

A2:2017 – Broken Authentication

The Case of Aerticket



The broken authentication vulnerability was found in an email sent to clients with a link to retrieve and download a passenger itinerary receipt. This link ended with an eight-digit number and, since the documents were not protected, simply changing the numbers would give access to other travelers' tickets, invoices, routes and credit card numbers. Since this flaw has existed since 2011, in theory it could have exposed data from 1.5 million bookings made over the years.

<https://sputniknews.com/europe/201608021043860684-aerticket-security-passenger-details/>

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

A2:2017 – Broken Authentication





A2:2017-Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

- Brute force attacks
 - Dictionary based attacks
 - Credential stuffing
 - Automation tools to try to guess the default admin passwords.
- Session management attacks
 - Unexpired session token vulnerability

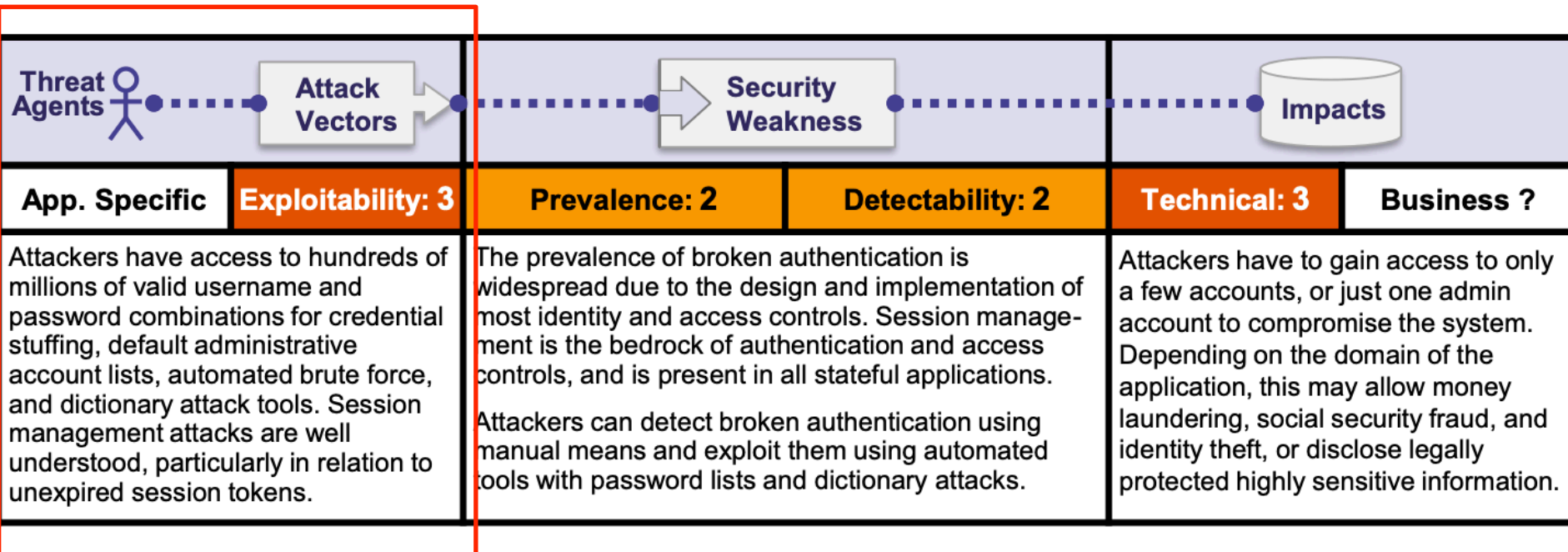
<https://resources.infosecinstitute.com/2017-owasp-a2-update-broken-authentication/#gref>

A2:2017 – Broken Authentication

 					
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
<p>Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.</p>		<p>The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications.</p> <p>Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.</p>		<p>Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.</p>	

