OWASP
Open Web Application
Security Project

FOR DEVELOPERS

2018 v 3.0

10 Critical Security Areas That Software Developers Must Be Aware Of

PROJECT LEADERS

KATY ANTON
JIM MANICO
JIM BIRD

Sobre a OWASP

A *Open Web Application Security Project* (OWASP) é uma organização sem fins lucrativos(501c3) dedicada a permitir organizações a desenhar, desenvolver, adquirir, operar e manter seu software seguro. Todas as ferramentas, documentos, forums e capítulos são gratuitos e abertos para qualquer um interessado em melhorar segurança de aplicações. Informações sobre a OWASP podem ser encontradas em www.owasp.org.

OWASP é um novo tipo de organização. Nossa liberdade comercial nos permite fornecer informações sobre segurança de aplicações que são práticas, com bom custo benefício e não-enviesadas.

OWASP não é afiliada com nenhuma empresa de tecnologia. Similarmente a diversos projetos *open-source*, OWASP produz vários tipos de materiais de forma colaborativa e aberta. A fundação OWASP é uma organização sem fins lucrativos que garante o sucesso dos projetos a longo prazo.

APRESENTAÇÃO

Software inseguro esta minando nossa estrutura financeira, sanitária, de defesa, energética e de outras críticas áreas de forma global. A medida em que nossa estrutura digital e global aumenta em complexidade e se interconecta a dificuldade em garantir uma aplicação segura aumenta exponencialmente. Nós não podemos mais nos dar o luxo de tolerar nem os problemas mais simples de segurança.

OBJETIVOS

O objetivo do OWASP Top 10 Controles Proativos de Projeto é despertar consciência sobre segurança de aplicações descrevendo as áreas mais preocupantes que desenvolvedores de software devem estar cientes. Nós encorajemos você a usar os Controles Proativos da OWASP para iniciar os seus desenvolvedores em segurança de aplicações. Desenvolvedores podem aprender a partir de erros de outras organizações. Nós esperamos que os Controles Proativos da OWASP sejam úteis ao seus esforços em criar um software seguro.

CONTATO

Por favor não hesite em entrar em contato com o projeto sobre Controle Proativos da OWASP com perguntas, comentários e ideias, seja pela nossa lista de email ou de forma privada através de jim@owasp.org.

COPYRIGHT E LICENÇA

LIDERES DE PROJETO

Katy Anton
Jim Bird
Jim Manico

CONTRIBUIDORES

Chris Romeo
Dan Anderson
David Cybuck
Dave Ferguson
Josh Grossman
Osama Elnaggar
Colin Watson
Rick Mitchell
And many more...

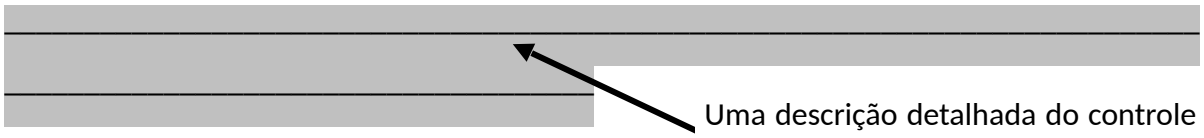
TRADUÇÃO

Wallace Soares

ESTRUTURA DO DOCUMENTO

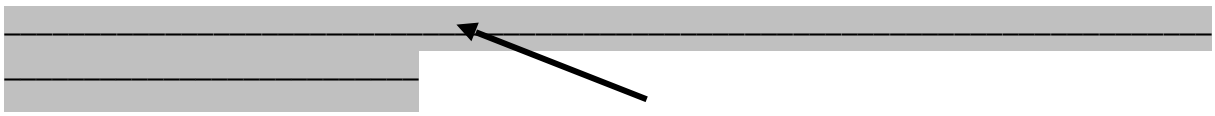
Este documento é estruturado como uma lista de controles de segurança. Cada controle é descrita como segue:

Descrição



Uma descrição detalhada do controle incluindo algumas boa práticas que devem ser consideradas.

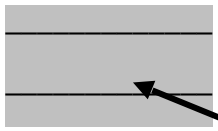
Implementação



Melhores práticas de implementação e exemplos para ilustrar como implementar cada controle.

Vulnerabilidades Prevenidas

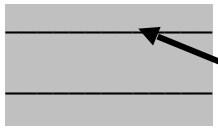
-
-



Lista de vulnerabilidades prevenidas e riscos endereçados. (OWASP TOP 10, CWE, etc.)

Referências

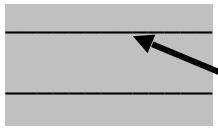
-
-



Lista de referências para estudos(OWASP Cheat sheet, Guias para garantir segurança, etc.)

Ferramentas

-
-



Conjunto de ferramentas/projetos que facilmente introduzem/integram controles de segurança no seu software.

INTRODUÇÃO

O OWASP Top 10 Controles Proativos 2018 é uma lista de técnicas de segurança que devem ser consideradas para o desenvolvimento de todo software. Este documento é escrito por desenvolvedores para ajudar àqueles que são iniciantes em desenvolvimento seguro.

