Paweł Rajba
pawel@ii.uni.wroc.pl
http://kursy24.eu/

# Application Security OWASP Top 10

# Agenda

- Introduction
- Security Risk
- OWASP Top 10

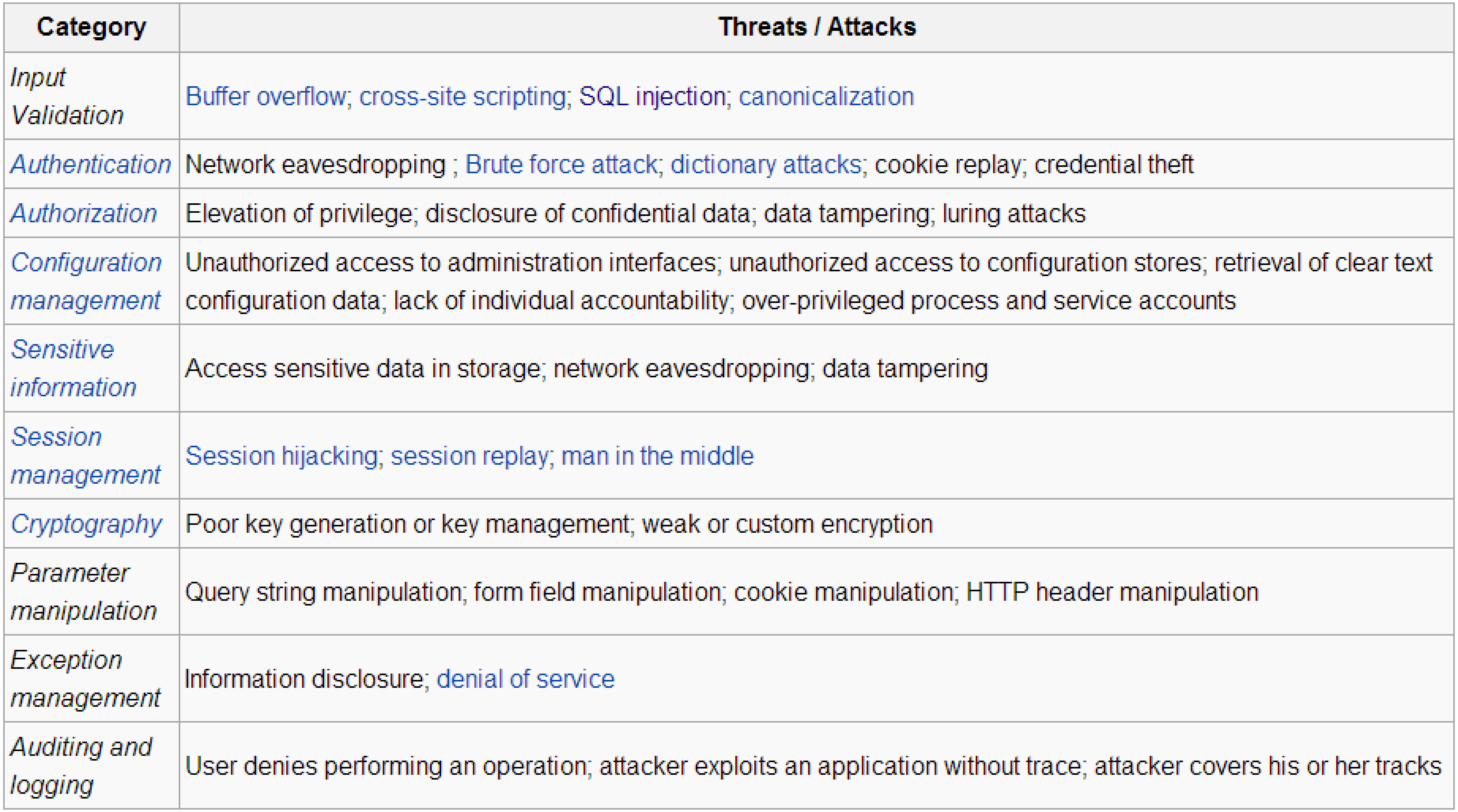
# Why bother?

- 75% of attacks happen in the Application Layer (Gartner)
- Perfectly secure environment can be compromised by a security hole in a web application
- There are numerous more difficult instructions allowing to hack a technology or a product
- Reputation is priceless
- Enterprises should assume that legal liability for poor security practices is on the horizon, and act accordingly  
(Gartner recommendation)

# Environment

- Elements of the web application environment
  - A Web server with an application
  - A Network infrastructure where the web server is placed (with all servers, firewalls, WAFs, IDSs, IPSs, ...)
  - A Web browser
  - A Communication channel (most common: HTTP & HTTPS)
- Everything outside the network infrastructure can be tampered by bad guys
  - My favourite Fiddler and breakpoints rules (F11 key)
  - Specialized tools instead of browsers
  - Scanners

# Common threats and attacks



| Category | Threats / Attacks |
|---|---|
| *Input Validation* | Buffer overflow; cross-site scripting; SQL injection; canonicalization |
| *Authentication* | Network eavesdropping ; Brute force attack; dictionary attacks; cookie replay; credential theft |
| *Authorization* | Elevation of privilege; disclosure of confidential data; data tampering; luring attacks |
| *Configuration management* | Unauthorized access to administration interfaces; unauthorized access to configuration stores; retrieval of clear text configuration data; lack of individual accountability; over-privileged process and service accounts |
| *Sensitive information* | Access sensitive data in storage; network eavesdropping; data tampering |
| *Session management* | Session hijacking; session replay; man in the middle |
| *Cryptography* | Poor key generation or key management; weak or custom encryption |
| *Parameter manipulation* | Query string manipulation; form field manipulation; cookie manipulation; HTTP header manipulation |
| *Exception management* | Information disclosure; denial of service |
| *Auditing and logging* | User denies performing an operation; attacker exploits an application without trace; attacker covers his or her tracks |

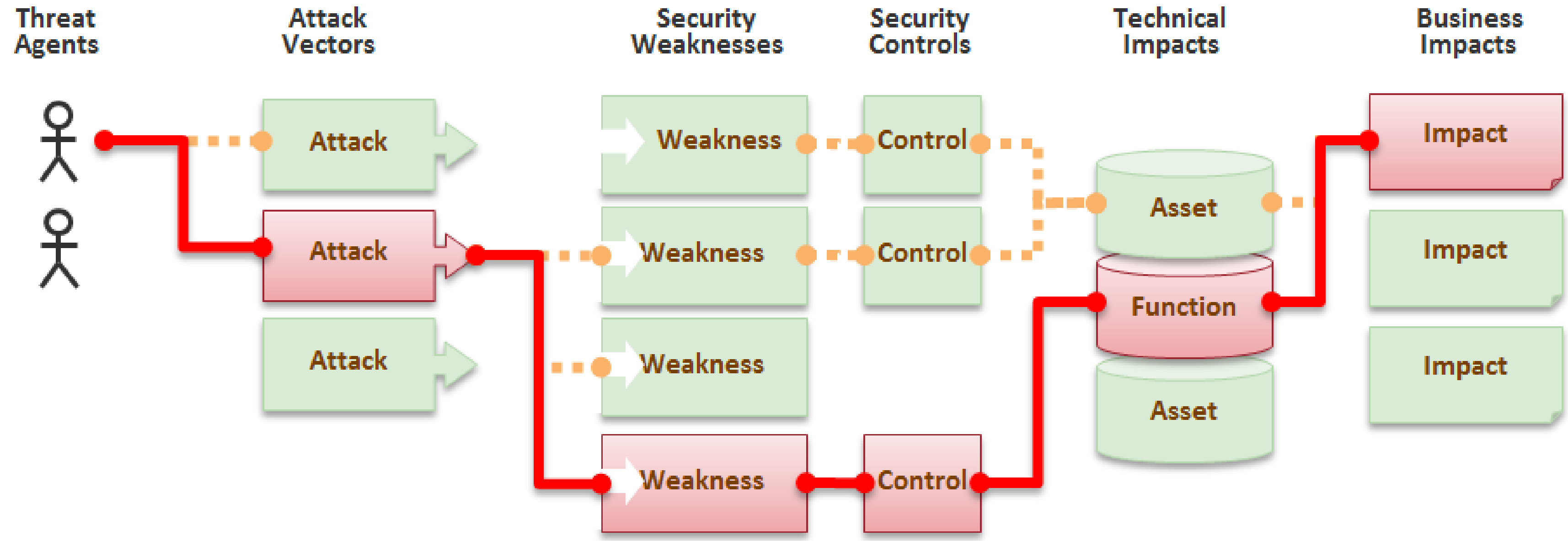
*Źródło:* http://en.wikipedia.org/wiki/Application_security

# Some initial terms

- Attack vector
  - A path or way in which a hacker can access computer system and exploit/reach a vulnerability
  - It can be one vulnerability with several attack vectors
    - More: http://searchsecurity.techtarget.com/definition/attack-vector
- Weakness Prevalence
  - How much a weakness is spread around
- Weakness detectability
  - Is it easy to find the weakness in an application?
- Technical and business impact