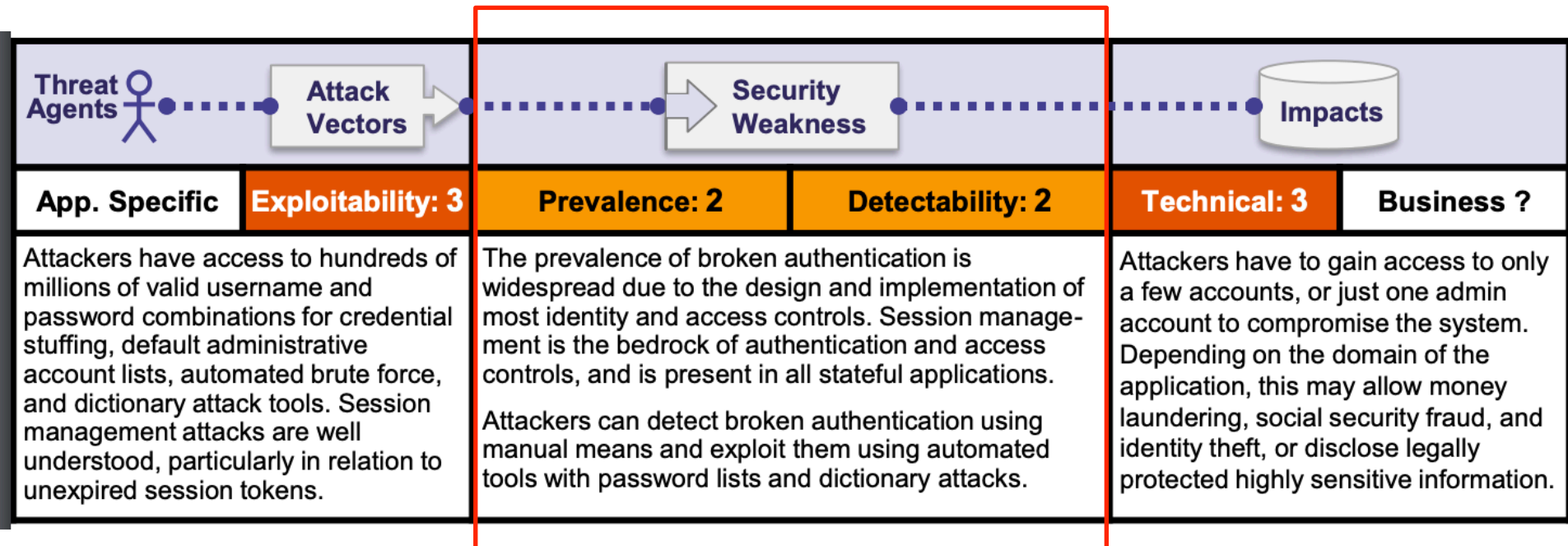
https://www.owasp.org/index.php/Top_10-2017_A2-Broken_Authentication

A2:2017 – Broken Authentication



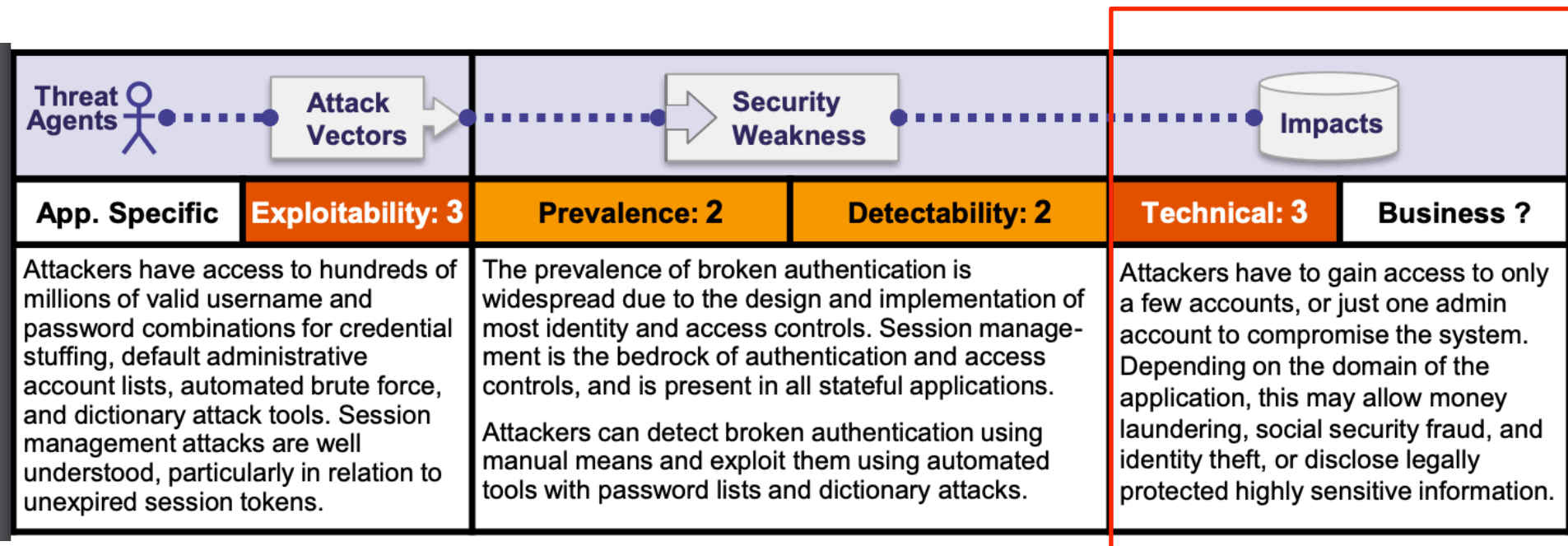
https://www.owasp.org/index.php/Top_10-2017_A2-Broken_Authentication