Um dos objetivos deste documento é prover um guia concreto e prático que ajude desenvolvedores a construir um software seguro. Estas técnicas devem ser aplicadas proativamente durante os momentos iniciais desenvolvimento para garantir o efetividade máxima.

Os Top 10 Controles Proativos

A lista é ordenada por ordem de importância sendo o item número 1 o mais importante:

C1: Defina os Requisitos de Segurança

C2: Aproveite os Frameworks e Bibliotecas de Segurança

C3: Mantenha o Acesso a Base de Dados Seguro

C4: Codifique e Escape os Dados

C5: Valide Todas as Entradas

C6: Implemente uma Identidade Digital

C7: Aplique Controles de Acesso

C8: Proteja os dados

C9: Implemente Logs e Monitoramento de Segurança

C10: Faça o Tratamento de Erros e Exceções

Como a lista foi criada

Esta lista foi originalmente criada pelo líder de projeto atual junto a contribuições de diversos voluntários. Este documento foi compartilhado globalmente e então até mesmo sugestões anônimas são consideradas. Centenas de mudanças foram aceitas a partir deste processo aberto à comunidade.

Publico-alvo

Este documento é primariamente escrito para desenvolvedores. Entretanto, gerentes de desenvolvimento, product owners, profissionais de QA, gerentes de projeto, e qualquer um envolvido em desenvolvimento de software pode também se beneficiar deste documento.

Como utilizar este documento

O propósito deste documento é despertar a atenção inicial por trás da construção de um software seguro. Este documento também fornecerá uma boa base de conhecimento em tópicos que ajudem a introduzir um treinamento para desenvolver um software seguro. Estes controles devem ser consistentemente e completamente em todas as aplicações. Porém este documento deve ser visto como ponto de partida e não um conjunto de técnicas e práticas. Um processo de desenvolvimento completamente seguro deve incluir requisitos a partir de padrões como o OWASP ASVS para que um conjunto de atividades de desenvolvimento de software, como OWASP SAMM e BSIMM, estejam presentes.

Link para o projeto OWASP Top 10

Os Top 10 Controles Proativos da OWASP é similar ao OWASP Top 10, porém é focado em técnicas de defesa e controle e não em riscos. Cada técnica ou controle neste documento irá mapear à um ou mais itens presente na lista do OWASP Top 10. Este mapeamento será incluído ao final de cada descrição de cada controle.

OWASP Proactive Controls v 3.0

C1: Defina os Requisitos de Segurança

Descrição

Um requisito de segurança é uma declaração de necessidade de segurança que garante que uma das muitas propriedades de segurança de software esta sendo satisfeita. Requisitos de segurança são derivados a partir de padrões da industria, leis e de um histórico de vulnerabilidade pré-existentes. Requisitos de segurança definem novas funcionalidades e adições a funcionalidades já existentes para resolver um problema de segurança especifico ou eliminar uma potencial vulnerabilidade.

Requisitos de segurança prover uma fundação de funcionalidades de segurança para uma aplicação. Ao invés de criar uma abordagem customizada para a segurança de cada aplicação, requisitos de padrões de segurança permitem à desenvolvedores reusar definições de controles de segurança e suas melhores práticas. Estes mesmo requisitos de segurança entregam soluções para problemas de segurança que já ocorreram no passado. Requisitos existem para previnir que falhas de segurança voltem a acontecer.

O OWASP ASVS

O OWASP Application Security Verification Standard (ASVS) é um catálogo de critérios de requisitos de segurança. OWASP ASVS pode ser uma fonte detalhada de requisitos de segurança para os times de desenvolvedores.

Requisitos de segurança são categorizados em diferentes pacotes baseados em funcionalidades de segurança em alto nível. Por exemplo, o ASVS contém categorias como autenticação, controle de acesso, tratamento de erros / logs e serviços web. Cada categoria contém uma coleção de requisitos que representam as melhores práticas para cada categoria criados como declarações verificáveis.

Melhorando os requisitos com estórias de usuários e casos de mal-uso

Os requisitos ASVS são declarações verificáveis que podem ser expandidas com o uso de estórias de usuários e casos de mal-uso. A vantagem do uso de estórias de usuário e casos de mal-uso é o que ligará a aplicação a exatamente o que o usuário ou um atacante pode fazer ao sistema, descrevendo também o que o sistema fornece ao usuário.

Aqui há um exemplo de expansão do um requisito ASVS 3.0.1. A partir da seção “Authentication Verification Requirements” do ASVS 3.0.1, o requisito 2.19 foca em senhas padrões.

> 2.19 Verifique que não existe senhas padrões em uso no framework da sua aplicação ou em qualquer componente usado pela mesma. (Ex.: admin/password)

Este requisito contém uma ação para verificar que não existe senhas padrão e também contém um guia explanando que nenhuma senha padrão deve estar presente na aplicação.