# Security risk

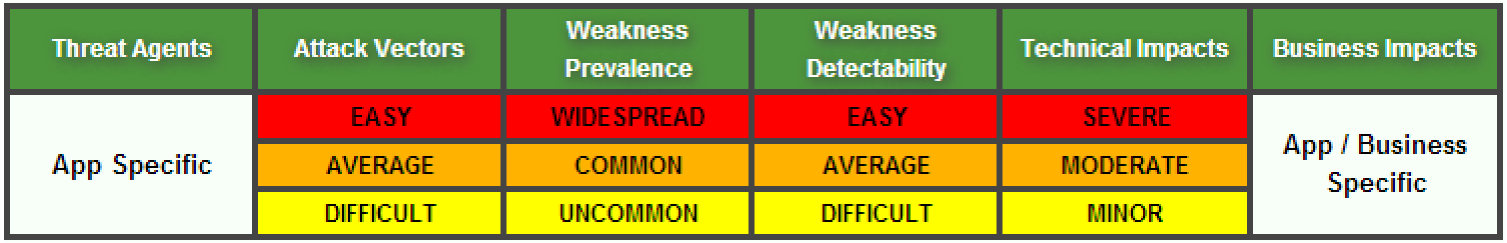
- Possible path: can be easy, can be difficult



*Źródło: https://www.owasp.org/index.php/Top_10_2013-Risk*

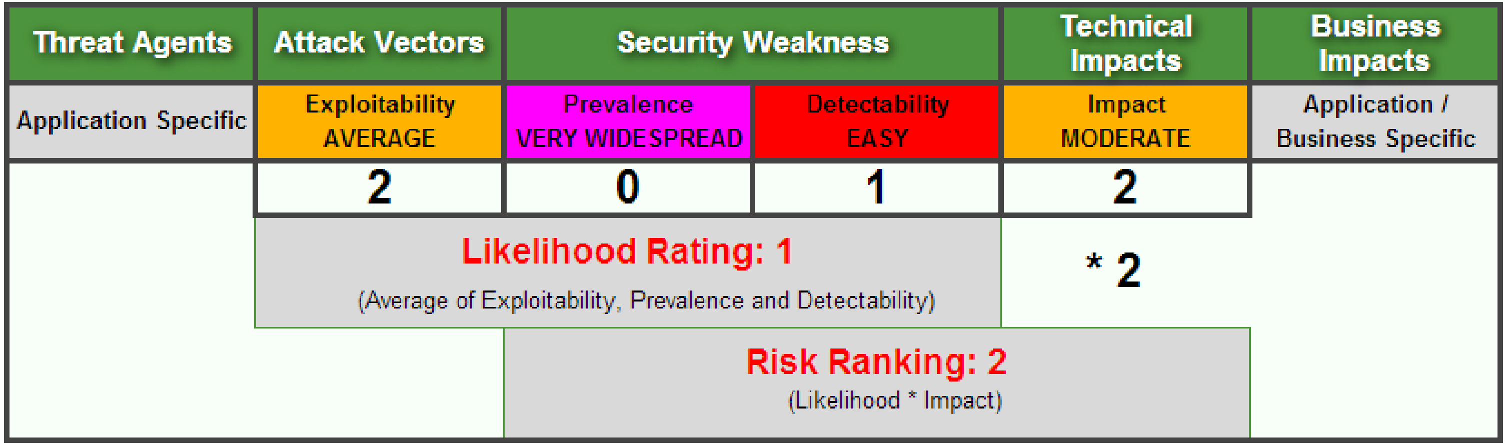
# Security Risk

- Every risk is assessed according to the following schema:



| Threat Agents | Attack Vectors | Weakness Prevalence | Weakness Detectability | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| App Specific | EASY | WIDESPREAD | EASY | SEVERE | App / Business Specific |
| | AVERAGE | COMMON | AVERAGE | MODERATE | |
| | DIFFICULT | UNCOMMON | DIFFICULT | MINOR | |

- The risk calculated as follows (an example):



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>AVERAGE | Prevalence<br>VERY WIDESPREAD | Detectability<br>EASY | Impact<br>MODERATE | Application / Business Specific |
| | 2 | 0 | 1 | 2 | |
| | Likelihood Rating: 1<br>(Average of Exploitability, Prevalence and Detectability) | | | * 2 | |
| | | Risk Ranking: 2<br>(Likelihood * Impact) | | | |

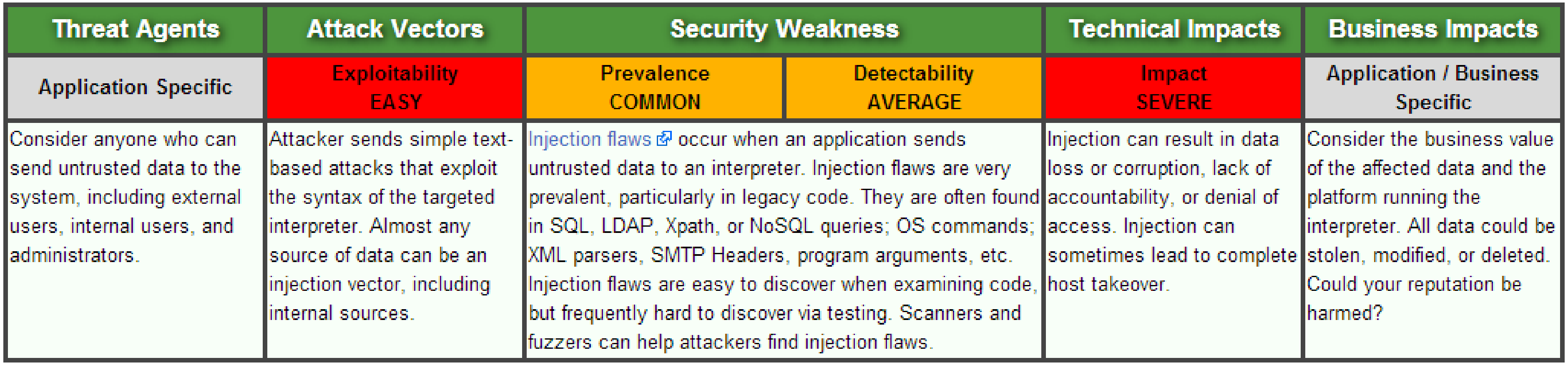
Źródła: https://www.owasp.org/index.php/Top_10_2013-Risk | https://www.owasp.org/index.php/Top_10_2013-Note_About_Risks

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Injections

- From OWASP:
  - Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>EASY | Prevalence<br>COMMON | Detectability<br>AVERAGE | Impact<br>SEVERE | Application / Business Specific |
| Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators. | Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources. | Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws. | | Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover. | Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed? |

# Injections

- Most popular: SQL injection
- Keep in mind that there are other ones:
  - LDAP injections
  - XPATH injections
  - Command injections
  - DOM injections
  - JSON injections
  - LOG spoofing
  - …

- But we will focus on SQL injections

# SQL Injection

- Occurs in the database layer of an application
- Simple sample
  - QUERY: "SELECT * FROM [Users] WHERE UserName ='"+Request["User"]+"' AND Password='"+Request["Pwd"]+"'"
  - Malicious UserName inputs:
    - ' OR 1=1; --
    - ';DROP TABLE [Users]; SELECT '1'
- Threats
  - Leak of information (at best)
  - Create, update or delete data
  - Grant access to hacker
  - Take over the OS

# SQL Injection

- A slightly dangerous attacks
  - Creating Windows account
    - SELECT * FROM [Users] WHERE UserName =''; exec master..xp_cmdshell 'net users username password /add'; --
      ' AND Password=''
  - Adding this account to the Administrators group
    - SELECT * FROM [Users] WHERE UserName =''; exec master..xp_cmdshell 'net localgroup Administrators username /add'; --
      ' AND Password=''
- Be carefull with stored procedure
  - Putting simply SQL code into stored procedure doesn’t make it secure
    - ...but can increase a security level

# SQL Injection

- Blind SQL injection
  - Occurs when application is vulnerable, but the results are not visible to the attacker
    - e.g. page doesn't display correctly, or some different content appears
  - It's a time-consuming type of attack
  - Sample scenario of such attack
    - Try to check if the site is vulnerable
      http://www.somesite.com/id=22 and 1=2
      - If something changed, e.g. an image or anything else is missing, we can continue
    - Check no. of returned columns
      http://www.somesite.com/id=22 order by n
      where instead of n you put 1,2,3,...
    - Check a vulnerable column
      http://www.somesite.com/id=-1 union select 1,2,version(),4,5
    - ... and continue guessing until you shoot something interesting

# SQL Injection

- How to protect?
  - Definitely use a parametrized statement:
    SELECT * FROM [Users] WHERE UserName = ? AND PASSWORD = ?
    - Then sp_executesql is used (in case SQLServer)
    - But some parts are not parametrized then you must be careful
  - Escape all potential risk fragments (e.g. – ; ' " \ etc.)
  - Validate input
    - Strong typing
      - e.g. id is an int – check whether it is really int
    - Business logic
      - e.g if an account no. is expected, check precisely if it's the account no.
  - Using stored procedures don't mitigate risk, but can reduce it
    - By e.g. typed parameters
  - Use the least privileges principle, i.e. run a query in a context of a user with least privileges needed to perform a specified action