A2:2017 – Broken Authentication



https://www.owasp.org/index.php/Top_10-2017_A2-Broken_Authentication

A2:2017 – Broken Authentication



https://www.owasp.org/index.php/Top_10-2017_A2-Broken_Authentication

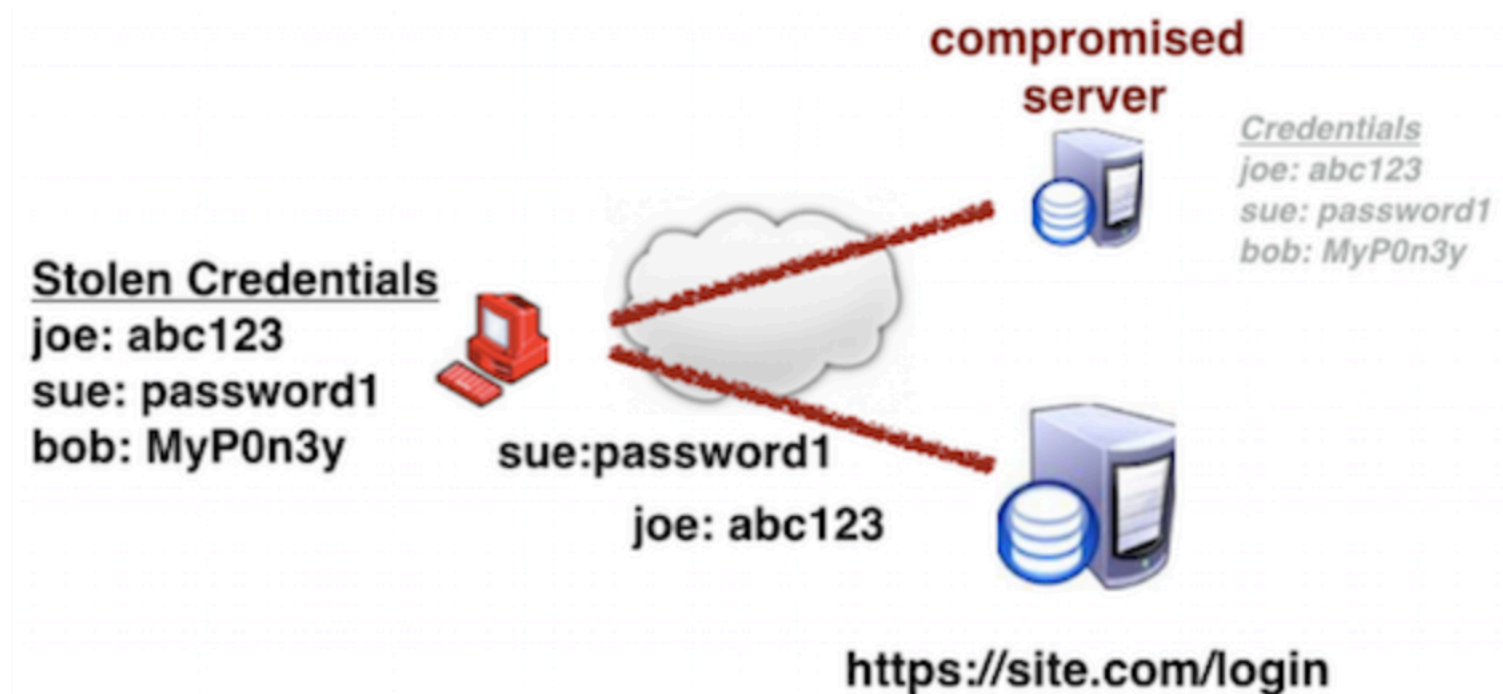
Example 1: Credential Stuffing



https://www.owasp.org/index.php/Credential_stuffing

Example 1: Credential Stuffing

Automated injection of breached username/password pairs in order to fraudulently gain access to user accounts.



Example 1: Credential Stuffing

Automated injection of breached username/password pairs in order to fraudulently gain access to user accounts.

1. The attacker acquires spilled username and passwords from a website breach or password dump site.
2. The attacker uses an account checker to test the stolen credentials against websites.
3. Successful logins (usually 0.1-0.2% of the total login attempts) allow the attacker to take over the account matching the stolen credentials.
4. The attacker can use the account information for nefarious purposes.

Example 2: Use of Passwords As a Sole Authentication Factor

Password rotation encourage users to use and reuse weak passwords.

Primary Defenses

Defense mechanisms are intended to be used in a layered approach. One single defense is in most of the cases inadequate.

