler.
- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances.
- **Virus**
  - Fragment of code embedded in an otherwise legitimate program.
  - Replicate itself (by infecting other programs).
- **Worm**
  - A program that copies itself onto another system.
  - Consume system resources, often blocking out other, legitimate processes.

Stack and Buffer Overflow

- The attacker do the following:
  - Exploits a bug in a program (e.g. failure to check bounds on inputs, arguments).
  - Write past arguments on the stack into the return address on stack.
  - When routine returns from call, returns to hacked address, which is pointed to code that executes some malicious action.
- Solution:
  - Adopting a better programming methodology by performing bounds checking.
  - Prevent the execution of code that is located in the stack segment of a process's address space via using special hardware support.

Stack and Buffer Overflow

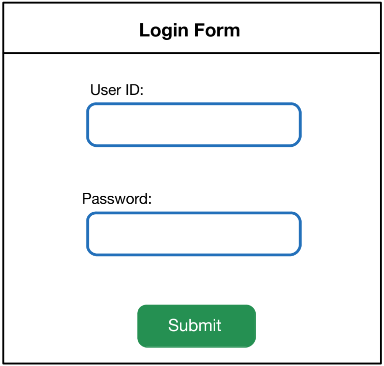
```
#include <stdio.h>
#define BUFFER SIZE 256
int main(int argc, char *argv[])
{
   char buffer[BUFFER SIZE];
   if (argc < 2)
       return -1;
   else {
       strcpy(buffer,argv[1]);
       return 0;
   }
}
```



SQL Injection

- One of the most serious threats for Web applications.
- Allows attackers to obtain unrestricted access to the databases underlying the applications.
- A class of code-injection attacks:
  part of the user's input is treated as SQL code
- Cause of SQL injection vulnerabilities: insufficient validation of user input.
- SQL Injection Mechanisms:
  - through user input
  - through cookies
  - through server variables

Example: A Simple Login Form




```
<form action="action.php" method="POST">
<label> User ID: </label>
<input type="text" name="uid" required/> "
<label> Password: </label>
<input type="password" name="passid" required/>
<input type="submit" value="Submit"/>
</form>
```

Example: A Simple Login Form

King's College London
University of London

action.php

```
…
$uid = $_POST['uid'];
$pid = $_POST['passid'];

$query = "SELECT * FROM user_info WHERE userid = '$uid'
and password = '$pid' ";

$result = mysql_query($query);

if (mysql_num_rows($result) > 0)
{
  while($row = mysql_fetch_row($result))
   {
     code to print the data of record $row
   }
}
else
{
   echo "Invalid user id or password";
}
…
```

user_info

Field	Type
userid	varchar(16)
password	varchar(16)
first_name	varchar(100)
last_name	varchar(100)
…	

Login Form

User ID:

lina

Password:

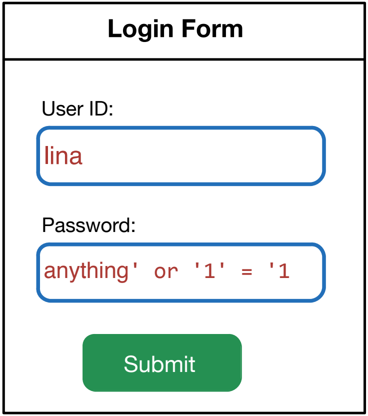
osc

Submit

```
SELECT * FROM user_info WHERE
userid = 'lina' and password = 'osc'
```

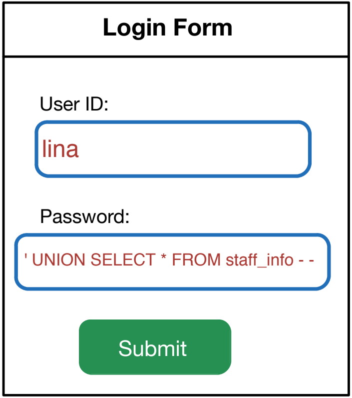
SQL Injection: Tautologies

- Purpose: Bypass authentication; Extract data; Identify injectable parameters




```
SELECT * FROM user_info WHERE
userid = 'lina'
and password = 'anything' or '1' = '1'
```

