Guide de sécurisation des applications

Introduction

Objectif	Ce guide à pour objectifs de faciliter la sécurisation des applications et sites web installés à l'université de Genève. Il décrit les points essentiels à respecter et donne des moyens pour répondre aux exigences de sécurité.
Public	Ce guide est à destination des personnels techniques qui ont pour responsabilité de déployer/de développer un site web/application.
Hors périmètre	Le respect des points présentés dans ce guide ne représente pas une garantie que l'application/le site web est suffisamment sécurisé. En cas de doute ou de question merci de prendre contact avec l'équipe sécurité cybersecurite@unige.ch.
Version	1.0, $action.dateFormatter.formatGivenString("dd.MM.yyyy", $content.currentDate)
Contacts	Si vous avez des questions ou si vous désirez modifier le guide merci de prendre contact soit avec l'équipe sécurité (architecture-si@listes.unige.ch) soit avec l'équipe architecture (cybersecurite@unige.ch)

Table des matières

Quick Guide

[Configuration WEB] [Contrôle de l'information] [IAM] [Développement] [Exploitation] [Firewall Applicatif] [Application] [Outils]

Configuration WEB

Protection des données

Chiffrement des données en transit

Le chiffrement des données en transit est essentiel pour garantir que les informations sensibles échangées entre les utilisateurs et le site web soient sécurisées et ne peuvent pas être interceptées par des tiers malveillants. Pour cela, vous devez activer HTTPS en utilisant des certificats SSL/TLS valides. Ainsi:

- Les communications doivent être par défaut en HTTPS (sauf exception dûment justifiée). Pour plus d'information sur la gestion des certificats voir https://plone.unige.ch/distic/support-si/service/service-metier/securite/securite/certificats-electroniques.
- Les certificats ne doivent pas être auto-signés.
- Les certificats ne doivent pas être expirés.
- Les sites web implémente HSTS (HTTP Strict Transport Security), voir https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.md.
- Une redirection immédiate est faite du HTTP vers le HTTPS (Ne pas chainer plusieurs redirection en http pour finir attendre l'URL cible en HTTPS. Porter une attention particulière sur la configuration HSTS dans le cas d'une redirection, d'un chainage (voir https://www.sentinelstand.com/article/http-strict-transport-security-hsts-canonical-www-redirects).
- Les algorithmes de chiffrements doivent être forts.
- L'agrafage OCSP est mise en place.

Pour aider à la création et valider une configuration pour un niveau de sécurisation élevé , les outils suivants peuvent aider :

- Mozilla SSL Configuration Generator (voir https://mozilla.github.io/server-side-tls/ssl-config-generator/). Choisir par défaut une configuration Moderne (algorithmes de chiffrement fort), si cela n'est pas possible choisir Intermediate. Sélectionner également HSTS et OCSP Stapling.
- La configuration HTTPS/TLS est testée avec https://observatory.mozilla.org et/ou https://www.ssllabs.com/ssltest/analyze.html. L'objectif est d'avoir une note A+.

> ⚠ La redirection de HTTP vers HTTPS est à réaliser vers le même FQDN (http://monsite.unige.ch https://monsite.unige.ch). Dans le cas où l'usager.e doit être redirigé.e vers un autre FQDN, cette première étape est tout de même nécessaire (http://monsite.unige.ch https://monsite.unige.ch https://www.unige.ch/monsite).

Protection des données sensibles au repos

[To Do]

Protection contre les vulnérabilités

Sécurisation des en-têtes HTTP

L'utilisation de certains en-têtes HTTP renforce la sécurité des sites web en empêchant diverses attaques et en renforçant le contrôle de la manière dont les navigateurs interagissent avec les sites. Voici quelques en-têtes HTTP à sécuriser pour améliorer la protection contre les vulnérabilités :

- Mise en œuvre des header X-Content-Type-Options, X-Frame-Options, X-XSS-Protection
- Mise en œuvre de Content Security Policy (CSP) dès le développement de l'application (Voir https://infosec.mozilla.org/guidelines/web_security#content-security-policy et https://content-security-policy.com/). Pour un site web déjà en production, il peut-être fastidieux voire impossible d'implémenter totalement CSP.

REM: si l'implémentation de CSP pour un ancien site déjà en production n'est pas possible, considérer les mesures ci-dessous suivant les cas pour réduire le risque de certaines attaques:

- Mise en œuvre des en-têtes X-Content-Type-Options, X-Frame-Options, X-XSS-Protection
- Mise en œuvre de SRI (Subresource Integrity) qui permet d'assurer l'intégrité des ressources chargées à partir de sources externes. Pour plus de détail :https://developer.mozilla.org/fr/docs/Web/Security/Subresource_Integrity
- Mise en œuvre de CORS (Cross-Origin Ressource Sharing) avec des en-têtes comme Access-Control-Allow-Origin, Access-Control-Allow-Methods, Access-Control-Allow-Headers, et Access-Control-Allow-Credentials. Pour plus de détail : https://developer.mozilla.org/fr/docs/Web/HTTP/CORS
- Mise en oeuvre de mesures de sécurité côté serveur comme la validation des entrées pour se protéger de vulnérabilité comme des injections SQL ou des XSS
- Garder un serveur web à jour (plateforme et plugin)

Pour obtenir de l'aide quant à l'implémentation de ces mesures et vérifier le niveau de sécurité d'un serveur web, vous pouvez vous aider de l'outil suivant:

- Mozilla Observatory (voir https://observatory.mozilla.org)

Injection SQL et XSS

[To Do]

Accès aux ressources externes

Img, script (js...) par défaut sous l'arborescence du site sauf exception justifiée

Contrôle : déclarer les ressources externes dans le CSP (voir Low Severity related to CSP in SSC)

Serveur Web

- Seuls les éléments nécessaires sont installés
  - seuls les modules nécessaires sont activés,
  - les applications exemples et la documentation ne sont pas installées
- Le serveur web à les droits minimaux pour s'exécuter
- Le mode production du serveur web est activé
- Les interfaces d'administration (phpinfo, server_info, etc.) ne sont accessibles qu'aux seuls administrateurs

Plusieurs contrôles peuvent aider à restreindre l'accès aux interfaces d'administration.

Un connecteur, sur un port spécifique non publié sur l'internet et bénéficiant d'une règle restrictive dans le pare-feu local, peut être dédié aux applications d'administration.

A défaut les serveurs web proposent d'autres mécanismes pour limiter l'accès aux applications d'administration aux seules adresses légitime : la directive *Require* dans une configuration Apache, une *Valve* dans Tomcat, etc

Contrôle de l'information

- L'application ne divulgue pas d'informations exploitables aux utilisateurs (technologies, versions, etc.)
- Tester avec un outil (https://www.wappalyzer.com/) que l'application ne divulgue pas d'information (en-têtes http, balise HTML meta, etc.)
- Les pages d'erreurs ne divulguent pas d'information (vérifié tous les éléments: le serveur web, le serveur applicatif, l'application, etc.)
- Pour les sites publics, vérifier que le fichier robot.txt est correctement configuré

IAM

- Les utilisateurs sont identifiés à l'aide d'une des méthodes supportées par l'institution: Shibboleth, OIDC, LDAP ou AD
- L'application n'utilise pas de comptes locaux
- Les droits applicatifs généraux (administrateur, etc.) sont gérés dans grouper
- Le logoff invalide l'ensemble des moyens d'accès: SSO, jeton OIDC, session applicative, etc.

Développement

- La protection contre le CSRF est activée
- Les accès cross-domain sont limités à ce qui est strictement nécessaire. Idéalement l'application ne fait pas d'appels cross-domain et les appels cross-domains sont interdits par un en-tête CSP
- Lors de mises à jour, les données sont validées et nettoyées (protection contre le XSS, l'injection)
- Les cookies sont correctement configurés
- Les pages d'erreurs ne divulguent pas d'information (Stack Trace)

Software Factory

- Le code est versionné sur le référentiel du code (https://gitlab.fl.unige.ch ou https://gitlab.unige.ch)
- Le code est analysé avec SonarQube (https://sonar.fl.unige.ch)
- Les packages sont versionnés sur Nexus (https://nexus.fl.unige.ch)
- Les failles de sécurité dans les dépendances sont analysées

Tests

- Les tests unitaires sont en place et sont exécutés lors de chaque build
- Le code coverage est activé et est visible dans SonarQube

Exploitation

- Les backups sont en place.
- Les failles de sécurités (application, serveur web, DB, etc.) sont suivies et font l'objet de mises à jour journalières
- Les composants installés (application, serveur web, DB, etc.) sont mis à jour régulièrement. Les versions utilisées sont des versions supportées
- L'application est scannée à intervalles réguliers et les alertes communiquées aux responsables applicatifs

Monitoring

- La journalisation (log) des accès et des erreurs est en place
- La journalisation (log) des accès enregistre dans le journal l'identifiant de l'utilisateur connecté
- Le monitoring de l'application est en place
- Les analytiques sont correctement configurées: anonymisation de l'adresse IP, suppression des données individuelles, information aux utilisateurs, etc. Pour plus d'information, consultez https://www.ge.ch/ppdt/fiches-info/doc/Cloud_computing.pdf

Firewall Applicatif

- Pour les applications anciennes et/ou n'ayant pas le niveau de sécurité adéquat un firewall applicatif (WAF) est en place et filtre les échanges
- Si l'activation du firewall applicatif comprend des risques, le WAF est activé en mode rapport
- Les logs du WAF sont archivés
- Pour plus d'information: https://modsecurity.org/, https://coreruleset.org/

Application

- Dans le cadre d'un progiciel, les bonnes pratiques propres à l'application sont revues
- L'application est configurée en accord avec les bonnes pratiques

Outils

- Vérifier la configuration HTTP à l'aide de https://observatory.mozilla.org/
- Vérifier la configuration TLS à l'aide de https://www.ssllabs.com/ssltest/analyze.html
- Vérifier les failles de sécurités à l'aide de nessus en contactant stic-securite@unige.ch
- Générer la configuration TLS à l'aide de https://mozilla.github.io/server-side-tls/ssl-config-generator/

Tests sécurité

- A01:2017-Injection
- A02:2017-Broken Authentication
- A03:2017-Sensitive Data Exposure
- A04:2017-XML External Entities (XXE)
- A05:2017-Broken Access Control
- A06:2017-Security Misconfiguration
- A07:2017-Cross-Site Scripting (XSS)
- A08:2017-Insecure Deserialization
- A09:2017-Using Components with Known Vulnerabilities
- A10:2017-Insufficient Logging & Monitoring
- A11: other control

A01:2017-Injection

OWASP A1-Injection

[Description] [Défenses] [Outils]

Description

Les attaques de type injection cherchent à faire exécuter du code par le programme. Les attaques se basent sur l'envoi d'instruction au travers de l'interface. Il s'agit souvent de faire exécuter du code SQL mais d'autres langages utilisés par le programme sont susceptibles d'être exploités: XML, LDAP, XPATH, etc.

En règle générale la faille vient de la création d'une expression par concaténation d'une variable provenant de l'utilisateur.

```
String sql = "SELECT * FROM table WHERE id=" + id;
```

Dans l'exemple précédent si la variable id est fournie par l'interface graphique sous forme de string sans vérification il est possible de transformer l'instruction pour obtenir d'autres effets

```
$x = "1 OR TRUE"
http://www.example.com/product?id=$x
```

Défenses

Software Factory

Des outils d'analyse du code (SonarQube) permettent de détecter les erreurs de développement les plus évidentes. Ces outils sont simples à mettre en place et l'analyse du code peut être automatisée.

Le code est analysé pour identifier des failles (SonarQube)

- Le code est versionné dans le référentiel de code
- Le code est analysé automatiquement à chaque soumission par SonarQube
- Les rapports de Sonar sont revus régulièrement par les développeurs pour identifier les failles de sécurités et les corriger.
- Les rapports sont revus avant soumission du code en production
- Voir SonarQube

Les développeurs ont accès dans leur IDE à l'analyse du code

- Les développeurs ont accès à un outil d'analyse du code (SonarLint) dans leur IDE.
- Les développeurs font régulièrement des analyses de leur code avec un outil d'analyse du code, de préférence avant de soumettre leur code sur le référentiel central.
- Voir SonarLint

Pratiques de développement

Les variables sont typées

- Les variables qui ne sont pas de type caractère (string) sont typées: entier, date, etc.
- Les variables sont typées dès l'introduction dans le système: soumission par l'utilisateur, import de fichier, etc.
- Dans le cadre de langage non typé (PHP) la conversion vers un type donné est forcée, ex: $var = (int)$var.