- Multi-Factor Authentication
- Multi-Step Login Process
- IP blacklist
- Device Fingerprinting
- Disallow Email Address as A User ID

Primary Defenses

Defense mechanisms are intended to be used in a layered approach. One single defense is in most of the cases inadequate.

- Multi-Factor Authentication
- Multi-Step Login Process
- IP blacklist
- Device Fingerprinting
- Disallow Email Address as A User ID

Multi-Factor Authentication

Use at least 2 different types of factors



Multi-Factor Authentication

Use at least 2 different types of factors

Type 1- Something you know

Include passwords, PINs, combinations, codewords, secret handshakes. Anything that you can remember and then type.

Type 2 – Something You Have

All items that are physical objects, such as keys, smartphones, smart cards, USB drives and token devices

Type 3- Something you are

Any part of the human body that can be offered for verification, such as fingerprints, palm scanning, facial recognition, retina scans, iris scans and voice verification.

Multi-Step Login Process

- It does not necessarily rely on different types of authentication factors but requires at least 2 steps.
- **Examples:** A multi-step authentication scheme which requires two physical keys, or two passwords, or two forms of biometric identification

Primary Defenses

Defense mechanisms are intended to be used in a layered approach. One single defense is in most of the cases inadequate.

- Multi-Factor Authentication
- Multi-Step Login Process
- IP blacklist
- Device Fingerprinting
- Disallow Email Address as A User ID

IP Blacklists

Block suspicious IPs

- Because the attacker requests will likely originate from a few (or one) IP, addresses attempting to log into multiple accounts can be blocked or sandboxed.

Reduce the number false positives

- Use the last several IPs that the user's account logged in from and compare them to the suspected "bad" IP.

Reduce the negative impact of false positives

- Making the IP bans temporary, say 15 minutes, would reduce the negative impact to the customer and business services significantly.

Primary Defenses

Defense mechanisms are intended to be used in a layered approach. One single defense is in most of the cases inadequate.

- Multi-Factor Authentication
- Multi-Step Login Process
- IP blacklist
- Device Fingerprinting
- Disallow Email Address as A User ID

Device Fingerprinting

By collecting information about the devices (device, browser, language) that access a web page it is possible to identify anomalies.

Example: A Facebook user who access Facebook from his/her Android phone in Dublin, using the En-US language, suddenly access Facebook from a MacOS device using Italian language from Brazil.

Device Fingerprinting

By collecting information about the devices (device, browser, language) that access a web page it is possible to identify anomalies.

- The most common collected information include Operating System + Geolocation + Language
 - The less variables are collected the higher is the probability of false positives.
- Dealing with mismatches:
 - blocking the account or blocking the account of the source IP attempts more than 3 user IDs.

Device Fingerprinting HowTo

In the javascript page

```
<script type="text/javascript">
  function language(){
    var userLang = navigator.language|| navigator.userLanguage;
    return userLang;
  }
</script>

<%= "IP: " + request.getRemoteAddr() +
  "<br>Host: " + request.getRemoteHost()+
  "<br>Agent "+request.getHeader("User-Agent") %>
<p>Language <script> document.write(language()) </script></p>
```

Device Fingerprinting HowTo

My Site

[Home](#) | [Product List](#) | [Product Search](#) | [My Account Info](#) | [Login](#) | [Feedback](#)

Home Page

This is demo Simple web application using jsp,servlet & Jdbc.

It includes the following functions:

- Login
- Storing user information in cookies
- Product List
- Create Product
- Edit Product
- Delete Product

IP: 0:0:0:0:0:0:0:1

Host: 0:0:0:0:0:0:0:1

Agent Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.119 Safari/537.36

Language en-US

Primary Defenses

Defense mechanisms are intended to be used in a layered approach. One single defense is in most of the cases inadequate.

- Multi-Factor Authentication
- Multi-Step Login Process
- IP blacklist
- Device Fingerprinting
- Disallow Email Address as A User ID

Disallow Email Addresses as User IDs

WHY?

- It is much easier to spoof user credentials especially when passwords are reused across websites (see Credential Stuffing).
- Users can also be more easily subjected to spearfishing attacks

Example 3: Weak Password Retrieval Mechanisms

- What is the information required to reset the password?
 - Are secret questions asked? How strong they are?
- How are reset password communicated to the users?
 - The password reset tool shows you the password VS forces the user to immediately change their passwords VS password reset is done via email
- Are reset passwords generated randomly?
 - The old password is visualized in clear text VS a new password is randomly generated.
- Is the reset password functionality requesting confirmation before changing the password?
 - To limit denial-of-service attacks the application should email a link to the user with a random token, and only if the user visits the link then the reset procedure is completed.

Implement Forgot Password Mechanism

There is no industry standard for implementing a Forgot Password feature.

Step 1) Gather Identity Data or Security Questions

Ask the user for multiple pieces of hard data that should have been previously collected (generally when the user first registers).