Uma estória de usuário foca na perspectiva de usuário, administrador, ou atacante do sistema e descreve as funcionalidades baseadas no que o usuário quer que o sistema entregue. Uma estória de usuário tem a forma de "Como usuário, eu posso fazer x, y, e z".

```
Como usuário, eu posso colocar meu usuário e senha para obter acesso à aplicação.

Como usuário, eu posso definir uma senha com no máximo 1023 caracteres.
```

Quando uma estória do usuário é focada no atacante e nas suas ações, ele é comumente referido como caso de mal-uso.

```
Como atacante, eu posso preencher um usuário e senha padrão e obter acesso.
```

Esta estória contém a mesma mensagem como um requisito tradicional do ASVS, com o adicional de detalhes a partir de usuários e atacantes que ajudam durante o teste do requisito.

Implementação

Um uso sucedido de requisitos de segurança envolve quatro passos. O processo inclui descobrir / selecionar, documentar, implementar e confirmar a correta implementação de novas funcionalidades de segurança da aplicação.

Descoberta e Seleção

O processo começa com a descoberta e seleção dos requisitos de segurança. Nesta fase, o desenvolvedor esta entendendo os requisitos de segurança a partir de padrões determinados como o ASVS e escolhe qual requisito incluir em cada *release* da aplicação. O ponto de descobrir e selecionar os requisitos é escolher um número de requisitos de segurança gerenciáveis para cada *release* ou *sprint*, e assim continuar a iteração de cada *sprint*, adicionando funcionalidades de segurança ao passar do tempo.

Investigação e documentação

Durante a investigação e documentação, o desenvolvedor revisa a aplicação existente por cima do novo conjunto de requisitos de segurança para determinar se a aplicação atinge os objetivos ou se é necessário mais desenvolvimento. A investigação alimentará a documentação a partir da revisão.

Implementação e teste

Após determinar a necessidade de desenvolvimento, o desenvolvedor deve agora modificar a aplicação de uma forma à adicionar a nova funcionalidade ou eliminar a opção insegura. Nesta fase o desenvolvedor deve primeiro determinar o design requerido para endereçar o requisito, e assim completar as modificações no código para que seja atingido os requisitos. Casos de teste devem ser criados para confirmar a existência da nova funcionalidade ou para refutar a existência da opção insegura.

Vulnerabilidades prevenidas

Requisitos de segurança definem funcionalidades de segurança de uma aplicação. Uma boa segurança criada a partir do começo da vida de uma aplicação resulta na prevenção de vários tipos de vulnerabilidades.

Referências

- OWASP Application Security Verification Standard (ASVS)
- OWASP Mobile Application Security Verification Standard (MASVS)
- OWASP Top Ten

OWASP Proactive Controls v 3.0

C2: Aproveite os Frameworks e Bibliotecas de Seguras

Descrição

Bibliotecas e frameworks incorporadas com software seguro ajudam desenvolvedores a se proteger contra falhas de design e implementação. Um desenvolvedor escrevendo uma aplicação do zero pode não ter conhecimento, tempo ou orçamento suficientes para implementar e manter propriamente funcionalidades de segurança. Se aproveitar de frameworks de segurança ajudam a cumprir estes objetivos de forma mais efetiva.

Melhores práticas

Quando incorporar bibliotecas e frameworks de terceiros no seu software, é importante considerar as seguintes melhores práticas:

1. Utilize bibliotecas e frameworks a partir de fontes seguras e que são ativamente mantidas e utilizadas por muitas aplicações.
2. Crie e mantenha um inventário com todas as bibliotecas de terceiros.
3. Proativamente mantenha as bibliotecas e componentes atualizados. Utilize ferramentas como OWASP Dependency Check e Retire.JS para identificar dependências de projeto e cheque se existem vulnerabilidades publicadas para todo código de terceiro.
4. Reduza a superficie de ataque encapsulando a biblioteca e expondo somente o comportamento requerido pelo seu software.

Vulnerabilidades prevenidas

Frameworks e bibliotecas seguras podem ajudar a previnir uma grande gama de vulnerabilidades em aplicações web. É critico manter frameworks e bibliotecas atualizadas como descrito em usando componentes com vulnerabilidades conhecidas no Top Ten 2017.

Ferramentas

- OWASP Dependency Check - identifica dependências de projeto e checa se há vulnerabilidades conhecidas.
- Retire.JS um scanner de bibliotecas JavaScript.

OWASP Proactive Controls v 3.0

C3: Acesso seguro à base de dados

Descrição

Esta seção descreve o acesso seguro a todos os banco de dados, incluindo bases de dados relacional e não-relacional. Algumas áreas para considerar:

1. Queries seguras
2. Configurações seguras
3. Autenticação segura
4. Comunicação segura

Queries seguras

SQL Injection ocorre quando uma entrada não confiável é dinamicamente adicionada à query SQL de forma insegura, e geralmente por uma concatenação básica. SQL Injection é uma dos mais perigosos riscos para uma aplicação. SQL Injection é fácil de ser explorado e pode levar a um roubo, deleção e/ou modificação de banco de dados inteiro. A aplicação pode até ser usada para rodar comandos contra o sistema operacional host e portanto dar acesso ao atacante à sua rede.

Para mitigar SQL Injection a entrada não confiável não deve ser interpretada como parte do comando SQL. A melhor forma de fazer isso é com uma técnica de programação conhecida como parametrização de query. Esta defesa deve ser aplicada a SQL, OQL e também a procedimentos de construção do banco.