Les variables externes ne sont pas exécutées directement

- Les variables externes ne sont pas concaténées pour construire une expression (SQL)
- Le passage de variables externes se fait au travers de mécanismes sécurisés, ex: prepared statement avec passage de variable
- Les librairies utilisées pour exécuter des expressions sont réputées sans failles de sécurité

Les contenus externes sont nettoyés

S'il est nécessaire de construire une expression par concaténation les variables de type string sont préalablement échappées. C'est-à-dire que les caractères réputés dangereux sont supprimés de la variable. Il s'agit typiquement des caractères d'échappement (", ') ainsi que les caractères de terminaison de commandes: ";", "GO", etc.

- S'il cela est possible il est préférable d'utiliser une liste d'expressions autorisées (white list)
- Si les listes blanches ne sont pas applicables, appliquez une liste d'expression non autorisée (black list). Une liste noire inclut typiquement les caractères d'échappements ainsi que des terminaisons de commande.

- Utiliser de préférence une librairie réputée pour échapper les caractères dangereux
- Les caractères dangereux sont spécifiques au type d'expression utilisée: SQL, LDAP, XML, etc.

Framework de développement

Les parsers XML sont correctement configurés

- Lorsque c'est possible, préférez l'usage de JSON
- S'assurer que les requêtes sur des API JSON n'acceptent pas du XML
- Lorsqu'il est nécessaire de traiter du XML s'assurer que les parsers XML sont correctement configurés et que, en particulier, le traitement des entités externes est désactivé (https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)

Serveur Web

Seuls les modules nécessaires sont activés

- Seuls les modules utilisés par l'application sont activés
- En particulier les modules à risques (module CGI, module include) sont désactivés

Firewall Applicatif

- Pour les applications ne pouvant être corrigées, un firewall applicatif est activé et filtre les attaques de type injection

Outils

Software factory

- https://fl.unige.ch/
- SonarLint

Bonnes pratiques

- https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet_in_Java
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet
- https://www.owasp.org/index.php/ASVS_V5_Input_validation_and_output_encoding

A02:2017-Broken Authentication

OWASP A2 Broken Authentication

[Description] [Défenses] [Outils]

Description

Les attaques de type "broken authentication" cherchent à exploiter des failles dans le système d'identification (login, gestion des sessions, etc.). Des failles existent quand:

- Les logins et mots de passes sont simples à deviner
- Les jetons de session peuvent être devinés (attaque de type force brute)
- Les sessions ne sont pas invalidées (SSO)
- etc.

Défenses

La meilleure défense de déléguer l'identification aux mécanismes institutionnels: Shibboleth (LDAP si nécessaire), OIDC Connect. Idéalement il s'agit de déléguer l'identification et la gestion des sessions.

Architecture

Les accès sont contrôlés par le serveur web (Apache)

- Apache est mis en front devant l'application. Les accès directs à l'application ne sont pas possibles.

Les points d'entrée dans l'application sont clairement identifiés

- Les points d'entrées dans l'application sont clairement identifiés (ex.: utiliser une approche MVC)
- Les accès publics sont clairement identifiés et documentés.
- Les accès utilisant des mécanismes alternatifs sont clairement identifiés et documentés (ex: OIDC)

Les accès cross domains sont limités à ce qui est strictement nécessaire

- Si cela est possible ne pas faire d'appels cross domain. Le navigateur n'utilise que des ressources (API) qui proviennent du même domaine.
- Si des accès cross domain (CORS) sont nécessaires, les accès droits d'accès sont limités strictement à ce qui est nécessaire.
- Les ressources (JavaScript, CSS, Images) sont chargées depuis le même domaine.
- Si des ressources doivent être chargées depuis d'autres domaines (Google Analytics, CDN, etc.) les accès sont limités à ce qui est nécessaire au travers d'en en-tête CSP (En-têtes HTTP). Ce type d'accès est limité à ce qui est strictement nécessaire.

Les accès sont testés

- Des tests automatiques sont en place pour vérifier que les accès sur des pages/api privées ne sont accessibles que par un utilisateur identifié

Les accès alternatifs sont sécurisés

- Les accès aux API qui ne sont pas shibboléthisés utilisent OIDC
- Les méthodes d'accès alternatives sont sécurisées, en particulier ces accès répondent aux critères du guide de sécurisation OWASP concernant les éléments liés à l'identification (tableau ci-dessous)

Serveur Web

Les échanges de données sont sécurisés

- Le protocole utilisé pour les échanges est crypté (HTTPS)
- L'ensemble des ressources est chargé en HTTPS (CSS, image, JavaScript, etc.).
- L'en-tête HTTP "HTTP Strict Transport Security" est activé (En-têtes HTTP)
- Les certificats sont renouvelés régulièrement
- La configuration TLS est appropriée (https://observatory.mozilla.org/)

Les certificats sont valides

- Les certificats sont issus d'un prestataire dont le certificat racine est reconnu par les navigateurs (QuoVadis, à défaut letsencrypt https://letsencrypt.org/)

Framework de développement

Les protections contre le CSFR sont activées

- Les verbes HTTP pour modifier des données sont POST et PUT. GET n'est pas utilisé pour modifier des données.
- Les échanges se font en HTTPS
- L'application vérifie que les en-têtes HTTP "orginin" et "referer" sont les mêmes
- Un cookie de session pour le CSFR est généré et transmis avec les requêtes POST et PUT. Le serveur vérifie que les données transmises dans la requête et le cookie sont identiques.
- La valeur du cookie CSFR ne peut pas être devinée, elle est générée de façon pseudo aléatoire par un générateur et la valeur est élevée.

Les cookies sont correctement paramétrés

- Les cookies utilisés sont limités à ce qui est nécessaire: idéalement seuls le cookie de session (Shibboleth) et le cookie de CSFR sont utilisés.
- Le cookie de CSFR répond aux critères de sécurisation des cookies (Testing for cookies attributes): limité à la racine applicative (URL), HTTPS. Dans le cas du CSFR l'accès par JavaScript est autorisé, car nécessaire.
- Si l'ajout d'autres cookies est nécessaire, ils sont correctement configurés (Testing for cookies attributes).

Gestion de l'identification

L'identification se base sur les mécanismes institutionnels

- Pour les applications web, Shibboleth est utilisé pour l'identification (à défaut LDAP peut être utilisé).
- Pour les accès à des API, si Shibboleth n'est pas utilisable, le mécanisme d'identification est OIDC.
- Il n'y a pas de moyen d'identification complémentaire, en particulier:
  - Dans le cadre d'un progiciel les comptes applicatifs ne sont pas utilisés/sont désactivés.
  - Il n'y a pas de mécanisme ad hoc d'accès (backdoor)

Shibboleth est activé au plus haut niveau possible

- Shibboleth est activé au plus haut niveau possible (idéalement à la racine de l'application) et sécurise l'ensemble des accès sous son contrôle. Lorsqu'il n'est pas possible de sécuriser l'application à la racine, Shibboleth est activé au niveau le plus élevé possible.
- Les accès n'étant pas sous la protection de Shibboleth (API, accès public) sont bien identifiés, idéalement une séparation est faite au niveau de l'URL.

Gestion de la session

L'application est stateless

- L'application est stateless, elle ne stocke pas de données côté serveur en session. L'ensemble des valeurs nécessaires pour exécuter une opération est transmis par le client.

La session est gérée par les mécanismes institutionnels

- La session est gérée par les mécanismes institutionnels: Shibboleth, OIDC.
- L'application n'utilise pas de mécanisme de session propre. Si nécessaire le mécanisme de session propre répond aux critères de sécurité (voir ci-dessous)

Les applications qui utilisent un mécanisme de session propre suivent les règles de sécurité

- Le mécanisme de création de sessions est centralisé (page de login, filtre, etc.).
- Le cookie de session est correctement paramétré: il est limité à la racine applicative, il n'est pas accessible depuis JavaScript, etc. (voir Testing for cookies attributes)
- Le mécanisme de session (cookie) ne peut pas être deviné : valeur de la session suffisamment grande (ne peut pas être attaquée par force brute), générée de façon aléatoire (pas de risque de reverse engineering), etc.
- La durée de la session est limitée et suit les règles de l'institution.

La persistance de la session suit les règles institutionnelles

- Il n'y a pas de mécanisme de persistance de session longue, type "remember me".
- La durée de la session est limitée à la durée institutionnelle.

Le logout de l'application gère SSO

- Une page de logout standard est affichée et explicite le mécanisme de logout du SSO (fermer tous les onglets du navigateur, etc.)

Le logout invalide l'ensemble des mécanismes d'accès (SSO, applicatif, etc.)

- Si une session applicative est utilisée, la page de logout désactive la session applicative.
- Si l'application utilise un mécanisme d'identification complémentaire (jeton JWT, cookie pour les API, jeton d'accès pour récupérer un fichier, etc.) la page de logout les invalides.
- La page de logout standard est affichée à l'utilisateur pour désactiver la session SSO.

Contrôle des accès

Le contrôle des accès se fait sur la base de l'identificateur de session

- Le contrôle pour savoir si un utilisateur est identifié se base seulement sur la présence de l'identificateur de session (cookie, etc.). D'autres variables (mise en cache, valeur de session, etc.) ne sont pas utilisées. Dans le cas de Shibboleth les champs d'identification (user-id, persistent-id) peuvent être utilisés.

Outils

Scanner

- http://http-observatory.unige.ch/
- https://observatory.mozilla.org/

Divers

- Testing for cookies attributes
- En-têtes HTTP

A03:2017-Sensitive Data Exposure

OWASP A3-Sensitive Data Exposure

[Description] [Défenses] [Outils]

Description

Plutôt que d'attaquer les systèmes de cryptographie, les attaquants essayent de récupérer des informations pour pénétrer les systèmes. Ces informations peuvent être récupérées lorsque l'échange d'information en fait en clair (en HTTP), lorsque les techniques de cryptage sont être faibles ou en s'interposant lors de l'échange de données (man in the middle). Un autre moyen pour récupérer des informations peut être d'interroger les caches.

Les données compromises peuvent être des mots de passe, des tokens, des données bancaires, des cartes de crédit, des données sensibles (information médicale, etc.).

Défenses

Architecture

L'application ne divulgue pas d'informations exploitables aux utilisateurs (technologies, versions, etc.)

- Les pages HTML et ressources (CSS, JavaScript, etc.) ne contiennent pas de commentaires ou de tags contenant des informations pouvant être exploitées (technologies utilisées, etc.).
- Les ressources (CSS, JavaScript) sont compilées (suppression des commentaires, minification, obfcucsation).
- L'application ne divulgue pas d'information aux utilisateurs sur les technologies utilisées et les versions (sauf pour les admins): API, pied de page, etc.
- L'application est scannée avec un outil pour s'assurer qu'il ne divulgue pas d'information (exemple: https://www.wappalyzer.com)

Les données ne sont pas mises en cache

- Les données envoyées au travers d'API (JSON) ne sont pas mises en cache (En-têtes HTTP).
- Les pages web (HTML) qui contiennent des données ne sont pas mises en cache (En-têtes HTTP).

L'indexation par des moteurs de recherche est conforme aux intentions

- Les applications ne devant pas être indexées par un moteur de recherche sont protégées:
  - Les applications sont sur réseau privé (pas d'accès)
  - Les applications sont protégées à la racine par Shibboleth à la racine (pas d'accès)
  - Les applications ont un fichier robot.txt qui spécifie ce qui est indexable
- Si nécessaire un tag méta robots est présent dans la page web

La configuration est testable par un scanner

- Une page en accès publique (non sécurisé) existe pour tester la configuration Web (https://observatory.mozilla.org/). La page peut être en IP privée (http://http-observatory.unige.ch/).

