Scripting Vulnerabilities

Scripting languages

▸ Construct commands (scripts) from predefined code fragments and user input at runtime

▸ Script is then passed to another software component where it is executed.

▸ It is viewed as a domain-specific language for a particular environment.

▸ It is referred to as very high-level programming languages

▸ Example:

  ▸ Bash, PowerShell, Perl, PHP, Python, Tcl, Safe-Tcl, JavaScript

Vulnerabilities

▸ An attacker can hide additional commands in the user input.

▸ The system will execute the malicious command without any awareness

Example 1: Command Injection

Consider a server running the following command
- ▸ `system`: takes a string as input, spawns shell, and executes the string as command in the shell.

```
void display_file(char* filename) {
  char cmd[512];
  snprintf(cmd, sizeof(cmd), "cat %s", filename);
  system(cmd);
}
```

Normal case:
- ▸ A client sets `filename`=hello.txt

<div align="center">

`cat hello.txt`

</div>

Compromised Input:
- ▸ The attacker sets `filename` = hello.txt; rm –rf /
- ▸ The command becomes:

<div align="center">

`cat hello.txt; rm -rf /`

</div>

- ▸ After displaying file, all files the script has permission to delete are deleted!

Defenses against Command Injection

Avoid shell commands

Use more secure APIs

▸ Python: subprocess.run()

▸ C: `execve( )`

Input inspection

▸ Sanitization: escape dangerous characters

▸ Validate and reject malformed input.

▸ Whitelist: only choose from allowed values

Drop privileges

▸ Run processes as non-root users.

Example 2: SQL Injection

Structured Query Language (SQL)

▸ A domain-specific language for managing data in a database

Basic syntax

▸ Obtain a set of records:

```
SELECT name FROM Accounts
SELECT * FROM Accounts WHERE name= 'Alice'
```

▸ Add or update data in the table:

```
INSERT INTO Accounts (name, age, password) VALUES ('Charlie', 32, 'efgh')
UPDATE Accounts SET password='hello' WHERE name= 'Alice'
```

▸ Delete a set of records or the entire table

```
DELETE FROM Accounts WHERE age >= 30
DROP TABLE Accounts
```

▸ Other syntax characters

▸ `--` single-line comments

▸ `;` separate different statements.

Accounts		
name	age	password
Alice	18	1234
Bob	23	5678
Eva	50	abcd

Example 2: SQL Injection

Consider a database that runs the following SQL commands

```
SELECT * FROM Accounts WHERE name= '$name'
```

▸ Requires the user client to provide the input **$name**

Normal case:

▸ A user sets **$name**=Bob:
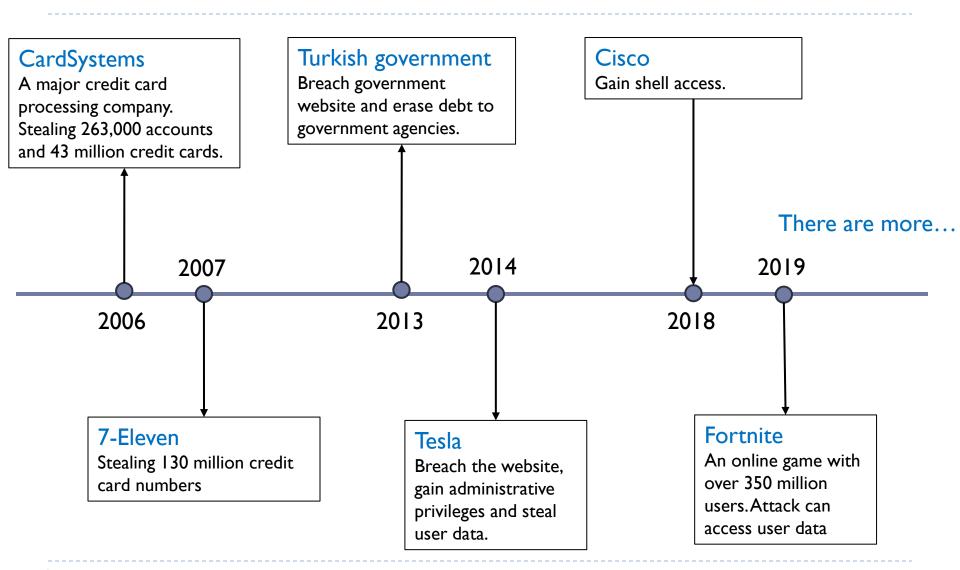
```
SELECT * FROM Accounts WHERE name= 'Bob'
```

Compromised inputs

▸ The attacker sets **$name** = ' OR 1=1 --

```
SELECT * FROM client WHERE name= '' OR 1=1 --'
```

1=1 is always true. The entire client database is selected and displayed!

▸ The attacker sets **$name** = '; DROP TABLE Accounts --

```
SELECT * FROM client WHERE name= ''; DROP TABLE ACCOUNTS --'
```

A new statement is injected, which deletes the entire table!

Real-World SQL Injection Attacks

CardSystems
A major credit card processing company. Stealing 263,000 accounts and 43 million credit cards.

Turkish government
Breach government website and erase debt to government agencies.

Cisco
Gain shell access.

There are more…

2007

2014

2019

2006

2013

2018

7-Eleven
Stealing 130 million credit card numbers

Tesla
Breach the website, gain administrative privileges and steal user data.

Fortnite
An online game with over 350 million users. Attack can access user data

Defenses against SQL Injection

Use parametrized queries

▸ Ensure that user input is treated as data, not command

▸ `cursor.execute("SELECT * FROM Accounts WHERE name= ?", (name))`

Object Relational Mapper (ORM)

▸ Abstract SQL generation and reduce risk of injection

```
class user(DBObject) {
  name =  Column(String(255));
  age =  Column(Integer);
  passsword =  Column(String(255));
}
```

Input inspection

▸ Sanitization: escape dangerous characters

▸ Validate and reject malformed input.

▸ Whitelist: only choose from allowed values

Example 3: Cross-Site Scripting (XSS)

JavaScript

▸ A programming language for web applications.

▸ The server sends the JavaScript code to the client, and the browser runs it.

▸ It makes the website more interactive.

JavaScript can be directly embedded in HTML with `<script>`

```html
<html>
  <body>
    <script>alert(Date())</script>
  </body>
</html>
```

AAA.html   ×   +

ile   C:/Users/tianwei/Desktop/AAA.html

This page says

Wed Jun 18 2025 20:06:10 GMT+0800 (Singapore Standard Time)

OK