Uma lista de exemplos de parametrização em ASP, ColdFusion, C#, Delphi, .NET, Go, Java, Perl, PHP, PL/SQL, PostgreSQL, Python, R, Ruby and Scheme pode ser encontrada em http://bobby-tables.com e no OWASP Cheat Sheet on Query Parameterization

Cuidados com a parametrização das queries

Algumas pontos da query não são parametrizáveis. Estes pontos são diferentes para cada banco de dados. Mantenha a validação para cada banco de dados ou escape de forma manual quando os parâmetros do banco não podem ser parametrizáveis. Enquanto, no geral, o uso de queries parametrizadas tem um impacto positivo no desempenho, algumas queries parametrizadas em algumas implementação podem afetar o desempenho de forma negativa. Garanta o teste de desempenho das queries, especialmente queries complexas como àquelas com uso extensivo de LIKE ou busca de textos.

Configuração segura

Infelizmente, sistemas de gerenciamento de banco de dados nem sempre entregam uma configuração "segura por padrão". Cuidados devem ser tomados para garantir que controles de segurança disponíveis a partir do Database Management System (DBMS) e da plataforma host estão habilitadas e propriamente configuradas. Existem padrões, guias, e benchmarks disponíveis para os DBMS mais comuns.

Autenticação segura

Todo acesso ao banco de dados deve ser propriamente autenticado. A autenticação ao DBMS deve ser adquirida de forma segura. Autenticação deve ser atingida somente através de um canal seguro. Credenciais devem ser propriamente seguras e disponíveis para uso.

Comunicação segura

Maior parte dos DBMS suportam uma variedade de métodos de comunicação (services, APIs, etc) - seguros (autenticados e encriptados) e inseguros(não autenticados e não encriptados). É uma boa prática usar somente comunicações seguras seguindo o *Protect Data Everywhere*.

Vulnerabilidades Prevenidas

- OWASP Top 10 2017- A1: Injection
- OWASP Mobile Top 10 2014-M1 Weak Server Side Controls

Referências

- OWASP Cheat Sheet: Query Parameterization
- Bobby Tables: A guide to preventing SQL injection
- CIS Database Hardening Standards

OWASP Proactive Controls v 3.0

C4: Codifique e Escape os Dados

Descrição

Codificar e escapar são técnicas de defesa que tem o objetivo de prevenir ataques de injeção. Codificar envolve traduzir caracteres especiais em comuns que sejam equivalentes e não ameacem o interpretador, por exemplo o simbolo "**<**" é traduzido para "<" quando escrito em paginas HTML.

Escapar envolve adicionar um caractere especial antes do caractere/string para evitar que seja mal-interpretado, por exemplo, adicionando um "\" antes de "**"**"(aspas duplas) evita que o caractere seja interpretado como fim de string.

Codificação é melhor quando aplicada ao conteúdo que é passado ao interpretador. Se esta forma de defesa for aplicada muito antes no processamento de um request a codificação ou o código escapado pode interferir no uso do conteúdo em outras partes do programa. Por exemplo se seu HTML escapa o conteúdo antes de salva-lo e sua UI automaticamente escapa o dado uma segunda vez então o conteúdo não será exibido de forma correta por ter sido escapado duas vezes.

Codificação de saída contextual

A codificação de saída contextual é uma técnica de programação de segurança crucial necessária para interromper o XSS. Essa defesa é executada na saída, quando você está construindo uma interface de usuário, no último momento antes que dados não confiáveis sejam adicionados dinamicamente ao HTML. O tipo de codificação dependerá da localização (ou contexto) no documento onde os dados estão sendo exibidos ou armazenados. Os diferentes tipos de codificação que seriam usados para construir interfaces de usuário seguras incluem Codificação de Entidade HTML, Codificação de Atributo HTML, Codificação JavaScript e Codificação de URL.

Exemplos de codificação em Java

Para exemplos da OWASP Java Encoder veja: OWASP Java Encoder Project Examples

Exemplos de codificação em .NET

Começando com o .NET 4.5, a biblioteca de Anti-Cross Site Scripting é parte do framework, mas não é habilitada por padrão. Você pode especificamente utilizar o AntiXssEncoder desta biblioteca como codificador padrão para toda a sua aplicação usando a configuração web.conf. Quando aplicado é importante codificar contextualmente sua saída - isto significa utilizar a função correta do AntiXSSEncoder apropriadamente para cada dado.

Exemplos de codificação em PHP

Zend Framework 2

No framework Zend 2 (ZF2), Zend\Escaper pode ser utilizado para codificar a saída. Para exemplos de codificação contextual veja Context-specific escaping with zend-escaper.

Outros tipos de codificação e defesa de injeção

Codificar/Escapar pode ser utilizado para neutralizar o conteúdo contra outras formas de injeção. Por exemplo, é possível neutralizar certos meta-caracteres especiais adicionando uma entrada a um comando do sistema operacional. Isto é chamado de "OS command escaping" ou "shell escaping". Esta forma de defesa pode ser usada para vulnerabilidades do tipo "Command Injection".