Framework de développement

Le cryptage des données sensibles est fait avec des composants réputés sûrs

- Le cryptage des données est fait avec des librairies reconnues (ex.: https://en.wikipedia.org/wiki/Bouncy_Castle_(cryptography)). Pas de cryptage ad hoc conçu par les développeurs.
- Les clefs et certificats sont stockés de façon sécurisée, dans les containers système prévus à cet effet (trousseau/key store, etc.).
- Les mots de passe par défaut (notamment pour les trousseaux de clefs) sont changés.
- Le niveau de cryptage est suffisant.

L'application ne divulgue pas d'information exploitable aux utilisateurs (technologies, versions, code d'erreur, etc.)

- L'application ne retourne pas d'information exploitable dans les pages d'erreurs: stacktrace, numéro d'erreur ODBC, système, etc.
- Le nom des cookies ne divulgue pas d'information sur les technologies utilisées. Si nécessaire les cookies sont renommés.
- Les en-têtes HTTP ne divulguent pas d'information sur les technologies: x-powered-by, x-generator, server, etc.
- Les pages web ne contiennent pas d'information (commentaire, tag méta, etc.) sur les technologies utilisées
- Les URL ne contiennent pas de nom permettant d'identifier la technologie utilisée (.php, .asp, etc.)
- La structure des URL ne donne pas d'information sur les technologies utilisées.

Serveur Web

Les échanges de données sont sécurisés

- Le protocole utilisé pour les échanges est crypté (HTTPS).
- L'ensemble des ressources (CSS, image, JavaScript, etc.) est chargé en HTTPS.
- L'en-tête HTTP "HTTP Strict Transport Security" est activé (En-têtes HTTP).
- Les certificats sont renouvelés régulièrement.
- La configuration TLS est appropriée (https://observatory.mozilla.org/).
- Le plus haut niveau de configuration possible est appliqué, idéalement la configuration "moderne" est appliquée (https://observatory.mozilla.org/).

Le serveur de divulgue pas d'information (technologies, versions, code, configuration, etc.)

- Les pages d'erreurs par défaut ne divulguent pas d'information à l'utilisateur, en particulier les pages d'erreur ne contiennent pas de stack traces ou d'autres informations. Vérifier l'appel à des URL qui n'existent pas.
- Les modules d'informations ne sont accessibles que par les seuls administrateurs: phpinfo, serverinfo, etc.
- Les fichiers de configuration (config, redirect, etc.) et le code source ne sont pas téléchargeables.
- La structure de répertoires n'est pas navigable, les fichiers ne sont pas téléchargeables par défaut.
- Les fichiers de configuration ne sont pas partagés sur le réseau.
- Les en-têtes HTTP ne divulguent pas d'information sur les technologies utilisées (x-powered-by, x-generator, etc., voir En-têtes HTTP)
- Le site est scanné avec un outil pour s'assurer qu'il ne divulgue pas d'information (exemple: https://www.wappalyzer.com)

Outils

- https://www.wappalyzer.com
- https://observatory.mozilla.org/ ou http://http-observatory.unige.ch/
- En-têtes HTTP

A04:2017-XML External Entities (XXE)

[Description] [Défenses] [Outils]

OWASP A4 XML External Entities (XXE)

Description

Les attaquants peuvent souvent exploiter les vulnérabilités des processeurs XML s'ils peuvent envoyer un fichier à l'application qui contient du contenu hostile.

Par défaut la plupart des processeurs XML anciens autorisent l'intégration de contenu externe au travers d'URI (entité externe).

Ces failles peuvent être utilisées pour exécuter une requête depuis le serveur, récupérer des données, scanner le serveur, etc.

Défenses

Architecture

Les échanges de données se basent sur des formats d'échange simples

- Quand cela est possible, privilégiez un format d'échange simple (JSON) comme format d'échange.
- N'utiliser des formats plus complexes (XML) que lorsque c'est nécessaire.

Les composants utilisés sont à jour

- Utiliser des parsers XML à jour.
- Utiliser les dernières versions des formats d'échange (SOAP 1.2).

La configuration XML est sécurisée

- Désactiver les entités externes et les DTD dans les parsers (https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet).
- Si cela est possible, validez les fichiers XML et XSL à l'aide d'un schéma.

Les règles de validation des données sont appliquées indépendamment de la source et du format

- Les mêmes règles de validation s'appliquent indépendamment de la source et du format: API, import de fichier, envoi de formulaire, JSON, XML, etc.
- Lors de l'envoi d'un document structuré les données sont validées: application des règles métiers, protection contre l'injection (A01:2017-Injection), etc.

Les champs/variables sont typés et ont le typage le plus spécifique possible

- Les champs importés et les variables sont typés (entier, date, etc.).
- Le typage est le plus spécifique possible: date plutôt que string, entier plutôt qu'objet.
- Lorsque le langage est typé dynamiquement (PHP, etc.), s'assurer que le type de la donnée est spécifique au moment de l'entrée dans le système (import de fichier, envoi de données par l'interface, etc.). Exemple: $var = (int)$var.

Les champs traités par le service sont les champs attendus

- Lors de la réception de données (import de fichier, appel sur un service web, envoi d'un formulaire WEB, etc.) seuls les champs attendus par l'action sont traités.
- Les champs supplémentaires éventuellement transmis sont ignorés et ne sont pas propagés dans l'application.
- L'application n'utilise pas de mécanisme de reprise automatique de l'ensemble des champs transmis (déserialisation automatique, JPA merge, etc.) sans vérification des données.

Software factory

Les dépendances sont vérifiées pour s'assurer qu'elles ne contiennent pas de failles de sécurité

- Vérifier que les librairies incluses dans le projet ne contiennent pas de faille de sécurité.

Framework de développement

Les API JSON n'acceptent pas de documents XML

- Vérifier que les API Web JSON n'acceptent pas le mime type XML. (Certains frameworks autorisent l'envoi de données dans plusieurs types mime (json, xml, etc. Le choix du type est déterminé par l'en-tête "Accept".)

Outils

- https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet

A05:2017-Broken Access Control

OWASP A5 Broken Access Control

[Description] [Défenses] [Voir aussi]

Description

Le contrôle des accès garantit que les utilisateurs ne puissent pas agir en dehors de leurs autorisations. Un échec entraîne généralement la divulgation non autorisée d'informations, la modification ou la destruction des données ou l'exécution d'une fonction métier en dehors des limites de l'utilisateur. Les vulnérabilités usuelles du contrôle d'accès incluent:

- Contournement des contrôles en modifiant l'URL, l'état interne de l'application ou la page HTML, ou simplement en utilisant un outil d'attaque API.
- Permettre la modification de la clé primaire dans l'enregistrement d'un autre utilisateur, ce qui permet de visualiser ou d'éditer le compte de quelqu'un d'autre.
- Élévation de privilège. Agir en tant qu'utilisateur sans être connecté, ou agir comme un administrateur lorsqu'il est connecté en tant qu'utilisateur.
- Manipulation de métadonnées, telle que la relecture ou la falsification d'un jeton de contrôle d'accès JSON Web Token (JWT) ou d'un cookie ou d'un champ masqué manipulé pour élever des privilèges ou abuser de l'invalidation JWT.
- La mauvaise configuration de CORS permet un accès aux API non autorisé.
- Forcez la navigation vers des pages authentifiées en tant qu'utilisateur non authentifié ou vers des pages privilégiées (admin) en tant qu'utilisateur standard.
- Accès à des API avec des contrôles d'accès manquants pour les verbes HTTTP: POST, PUT et DELETE. etc.

Défenses

Architecture

La sécurité est appliquée au niveau du serveur (back end)

- Les contrôles sur les droits d'accès sont appliqués au niveau du serveur (back end).
- Le client est traité comme potentiellement hostile.

Les contrôles sont fondés sur la base des attributs du principal

- Le principal est construit sur la base des attributs (données démographiques, rôles) et de l'identité fournis par le mécanisme d'identification (SSO).
- Le principal est immutable.
- Les contrôles d'accès sont appliqués sur la base des attributs du principal.
- Les contrôles d'accès ne sont pas appliqués sur la base d'autres attributs: par exemple des valeurs stockées en session.

Les règles métiers sont appliquées au niveau du modèle et réutilisées

- Les règles métiers sont appliquée au niveau du modèle (Object Model, Data Access Object, etc.).
- Le modèle et les règles de validation sont réutilisés pour valider la conformité des données.
- Les règles métiers ne sont pas répliquées.

Les accès sont testés

- Des tests automatiques sont en place pour vérifier que les accès sur des pages/api privées ne sont accessibles que par un utilisateur identifié et avec le niveau d'autorisation adéquat.

Contrôle des accès

L'accès à l'application est limité au plus petit groupe possible sur la base des attributs démographiques

- L'accès à l'application est limité sur la base des attributs des utilisateurs: institution, type d'institution, affiliation (étudiant, staff, etc.), etc.
- L'accès à l'application est limité au plus petit groupe devant accéder à l'application.
- Une attention particulière est apportée aux accès d'utilisateurs ne provenant pas d'une institution (compte créer manuellement avec Switch Edu ID) et des comptes interfédération.
- L'application des droits se fait à la racine de l'application. Le mécanisme privilégié pour l'application des droits est le serveur Web.

Les droits utilisateurs sont les droits minimaux

- Les droits utilisateurs sont les droits nécessaires et minimaux pour réaliser les tâches confiées.

La gestion des rôles est gérée par les mécanismes institutionnels

- La gestion des rôles (attribution des groupes) est gérée par les mécanismes institutionnels (grouper).
- En particulier les rôles majeurs (administrateur, super utilisateur, etc.) sont gérés dans grouper.
- Les rôles sont transmis au travers du mécanisme d'identification.

Les droits d'accès sont fermés par défaut et ouverts au besoin

- Hormis les accès publics (ressources web, JavaScript, CSS, etc.) l'accès à l'application est fermé par défaut.
- Les droits d'accès sont ouverts au besoin.

Les droits sont appliqués au niveau de l'objet

- Les droits (accès/modification/suppression) sont vérifiés pour chaque objet/enregistrement.
- En particulier, quand l'utilisateur utilise un identifiant pour obtenir un objet au travers d'une API, vérifier qu'il a bien les droits sur cet objet.
- Dans le cas d'une structure hiérarchique, les droits sont appliqués pour chacun des objets inclus.
- En particulier l'application n'utilise pas de mécanisme de reprise automatique de l'ensemble des champs transmis (déserialisation automatique, JPA merge, etc.) sans vérification.
- L'utilisateur a les droits sur les objets retournés par une recherche.

Les droits sont appliqués à l'ensemble des verbes HTTP qu'elle utilise

- L'application ne répond pas aux verbes HTTP (GET, POST, HEADER, PUT) qu'elle n'utilise pas.
- Les droits d'accès sont testés pour l'ensemble des verbes HTTP utilisables.

Les interfaces n'utilisent pas de chemin (URL, répertoires, etc.) dans les échanges

- Les interfaces (pages web, API, import de fichier) n'utilisent pas de chemin (URL, répertoire, etc.) dans les paramètres utilisés.
- S'il est nécessaire de passer un chemin, utilisez un mécanisme de protection: liste blanche, identifiant numérique dans une liste, etc.
- S'il est indispensable de passer un chemin, nettoyez les valeurs (enlever les paramètres permettant de naviguer les répertoires .., /., /.. ,etc.)

Les accès aux fichiers (fichiers temporaires, rapports, etc.) sont sécurisés

- Les fichiers fournis par l'application ne sont pas accessibles au travers d'un chemin. Par exemple en déposant le fichier sur un répertoire et en donnant accès au travers du serveur Web.
- Un accès au travers d'un identifiant numérique est à privilégier.
- L'accès au fichier demande d'être identifié.

Les échecs sur le contrôle des accès sont journalisés

- Les échecs sur le contrôle des accès ne devant pas se produire sont journalisés (log).
- Le journal des erreurs contient l'identifiant de l'utilisateur.
- En cas d'erreurs répétées, les administrateurs sont prévenus.

Serveur Web

Le serveur de divulgue pas d'information

- Désactiver la navigation des répertoires et fichiers sur le serveur web.