- You should have collected some data that will allow you to send the password reset information to some out-of-band side-channel, such as a (possibly different) email address or an SMS text number
 - **Examples:** email address, last name, date of birth, account number, customer number, last 4 digits of social security number, zip code and street number for address

Implement Forgot Password Mechanism

Step 2) Verify Security Questions

Verify that each piece of data is correct for the given username.

- How many answer can be given to a question?
- How many guesses should the user be allowed to provide?

Warnings

- Do not provide a drop-down list of the user to select the questions he wants to answer.
- Avoid sending the username as a parameter (hidden or otherwise) when the form on this page is submitted.

Implement Forgot Password Mechanism

Step 3) Send a Token over a side-Channel

Lock out the user's account immediately. Then SMS or utilize some other multi-factor token challenge with a randomly-generated code having 8 or more characters.

- Out-of-band communication
- Random token valid only for a limited time period (e.g., 20 mins)

Implement Forgot Password Mechanism

Step 4) Allow user to change password in the existing session

- Display a simple HTML form with one input field for the code, one for the new password, and one to confirm the new password.
- Verify the correct code is provided and be sure to enforce all password complexity requirements that exist in other areas of the application.

As before, avoid sending the username as a parameter when the form is submitted.

Implement Forgot Password Mechanism

Step 5) Logging

Keep audit records when password change requests were submitted.

- whether or not security questions were answered
- when reset messages were sent to users and when users utilize them
- Failed logs
- Attempted use of expired tokens.
- Time, IP address, and browser information can be used to spot trends of suspicious use.

Example 3: Application session timeouts are not set properly

- A user uses a public computer to access an application.
- Instead of selecting “logout” the user simply closes the browser tab and walks away
- An attacker uses the browser an hour later and the user is still authenticated.

Session Management

Web session: sequence of network HTTP request and response transactions associated with the same user.

- Keep track of anonymous users after the very first user request (e.g., maintain language preference).
- Once the user is authenticated:
 - to identify the user on any subsequent request
 - apply security access controls
 - authorized access to the user private data
 - to increase the usability of the application

Session Management

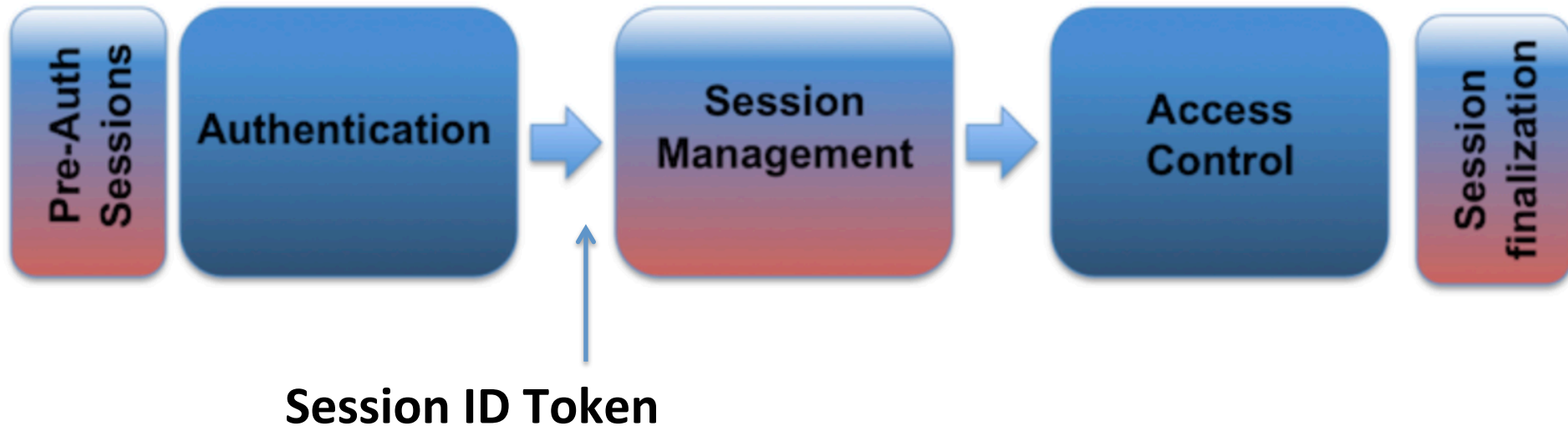


Session ID Token

binds the user authentication credentials (in the form of a user session) to the user HTTP traffic and the appropriate access controls enforced by the web application.

Implementation is in the developers hands!

Session Management



The disclosure, capture, prediction, brute force, or fixation of the session ID will lead to session hijacking (or sidejacking) attacks, where an attacker is able to fully impersonate a victim user in the web application.