# SQL Injection

- DEMO
- Let’s fastly review
  - http://www.securiteam.com/securityreviews/5DP0N1P76E.html
  - http://weblogs.sqlteam.com/mladenp/archive/2011/02/16/sql-server-sql-injection-from-start-to-end.aspx
  - http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
  - http://blogs.msdn.com/b/raulga/archive/2007/01/04/dynamic-sql-sql-injection.aspx
  - http://www.breakthesecurity.com/2010/12/hacking-website-using-sql-injection.html
  - https://www.owasp.org/index.php/Blind_SQL_Injection
  - http://www.breakthesecurity.com/2010/12/hacking-website-using-sql-injection.html

# References

- SQL Injection
  - http://www.unixwiz.net/techtips/sql-injection.html
  - http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
  - http://weblogs.sqlteam.com/mladenp/archive/2011/02/16/sql-server-sql-injection-from-start-to-end.aspx
  - http://www.securiteam.com/securityreviews/5DP0N1P76E.html
  - http://blogs.msdn.com/b/raulga/archive/2007/01/04/dynamic-sql-sql-injection.aspx
  - http://www.securiteam.com/securityreviews/5DP0N1P76E.html
  - http://download.oracle.com/oll/tutorials/SQLInjection/index.htm
  - http://www.sommarskog.se/dynamic_sql.html
  - http://msdn.microsoft.com/en-us/library/cc716760.aspx
- Blind SQL Injection sample scenarios
  - https://www.owasp.org/index.php/Blind_SQL_Injection
  - http://www.breakthesecurity.com/2010/12/hacking-website-using-sql-injection.html
  - http://forum.internot.org/web-hacking-war-games/818-blind-sql-injection.html

# XPath & LDAP Injection

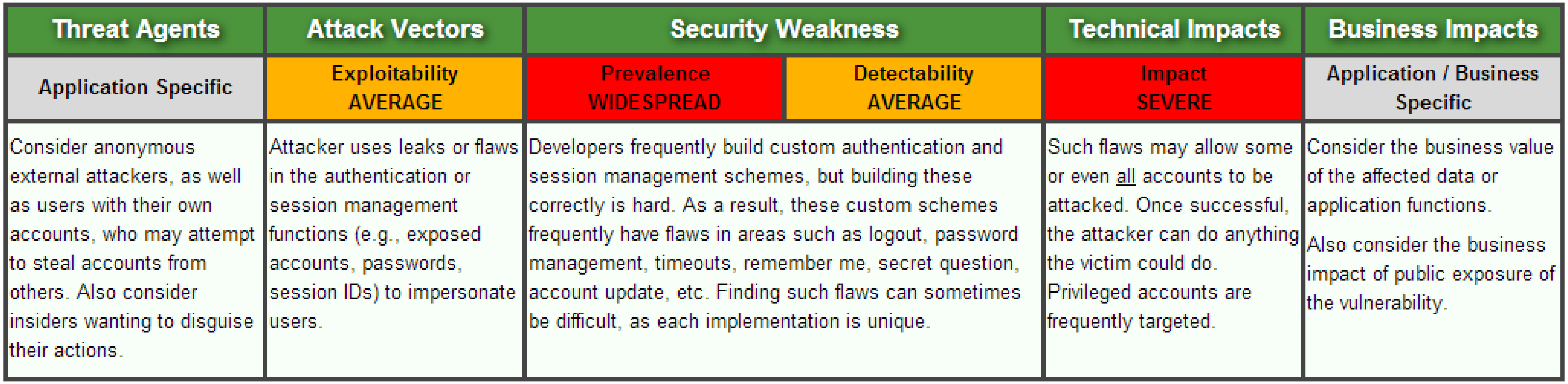
- Short review
  - https://www.owasp.org/index.php/XPATH_Injection
  - https://www.owasp.org/index.php/LDAP_injection

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Broken Authentication and Session Management

- From OWASP:
  - Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>AVERAGE | Prevalence<br>WIDESPREAD | Detectability<br>AVERAGE | Impact<br>SEVERE | Application / Business Specific |
| Consider anonymous external attackers, as well as users with their own accounts, who may attempt to steal accounts from others. Also consider insiders wanting to disguise their actions. | Attacker uses leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to impersonate users. | Developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc. Finding such flaws can sometimes be difficult, as each implementation is unique. | | Such flaws may allow some or even all accounts to be attacked. Once successful, the attacker can do anything the victim could do. Privileged accounts are frequently targeted. | Consider the business value of the affected data or application functions.<br>Also consider the business impact of public exposure of the vulnerability. |

# Broken Authentication and Session Management

- Where are the threats?
  - User credentials aren’t protected enough and can be stolen
    - e.g. SQL Injection, guessing or just brute force attack, **keyloggers**
  - Credentials can be guessed or overwritten through weak account management functions
    - e.g., account creation, change password, **recover password**
  - Session IDs
    - are exposed in the URL (e.g., URL rewriting).
    - are vulnerable to session fixation attacks.
    - don’t timeout, or user sessions or authentication tokens, particularly single sign-on (SSO) tokens, aren’t properly invalidated during logout.
    - aren’t rotated after successful login.
  - Passwords, session IDs, and other credentials are sent over unencrypted connections (man-in-the-middle attack)

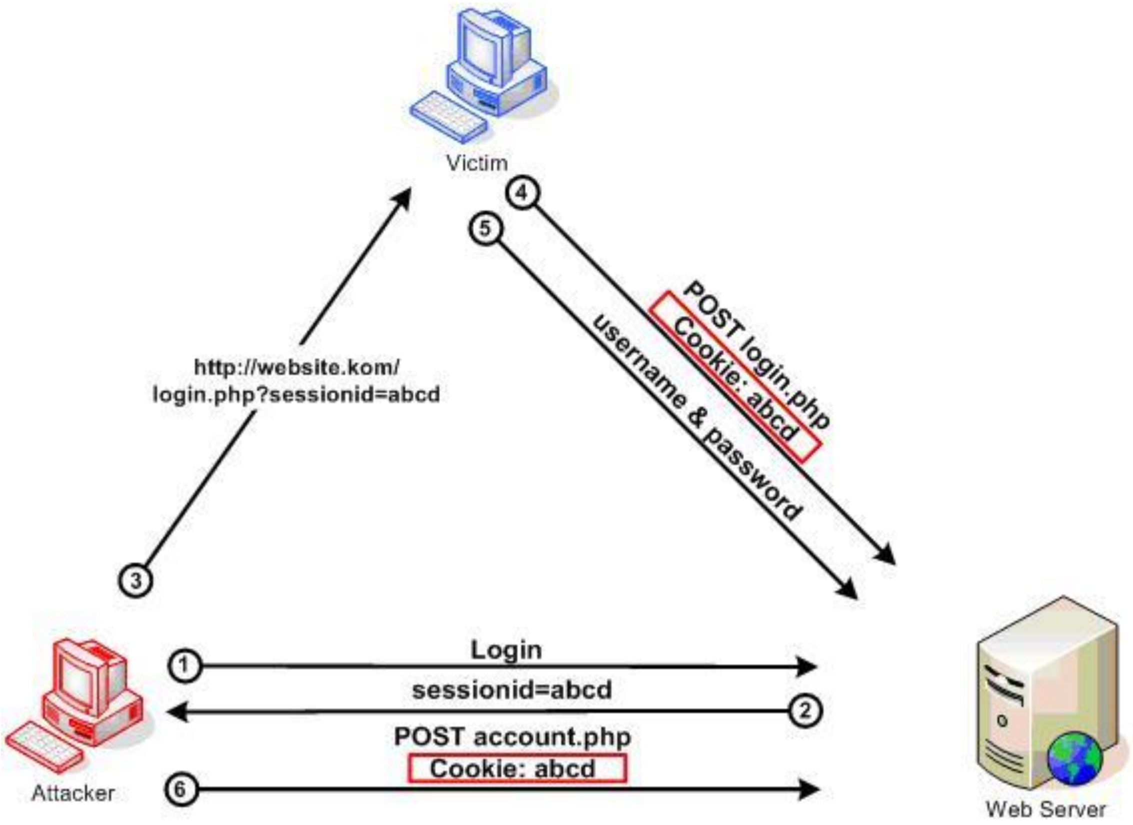
*Źródło:* *https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management*

# Broken Authentication and Session Management

- How to protect ourselves?
  - Credentials should be stored using hashing or encryption
    - Also credentials to databases
    - Consider using „slow" function (e.g. PBKDF2) – prevents brute force
    - Mandatory using salt in hashes – prevent rainbow tables attack
  - Credentials shouldn't be easy to guess or overwrite
    - Some policies should be applied
    - Brute force (e.g. weak passwords)
  - Be carefull with password recovery – some approaches can be risky
  - TLS should be used to send sensitive information (including credentials and session IDs)
    - Ensure certificate is valid
  - Consider different approach than user/password
    - PKI, Fingerprint
    - Digipass, one-time codes
    - Masked password (a little bit controversial)
    - Multi-factor authentication
  - Session management should be done carefully
    - Not in URL,rotation of session's Ids, using good timeouts and proper logout
- Sometimes it's difficult to protect
  - e.g. keyloggers, but a display keyboard can be solution…
    … O RLY? Who will use it??