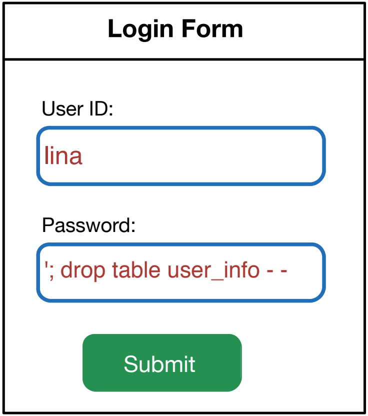
- Purpose: Bypass authentication; Extract data




```
SELECT * FROM user_info WHERE
userid = 'lina' and
password = '' UNION SELECT * FROM staff_info --'
```

SQL Injection: Piggy-Backed Queries

- Purpose: Extract data; add or modify data; execute remote commands; denial of service




```
SELECT * FROM user_info WHERE
userid = 'lina' and
password = ''; drop table user_info --'
```

SQL Injection: Illegal/Logically Incorrect Queries

- Purpose: Identify injectable parameters; Identify database; Extract data

Pin Code

convert(int, (select top 1 name from sysobjects where xtype= 'u'))

- Error Message:

Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value 'CreditCards' to a column of data type int.

SQL Injection: Inference

- Purpose: Identify injectable parameters; Identify database; Extract data

Login Form

User ID:

`' or 1 = 1 --`

Password:

anything

Submit

```
SELECT * FROM user_info WHERE
userid = '' or 1 = 1 --'
and password = 'anything'
```

Login Form

User ID:

`' or 1 = 0 --`

Password:

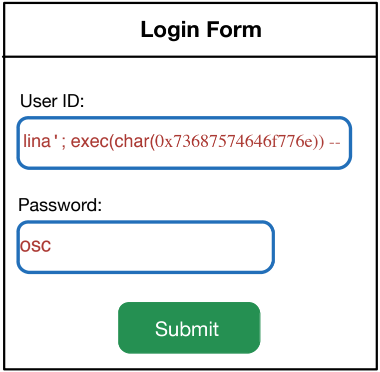
anything

Submit

```
SELECT * FROM user_info WHERE
userid = '' or 1 = 0 --'
and password = 'anything'
```

SQL Injection: Alternate Encodings

- Purpose: Evade detection
- In the example below, $char(0x73687574646f776e)$ translates to SHUTDOWN.




```
SELECT * FROM user_info WHERE
userid = 'lina'; exec(char(0x73687574646f776e)) --'
and password = 'osc'
```

Defending Against SQL Injection

King's College London — University of London

- Use **parametrised queries**:
  - specify placeholders for parameters
  - allow treating input as data rather than part of an SQL command

Not Parametrised Query

```
String query = "SELECT * FROM user_info WHERE userid= '" + request.getParameter("uid")+ "' "
AND password='" + request.getParameter("passid") + "'";

Statement statement= connection.createStatement();
ResultSet rs= statement.executeQuery(query);
```

Parametrised Query

```
String query = "SELECT * FROM user_info WHERE userid= ? AND password= ?";

PreparedStatement statement=
                   connection.prepareStatement(query);
statement.setString(1,request.getParameter("uid"));
statement.setString(2,request.getParameter("passid"));
ResultSet rs= statement.executeQuery();
```

Defending Against SQL Injection

- Input Validation
  - Simple input check can prevent many attacks.
  - Always validate user input by checking type, length, range, etc.
  - Check for certain characters and character sequences, such as *query delimiter* (;), *string delimiter* ('), *comment delimiter* (- -), etc.
- Access Rights/User Permissions
  - Create *low privileged* accounts for use by applications
- Configure database error reporting