Voir aussi

- Les accès cross domains sont limités à ce qui est strictement nécessaire (A06:2017-Security Misconfiguration)
- L'identification se base sur les mécanismes institutionnels (A02:2017-Broken Authentication)
- La session est gérée par les mécanismes institutionnels (A02:2017-Broken Authentication)
- Les champs traités sont les champs attendus (A08:2017-Insecure Deserialization)

A06:2017-Security Misconfiguration

OWASP A6 Security Misconfiguration

[Description] [Défenses] [Outils]

Description

Les attaquants vont souvent essayer d'exploiter un problème de configuration. Il peut s'agir de répertoires non protégés, de comptes avec des valeurs par défaut, etc. Ces failles peuvent être exploitées pour obtenir des informations sur le système ou des accès non autorisés. Ces failles peuvent compromettre l'ensemble du système.

Défenses

Architecture

Les composants installés sont les composants nécessaires

- Les ports ouverts sont les seuls ports nécessaires.

L'application ne divulgue pas d'information aux utilisateurs sur les technologies et versions

- Les informations techniques ne sont pas accessibles par les utilisateurs. Que ce soit au travers de l'interface graphique (pied de page) ou de services web (API). À titre d'exemple les informations suivantes ne doivent pas être accessibles par l'utilisateur:
  - Le framework utilisé
  - Le serveur web utilisé
  - Les librairies utilisées
  - Le système d'exploitation
  - Les numéros de version, de l'application, du serveur web, etc.
  - ..

Les droits attribués sont les droits minimaux

- L'utilisateur a les droits d'accès nécessaires et minimaux pour réaliser les tâches qui lui sont confiées.
- L'application a les droits nécessaires et minimaux pour réaliser les tâches qui lui sont confiées. En particulier les connexions à la base de données et les droits aux API externes (services web) ont les droits minimaux.

L'application répond aux seuls verbes HTTP qu'elle utilise

- L'application ne répond pas aux verbes HTTP (GET, POST, HEADER, PUT) qu'elle n'utilise pas.
- Privilégié un framework déclaratif, c'est-à-dire un framework qui ne répond par défaut qu'aux verbes HTTP qui ont été déclarés comme utilisés.
- Les droits d'accès sont testés pour l'ensemble des verbes HTTP utilisables.

Les accès cross domains sont limités à ce qui est strictement nécessaire

- Si cela est possible ne pas faire d'appels cross domain.
- S'il est nécessaire de faire des appels cross domain limiter ceux-ci à ce qui est strictement nécessaire. Appliquer les en-têtes HTTPS CORS et CSP pour contrôler les accès. En particulier ne pas ouvrir à tous les domaines (CORS *) ou tous les sous-domaines.
- Privilégier les technologies dominantes pour le contrôle cross domain (CORS, CSP) et éviter des technologies marginales (Flash, RIA, crossdomain.xml).

L'infrastructure est cartographiée et revue

- Les éléments qui composent l'infrastructure sont connus et documentés: serveur web, serveur applicatif, base de données, cache, connexions réseau, interdépendances, etc.
- Les éléments qui composent l'infrastructure sont revus et configurés en accord avec les règles de sécurité.
- Les interconnexions réseau (dépendances entre systèmes, URL, ports, etc.) sont documentées et configurées en accord avec les règles de sécurité.

Les interfaces d'administration ne sont accessibles que par les administrateurs

- Les interfaces d'administration ne sont accessibles qu'aux administrateurs.
- Les accès aux interfaces d'administrations sont contrôlés au travers de tests.

Le mécanisme d'identification est adapté au contexte

- Lorsque c'est possible, utilisé un certificat pour l'identification plutôt qu'un compte et mot de passe (exemple: accès SSH).

Les secrets sont sécurisés

- Les secrets (compte, mot de passe, clefs, jeton, string de connexion, certificats, etc.) sont stockés de façon sécurisée (coffre-fort/trousseau: https://hestia.unige.ch/, java keystore, etc.).
- Les secrets ne sont pas réutilisés entre applications et ne sont accessibles qu'aux applications qui en ont besoin.
- L'environnement de production n'est accessible que par l'équipe d'exploitation.

Les mots de passe sont robustes

- Les mots de passe techniques sont robustes (https://wadme80.unige.ch/cgi-bin/genpass.cgi, https://wikisecu.unige.ch/dokuwiki/doku.php?id=mots_de_passe)
  - Au moins 12 caractères
  - Contiens au moins une lettre et un chiffre
  - Contiens au moins une majuscule
  - Contiens au moins une minuscule
  - Générer de façon aléatoire

Framework

L'application ne divulgue pas d'information exploitable aux utilisateurs (technologies, versions, code d'erreur, etc.)

- L'application ne retourne pas d'information exploitable dans les pages d'erreurs: stacktrace, numéro d'erreur ODBC, système, etc.

Les paramètres sécurités sont correctement configurés

- Les sécurités du framework sont activées (CSFR).
- Les en-têtes HTTP sécurités sont activés (En-têtes HTTP).
- Les cookies sont correctement configurés (Testing for cookies attributes).

Serveur Web

Les composants installés sont les composants nécessaires

- Les modules web (module Apache, module ISAPI, etc.) activés sont les seuls modules nécessaires.
- Les applications exemples et la documentation ne sont pas installées.

Les serveurs ne contiennent pas de fichiers non référencés

- Le système ne contient pas de fichiers versionnés à la main, par exemple: monfichier.bak.php, monfichier1.php, monfichier.php.old, etc.
- Les fichiers temporaires générés ne sont accessibles que par l'utilisateur qui l'a demandé, les fichiers temporaires sont effacés régulièrement.
- Les fichiers de backups sont tous référencés et ne sont accessibles que par les administrateurs. Les fichiers de backups qui ne sont plus nécessaires sont effacés.

Les droits donnés au serveur web sont les droits minimaux

- Le serveur s'exécute avec les droits minimaux.
- Les droits donnés aux répertoires sont les droits minimaux www.owasp.org/index.php/Test_File_Permission_(OTG-CONFIG-009))

Les interfaces d'administration ne sont accessibles que par les administrateurs

- Les interfaces d'administration sont connues et documentées.
- Les interfaces d'administration ne sont accessibles que par les administrateurs des plateformes. Il s'agit des interfaces d'administration pour le serveur lui-même ainsi que pour les modules et composants installés: phpmyadmin, mysqladmin, AdminRealm, etc.

Les paramètres sécurités sont correctement configurés

- Les en-têtes HTTP sécurités sont activés (En-têtes HTTP).
- HTTPS est configuré avec le niveau de sécurité le plus élevé (https://observatory.mozilla.org/).
- Les paramètres par défauts sont revus, en particulier les mots de passes et comptes d'administration par défaut sont modifiés.

Outils

- En-têtes HTTP
- https://observatory.mozilla.org/
- https://hestia.unige.ch/
- https://wadme80.unige.ch/cgi-bin/genpass.cgi

A07:2017-Cross-Site Scripting (XSS)

OWASP A7 Cross-Site Scripting (XSS)

[Description] [Défenses] [Outils]

Description

Les étapes de type XSS consistent à injecter du code (JavaScript) dans le navigateur. On distingue trois types:

- Stored XXS: le contenu malicieux est stocké en base de données
- Reflected XXS: le contenu malicieux est passé au travers d'un paramètre (pas de stockage en base)
- DOM XSS: le contenu malicieux est passé au travers d'une API (Json, etc.) avant d'être exécuté par un framework de présentation.

Les attaques de type XSS permettent entre autres de: voler la session, de télécharger des logiciels malveillants, d'installer des key loggers, etc.

Défenses

Architecture

L'application utilise des ressources locales

- L'application n'utilise pas de ressources externes (JavaScript, Image, CSS, etc.). Les ressources font partie de l'application et sont disponibles depuis la même URL de base.
- S'il est nécessaire d'utiliser des ressources externes (Google Analytics) les droits d'accès sont spécifiques à ces ressources (liste blanche). Il n'y a pas d'accès générique (wildcards) à des ressources externes.

L'application limite le nombre de technologies de présentation utilisée et privilégie les technologies standards

- Ne pas utiliser de technologies de type Flash, Applet, Object, etc. si cela n'est pas nécessaire.
- Quand cela est possible, l'utilisation de technologies non utilisées est désactivée (voir Content-Security-Policy)

Pratiques de développement

Les variables/données sont typées

- Les variables qui ne sont pas de type caractère (string) sont typées: entier, date, etc.
- Les variables sont typées dès l'introduction dans le système: soumission par l'utilisateur, import de fichier, etc.
- Dans le cadre de langage non typé (PHP) la conversion vers un type donné est forcée, ex: $var = (int)$var.

Les contenus externes de type chaîne de caractères sont nettoyés

- Les contenus externes (soumission, import de fichier, etc.) de type string sont nettoyés. Les contenus de type HTML, CSS, JavaScript ne sont pas acceptés.
- Quand cela est possible, il est préférable d'utiliser une liste d'expressions autorisées (liste blanche).
- Le nettoyage se fait avec l'aide de librairies réputées sûres (ne pas créer soit même de librairie): http://html-tidy.html-tidy.org/, https://www.owasp.org/inOWASP Java HTML Sanitizer, dex.php/OWASP_Java_HTML_Sanitizer_Project,
- S'il est nécessaire de conserver du formatage HTML, ne garder que les tags nécessaires et utiliser une librairie pour nettoyer le code.

Les paramètres externes sont nettoyés

- Les paramètres passés à une requête sont nettoyés même s'ils ne sont pas stockés en base, ex: paramètres d'URL, en-têtes HTTP, etc.

Les modifications du DOM se font au travers d'un framework

- Ne pas générer de page et de ressources (CSS) dynamiquement par concaténation, mais les mécanismes de présentation du framework pour échapper les variables.
- Ne pas modifier directement le contenu (document.write, etc.), mais utiliser les mécanismes de présentation du framework.

Framework de développement

Les contenus sont affichés par défaut comme du texte

- Utilisez un framework de présentation qui, par défaut échappe le contenu qui est affiché dans les pages Web (ex: Angular), et traite le contenu comme étant du pur texte.

En-têtes HTTP

Activer l'en-tête content security policy

- L'en-tête HTTP Content-Security-Policyest activé au niveau de restriction le plus élevé.
- En particulier le code JavaScript en ligne (inline) est désactivé.

Activer l'en-tête XSS

- L'en-tête X-XSS-Protection est activé.

Activer l'en-tête X-Content-Type-Options

- L'en-tête X-Content-Type-Options est activé.

Serveur Web

Firewall Applicatif

- Pour les applications ne pouvant être corrigées, un firewall applicatif est activé et filtre les attaques de type injection pour le JavaScript, le CSS, le HTML.

Outils

- https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet_in_Java#HTML.2FJavaScript.2FCSS

A08:2017-Insecure Deserialization

OWASP A8 Insecure Deserialization

[Description] [Défenses] [Outils]

Description

Les applications et les API seront vulnérables si elles déserialisent des objets hostiles ou falsifiés fournis par un attaquant. Cela peut entraîner deux principaux types d'attaques:

- Des attaques liées aux objets et à la structure de données où l'attaquant modifie la logique de l'application. Cela arrive quand les objets (données et code) sont désérialisés (exemple: sérialisation d'objets en Java). Ces attaques permettent l'exécution de code arbitraire à distance.
- Les attaques de falsification de données. Dans ce cas, seules les données sont modifiées (exemple: POST d'une structure de données JSON falsifiées sur une API).

La sérialisation peut être utilisée dans des applications pour:

- Communication à distance et interprocessus (RPC / IPC)
- Services Web, envois de messages
- Mise en cache/persistance
- Bases de données, serveurs de cache, systèmes de fichiers
- Cookies HTTP, paramètres de formulaire HTML, jetons d'authentification API

Défenses

Architecture

Les échanges de données se basent sur des formats d'échange simples

- Les échanges de données se basent sur des types primitifs et/ou des structures de données basées sur des types primitifs (exemple: JSON, Texte, etc.)
- Les échanges de données se basent sur des formats qui ne permettent pas la déserialisation de code

Les champs/variables sont typés et ont le typage le plus spécifique possible

- Les champs importés et les variables sont typés (entier, date, etc.).
- Le typage est le plus spécifique possible: date plutôt que string, entier plutôt qu'objet.
- Lorsque le langage est typé dynamiquement (PHP, etc.), s'assurer que le type de la donnée est spécifique au moment de l'entrée dans le système (import de fichier, envoi de données par l'interface, etc.). Exemple: $var = (int)$var.