# Broken Authentication and Session Management

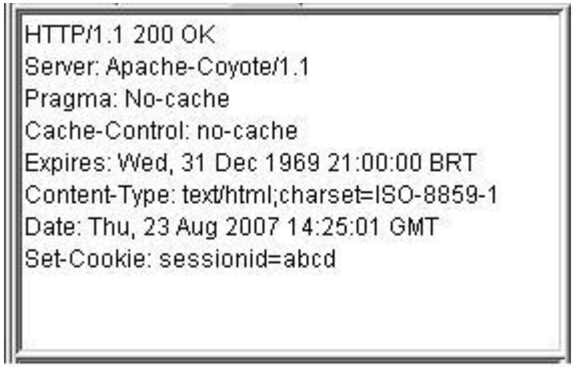
- Session fixation



*Źródło: https://www.owasp.org/index.php/Session_fixation*

# Broken Authentication and Session Management

- Session fixation
  - Other similar possibilities
    - XSS on client side
      - http://website.kom/<script>document.cookie=”sessionid=abcd”;</script>
    - XSS on DOM
      - http://website.kon/<meta http-equiv=Set-Cookie content=”sessionid=abcd”>
    - Insert HTTP header



```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Pragma: No-cache
Cache-Control: no-cache
Expires: Wed, 31 Dec 1969 21:00:00 BRT
Content-Type: text/html;charset=ISO-8859-1
Date: Thu, 23 Aug 2007 14:25:01 GMT
Set-Cookie: sessionid=abcd
```

*Źródło: https://www.owasp.org/index.php/Session_fixation*

# References

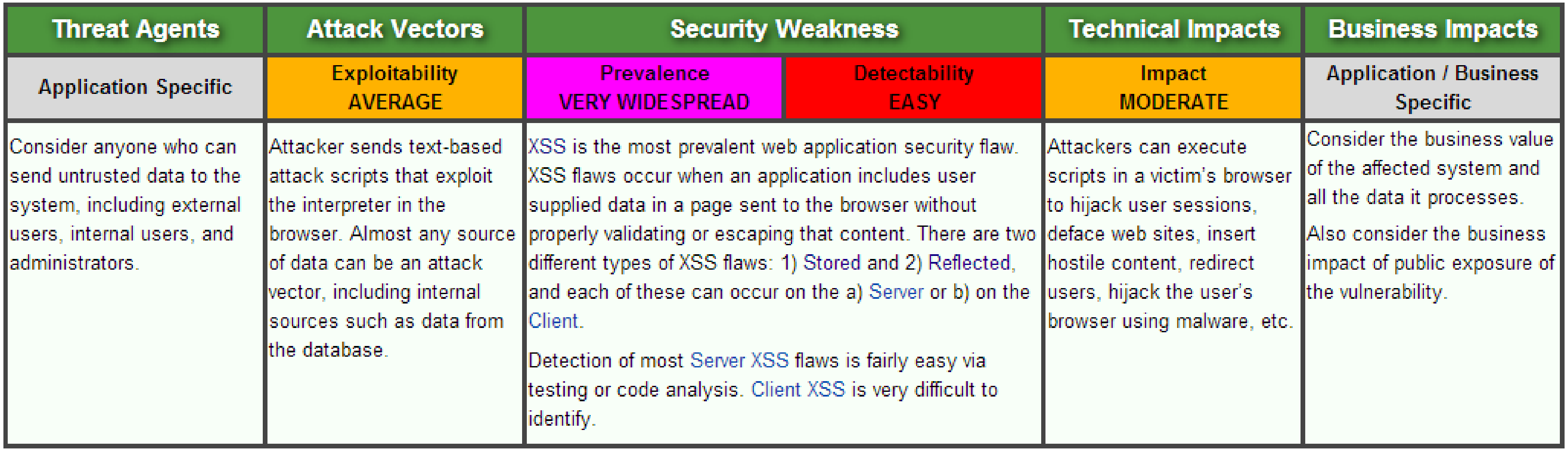
- Session fixation
  - https://www.owasp.org/index.php/Session_fixation
  - http://en.wikipedia.org/wiki/Session_fixation
  - https://www.owasp.org/index.php/Session_Management_Cheat_Sheet
  - http://software-security.sans.org/blog/2009/06/14/session-attacks-and-aspnet-part-1/
  - http://software-security.sans.org/blog/2009/06/24/session-attacks-and-aspnet-part-2

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Cross-Site Scripting (XSS)

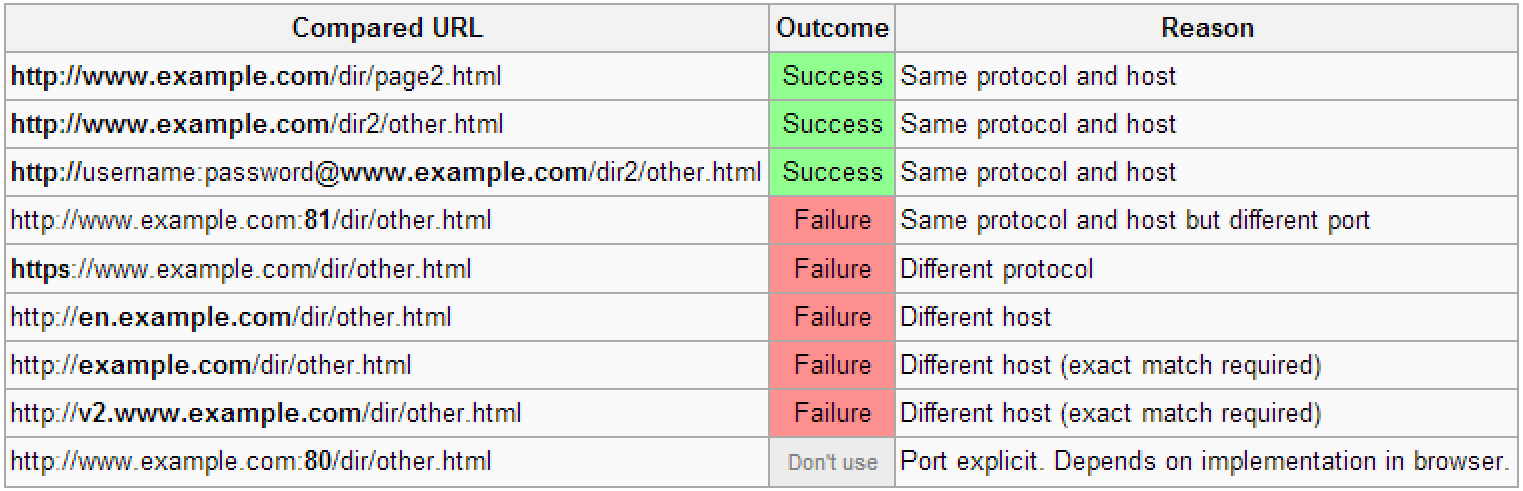
- From OWASP
  - XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>AVERAGE | Prevalence<br>VERY WIDESPREAD | Detectability<br>EASY | Impact<br>MODERATE | Application / Business Specific |
| Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators. | Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database. | XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are two different types of XSS flaws: 1) Stored and 2) Reflected, and each of these can occur on the a) Server or b) on the Client.<br>Detection of most Server XSS flaws is fairly easy via testing or code analysis. Client XSS is very difficult to identify. | | Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc. | Consider the business value of the affected system and all the data it processes.<br>Also consider the business impact of public exposure of the vulnerability. |

# Cross-Site Scripting (XSS)

- Same Origin Policy
  - Define a rule where only requests to the same „site” are allowed
  - It’s applied mostly to JavaScript code



| Compared URL | Outcome | Reason |
|---|---|---|
| **http://www.example.com**/dir/page2.html | Success | Same protocol and host |
| **http://www.example.com**/dir2/other.html | Success | Same protocol and host |
| **http**://username:password@**www.example.com**/dir2/other.html | Success | Same protocol and host |
| http://www.example.com:**81**/dir/other.html | Failure | Same protocol and host but different port |
| **https**://www.example.com/dir/other.html | Failure | Different protocol |
| http://**en.example.com**/dir/other.html | Failure | Different host |
| http://**example.com**/dir/other.html | Failure | Different host (exact match required) |
| http://**v2.www.example.com**/dir/other.html | Failure | Different host (exact match required) |
| http://www.example.com:**80**/dir/other.html | Don't use | Port explicit. Depends on implementation in browser. |

# Cross-Site Scripting (XSS)

- Relaxing SOP
  - Cross-origin resource sharing
    - Let’s review
      - http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
  - Web Messaging
    - Let’s review
      - http://en.wikipedia.org/wiki/Cross-document_messaging

# Cross-Site Scripting (XSS)

- XSS stands for cross site scripting
- Attack on a the client side (web browser)
  - Force a web browser to execute an unwanted JavaScript code
- There are 3 main types of attacks
  - DOM based XSS (type 0)
    - The code is executed as a result of dynamic modification of a DOM by another JavaScript code
  - Reflected (type1)
    - The code is attachted somehow to a link and the user is tricked to follow this link
  - Stored or persistent (type 2)
    - The code is stored in a database and shown to a user while a web page is opening