Session ID Properties

- Name Fingerprinting
- Session ID Length
- Entropy
- Content

Session ID Name Fingerprinting

- The name used by the session ID should not be extremely descriptive
- The session ID names used by the most common web application development frameworks can be easily fingerprinted, such as PHPSESSID (PHP), JSESSIONID (JEE), CFID & CFTOKEN (ColdFusion), ASP.NET_SessionId (ASP .NET),
 - The session ID name can disclose the technologies and programming languages used by the web application.
- Change the default session ID name of the web development framework to a generic name, such as id.

Session ID Length

- must be long enough to prevent brute force attacks
- must be at least 128 bits (16 bytes).

Session ID Entropy

- must be unpredictable (random enough) to prevent guessing attacks. For this purpose, a good PRNG (Pseudo Random Number Generator) must be used.

Session ID Content (or Value)

- Must be meaningless to prevent information disclosure.
- Additional Information associated with the session ID must be stored on the server side
 - client IP address, User-Agent, e-mail, username, user ID, role, privilege level, access rights, language preferences, account ID, current state, last login, session timeouts, and other internal session details.
- If the session objects and properties contain sensitive information, such as credit card numbers, duly encrypt and protect the session management repository.
- Create cryptographically strong session IDs through the usage of cryptographic hash functions such as SHA1 (160 bits)

Session Management Implementation

The exchange mechanism used between the user and the web application to share and continuously exchange the session ID.

- cookies (standard HTTP header),
 - URL parameters (URL rewriting – RFC2396),
 - URL arguments on GET
 - ...
- Should allow defining advanced token properties, such as the token expiration date and time, or granular usage constraints.
- Use built-in frameworks: JEE, ASP.NET, PHP

Used VS Accepted Session ID Exchange Mechanisms

A web application should make use of cookies for session ID exchange management.

Other exchange mechanism can be accepted:

confirm via thorough testing all the different mechanisms currently accepted by the web application when processing and managing session IDs, and limit the accepted session ID tracking mechanisms to just cookies.

- It is mandatory to use an encrypted HTTPS (SSL/TLS) connection for the entire web session.

Cookies

Cookies provides multiple security features in the form of cookie attributes that can be used to protect the exchange of the session ID:

- **Secure cookie attribute** ensure the session ID is only exchanged through an encrypted HTTPS (SSL/TLS) connection.
- **Non-persistent cookies:** It does not have an EXPIRES attribute. Ensures that session ID does not remain on the web client cache for long periods of time, from where an attacker can obtain it.

Session Expiration

Set session expiration timeouts

- minimize the time period an attacker can launch attacks over active sessions and hijack them
- The session expiration timeout values must be set accordingly with the purpose and nature of the web application, and balance security and usability (2-5 minutes for high-value applications and 15-30 minutes for low risk applications).
- When a session expires, the web application must take active actions to invalidate the session on both sides, client and server.

Automatic Session Expiration

Idle Timeout

This timeout defines the amount of time a session will remain active in case there is no activity in the session

Absolute Timeout

This timeout defines the amount of time a session will remain active in case there is no activity in the session

Renewal Timeout

session ID is automatically renewed, in the middle of the user session, and independently of the session activity and, therefore, of the idle timeout.

Manual Session Expiration

Allow users to actively close their session once they have finished using the web application.

Web applications must provide a visible and easily accessible logout (logoff, exit, or close session) button.

Web Content Caching

Caching web application contents is allowed, the session IDs must never be cached, so it is highly recommended to use the Cache-Control: no-cache="Set-Cookie, Set-Cookie2" directive, to allow web clients to cache everything except the session ID.

Is The Application Vulnerable? (1/2)

- Permits automated attacks such as **credential stuffing**, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as “Password1” or “admin/admin”
- Uses weak or ineffective credential recovery and forgot-password processes, such as “knowledge-based answers”, which can be easily guessed.

Is The Application Vulnerable? (2/2)

- Use plain text, encrypted, or weakly hashed passwords.
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL
- Does not rotate session ID after successful login
- Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on tokens) aren't properly invalidated during logout or a period of inactivity.

How To Prevent

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the [top 10000 worst passwords](#)↗.
- Align password length, complexity and rotation policies with [NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets](#)↗ or other modern, evidence based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

Implement A Strong Session Management

Several Ways To Exchange Session IDs

- **User Authentication:** Authentication credentials are provided from the login page. Won't work if the same user is logged in different browsers.
- **HTML Hidden Field:** A unique hidden field in the HTML that is used to keep track of the session.
- **URL Rewriting:** A session identifier parameter is appended with every request and response to keep track of the session.
- **Cookies:** Cookies are small piece of information that is sent by the web server in response header and gets stored in the browser cookies. When a client make further request, it adds the cookie to the request header and we can utilize it to keep track of the session.