Les champs traités sont les champs attendus

- Lors de la réception de données (import de fichier, appel sur un service web, envoi d'un formulaire WEB, etc.) seuls les champs attendus par l'action sont traités.
- Les champs supplémentaires éventuellement transmis sont ignorés et ne sont pas propagés dans l'application.
- L'application n'utilise pas de mécanisme de reprise automatique de l'ensemble des champs transmis (déserialisation automatique, JPA merge, etc.) sans vérification des données.

Les règles métiers sont appliquées au niveau du modèle et réutilisées

- Les mêmes règles de validation s'appliquent indépendamment de la source et du format: API, import de fichier, envoi de formulaire, JSON, XML, etc.
- Lors de l'envoi d'un document structuré les données sont validées: application des règles métiers, protection contre l'injection (A01:2017-Injection), etc.

Outils

- https://www.owasp.org/index.php/Deserialization_Cheat_Sheet
- https://speakerdeck.com/pwntester/surviving-the-java-deserialization-apocalypse

A09:2017-Using Components with Known Vulnerabilities

[Description] [Défenses] [Voir aussi] [Outils]

OWASP A9 Using Components with Known Vulnerabilities

Description

Les attaquants exploitent fréquemment des failles de sécurités présentent dans les composants utilisés.

Une application est à risque:

- Si on ne connaît pas les versions de tous les composants (côté client et côté serveur). Cela inclut les composants utilisés directement ainsi que les dépendances imbriquées.
- Si le logiciel comporte une faille, n'est plus supporté ou est obsolète. Cela inclut le système d'exploitation, le serveur Web / d'application, le système de gestion de base de données (SGBD), les applications, les API, l'environnement d'exécution et les bibliothèques.
- S'il n'y a pas de surveillance régulière des vulnérabilités.
- S'il n'y a pas de mise à jour et de patch régulier sur l'ensemble de la chaîne (plateforme, frameworks, dépendances, etc.).
- Si les développeurs ne testent pas la compatibilité des bibliothèques mises à jour, améliorées ou corrigées.
- Si les composants ne sont pas correctement configurés.

Défenses

Exploitation

Les applications et services déployés sont maintenus

- Les développements internes sont maintenus. Les développements sont mis à jour à intervalles réguliers en accord avec le plan de maintenance. En particulier les librairies et frameworks sont mis à jour vers les dernières versions.
- Les développements externes sont maintenus. Les développements sont mis à jour à intervalles réguliers en accord avec le plan de maintenance. La maintenance fait l'objet d'un contrat de maintenance externe ou la maintenance est reprise en interne.
- Les applications déployées sont maintenues. La migration vers des versions récentes se fait à intervalle régulier en accord avec le plan de maintenance.

Les failles de sécurités font l'objet d'une surveillance

- Les failles de sécurités sont surveillées. Les failles de sécurité majeures font l'objet d'une alerte.

Architecture

Les composants installés sont les composants nécessaires

- L'application n'a pas de dépendance sur des composants qui ne sont pas nécessaires: librairies, fichiers, exemples, documentation, etc.
- Une application utilise un seul composant pour répondre à un besoin (affichage, formatage, etc.). Plusieurs composants distincts ne sont pas utilisés pour répondre à un même besoin.
- Les fonctionnalités activées sont les fonctionnalités nécessaires.

Les composants utilisés sont supportés

- Les composants (librairies, frameworks, etc.) utilisés sont supportés, soit par l'éditeur, soit par la communauté.
- Les composants utilisés sont activement maintenus: les mises à jour des composants sont fréquentes, les mises à jour pour les failles de sécurités sont rapides, etc.
- L'institution à accès aux dernières versions. Dans le cadre des logiciels payant l'institution à un contrat qui lui donne accès aux dernières versions.

Les versions des dépendances sont connues

- Les applications utilisent un système de gestion des versions pour les dépendances (maven, composer, npm, etc.).
- Les versions des dépendances sur des ressources client (JavaScript, CSS, etc.) sont connues.

Les montées de version des dépendances sont simples à réaliser

- Les applications utilisent un mécanisme de gestion des dépendances (packet manager) qui permet de modifier les versions des dépendances utilisées.
- Le processus de mise à jour des dépendances (téléchargement, remplacement) est automatique.

Les dépendances applicatives sont récupérées depuis le référentiel institutionnel

- Les dépendances applicatives sont récupérées depuis le référentiel institutionnel (https://nexus.fl.unige.ch/).
- Il n'y a pas d'accès direct à un référentiel extérieur pour les dépendances applicatives (npm, maven central, etc.).

Les applications et dépendances proviennent de sources officielles

- Les applications proviennent de sources officielles (moodle.org, etc.)

La qualification des développements est simple à réaliser

- Les développements disposent de test automatique (unitaire et d'intégration).
- La couverture des tests automatiques est suffisante pour assurer la qualification de l'application lors d'une mise à niveau des dépendances.

Les applications sont versionnées et stockées sur le référentiel institutionnel

- Les applications et librairies développées sont versionnées.
- Les livrables sont stockés sur le référentiel institutionnel (https://nexus.fl.unige.ch/).

Les applications non maintenues sont isolées

- Les applications non maintenues sont isolées dans une zone dédiée (machine virtuelle) avec des accès restreints.
- Les accès WEB se font au travers d'un serveur WEB à jour.
- Un firewall applicatif est installé et configuré sur le serveur WEB.
- Si nécessaire envisager le déploiement d'un patch virtuel.

Software Factory

Les applications et dépendances proviennent de sources officielles

- Le référentiel institutionnel (https://nexus.fl.unige.ch/) ne sert de miroir que pour des référentiels officiels (maven central, maven oracle, npm, etc.)

Les dépendances (librairies, framework) sont analysées pour détecter la présence de failles de sécurité

- Les dépendances (librairies) sont analysées pour détecter la présence de failles de sécurité
- Les rapports sont affichés dans SonarQbue

Les containers (Dockers) sont analysés pour détecter la présence de failles de sécurité

- Les applications qui utilisent des containers (Docker) analysent les containers pour la présence de failles de sécurités
- Les analyses sont automatiques et font partie du processus de construction.
- Voir https://github.com/coreos/clair

Les dépendances (librairies, framework) sont analysées pour détecter la présence de versions plus récente

- Le processus de construction (build) scan les dépendances pour la présence de versions plus récentes
- Pour Maven (http://www.mojohaus.org/versions-maven-plugin/) pour JavaScript (https://github.com/retirejs/retire.js/)

Les tests sont exécutés automatiquement lors des builds

- Les tests unitaires et d'intégration sont exécutés automatiquement lors des builds.

Les analyses sont réalisées à intervalles réguliers

- Le processus de construction (build) et les analyses (faille de sécurités, etc.) sont réalisés à intervalles réguliers.
- Les résultats des analyses (notamment les failles de sécurités) font l'objet d'une surveillance.

Serveur Web

Les composants sont à jour

- Le serveur est mis à jour régulièrement.
- Les patchs sécurités sont appliqués quotidiennement.

Les composants installés sont les composants nécessaires

- Les modules web (module Apache, module ISAPI, etc.) activés sont les seuls modules nécessaires.
- Les applications exemples et la documentation ne sont pas installées.

Les composants utilisés sont supportés

- Le serveur et sa version sont supportés.

Système d'exploitation

Les composants sont à jour

- Le système d'exploitation est mis à jour régulièrement.
- Les patchs sécurités sont appliqués quotidiennement.

Les composants utilisés sont supportés

- Le système d'exploitation et sa version sont supportés.

Voir aussi

- Les applications sont correctement configurées (A06:2017-Security Misconfiguration)
- Les applications ne divulguent pas d'information concernant leurs versions ainsi que les technologies concernées (A03:2017-Sensitive Data Exposure)

Outils

Sotware factory

- https://fl.unige.ch/

Analyse de faille des sécurités dans les dépendances

- https://www.owasp.org/index.php/OWASP_Dependency_Check
- https://github.com/coreos/clair

Analyse de versions plus récentes dans les dépendances

- http://www.mojohaus.org/versions-maven-plugin/
- https://github.com/retirejs/retire.js/
- https://chrome.google.com/webstore/search/retirejs?hl=fr

Référentiels

- https://nodesecurity.io/advisories
- https://www.cvedetails.com/version-search.php
- https://nvd.nist.gov/

Firewall applicatif

- https://modsecurity.org/
- https://coreruleset.org/
- https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project

A10:2017-Insufficient Logging & Monitoring

[Description] [Défenses] [Outils]

OWASP A10 Insufficient Logging & Monitoring

Description

La journalisation, la détection, la surveillance et la réponse sont insuffisantes si:

- Les événements vérifiables, tels que les connexions, les échecs de connexion et les transactions importantes, ne sont pas consignés.
- Les avertissements et les erreurs génèrent des messages inadéquats, peu clairs, ou ne génèrent pas de messages.
- Les journaux des applications et des API ne sont pas surveillés pour détecter une activité suspecte.
- Les journaux ne sont stockés que localement.
- Les seuils d'alerte et les processus d'escalade de ne sont pas en place ou effectifs.
- Les tests de pénétration et les analyses effectuées par les outils de type DAST (tels que OWASP ZAP) ne déclenchent pas d'alertes.
- L'application est incapable de détecter, d'escalader ou d'alerter les attaques en temps réel ou en temps quasi réel.

Vous êtes vulnérable aux fuites d'informations si vous rendez visibles les événements de journalisation et d'alerte à un utilisateur ou à un attaquant (voir A03:2017-Sensitive Data Exposure).

Défenses

Architecture

Les adresses web (URL) ne contiennent pas d'informations systématiques permettant de tracer l'utilisateur

- Les adresses web (URL) ne contiennent pas d'informations systématiques permettant de tracer l'utilisateur telles que: identifiant, adresse mail, etc.

Les modifications des données sensibles sont auditées (développement)

- Les données sensibles sont auditées: qui a fait la modification, quand la modification a été faite.
- Pour les données ayant une importance particulière la nature de la modification est auditée: table journalisée, table des historiques, etc.

Les accès sont contrôlés par un reverse proxy (Apache)

- Un reverse proxy (Apache) est mis en front devant l'application. Les accès directs à l'application ne sont pas possibles.

Journalisation

Les requêtes HTTP sont journalisées

- Les requêtes HTTP sont journalisées.
- Le journal contient l'identifiant de l'utilisateur connecté.

Le format des journaux est exploitable par un agent

- Les journaux générés peuvent être importés par un agent (programme informatique) et centralisés.

Les journaux sont exploitables

- Les événements critiques devant être suivis sont formalisés et facilement identifiables dans les journaux.
- Les journaux ne contiennent pas d'événements sans valeur (bruit). En particulier les avertissements sont traités pendant la phase de test et ne sont pas reconduits dans l'environnement de production.
- Les événements sont liés à l'application. Lorsque les journaux sont agrégés (par exemple dans le cas d'un serveur applicatif) l'application ayant généré l'événement est inscrite dans le journal.
- Lorsqu'un utilisateur est connecté, son identifiant est inscrit dans les journaux.

Les journaux sont archivés

- Les journaux sont compressés et archivés.

Google Analytics est correctement configuré

- Les adresses IP sont anonymisées (https://support.google.com/analytics/answer/2763052).
- La durée de conservation des données brutes est limitée à ce qui est nécessaire sans plus. Les données agrégées peuvent être conservées aussi longtemps que nécessaire (https://support.google.com/analytics/answer/7667196)
- Les comptes sont supprimés lorsqu'ils ne sont plus nécessaires (https://developers.google.com/analytics/devguides/collection/analyticsjs/cookie-usage).
- Les utilisateurs sont informés.