# Cross-Site Scripting (XSS)

- Classic scenarios
  - Someone is tricked to follow a link with malicious code attached (type 0 and 1)
  - Hacker adds a malicious code in a blog comment
    - Everyone who opens an article, executes malicious code

# Cross-Site Scripting (XSS)

- What the attacker can do using XSS hole
  - Steal your cookies (sometimes they can be really delicious ☺)
  - Intercept your login and password
  - Generally embed any malware
  - XSS is a start point for other attacks (e.g. CSRF)
- How can we protect ourselves?
  - Escape everything sent to a browser
    - Sometimes we want to sent an unescaped stuff – it should be then under full control
  - Use frameworks or tools to do it automatically
    - In this case a good knowledge of the tools is important
  - Validate any input provided by an user
    - Use whitelists instead of blacklists
  - Use httpOnly
  - XSS regarding the ASP.NET
    - http://msdn.microsoft.com/en-us/library/ff649310.aspx
  - Other good resources:
    - https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
    - https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

# Cross-Site Scripting (XSS)

- But how information can be stolen?
  - Requests are limited by the same origin policy and alerts don't look dangerous…
  - … but we can use img and iframe tags
  - … and there are many other techniques
- Surprising how many big companies has been xssed
  - Nice web site: [http://xssed.com/](http://xssed.com/)

# Cross-Site Scripting (XSS)

- DEMO: XSS0, XSS1

# References

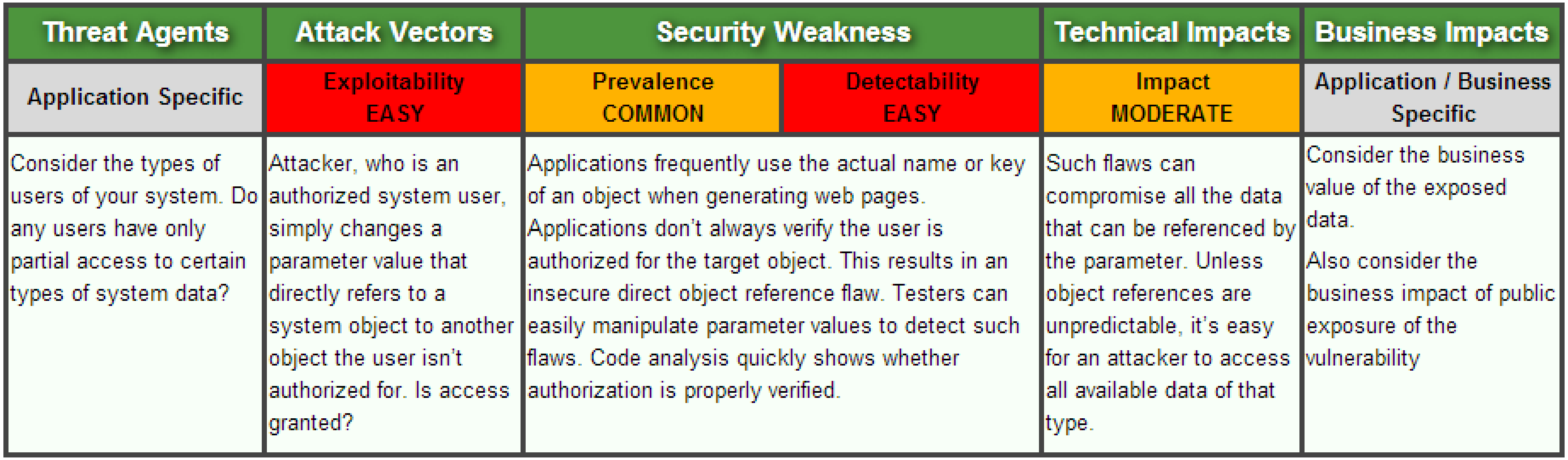
- SOP, CORS, Web Messaging
  - http://en.wikipedia.org/wiki/Same-origin_policy
  - http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
  - http://en.wikipedia.org/wiki/Cross-document_messaging
- XSS
  - https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
  - https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
  - http://projects.webappsec.org/w/page/13246920/Cross%20Site%20Scripting
  - http://excess-xss.com/
  - http://www.webappsec.org/projects/articles/071105.shtml
  - https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
  - https://www.acunetix.com/websitesecurity/xss/
  - http://www.isaca.org/chapters5/Venice/Events/Documents/ISACAVENICE-OWASP-UNIVE-2013-1%20-%20DiPaola.pdf
  - http://www.thegeekstuff.com/2012/02/xss-attack-examples/
  - https://addons.mozilla.org/en-US/firefox/addon/xss-me/
- Browser Security Handbook
  - https://code.google.com/p/browsersec/wiki/Main

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Insecure Direct Object References

- From OWASP
  - A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>EASY | Prevalence<br>COMMON | Detectability<br>EASY | Impact<br>MODERATE | Application / Business Specific |
| Consider the types of users of your system. Do any users have only partial access to certain types of system data? | Attacker, who is an authorized system user, simply changes a parameter value that directly refers to a system object to another object the user isn't authorized for. Is access granted? | Applications frequently use the actual name or key of an object when generating web pages. Applications don't always verify the user is authorized for the target object. This results in an insecure direct object reference flaw. Testers can easily manipulate parameter values to detect such flaws. Code analysis quickly shows whether authorization is properly verified. | | Such flaws can compromise all the data that can be referenced by the parameter. Unless object references are unpredictable, it's easy for an attacker to access all available data of that type. | Consider the business value of the exposed data.<br>Also consider the business impact of public exposure of the vulnerability |

# Insecure Direct Object References

- Occurs when one can get access to an object which shouldn’t be available to the user
  - e.g. http://site.com/accountInfo?accno=**not_my_account_no**
- In order to perform this attack, a family of web parameter tampering techniques can be used
  - Query string tampering
  - Form’s hidden field tampering
  - Cookie tampering
- Some general guidelines
  - Check if all resources are protected well enough and if there is always an authorization when needed
  - Usually automatic tests are not sufficient because they can’t recognize if sth should be available to the user or not
  - So, security code review is the better approach

# Insecure Direct Object References

- Query string tampering
  - Simple sample
    - Link
      http://host/product/1332/view
      can be replaced by
      http://host/product/1335/view
    - Next, attacker can try
      http://host/product/1332/delete
  - Mitigations
    - Nothing – just perform a proper authorization on the server side
    - Use POST instead of GET
    - Protect link from change
      - Encrypting the link
      - Singing the link by attaching some salted hashcode
        - Usability can be poor
        - One cannot send link to a friend
        - Not SEO friendly

# Insecure Direct Object References

- Form’s hidden fields tampering
  - Simple sample
    <input type="hidden " id= "12345” name= "balance" value= "1200" />
  - Mitigations
    - Rebuild the app and don’t store such information on client side
    - Use encryption
      - It’s safe
      - Doesn’t decrease user experience
- Cookie tampering
  - Analogous as in the point above
  - There are 3 main problems:
    - Cookie Theft
    - Cookie Poisoning
    - Cross Site Cooking

# References

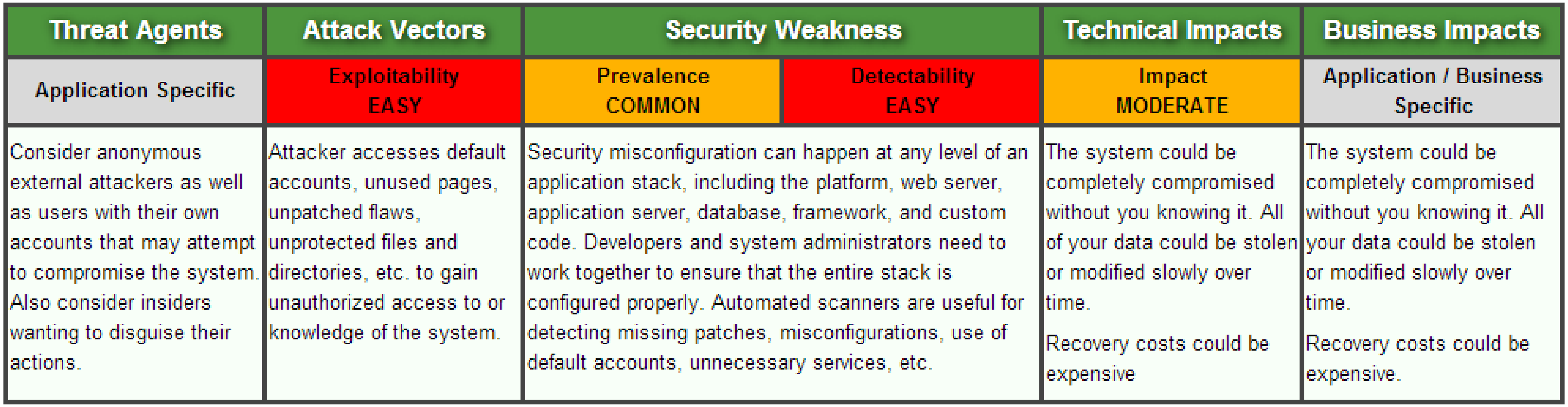
- https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References
- http://www.cisodesk.com/web-application-security/threats-mitigation/insecure-direct-object-references/
- http://cwe.mitre.org/data/definitions/22.html