Existem outras formas de escape que podem ser utilizadas para se defender contra injeção de conteúdo como escape de atributos XML que impede varias formas XML e injeção de caminhos XML, assim como também escape de nomes em LDAP que podem ser utilizados para prevenção de várias formas de injeção em LDAP.

Codificação de caracteres e canonicalização

Codificação Unicode é uma método de salvar caracteres em múltiplos bytes. Sempre que uma entrada é permitida, o dado pode ser colocado usando Unicode para disfarçar código malicioso e permitir uma variedade de ataques. RFC 2279 referencia várias formas de como o texto pode ser codificado.

Canonicalização é um método em sistemas convertem os dados em formas simples ou padronizadas. Aplicações web comumente usam caracteres canonicalizados para garantir que todo o conteúdo possui o mesmo tipo de caractere quando salvo ou exibido.

Para se prevenir contra ataques relacionados a canonicalização uma aplicação deve estar protegida quando Unicodes e outros tipos de entrada mal formadas são dadas como entrada.

Vulnerabilidades prevenidas

- OWASP Top 10 2017 - A1: Injection
- OWASP Top 10 2017 - A7: Cross Site Scripting (XSS)
- OWASP Mobile_Top_10_2014-M7 Client Side Injection

Referência

- XSS - Informações gerais
- OWASP Cheat Sheet: XSS Prevention - Parando o XSS em seu aplicativo da web
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: Injection Prevention

Ferramentas

- OWASP Java Encoder Project
- AntiXSSEncoder
- Zend\Escaper - exemplos de codificação contextual

OWASP Proactive Controls v 3.0

C5: Valide todas as entradas

Descrição

A validação de entrada é uma técnica de programação que garante que apenas dados formatados corretamente possam entrar em um componente do sistema de software.

Validade de Sintaxe e Semântica

Uma aplicação deve verificar se os dados são sintaticamente e semanticamente válidos (nessa ordem) antes de usá-los de qualquer forma (incluindo exibi-los de volta ao usuário).

Validade de sintaxe significa que os dados estão no formato esperado. Por exemplo, uma aplicação pode permitir que um usuário selecione um “ID de conta” de quatro dígitos para realizar algum tipo de operação. A aplicação deve assumir que o usuário está inserindo uma injeção SQL e deve verificar se os dados inseridos pelo usuário têm exatamente quatro dígitos e consistem apenas em números (além de utilizar a parametrização de queries adequada).

A validade semântica inclui apenas a aceitação de entrada que esteja dentro de um intervalo aceitável para a funcionalidade e o contexto da aplicação fornecido. Por exemplo, uma data de início deve ser anterior a uma data de término ao escolher intervalos de datas.

Allowlisting vs Blocklisting

Existem duas abordagens gerais para realizar a validação da sintaxe de entrada, comumente conhecidas como blocklist e allowlist:

- A blocklist ou validação de blocklist tenta verificar se os dados fornecidos não contêm conteúdo "malicioso". Por exemplo, uma aplicação da Web pode bloquear a entrada que contém o texto exato <SCRIPT> para ajudar a evitar XSS. No entanto, essa defesa pode ser evitada com uma tag de script em minúsculas ou uma tag de script de maiúsculas e minúsculas.
- A allowlist ou validação de allowlist tenta verificar se um dado dado corresponde a um conjunto de regras conhecidas como “válidas”. Por exemplo, uma regra de validação de allowlist para um estado dos EUA seria um código de 2 letras que é apenas um dos estados dos EUA válidos.

Ao criar software seguro, a lista de permissões é a abordagem mínima recomendada. A lista negra é propensa a erros e pode ser contornada com várias técnicas de evasão e pode ser perigosa quando dependente de si mesma.

Embora a blocklist possa ser evitada, muitas vezes pode ser útil para ajudar a detectar ataques óbvios. Portanto, enquanto a allowlist ajuda a limitar a superfície de ataque, garantindo que os dados tenham a validade sintática e semântica correta, a blacklist ajuda a detectar e potencialmente interromper ataques óbvios.

Validação do lado do cliente e do lado do servidor

A validação de entrada deve sempre ser feita no lado do servidor para segurança. Embora a validação do lado do cliente possa ser útil para fins funcionais e de segurança, ela pode ser facilmente "*bypassada*". Isso torna a validação do lado do servidor ainda mais fundamental para a segurança. Por exemplo, a validação de JavaScript pode alertar o usuário de que um determinado campo deve consistir em números, mas a aplicação do lado do servidor deve validar que os dados enviados consistem apenas em números no intervalo numérico apropriado para esse recurso.

Expressões regulares

As expressões regulares oferecem uma maneira de verificar se os dados correspondem a um padrão específico. Vamos começar com um exemplo básico.

A expressão regular a seguir é usada para definir uma regra de lista de permissões para validar nomes de usuário.

```
^[a-z0-9_]{3,16}$
```

Essa expressão regular permite apenas letras minúsculas, números e o caractere sublinhado. O nome de usuário também é restrito a um comprimento de 3 e 16 caracteres.

Cuidado: Potencial de negação de serviço

Deve-se ter cuidado ao criar expressões regulares. Expressões mal projetadas podem resultar em possíveis condições de negação de serviço (também conhecidas como ReDoS). Várias ferramentas podem testar para verificar se as expressões regulares não são vulneráveis ao ReDoS.