Gestion de l'identification

Les identifications sont journalisées

- Les identifications et tentatives d'identification sont journalisées

Outils

Firewall applicatif

- https://modsecurity.org/
- https://coreruleset.org/
- https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project

A11: other control

[Défenses]

Défenses

Architecture

Les requêtes sont considérées comme potentiellement hostiles

- La sécurité est implémentée au niveau du serveur (back-end). Le client est considéré comme potentiellement hostile.
- Les valeurs passées par le client (headers, verbes HTTP, contenus, etc.) sont validées au moment de la réception.

Les IFRAMES ne sont pas utilisées

- Les IFRAMES ne sont pas utilisées pour construire des applications

Les points d'entrées sont répertoriés

- Les points d'entrées dans l'application sont connus: API, intégration avec d'autres applications, clients web, etc.
- La même architecture est réutilisée entre les applications
- Les API à disposition d'applications externes sont documentées

Pratiques de développement

Les fichiers uploadés sont vérifiés

- Les types de fichiers acceptés sont limités (liste blanche)
- Les fichiers sont downloadés avec l'en-tête X-Content-Type-Options
- Les noms de fichier n'acceptent pas de caractères de navigation (/, /.., .., etc.)
- Les fichiers compressés soit ne contiennent pas de chemin, soit ne sont pas décompressés (Zip bombe)
- Les fichiers stockés sur le serveur n'ont pas de droit d'exécution

Framework de développement

La configuration contre le CSRF est activée

- Les modifications ne sont pas autorisées sur le verbe HTTP GET
- Les données sont transmises en HTTPS
- Un cookie de protection contre le CSFR est présent et est validé lors des requêtes

La protection contre le clickjacking est activée

- Un en-tête HTTP X-Frame-Options est présents, éventuellement ajouter un script de framebursting

Journalisation

Les événements non désirés sont journalisés

- Les événements qui ne devraient pas se produire sont journalisés avec l'identifiant de l'utilisateur
- En cas de problème à répétition, une alerte est levée à destination des administrateurs

Annexes

En-têtes HTTP

[En-têtes] [Outils]

En-têtes

- Cache-Control
- Content-Security-Policy
- Expect-CT
- HTTP Strict Transport Security
- Public Key Pinning Extension for HTTP (HPKP)
- Referrer-Policy
- X-Content-Type-Options
- X-Frame-Options
- X-Permitted-Cross-Domain-Policies
- X-Powered-By, Server, etc.
- X-XSS-Protection

Outils

- https://observatory.mozilla.org/
- https://infosec.mozilla.org/guidelines/web_security#web-security-cheat-sheet
- https://www.owasp.org/index.php/OWASP_Secure_Headers_Project#tab=Headers
- https://docs.spring.io/spring-security/site/docs/current/reference/html/headers.html

Cache-Control

Nom	`Cache-Control: no-cache, no-store, must-revalidate`
	
	`Cache-Control: public, max-age=31536000`

Avantage	MOYEN
Difficulté	MOYEN
Poids	10
Description	• Garantit la bonne gestion des caches • Garantit que les données (API ou page HTML) ne sont pas mises en cache, et donc pas accessibles après que la session soit fermée (ex: cyber-café) • Garantit que les ressources publiques (CSS, Images, etc.) sont mises en cache
Documentation	• https://developer.mozilla.org/fr/docs/Web/HTTP/Headers/Cache-Control
Utilisation	• ✅ De façon systématique pour les API et les pages HTML contenant des données
Activation	• Au niveau du framework
Remarques	

Exemple	# Empêche la mise en cache
	`Cache-Control: no-cache, no-store, must-revalidate`
	# Mise en cache des fichiers statiques
	`Cache-Control: public, max-age=31536000`

Content-Security-Policy

Nom	Content-Security-Policy
Avantage	ELEVÉ
Difficulté	ELEVÉ
Poids	25

Description	• Garantit que les ressources chargées proviennent bien du bon site
	• Protège contre les attaques de type XSS
	• Contrôle les ressources de type
	  ◦ Image
	  ◦ StyleSheet/CSS
	  ◦ JavaScript
	  ◦ Ajax

Documentation	• https://infosec.mozilla.org/guidelines/web_security#content-security-policy • https://content-security-policy.com/ • Évaluer avec https://csp-evaluator.withgoogle.com/ • Une première version de l'en-tête CSP peut être générée avec https://addons.mozilla.org/en-US/firefox/addon/laboratory-by-mozilla/
Utilisation	• ✅ De façon systématique pour les développements internes (nouvelles applications) • ✅ Fortement recommandé pour applications sensibles • ❓ Pour les progiciels et les applications anciennes peut être difficiles à implémenter. À évaluer au cas par cas.
Activation	• Au niveau du serveur web, par application • Pour les applications nouvelles, activer l'en-tête CSP dès le début du développement
Remarques	• Les applications anciennes sont difficiles à adapter • L'usage d'api externes (Google, Twitter, etc.) nécessite d'adapter la politique en conséquence • On peut activer le mode "report" pour vérifier si une application est compatible avec une politique

Exemple	# This policy allows images, scripts, AJAX, and CSS from the same origin, and does not allow any other resources to load (eg object, frame, media, etc). It is a good starting point for many sites.
	`default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self';`
	# For google analytics
	`script-src 'self' www.google-analytics.com`
	# Deactivate objects (flash, applet, etc.)
	object-src 'none'
	# https only
	`default-src https:`

Expect-CT

Nom	Expect-CT
Avantage	FAIBLE
Difficulté	MOYENNE
Poids	05
Description	• Améliore la transparence des certificats
Documentation	• https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expect-CT
Utilisation	• ✅ Activez de façon systématique pour les certificats publics (QuoVadis, etc.) • ❌ Ne pas activer pour les certificats autosignés ou signer par l'institution
Activation	• Au niveau du serveur web (Apache)
Remarques	▪ Peut être déployé en mode "rapport" ▪ Vérifier la compatibilité en activant le mode rapport ▪ Commencer en spécifiant une durée de cache courte
Exemple	`Expect-CT: max-age=86400`

HTTP Strict Transport Security

Nom	HTTP Strict Transport Security (HSTS)
Avantage	ELEVÉ
Difficulté	FAIBLE
Poids	25
Description	• L'usage de HTTPS est obligatoire. • ⚠️ En cas d'erreur sur le certificat, le certificat est banni pour la durée spécifiée dans le cache. Une remise à zéro du cache du navigateur n'invalide pas le cache HSTS. Il est en conséquence important que la gestion des certifications soit en place. • ⚠️ Il n'est pas possible d'ajouter une exception en cas d'erreur sur le certificat. En conséquence ,le certificat ne peut pas être autosigné.
Documentation	https://infosec.mozilla.org/guidelines/web_security#http-strict-transport-security https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.md
Utilisation	• ✅ De façon systématique pour les applications sensibles • ❌ Ne pas activer pour les certificats autosignés ou issus d'une autorité de certification non connue
Activation	▪ Au niveau du serveur web ▪ Utiliser le générateur https://ssl-config.mozilla.org/ et cocher "HTTP Strict Transport Security" dans Miscelleanous
Remarques	• Nécessite un certificat valide • Commencer par une durée cache courte
Exemple	*Phase de test (lors de la validation de la configuration, spécifier une valeur de l'ordre d'une heure)* # Only connect to this site via HTTPS for few minutes (recommended) # l'attribut `includeSubDomains` permet d'appliquer la directive aux sous-domaines. Strict-Transport-Security: max-age=3600; includeSubDomains *Mode run* # Only connect to this site via HTTPS for the two years (recommended) # l'attribut `includeSubDomains` permet d'appliquer la directive aux sous-domaines. Strict-Transport-Security: max-age=63072000; includeSubDomains

Public Key Pinning Extension for HTTP (HPKP)

Nom	Public Key Pinning Extension for HTTP (HPKP)
Avantage	FAIBLE
Difficulté	MAXIMUM
Poids	00
Description	• Lie un certificat à un site • ⚠️ Dangereux à réaliser
Documentation	• https://infosec.mozilla.org/guidelines/web_security#http-public-key-pinning
Utilisation	• ❌ Ne pas utiliser sauf pour les sites les plus sensibles • ⚠️ A risque
Activation	• Au niveau du serveur Web (Apache)
Remarques	
Exemple	

Referrer-Policy

Nom	Referrer-Policy
Avantage	FAIBLE
Difficulté	FAIBLE
Poids	01
Description	• Permets de spécifier quand le referer (provenance) est envoyé • Améliore la confidentialité
Documentation	
Utilisation	• ? Optionnel
Activation	▪ Au niveau du serveur web (Apache)
Remarques	
Example	# On example.com, only send the Referer header when loading or linking to other example.com resources Referrer-Policy: same-origin

X-Content-Type-Options

Nom	X-Content-Type-Options
Avantage	FAIBLE
Difficulté	FAIBLE
Poids	05
Description	• S'assure que seules les ressources envoyées avec le bon mime-type (css, javascript, etc.) sont interprétées comme tel • Protège contre des attaques de type XSS
Documentation	• https://infosec.mozilla.org/guidelines/web_security#x-content-type-options
Utilisation	• ✅ De façon systématique pour les développements internes • ❓ Utile pour les progiciels, peut avoir des effets de bords, à tester
Activation	▪ Au niveau du serveur web (Apache)
Remarques	• Si l'application est mal développée, il y a un risque que certaines ressources ne soient plus interprétées (JavaScript, CSS) et ne fonctionnent plus. Si c'est le cas, il faut le traiter comme un bug et le corriger. • On peut s'en assurer en vérifiant que les ressources (CSS, JavaScript) transmissent par l'application ont le bon mime-type.
Example	# Prevent browsers from incorrectly detecting non-scripts as scripts X-Content-Type-Options: nosniff

X-Frame-Options

Nom	X-Frame-Options
Avantage	ELEVÉ
Difficulté	FAIBLE
Poids	20
Description	• Empêche d'afficher le site dans une iframe • Protège contre le click-jacking
Documentation	• https://infosec.mozilla.org/guidelines/web_security#x-frame-options
Utilisation	• ✅ De façon systématique pour les applications n'étant pas affichée dans le portail/ne devant pas être affichée dans dans une iframe • ❌ Pour les applications devant être affichée dans le portail utilisez l'en-tête CSP
Activation	▪ Au niveau du serveur web (Apache)
Remarques	• S'applique également dans la CSP • L'affichage d'applications dans une iframe est à éviter
Exemple	# Block site from being framed with X-Frame-Options and CSP X-Frame-Options: DENY X-Frame-Options: SAMEORIGIN Content-Security-Policy: frame-ancestors 'none'

X-Permitted-Cross-Domain-Policies

Nom	X-Permitted-Cross-Domain-Policies
Avantage	FAIBLE
Difficulté	FAIBLE
Poids	00
Description	• Liste les sites auxquelles les technologies comme Flash et PDF peuvent accéder
Documentation	
Utilisation	• ❌ Inutile si n'utilise aucune de les technologies Flash ou PDF
Activation	• Au niveau du serveur web (Apache)
Remarques	• Flash et PDF ne sont pas des technologies recommandés
Exemple	

X-Powered-By, Server, etc.