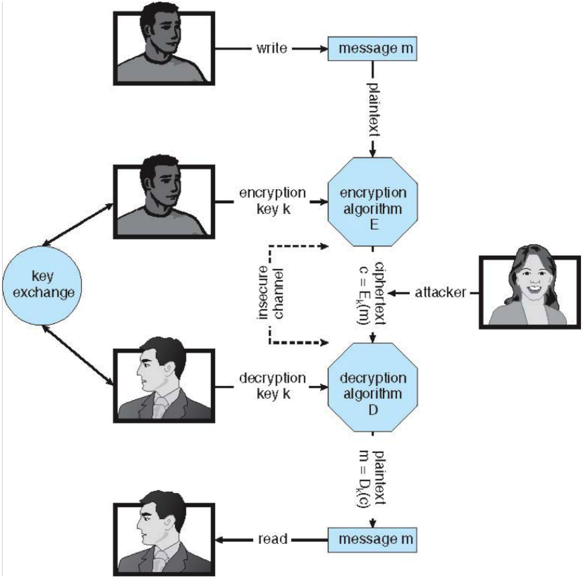
Cryptography

- Constrain the potential senders and/or receivers of a message.
- Based on **secrets** (keys)
- Good for:
  - **Confidentiality** — others cannot read the content of message
  - **Authentication** — determine origin of message
  - **Integrity** — verify that message has not been modified
  - **Nonrepudiation** — sender should not be able to falsely deny that a message was sent
- Terminology
  - M — message or plaintext
  - C — ciphertext
  - E_k — encrypting function, using key k
  - D_k — decrypting function, using key k

Symmetric Cryptography

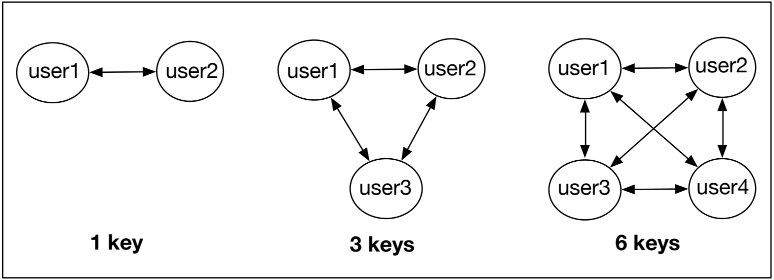
- **Same key**, k, for encryption and decryption
  - $C = E_k(M), \quad M = D_k(C)$
  - key must be kept **secret**
- Examples:
  - DES
  - 3DES
  - AES
  - RC5
- Key length determines number of possible keys
  - DES (56-bit key) — $2^{56} = 7.2 \times 10^{16}$ possible keys
  - AES (256-bit key) — $2^{256} = 1.1 \times 10^{77}$ possible keys

Symmetric Cryptography



Symmetric Cryptography - Key Distribution

- Each pair of users needs a separate key for secure communication.
- n users : $\frac{n(n-1)}{2}$ keys.



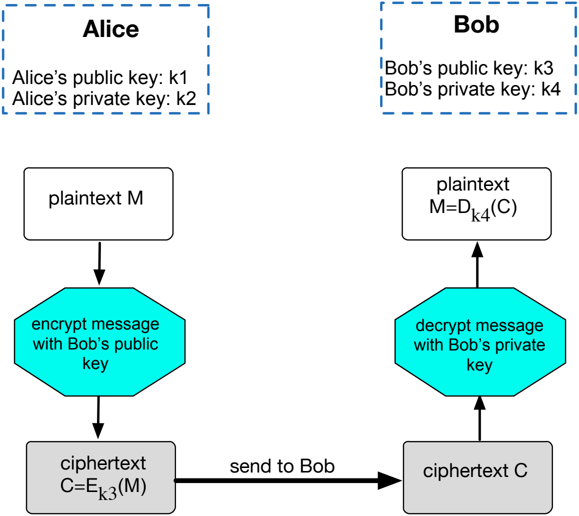
- Secure key distribution is the biggest problem with symmetric cryptography.

Asymmetric Cryptography

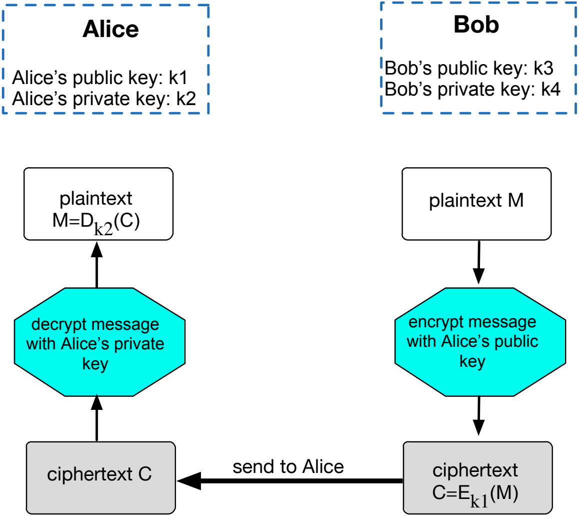
- Also called Public-key Cryptography
- Each user has **two** keys:
  - **public key** k_1 — published key, known to everyone
  - **private key** k_2 — only known to the user
- $C_1 = E_{k_1}(M), \quad M = D_{k_2}(C_1)$
- $C_2 = E_{k_2}(M), \quad M = D_{k_1}(C_2)$
- Unlike symmetric cryptography, not every number is a valid key.
- Examples:
  - RSA
  - DSS

Asymmetric Cryptography

Alice wants to send a **confidential (secret)** message to Bob.