Exchange Session IDs using Cookies

Client

Server



Creates a session id

Exchange Session IDs using Cookies

Client

Server



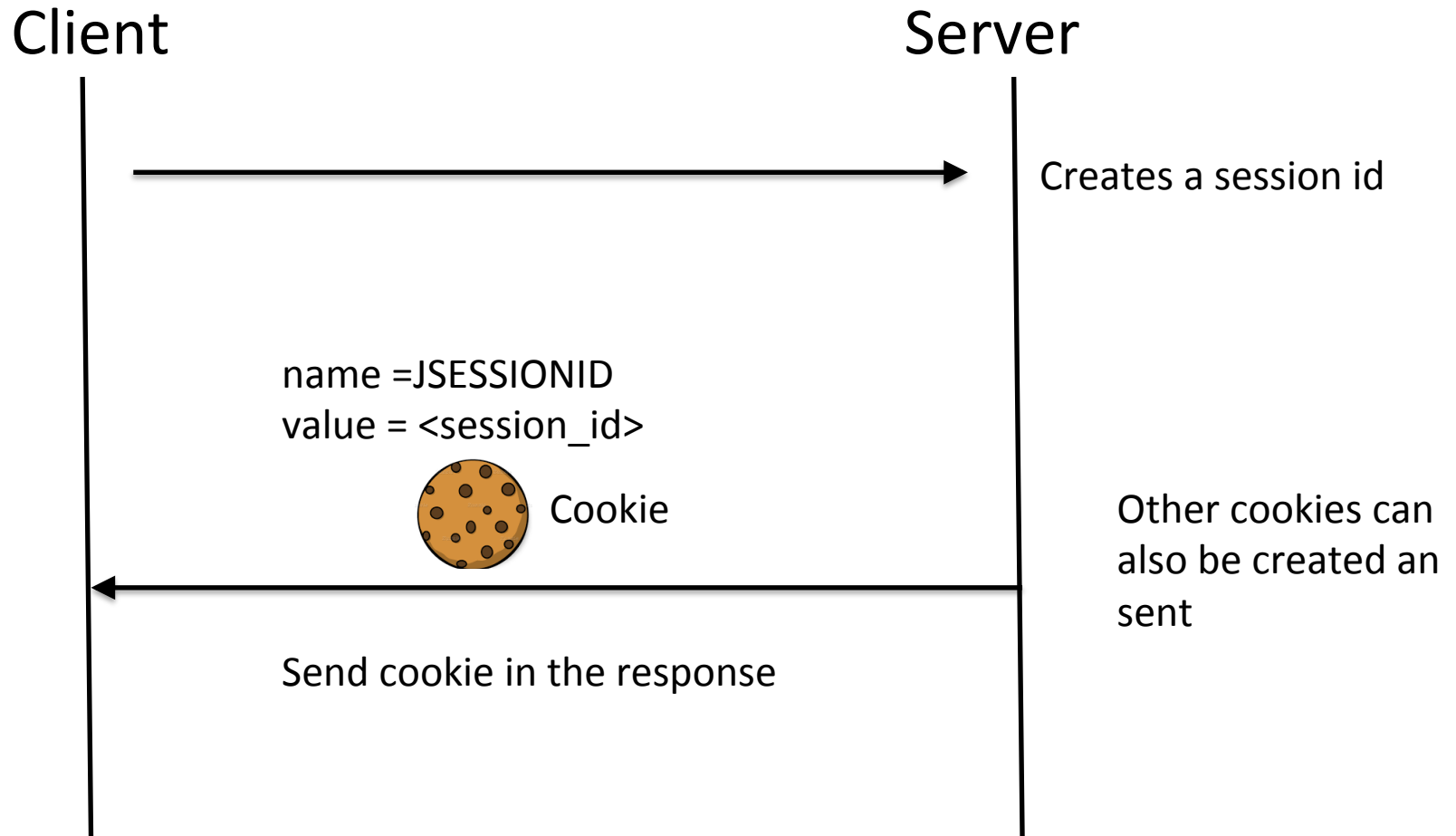
Creates a session id



Cookie

name =JSESSIONID
value = <session_id>

Exchange Session IDs using Cookies



Create a New WebApp Using the Code inside Cookie Tutorial.

What are the Problems?

- A password is stored locally in clear text.
- Lack of session management. JSESSIONID cookie is only created when an HttpSession object is created.

Session in Java Servlet

- Servlet API provides Session management through **HttpSession** interface
- A session can be obtained from `HttpServletRequest` object using:
 - **`HttpSession getSession()`**: returns the session object attached with the request, if the request has no session attached, then it creates a new session and return it.
 - **`HttpSession getSession(boolean flag)`**: This method returns `HttpSession` object if request has session else it returns null.

Some Important Methods

String getId(): Returns a string containing the unique identifier assigned to this session.