Atenção: Complexidade

As expressões regulares são apenas uma maneira de realizar a validação. Expressões regulares podem ser difíceis de manter ou entender para alguns desenvolvedores. Outras alternativas de validação envolvem escrever métodos de validação programaticamente que podem ser mais fáceis de manter para alguns desenvolvedores.

Limites de validação de entrada

A validação de entrada nem sempre torna os dados “seguros”, pois certas formas de entrada complexa podem ser “válidas”, mas ainda assim perigosas. Por exemplo, um endereço de e-mail válido pode conter um ataque de injeção SQL ou um URL válido pode conter um ataque Cross Site Scripting. Defesas adicionais além da validação de entrada devem sempre ser aplicadas a dados como parametrização de query ou escape.

Desafios da validação de dados serializados

Algumas formas de entrada são tão complexas que a validação só pode proteger minimamente uma aplicação. Por exemplo, é perigoso desserializar dados não confiáveis ou dados que podem ser manipulados por um invasor. O único padrão de arquitetura seguro é não aceitar objetos serializados de fontes não confiáveis ou apenas desserializar em capacidade limitada apenas para tipos de dados simples. Você deve evitar o processamento de formatos de dados serializados e usar formatos mais fáceis de defender, como JSON, quando possível.

Se isso não for possível, considere uma série de defesas de validação ao processar dados serializados.

- Implemente verificações de integridade ou criptografia dos objetos serializados para evitar a criação de objetos hostis ou adulteração de dados.
- Imponha restrições de tipo estritas durante a desserialização antes da criação do objeto; normalmente o código espera um conjunto de classes definível. Os desvios para esta técnica foram demonstrados.
- Isole o código que desserializa, de modo que seja executado em ambientes com privilégios muito baixos, como contêineres temporários.
- Log exceções e falhas de desserialização, como quando o tipo de entrada não é o tipo esperado ou a desserialização gera exceções.
- Restrinja ou monitore a conectividade de rede de entrada e saída de contêineres ou servidores que desserializam.
- Monitore a desserialização, alertando se um usuário desserializar constantemente.

Entrada inesperada do usuário (atribuição em massa)

Algumas estruturas oferecem suporte à associação automática de parâmetros de solicitações HTTP a objetos do lado do servidor usados pela aplicação. Esse recurso de ligação automática pode permitir que um invasor atualize objetos do lado do servidor que não deveriam ser modificados. O invasor pode modificar seu nível de controle de acesso ou contornar a lógica de negócios pretendida da aplicação com esse recurso.

Este ataque tem vários nomes, incluindo: atribuição em massa, ligação automática e injeção de objetos.

Como um exemplo simples, se o objeto de usuário tiver um privilégio de campo que especifica o nível de privilégio do usuário na aplicação, um usuário mal-intencionado pode procurar páginas onde os dados do usuário são modificados e adicionar privilégio=admin aos parâmetros HTTP enviados. Se a vinculação automática estiver habilitada de maneira insegura, o objeto do lado do servidor que representa o usuário será modificado de acordo.

Duas abordagens podem ser usadas para lidar com isso:

- Evite vincular a entrada diretamente e use objetos de transferência de dados (DTOs).
- Ative a vinculação automática, mas configure regras de lista de permissões para cada página ou recurso para definir quais campos podem ser vinculados automaticamente.

Mais exemplos estão disponíveis no OWASP Mass Assignment Cheat Sheet.

Validando e Sanitizando HTML

Considere uma aplicação que precisa aceitar HTML de usuários (por meio de um editor WYSIWYG que representa conteúdo como HTML ou recursos que aceitam HTML diretamente na entrada). Nesta situação, validação ou escape não ajudará.

- Expressões regulares não são expressivas o suficiente para entender a complexidade do HTML5.
- Codificar ou escapar HTML não ajudará, pois fará com que o HTML não seja renderizado corretamente.

Portanto, você precisa de uma biblioteca que possa analisar e limpar texto formatado em HTML. Consulte XSS Prevention Cheat Sheet on HTML Sanitization para obter mais informações sobre sanitização de HTML.

Funcionalidade de validação em bibliotecas e frameworks

Todas as linguagens e a maioria dos frameworks fornecem bibliotecas ou funções de validação que devem ser aproveitadas para validar dados. As bibliotecas de validação geralmente cobrem tipos de dados comuns, requisitos de comprimento, intervalos de números inteiros, verificações de "é nulo" e muito mais. Muitas bibliotecas e estruturas de validação permitem que você defina sua própria expressão regular ou lógica para validação personalizada de uma maneira que permite que o programador aproveite essa funcionalidade em toda a aplicação.

Exemplos de funcionalidade de validação incluem as funções de filtro do PHP ou o Hibernate Validator for Java. Exemplos de HTML Sanitizers incluem o método de sanitização Ruby on Rails, OWASP Java HTML Sanitizer ou DOMPurify.