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Security Misconfiguration

- From OWASP
  - Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>EASY | Prevalence<br>COMMON | Detectability<br>EASY | Impact<br>MODERATE | Application / Business Specific |
| Consider anonymous external attackers as well as users with their own accounts that may attempt to compromise the system. Also consider insiders wanting to disguise their actions. | Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system. | Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code. Developers and system administrators need to work together to ensure that the entire stack is configured properly. Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc. | | The system could be completely compromised without you knowing it. All of your data could be stolen or modified slowly over time.<br>Recovery costs could be expensive | The system could be completely compromised without you knowing it. All your data could be stolen or modified slowly over time.<br>Recovery costs could be expensive. |

# Security Misconfiguration

- Occurs when some backdoor is available through the configuration
  - e.g. default settings, default accounts
- It's related to another type of hole: information leakage and improper error handling
- Google can index details about DB
  - e.g. full connection string with password
- Rule of thumb: any information exposed to the hacker can be used to hack the site

# Security Misconfiguration

- How to protect, part 1
  - Don’t expose information about your system
    - Especially turn off directory browsing
  - Remove passwords from source code
  - Delete unused user accounts and pages
  - Turn off unused services
  - Messages from database or application should be as minimal as possible
    - Notice that these information can be indexed by Google!
    - Also can be stored in logs

# Security Misconfiguration

- How to protect, part 2
  - By default, many systems have a bad configuration, e.g.
    - Access logs are available to public
    - Directory listing is enabled
  - Be careful with robots.txt – first file used by hackers, sometimes it's better to use access control
  - Be up-to-date with patches
  - Assume internal attacks
    - Although web.config isn't available to browsers, it can be read by employees
  - Perform a network penetration test (or other tests) and harden a server

# References

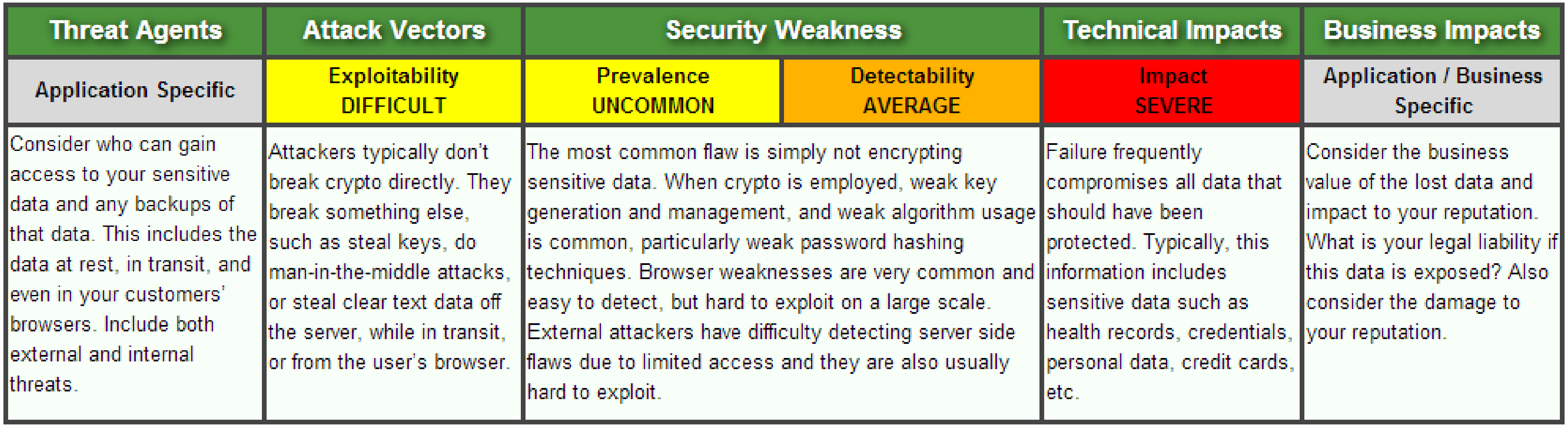
- https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration
- http://cwe.mitre.org/data/definitions/2.html
- http://msdn.microsoft.com/en-us/library/dtkwfdky.aspx

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Sensitive Data Exposure

- From OWASP:
  - Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>DIFFICULT | Prevalence<br>UNCOMMON | Detectability<br>AVERAGE | Impact<br>SEVERE | Application / Business Specific |
| Consider who can gain access to your sensitive data and any backups of that data. This includes the data at rest, in transit, and even in your customers' browsers. Include both external and internal threats. | Attackers typically don't break crypto directly. They break something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's browser. | The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server side flaws due to limited access and they are also usually hard to exploit. | | Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive data such as health records, credentials, personal data, credit cards, etc. | Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also consider the damage to your reputation. |

# Sensitive Data Exposure

- There are two main steps to mitigate a risk:
  - Make a data classification
  - Apply appropriate level of protection for every class of data
- Ways for gathering information
  - Exploring the network
  - Stealing computers and media
  - Breaking into computers and stealing the data
    - Also using stolen passwords
  - Eavesdroping and phishing network and emails
  - Social engineering

# Sensitive Data Exposure

- How to protect? (part 1 – data)
  - Appropriate strong encryption mechanisms are used
    - Use AES, Blowfish, 3DES
    - Use SHA-256, 512 instead of MD5
    - E.g. salted hashes vs. not salted hashes (3000 years vs. 4 weeks)
  - Decryption is available to the authorized users only
  - There are appropriate procedures
    - E.g. a decryption key should be stored in a different place than the encrypted data ☺
  - Again, encrypt the web.config

# Sensitive Data Exposure

- How to protect? (part 2 – communication)
  - Of course the SSL should be turned on, but it is important to ensure that a SSL is needed (because it costs)
  - If the SSL is used, all resources on the page should be requested using SSL
  - A SSL certificate has to be valid
  - In some cases some local certificate authority should be maintained with whole environment configuration
    - e.g. appropriate certificates should be added to each station
  - In some cases also client certificates are needed
  - And finally there is not only traffic between the browser and the web server, but also
    - SQL servers
    - Web Services
    - ...

# References

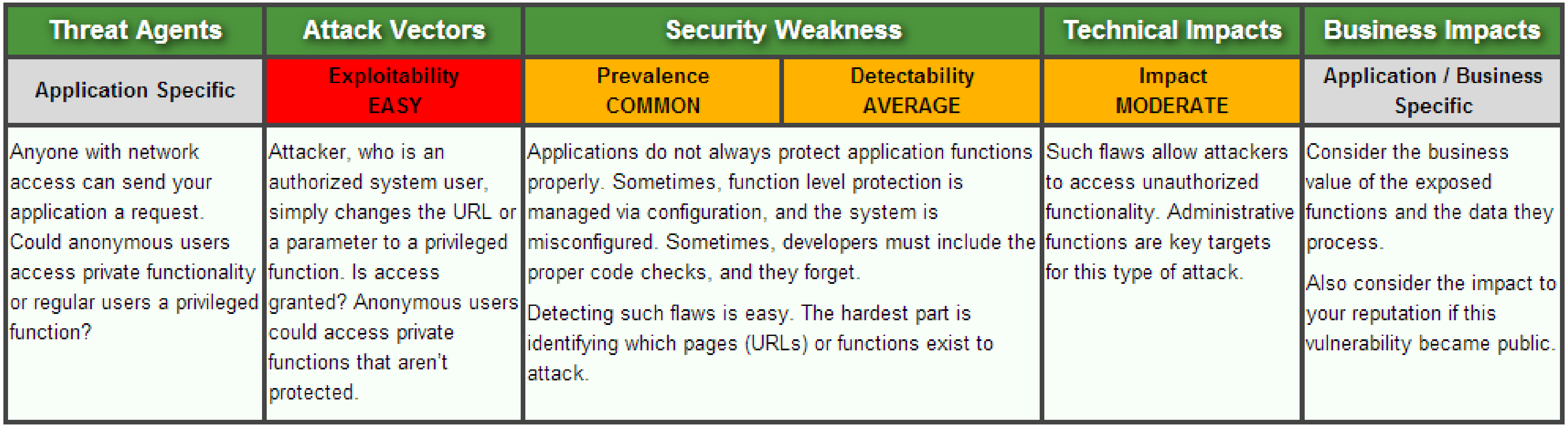
- https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Missing Function Level Access Control

- From OWASP
  - Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>EASY | Prevalence<br>COMMON | Detectability<br>AVERAGE | Impact<br>MODERATE | Application / Business Specific |
| Anyone with network access can send your application a request. Could anonymous users access private functionality or regular users a privileged function? | Attacker, who is an authorized system user, simply changes the URL or a parameter to a privileged function. Is access granted? Anonymous users could access private functions that aren't protected. | Applications do not always protect application functions properly. Sometimes, function level protection is managed via configuration, and the system is misconfigured. Sometimes, developers must include the proper code checks, and they forget.<br>Detecting such flaws is easy. The hardest part is identifying which pages (URLs) or functions exist to attack. | | Such flaws allow attackers to access unauthorized functionality. Administrative functions are key targets for this type of attack. | Consider the business value of the exposed functions and the data they process.<br>Also consider the impact to your reputation if this vulnerability became public. |