Object getAttribute(String name): – Returns the object bound with the specified name in this session, or null if no object is bound under the name.

long getCreationTime() – Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

Some Important Methods

setMaxInactiveInterval(int interval) – Specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

ServletContext getServletContext() – Returns ServletContext object for the application.

boolean isNew() – Returns true if the client does not yet know about the session or if the client chooses not to join the session.

void invalidate() – Invalidates this session then unbinds any objects bound to it

JSESSIONID Cookie

When we create a new HttpSession object, a Cookie is added to the response object with name JSESSIONID and value session id.

- This cookie is used to identify the HttpSession object in further requests from client.
- JSESSIONID cookie is used for session tracking, so we should not use it for our application purposes to avoid any session related issues.

Create a New WebApp Using the Code inside Session folder .

What are the Problems?

- The cookie containing the SESSION ID should be renamed not to disclose information about the platform.
- A password is still stored locally in clear text.
- If a session has not been established, it is always necessary to redirect the user to the login page.
- Transmission of cookies should only happen through HTTP request and response.
- The channel should be encrypted, for example, using SSL or TLS.

Rename JSESSIONID for Security

In the Tomcat startup script (*catalina.bat/catalina.sh*), change the session cookie name and the session ID

```
1  #!/bin/sh
2  # // .....
3  # -----
4  # Start/Stop Script for the CATALINA Server
5  # // .....
6  # -----
7  JAVA_OPTS="$JAVA_OPTS
8      -Dorg.apache.catalina.SESSION_COOKIE_NAME=MYJSESSIONID
9      -Dorg.apache.catalina.SESSION_PARAMETER_NAME=myjsessionId"
10 # // .....
```

Only Store Encrypted Password

```
package ie.ucd.servlets;

import java.io.IOException;

/**
 * Servlet implementation class LoginServlet
 */
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private final String userID = "admin";
    //no longer using a password in clear
    private final String hashedPassword = "5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8";

    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        // get request parameters for userID and password
        String user = request.getParameter("user");
        String pwd = request.getParameter("pwd");

        System.out.println(DigestUtils.sha1Hex(pwd));
        System.out.println(hashedPassword);

        if(userID.equals(user) && DigestUtils.sha1Hex(pwd).equals(hashedPassword)){
            //get the old session and invalidate
            HttpSession oldSession = request.getSession(false);
            if (oldSession != null) {
                oldSession.invalidate();
            }
        }
    }
}
```

SHA1 is used as hashing function

A Servlet Filter Redirects Requests to the Login Page if The Session Is Invalid

```
package ie.ucd.servlets;

import java.io.IOException;

public class AuthenticationFilter implements Filter {

    private ServletContext context;

    public void init(FilterConfig fConfig) throws ServletException {
        this.context = fConfig.getServletContext();
        this.context.log("AuthenticationFilter initialized");
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {

        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        HttpSession session = req.getSession(false);

        String path = req.getRequestURI();
        if(path.endsWith("login.html") || session != null || req.getServletPath().equals("/LoginServlet"))
            // pass the request along the filter chain
            chain.doFilter(request, response);
        else {
            if (session == null ) { //checking whether the session exists
                this.context.log("Unauthorized access request");
                res.sendRedirect("login.html");
            }
        }
    }

    public void destroy() {
        //close any resources here
    }
}
```


Authentication Filter

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml
  <display-name>ServletCookieExample</display-name>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

  <filter>
    <filter-name>AuthenticationFilter</filter-name>
    <filter-class>ie.ucd.servlets.AuthenticationFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>AuthenticationFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <session-config>
    <cookie-config>
      <http-only>true</http-only>
      <secure>true</secure>
    </cookie-config>
  </session-config>

</web-app>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml
  <display-name>ServletCookieExample</display-name>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

  <filter>
    <filter-name>AuthenticationFilter</filter-name>
    <filter-class>ie.ucd.servlets.AuthenticationFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>AuthenticationFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <session-config>
    <cookie-config>
      <http-only>true</http-only>
      <secure>true</secure>
    </cookie-config>
  </session-config>

</web-app>
```

Modify web.xml to enforce trasmission of cookies only through HTTP requests and responses.

You will need to enable SSL/TLS on Tomcat. To do so, please follow the tutorial at <https://tomcat.apache.org/tomcat-9.0-doc/ssl-howto.html>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml.
  <display-name>ServletCookieExample</display-name>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

  <filter>
    <filter-name>AuthenticationFilter</filter-name>
    <filter-class>ie.ucd.servlets.AuthenticationFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>AuthenticationFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <session-config>
    <cookie-config>
      <http-only>true</http-only>
      <secure>true</secure>
    </cookie-config>
  </session-config>

</web-app>
```

Enforces requests to be accepted only via an encrypted communication channel via SSL/TLS