Nom	• X-Powered-By • X-Generator • Server
Avantage	MOYENNE
Difficulté	FAIBLE
Poids	05
Description	• Les bannières permettent de savoir quelles technologies, et quelles versions sont utilisées. Ces informations sont utilisées par les attaquants pour cibler des failles. • Les numéros de version sont particulièrement sensible
Documentation	
Utilisation	• ✅ ⚠️ Les en-têtes doivent être enlevés
Activation	• Au niveau du serveur web (Apache) • Au niveau des serveurs applicatifs (Weblogic)
Remarques	▪ Par défaut les serveurs web affichent leur version ▪ Un outil comme wappalyzer (https://www.wappalyzer.com/) peut être utilisé pour identifier si des numéros de version ou des technologies sont transmis
Exemple	Passer Apache en mode production: • set *ServerTokens* to Off or Prod • set *ServerSignature* to Off

X-XSS-Protection

Nom	X-XSS-Protection
Avantage	FAIBLE
Difficulté	MOYENNE
Poids	10
Description	• Active le filtre XSS du navigateur • Le filtre XSS est normalement actif, l'ajout du header permet de s'assurer que l'utilisateur n'a pas désactivé le filtre • Empêche certaines attaques de type XSS
Documentation	• https://infosec.mozilla.org/guidelines/web_security#x-xss-protection
Utilisation	• ✅ De façon systématique pour les nouvelles applications
Activation	Au niveau du serveur web (Apache)
Remarques	• S'applique également dans la CSP
Example	# Block pages from loading when they detect reflected XSS attacks X-XSS-Protection: 1; mode=block

Outils

Nom	Url	Description
Observatory	• http://http-observatory.unige.ch/ • https://observatory.mozilla.org/	Validation de la configuration HTTPS et des en-têtes HTTP
SSL Labs	• https://www.ssllabs.com/ssltest/	• Permet de vérifier la configuration SSL/HTTPS
Zaproxy	• ZAP	• Facilite/Automatise les test de pénétrations • Peut-être intégré à la SF
Générer un mot de passe	• https://wadme80.unige.ch/cgi-bin/genpass.cgi	Générateur de mots de passe
Forge Logiciel	https://fl.unige.ch	Ensemble d'outils pour le développement logiciel: GitLab, Nexus, Sonar, etc.
Forge Logiciel - JaCoCo	JaCoCo	Outil de revue du code (code coverage) pour les tests
Forge Logiciel - Dependency Check	▪ https://github.com/stevespringett/dependency-check-sonar-plugin ▪ https://jeremylong.github.io/DependencyCheck/	Outils permettant de vérifier si les dépendances contiennent des failles de sécurité
ModSecurity	▪ https://coreruleset.org/ ▪ https://www.modsecurity.org/	Firewall Applicatif, permet d'intercepter des attaques et de protéger les applications
wappalyzer	▪ https://www.wappalyzer.com/	▪ Permet de vérifier que des informations de sécurité ne sont pas misent à disposition par erreur
Clair	▪ https://docs.gitlab.com/ee/topics/autodevops/#auto-container-scanninge ▪ https://github.com/coreos/clair	▪ Analyse des failles de sécurité dans les dépendances pour les containers (Docker)

ZAP

[Info] [Démarrage rapide]

Info

À utiliser sur une application en ⚠️ TEST ⚠️

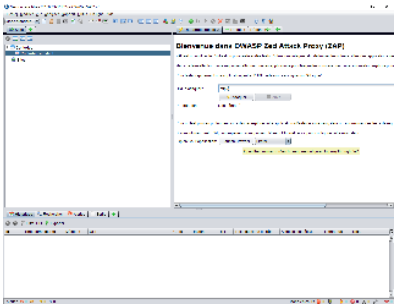
Liens:

- https://www.zaproxy.org/
- https://github.com/zaproxy/zaproxy
- https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
- https://docs.gitlab.com/ee/topics/autodevops/#auto-dast

Démarrage rapide

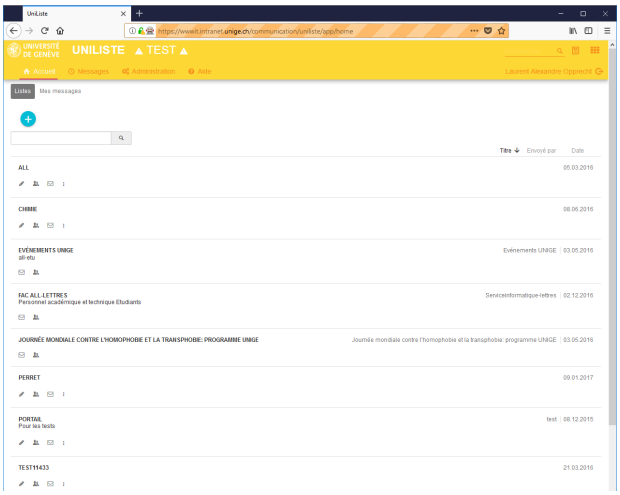
Accéder à l'application

Lancer le navigateur avec le bouton "Lauch Browser"



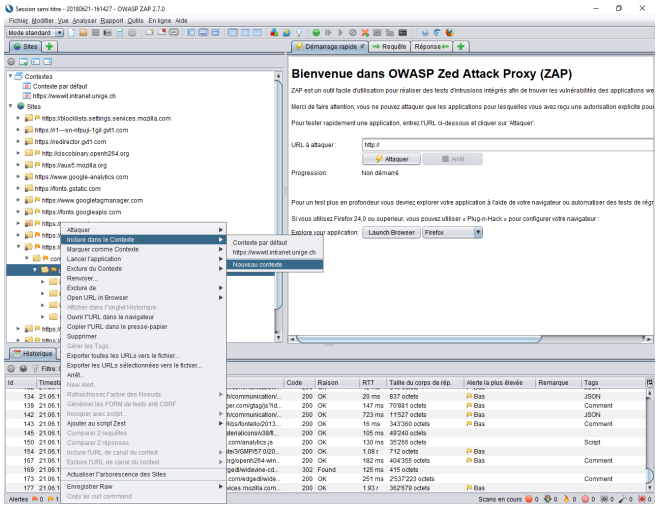
Entrer l'URL de l'application en test

Se connecter au SSO

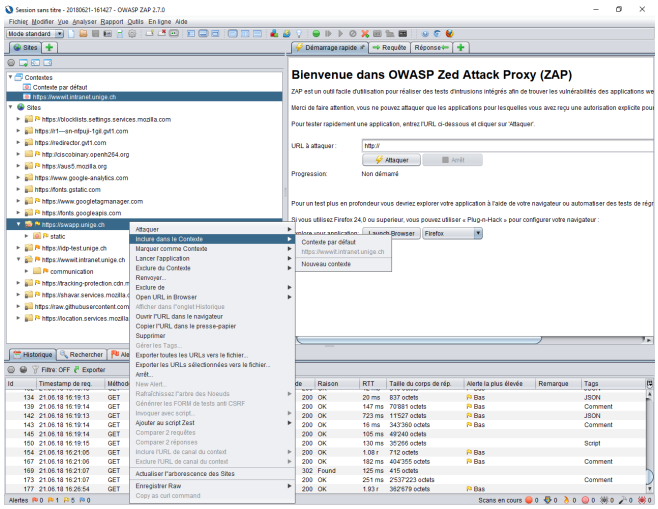


Créer le contexte

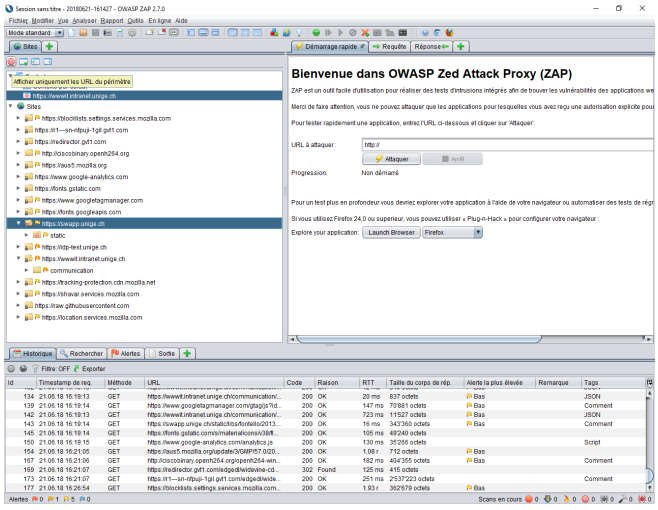
Dans la liste des sites visités, trouver le noeud à ajouter au contexte et créer un nouveau contexte



Ajouter éventuellement d'autres sites au contexte (par exemple cdn)

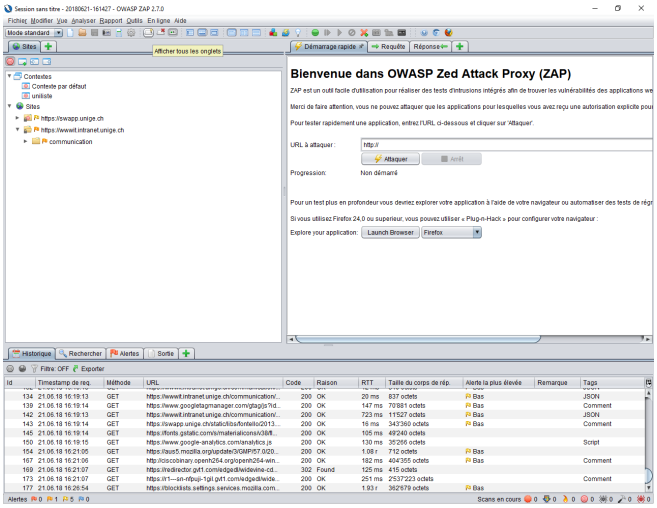


Filtrer les sites.

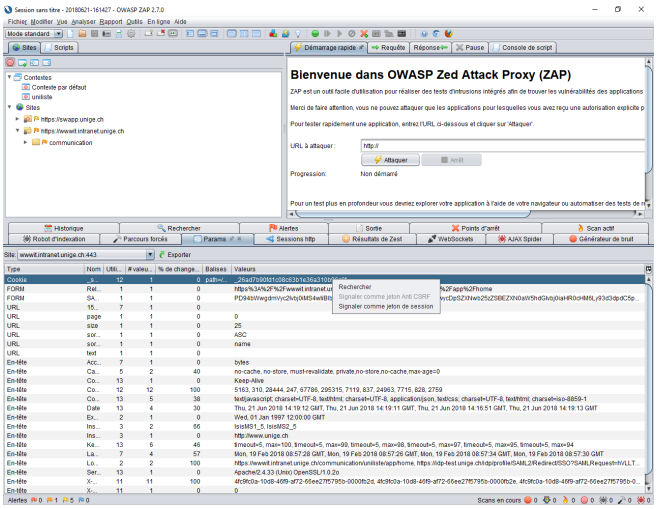


Identification

Afficher tous les onglets



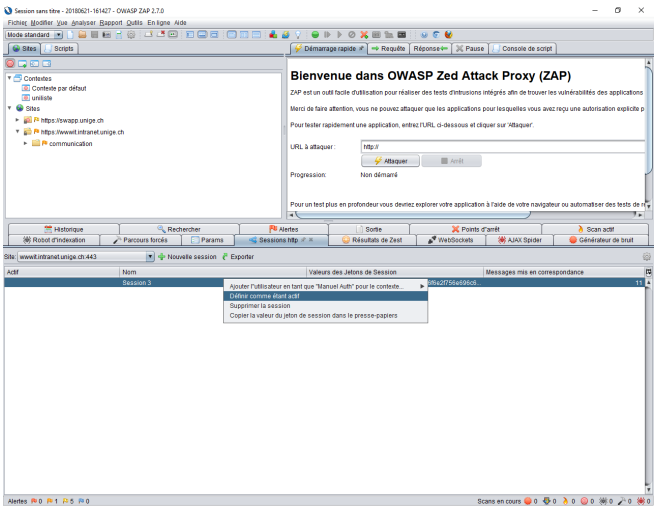
Dans l'onglet "params", sélectionner le site adéquat, identifier le cookie de session, le signaler comme jeton de session



Sélectionner l'onglet "Sessions HTTP", sélectionner le site adéquat

Rafraîchir le navigateur pour qu'il capture le jeton de session

Activer la session



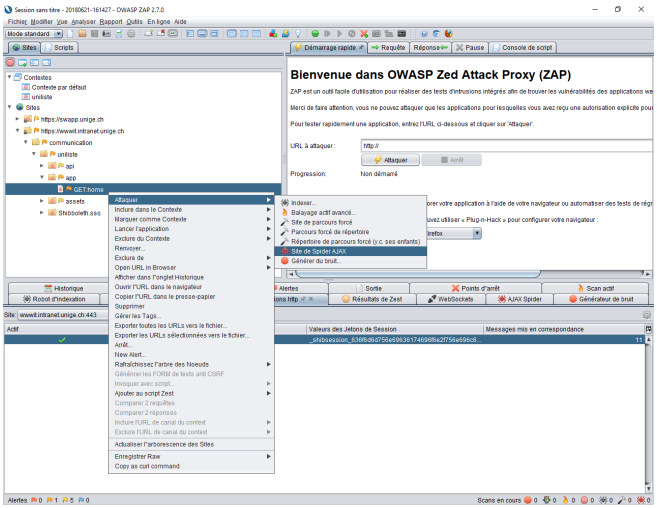
Exclure la page de logoff (pas nécessaire pour les applications full SSO)