Vulnerabilidades evitadas

- A validação de entrada reduz a superfície de ataque das aplicações e às vezes pode dificultar os ataques contra uma aplicação.
- A validação de entrada é uma técnica que fornece segurança a certas formas de dados, específicas para determinados ataques e não pode ser aplicada de forma confiável como uma regra geral de segurança.
- A validação de entrada não deve ser usada como o principal método de prevenção de XSS, SQL Injection e outros ataques.

Referências

- OWASP Cheat Sheet: Input Validation
- OWASP Cheat Sheet: iOS - Security Decisions via Untrusted Inputs
- OWASP Testing Guide: Testing for Input Validation

Ferramentas

- OWASP Java HTML Sanitizer Project
- Java JSR-303/JSR-349 Bean Validation
- Java Hibernate Validator
- JEP-290 Filter Incoming Serialization Data
- Apache Commons Validator
- PHP's filter functions

OWASP Proactive Controls v 3.0

C6: Implementar Identidade Digital

Descrição

A Identidade Digital é a representação única de um usuário (ou outro sujeito) enquanto se envolve em uma transação online. Autenticação é o processo de verificar se um indivíduo ou entidade é quem eles afirmam ser. O gerenciamento de sessão é um processo pelo qual um servidor mantém o estado de autenticação dos usuários para que o usuário possa continuar a usar o sistema sem reautenticar. A Publicação Especial NIST 800-63B: Diretrizes de Identidade Digital (Autenticação e Gerenciamento do Ciclo de Vida) fornece orientações sólidas sobre a implementação de controles de identidade digital, autenticação e gerenciamento de sessão.

Abaixo estão algumas recomendações para implementação segura.

Níveis de autenticação

O NIST 800-63b descreve três níveis de garantia de autenticação chamados de nível de garantia de autenticação (AAL). AAL nível 1 é reservado para aplicações de baixo risco que não contêm PII ou outros dados privados. No nível 1 da AAL, apenas a autenticação de fator único é necessária, normalmente por meio do uso de uma senha.

Level 1 : Senhas

As senhas são realmente muito importantes. Precisamos de política, precisamos armazená-los com segurança, às vezes precisamos permitir que os usuários os redefinam.

Requisitos de senha

As senhas devem cumprir, no mínimo, os seguintes requisitos:

- ter pelo menos 8 caracteres se a autenticação multi-fator (MFA) e outros controles também forem usados. Se a MFA não for possível, ela deve ser aumentada para pelo menos 10 caracteres
- todos os caracteres ASCII de impressão, bem como o caractere de espaço, devem ser aceitáveis em segredos memorizados
- incentivar o uso de senhas longas
- remover requisitos de complexidade, pois estes foram considerados de eficácia limitada. Em vez disso, recomenda-se a adoção de MFA ou comprimentos de senha mais longos.

Implementar mecanismo de recuperação de senha segura

É comum que uma aplicação tenha um mecanismo para um usuário obter acesso à sua conta caso esqueça sua senha. Um bom fluxo de design para um recurso de recuperação de senha usará elementos de autenticação multi-fator. Por exemplo, ele pode fazer uma pergunta de segurança - algo que eles sabem e, em seguida, enviar um token gerado para um dispositivo - algo que eles possuem.

Veja Forgot_Password_Cheat_Sheet e Choosing_and_Using_Security_Questions_Cheat_Sheet para mais detalhes.

Implementar armazenamento seguro de senha

Para fornecer controles de autenticação fortes, uma aplicação deve armazenar com segurança as credenciais do usuário. Além disso, controles criptográficos devem ser implementados de forma que, se uma credencial (por exemplo, uma senha) for comprometida, o invasor não tenha acesso imediato a essas informações.

Exemplo de PHP para armazenamento de senha

Abaixo está um exemplo de hash de senha segura em PHP usando a função password_hash() (disponível desde 5.5.0) que usa por padrão o algoritmo bcrypt. O exemplo usa um fator de trabalho de 15.

```
<?php
  $cost = 15;
  $password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" =>
$cost] );
?>
```

Por favor veja o OWASP Password Storage Cheat Sheet para mais detalhes

Nível 2: Autenticação multi-fator

O NIST 800-63b AAL nível 2 é reservado para aplicações de alto risco que contêm "PII auto-declaradas ou outras informações pessoais disponibilizadas on-line". No nível 2 da AAL, a autenticação multi-fator é necessária, incluindo OTP ou outras formas de implementação multi-fator. A autenticação multi-fator (MFA) garante que os usuários sejam quem afirmam ser, exigindo que eles se identifiquem com uma combinação de:

- Algo que você sabe – senha ou PIN
- Algo que você possui – token ou telefone
- Algo que você é – biometria, como uma impressão digital

O uso de senhas como único fator fornece segurança fraca. As soluções multi-fatores fornecem uma solução mais robusta, exigindo que um invasor adquira mais de um elemento para se autenticar no serviço.