# Missing Function Level Access Control

- This is tightly related to proper authorization
- We can consider 3 main areas:
  - As stated in OWASP desc., access control only in UI, but not repeated in server layer
  - Privileges elevation: one accessed a system as a common user, but is able to perform admin oper.
  - Don't forget about files like PDF, DOC, etc.
- It is important to create good security architecture with e.g.
  - RBAC (role based access control),
  - SRP (single responsibility principle),
  - LPP (least privileges principle)
- It is similar to A4, but here we have missing access control, but in A4 it is about parameters

# References

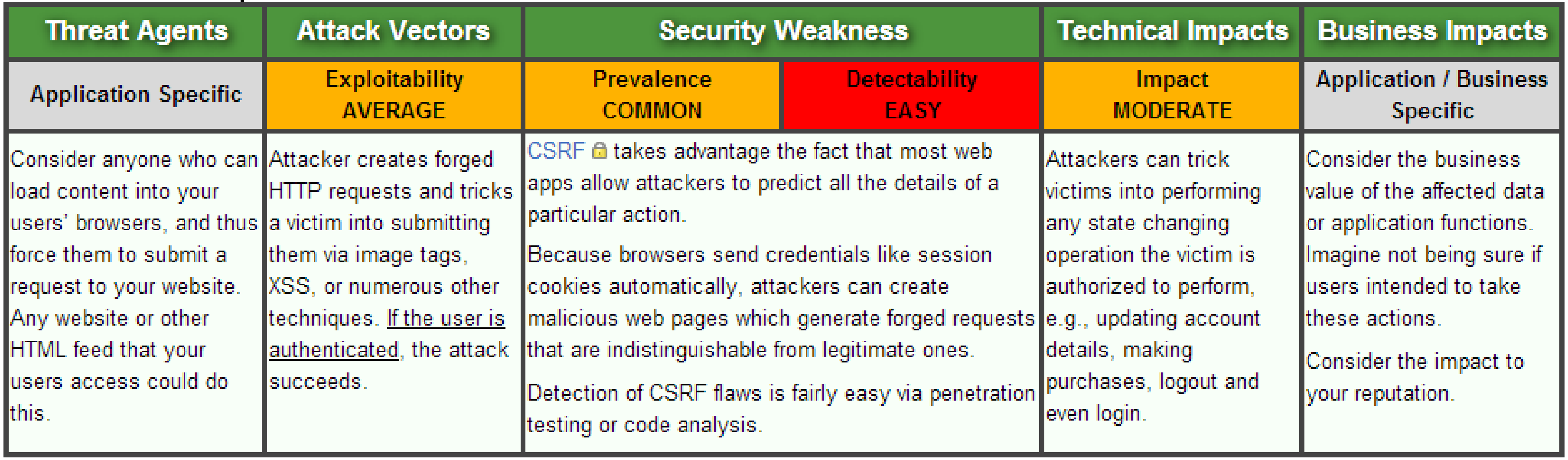
- https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control
- https://www.owasp.org/index.php/Guide_to_Authorization
- http://lists.owasp.org/pipermail/owasp-topten/2010-August/000694.html

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Cross-Site Request Forgery (CSRF)

- From OWASP
  - A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>AVERAGE | Prevalence<br>COMMON | Detectability<br>EASY | Impact<br>MODERATE | Application / Business Specific |
| Consider anyone who can load content into your users' browsers, and thus force them to submit a request to your website. Any website or other HTML feed that your users access could do this. | Attacker creates forged HTTP requests and tricks a victim into submitting them via image tags, XSS, or numerous other techniques. If the user is authenticated, the attack succeeds. | CSRF takes advantage the fact that most web apps allow attackers to predict all the details of a particular action.<br>Because browsers send credentials like session cookies automatically, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones.<br>Detection of CSRF flaws is fairly easy via penetration testing or code analysis. | | Attackers can trick victims into performing any state changing operation the victim is authorized to perform, e.g., updating account details, making purchases, logout and even login. | Consider the business value of the affected data or application functions. Imagine not being sure if users intended to take these actions.<br>Consider the impact to your reputation. |

# Cross-Site Request Forgery (CSRF)

- CSRF or XSRF: cross site request forgery
- It’s an attack on the server side of the web application
- Let’s see a sample scenario..

# Cross-Site Request Forgery (CSRF)

- Let’s see sample scenario
  - There is a bank called: bank
    - URL: http://www.bank.com/
  - This bank has a site to transfer money:
    - http://www.bank.com/transfer?toaccount=123&amount=1000
  - Alice logged into the bank site
  - Then Eve sent to Alice an email with the following link:
    - <a href="http://bit.ly/veryfunny">Very happy rabbit with big eggs</a>
    - As you can guess, http://bit.ly/veryfunny resolves to http://www.bank.com/transfer?toaccount=123&amount=1000
  - Alice is stupid and clicks the link
  - And due to Alice is still logged into the bank site, Eve becomes happy

# Cross-Site Request Forgery (CSRF)

- In general: attack occurs when a user unknowingly perform an action
  - Important remark: user has to be authorized
- Sometimes very difficult to track
  - Everything is performed in the context of authorized user, action and IP, everything is correct, oh, beside that use doesn’t know anything
- Usually combined with the XSS attack

# Cross-Site Request Forgery (CSRF)

- How to force the user to perform what the attacker wants?
  - Send him a link with a request which performs what the attacker wants
    - <a href="http://bank/transfer/no=123&amount=1000">Very funny</a>
  - Using a XSS embed a malicious code on some pages (e.g. a forum or anything else)
    - <img src="http://bank/transfer/no=123&amount=1000" alt="Very funny"/>

# Cross-Site Request Forgery (CSRF)

- So, if we force to use POST we are safe, are we?

# Cross-Site Request Forgery (CSRF)

- As you can guess, it’s not so easy
  - It is better to use POST instead of GET, but it doesn’t protect, because one can embed the following code (e.g. into iframe):

    -
      ```
      <div style="display:none">
      <form action="http://app.com/delete"
      method="POST">
      <input type="hidden" name= "id" value=
      "1"></form></div>
      <script>document.forms[0].submit()</script>
      ```

# Cross-Site Request Forgery (CSRF)

- How to protect?
  - Require a confirmation page before executing a potentially dangerous action
  - Require a reauthentication
    - E.g. in allegro.pl you have to authenticate again before doing sth sensitive
  - Use POST instead of GET although it doesn't mitigate fully
  - Add token to a form
    - Yahoo calls it crumb

- Crumb
  - Should be unique per user
    - Best approach: should be changed with every request
  - Remember, if there is XSS hole, crumb can be stolen!

# Cross-Site Request Forgery (CSRF)

- POST vs. GET
  - GET
    - One can achieve better user experience
    - The amount of data is limited to ca. 2KB
    - On the other hand, all information are shown explicitly in the URL
    - Don't use if any sensitive information is sent
      - e.g. user/password, session ID, …
    - All requests are stored in logs
    - … and in a browser's history
  - POST
    - Use when sensitive data is sent
    - Use when large amount of data should be sent
    - Helps prevent duplicate submission
    - Attacks are harder due to the fact that malicious script can't be injected directly
      - but they are not impossible!
    - Most of search engines don't crawl POST forms
    - Prevents unintentional actions

# References

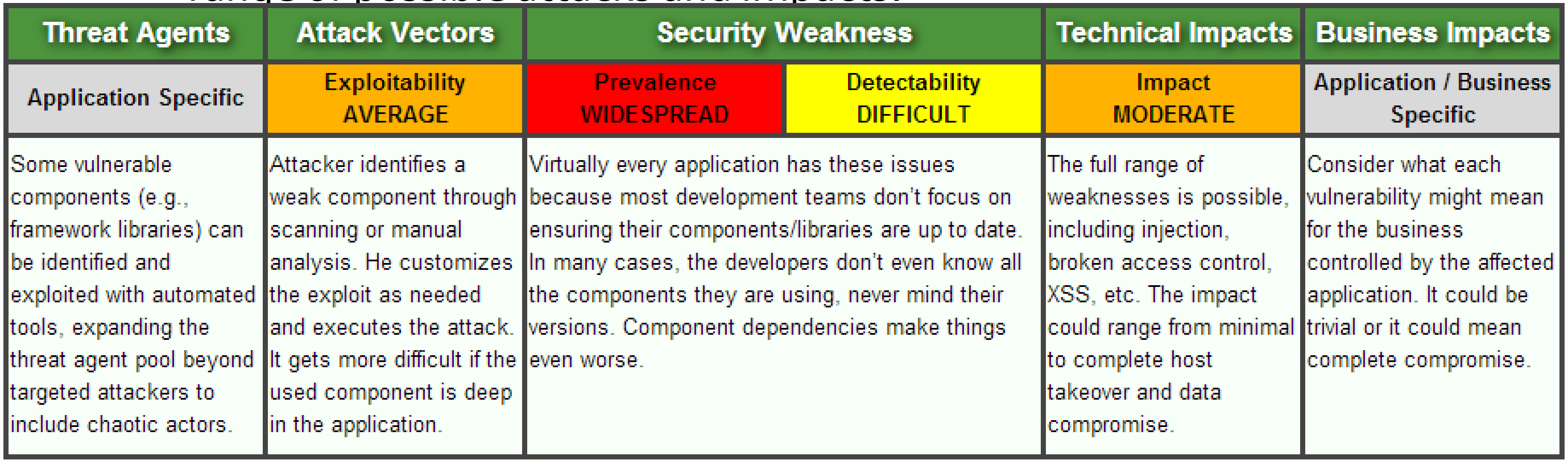
- https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_(CSRF)