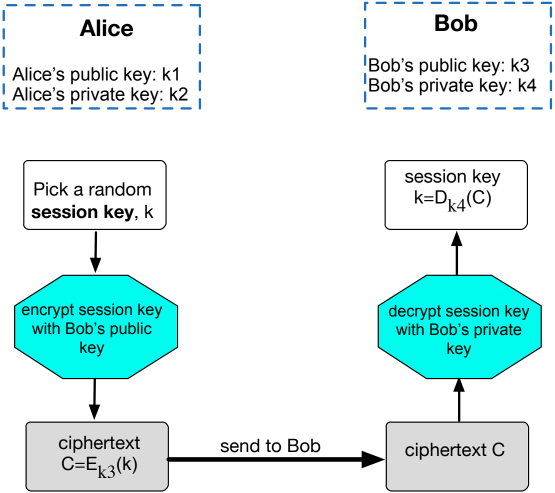
Bob wants to send a **confidential (secret)** message to Alice.



Asymmetric Cryptography

- While symmetric encryption is fast in its execution, asymmetric encryption tends to be slower in execution as a result of more complex algorithms which come with a high computation burden.
- Asymmetric encryption is not used for general-purpose encryption of large amounts of data.
- Asymmetric encryption is used for:
  - encryption of small amounts of data
  - key distribution
  - authentication and integrity
- **Hybrid Cryptosystems**:
  - Session key: randomly-generated key for one communication session.
  - Use a public key algorithm to send the session key.
  - Use a symmetric algorithm to encrypt data with the session key.

Session Key Exchange:



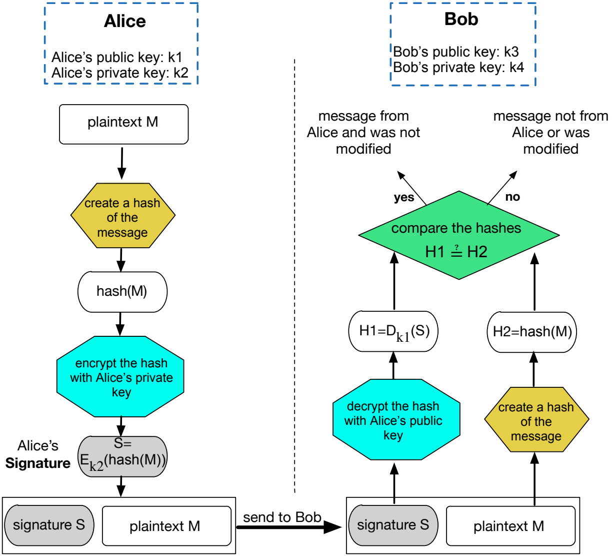
Asymmetric Cryptography: Authentication and Message Integrity

- **Authentication and Message Integrity**
  - Validate the creator of the content
  - Validate that the content has not been modified
  - The content itself does not have to be encrypted
- Encrypting a message with a private key is the same as **signing** it. **But**
  - We do not want to hide the content
  - What if the message is very long (asymmetric encryption is much slower than symmetric encryption)
- Solution: **Hash Functions**.

Hash Functions

- **Hash function**: a mathematical function which takes a message (a big number, a piece of text, or other data) and converts it into a small, fixed-length block of data.
- Properties
  - **Efficient**: computing $hash(M)$ should be computationally efficient.
  - **One-way function**: should be difficult to compute M, given $hash(M)$.
  - **Collision resistant**: must be infeasible to find an $M2 \neq M1$ such that $hash(M2) = hash(M1)$
- Examples:
  - MD5,— which produces a 128-bit hash,
  - SHA-1 — which outputs a 160-bit hash.

Asymmetric Cryptography: Digital Signature



Asymmetric Cryptography - Key Distribution

- Even the distribution of public keys requires some care
  - man-in-the-middle attack
- **Digital Certificates:**
  - Proof of who or what owns a public key
  - Public key digitally signed by a trusted party
  - Trusted party receives proof of identification from entity and certifies that public key belongs to entity
  - **Certificate authority** are trusted party — their public keys included with web browser distributions