Si une page de logoff existe la marquer comme étant hors contexte (pas nécessaire pour les applications full SSO)

Scanner (attaquer) le site (⚠ en test ⚠)

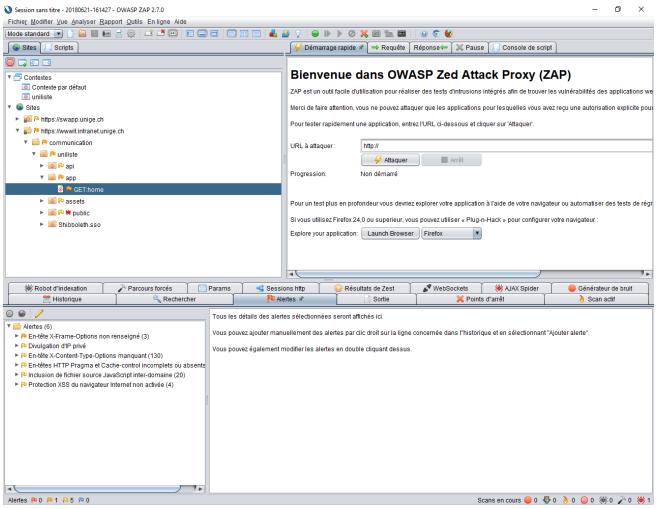
Sélectionner la page d'accueil dans l'arborescence

Pour les applications Single Page Application / AJAX utiliser le spider AJAX



Arrêter le scan en cliquant sur le bouton carré dans l'onglet à côté de "Démarrez AJAX Spider"

Aller sur l'onglet alerte pour visualiser les failles de sécurités



Ajouter une URL

Arrêter le spider

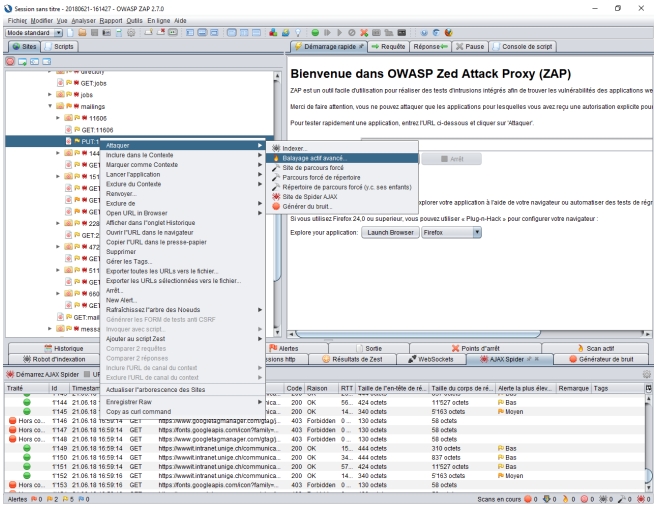
Accéder au navigateur (browser)

Utiliser le navigateur pour enregistrer une séance d'actions qui doivent être prises en compte

Attaque active d'une URL

Sélectionner une URL dans l'arborescence

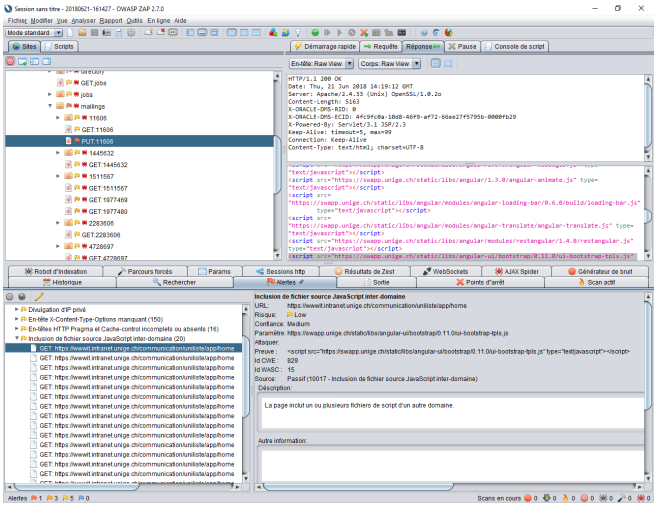
Sélection Attaquer scan actif avancé



Afficher une alerte

Sélectionner une alerte dans l'onglet alerte

Sélectionner l'onglet "requête" ou "réponse" selon le besoin



Configuration TLS et HTTP

[But du document] [Objectifs] [Artefacts] [Comment tester] [Comment configurer]

But du document

Ce document décrit comment configurer les parties SSL/TLS et HTTP pour les sites/applications web.

Objectifs

Les objectifs poursuivis sont d'obtenir

Domaine	Note	Remarques
TLS	A+	Avec l'outil SSL LABS
HTTP	(B)	Concernant la configuration HTTP, il n'y a pas de note minimale. L'objectif est de se rapprocher le plus possible de la note maximum compte tenu des contraintes applicatives. Pour les applications développées en interne, l'objectif est d'obtenir au moins la note de B. L'outil pour évaluer la configuration est Mozilla Observatory.

Artefacts

Nom	URL	Commentaires
SSL Labs	https://www.ssllabs.com/ssltest/analyze.html	Permet de tester la configuration SSL d'un site
	https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices	SSL and TLS Deployment Best Practices · ssllabs/research Wiki · GitHub
Observatory	https://observatory.mozilla.org/	▪ Test HTTP et SSL ▪ Pour les applications en accès public
SSL Configuration Generator	https://ssl-config.mozilla.org/	Outil de configuration pour la partie SSL.
Module Apache mod_headers	https://httpd.apache.org/docs/current/fr/mod/mod_headers.html	Personnalisation des en-têtes de requêtes et de réponses HTTP pour le serveur web Apache.

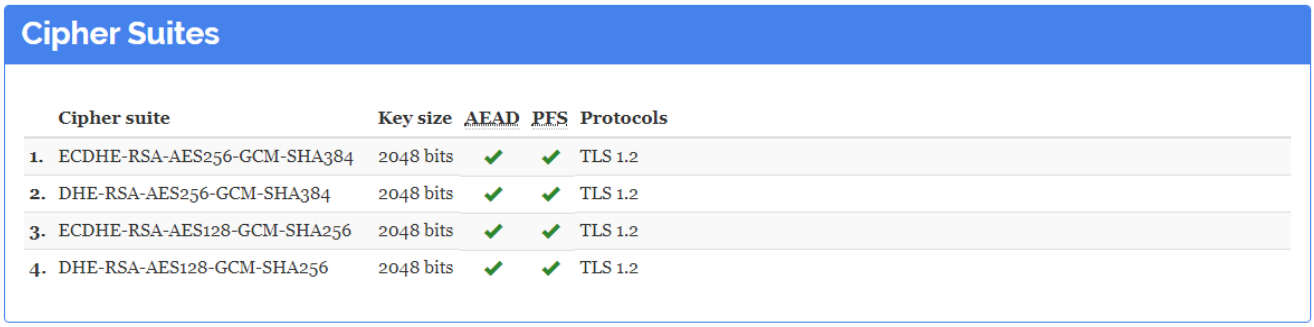
Comment tester

TLS

Les applications ayant un accès public sont à tester avec SSL Labs.

Les applications sur réseau privé sont à tester avec Observatory.

Exemple de résultat avec Observatory montrant que seul TLS 1.2 est activé

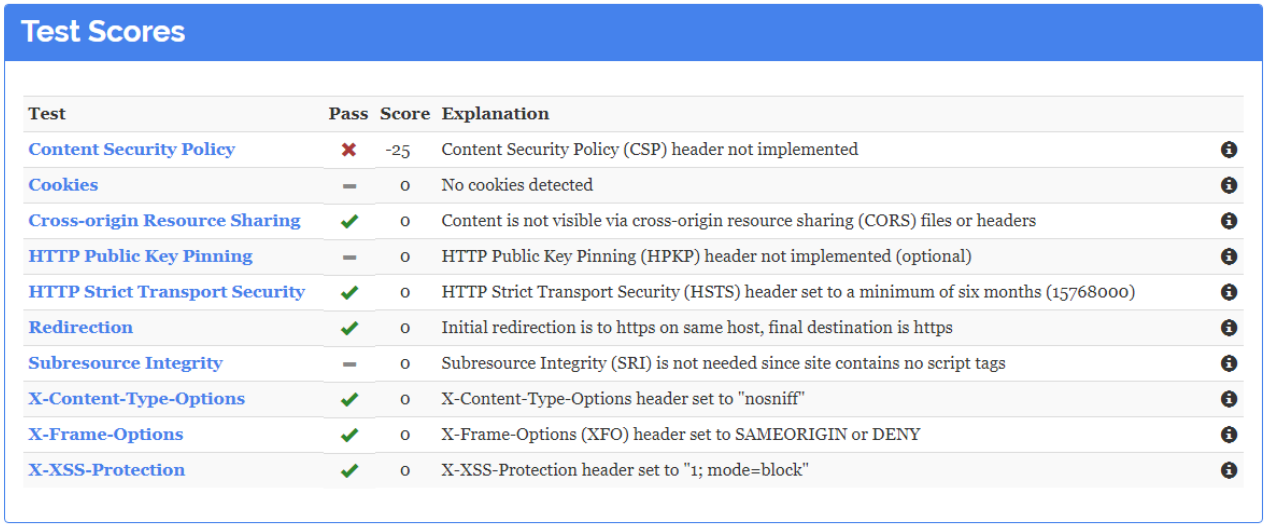


HTTP

Les applications sur réseau public sont à tester avec https://observatory.mozilla.org.

Les applications sur réseau privé sont à tester avec http://http-observatory.unige.ch. **(ne fonctionne plus => sujet stage)**

Exemple de résultat



Test Scores

Test	Pass	Score	Explanation
Content Security Policy	✖	-25	Content Security Policy (CSP) header not implemented
Cookies	–	0	No cookies detected
Cross-origin Resource Sharing	✔	0	Content is not visible via cross-origin resource sharing (CORS) files or headers
HTTP Public Key Pinning	–	0	HTTP Public Key Pinning (HPKP) header not implemented (optional)
HTTP Strict Transport Security	✔	0	HTTP Strict Transport Security (HSTS) header set to a minimum of six months (15768000)
Redirection	✔	0	Initial redirection is to https on same host, final destination is https
Subresource Integrity	–	0	Subresource Integrity (SRI) is not needed since site contains no script tags
X-Content-Type-Options	✔	0	X-Content-Type-Options header set to "nosniff"
X-Frame-Options	✔	0	X-Frame-Options (XFO) header set to SAMEORIGIN or DENY
X-XSS-Protection	✔	0	X-XSS-Protection header set to "1; mode=block"

Comment configurer

HSTS

L'outil SSL Configuration Generator permet de générer la configuration SSL pour la plupart des serveurs web du marché.

- Aller sur https://ssl-config.mozilla.org/
- Choisir la configuration "Modern" ou éventuellement "Intermediate"
- Faire attention au choix de l'option "Http Strict Transport Security" qui est activé par défaut.

HTTP

Les en-têtes HTTP manquants peuvent être ajoutés au niveau du serveur web. Pour Apache utilisé le module mod_headers.

Les recommandations sont à appliquer en fonction des besoins applicatifs.

A noter que

- L'en-tête HSTS représente un risque. Si le certificat n'est pas conforme, le site est banni au niveau du navigateur. Il est difficile de revenir en arrière. L'en-tête est à activer que si les certificats sont bien gérés. Pour plus d'information voir HTTP Strict Transport Security.
- L'en-tête HPKP est complexe, il ne doit être activé que pour les sites les plus exigeants en termes de sécurité. Pour l'essentiel des sites, il est préférable de ne pas l'activer. Pour plus d'information voir Public Key Pinning Extension for HTTP (HPKP)
- L'en-tête CSP peut être activé en mode reporting pour valider dans un premier temps la conformité. Pour plus d'information, voir Content-Security-Policy.
- Pour plus d'information, se référer à la documentation En-têtes HTTP.