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Using Components with Known Vulnerabilities

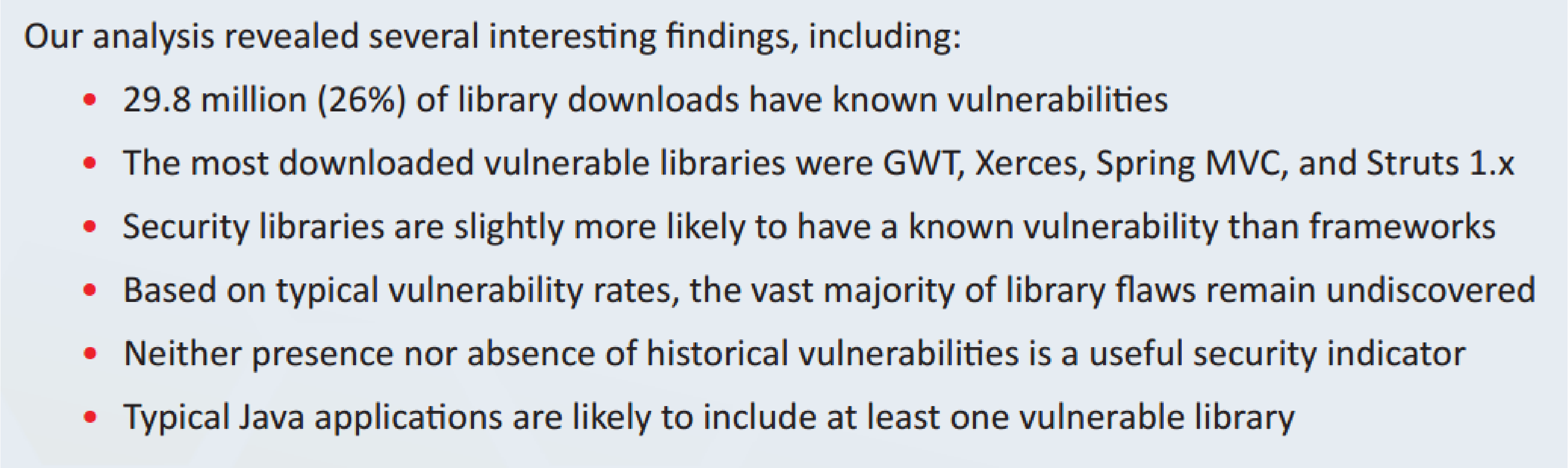
- From OWASP
  - Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>AVERAGE | Prevalence<br>WIDESPREAD | Detectability<br>DIFFICULT | Impact<br>MODERATE | Application / Business Specific |
| Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools, expanding the threat agent pool beyond targeted attackers to include chaotic actors. | Attacker identifies a weak component through scanning or manual analysis. He customizes the exploit as needed and executes the attack. It gets more difficult if the used component is deep in the application. | Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date. In many cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse. | | The full range of weaknesses is possible, including injection, broken access control, XSS, etc. The impact could range from minimal to complete host takeover and data compromise. | Consider what each vulnerability might mean for the business controlled by the affected application. It could be trivial or it could mean complete compromise. |

# Using Components with Known Vulnerabilities

- From Aspect Security report



Our analysis revealed several interesting findings, including:

- 29.8 million (26%) of library downloads have known vulnerabilities
- The most downloaded vulnerable libraries were GWT, Xerces, Spring MVC, and Struts 1.x
- Security libraries are slightly more likely to have a known vulnerability than frameworks
- Based on typical vulnerability rates, the vast majority of library flaws remain undiscovered
- Neither presence nor absence of historical vulnerabilities is a useful security indicator
- Typical Java applications are likely to include at least one vulnerable library

*Źródło: https://www.aspectsecurity.com/uploads/downloads/2012/03/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf*

# Using Components with Known Vulnerabilities

- How to protect?
  - Monitor security of components used in solution
    - e.g. in forums, security press
  - Establish policy related to component choice
    - e.g. required vendor to implement ISO certification, ITIL, SDLC practices, passing security tests,…
  - Keep versions updated
  - Use software from trusted vendors, if possible

# Using Components with Known Vulnerabilities

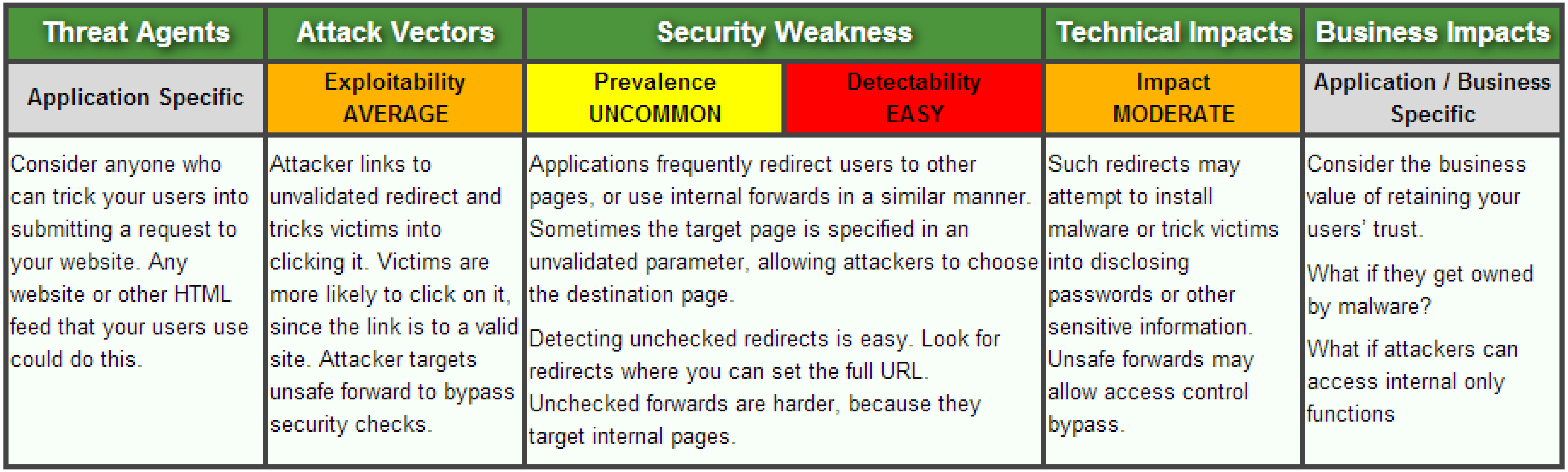
- https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities
- https://www.aspectsecurity.com/uploads/downloads/2012/03/Aspect-Security-The-Unfortunate-Reality-of-Insecure-Libraries.pdf

# OWASP TOP 10

- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- A10: Unvalidated Redirects and Forwards

# Unvalidated Redirects and Forwards

- From OWASP
  - Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>AVERAGE | Prevalence<br>UNCOMMON | Detectability<br>EASY | Impact<br>MODERATE | Application / Business Specific |
| Consider anyone who can trick your users into submitting a request to your website. Any website or other HTML feed that your users use could do this. | Attacker links to unvalidated redirect and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site. Attacker targets unsafe forward to bypass security checks. | Applications frequently redirect users to other pages, or use internal forwards in a similar manner. Sometimes the target page is specified in an unvalidated parameter, allowing attackers to choose the destination page.<br>Detecting unchecked redirects is easy. Look for redirects where you can set the full URL. Unchecked forwards are harder, because they target internal pages. | | Such redirects may attempt to install malware or trick victims into disclosing passwords or other sensitive information. Unsafe forwards may allow access control bypass. | Consider the business value of retaining your users' trust.<br>What if they get owned by malware?<br>What if attackers can access internal only functions |

# Unvalidated Redirects and Forwards

- A very good example is here:
  - http://www.asp.net/mvc/tutorials/security/preventing-open-redirection-attacks
- How to protect?
  - Perform again a verification of user input
  - Create a whitelist of allowed redirections
    - Or create a strict verification

# References

- http://www.asp.net/mvc/tutorials/security/preventing-open-redirection-attacks
- http://www.asp.net/mvc/tutorials/security/preventing-open-redirection-attacks
- http://stackoverflow.com/questions/13146032/redirect-to-requested-page-after-authentication

# General References

- OWASP Top 10
  - http://www.slideshare.net/xplodersuv/EducauseAnnualWebAppSecTutorialV3
  - http://www.slideshare.net/MaureenR/owasp-top-ten-in-practice
  - http://www.slideshare.net/tmd800/owasp-top-102013-25184337