Vale ressaltar que a biometria, quando empregada como fator único de autenticação, não são considerados segredos aceitáveis para autenticação digital. Eles podem ser obtidos on-line ou tirando uma foto de alguém com um telefone com câmera (por exemplo, imagens faciais) com ou sem seu conhecimento, levantadas de objetos que alguém toca (por exemplo, impressões digitais latentes) ou capturadas com imagens de alta resolução (por exemplo, íris padrões). A biometria deve ser usada apenas como parte da autenticação multifator com um autenticador físico (algo que você possui). Por exemplo, acessar um dispositivo de senha de uso único (OTP) multi-fator que gerará uma senha de uso único que o usuário insere manualmente para o verificador.

Nível 3: Autenticação baseada em criptografia

NIST 800-63b Authentication Assurance Level 3 (AAL3) é necessária quando o impacto de sistemas comprometidos pode levar a danos pessoais, perdas financeiras significativas, prejudicar o interesse público ou envolver violações civis ou criminais. AAL3 requer autenticação "baseada na prova de posse de uma chave por meio de um protocolo criptográfico". Esse tipo de autenticação é usado para obter o nível mais forte de garantia de autenticação. Isso geralmente é feito por meio de módulos criptográficos de hardware.

Gerenciamento de sessão

Uma vez que a autenticação inicial bem-sucedida do usuário tenha ocorrido, uma aplicação pode optar por rastrear e manter esse estado de autenticação por um período limitado de tempo. Isso permitirá que o usuário continue usando a aplicação sem precisar manter a re-autenticação a cada solicitação. O rastreamento desse estado de usuário é chamado de Gerenciamento de Sessão.

Session Generation and Expiration

O estado do usuário é rastreado em uma sessão. Esta sessão é normalmente armazenada no servidor para gerenciamento de sessão tradicional baseado na web. Um identificador de sessão é então fornecido ao usuário para que o usuário possa identificar qual sessão do lado do servidor contém os dados de usuário corretos. O cliente só precisa manter esse identificador de sessão, que também mantém os dados confidenciais da sessão do lado do servidor fora do cliente.

Aqui estão alguns controles a serem considerados ao criar ou implementar soluções de gerenciamento de sessão:

- Certifique-se de que o ID da sessão seja longo, único e aleatório.
- A aplicação deve gerar uma nova sessão ou pelo menos alternar o ID da sessão durante a autenticação e a re-autenticação.

- A aplicação deve implementar um tempo limite de inatividade após um período de inatividade e um tempo de vida máximo absoluto para cada sessão, após o qual os usuários devem se autenticar novamente. A duração dos tempos limite deve ser inversamente proporcional ao valor dos dados protegidos.

Por favor, veja mais detalhes em Session Management Cheat Sheet. A Seção 3 do ASVS cobre requisitos adicionais de gerenciamento de sessão.

Cookies do navegador

Os cookies do navegador são um método comum para a aplicação Web armazenar identificadores de sessão para aplicações Web que implementam técnicas padrão de gerenciamento de sessão. Aqui estão algumas defesas a serem consideradas ao usar cookies do navegador.

- Quando os cookies do navegador são usados como mecanismo para rastrear a sessão de um usuário autenticado, eles devem ser acessíveis a um conjunto mínimo de domínios e caminhos e devem ser marcados para expirar no período de validade da sessão ou logo após.
- A flag 'secure' deve ser definida para garantir que a transferência seja feita somente via canal seguro (TLS).
- A flag HttpOnly deve ser definida para evitar que o cookie seja acessado via JavaScript.
- A adição de atributos “samesite” aos cookies impede que alguns navegadores modernos enviem cookies com solicitações entre sites e oferece proteção contra falsificação de solicitações entre sites e ataques de vazamento de informações.

Tokens

As sessões do lado do servidor podem ser limitantes para algumas formas de autenticação. "Serviços sem estado" permitem o gerenciamento de dados de sessão do lado do cliente para fins de desempenho, de modo que o servidor tenha menos problemas para armazenar e verificar a sessão do usuário. Essas aplicações "sem estado" geram um token de acesso de curta duração que pode ser usado para autenticar uma solicitação de cliente sem enviar as credenciais do usuário após a autenticação inicial.

JWT (JSON Web Tokens)

JSON Web Token (JWT) é um padrão aberto (RFC 7519) que define uma maneira compacta e independente de transmitir informações com segurança entre as partes como um objeto JSON.

Essas informações podem ser verificadas e confiáveis porque são assinadas digitalmente. Um token JWT é criado durante a autenticação e é verificado pelo servidor (ou servidores) antes de qualquer processamento.

No entanto, os JWTs geralmente não são salvos pelo servidor após a criação inicial. Os JWTs geralmente são criados e entregues a um cliente sem serem salvos pelo servidor de forma alguma. A integridade do token é mantida através do uso de assinaturas digitais para que um servidor possa verificar posteriormente se o JWT ainda é válido e não foi adulterado desde sua criação.

Essa abordagem é sem estado e portátil, pois as tecnologias de cliente e servidor podem ser diferentes e ainda assim interagir.

Cuidado

Identidade digital, autenticação e gerenciamento de sessão são tópicos muito grandes. Estamos arranhando a superfície do tópico da Identidade Digital aqui. Garanta que seu talento de engenharia mais capaz seja responsável por manter a complexidade envolvida com a maioria das soluções de identidade.

Vulnerabilidades evitadas

- OWASP Top 10 2017 A2- Broken Authentication and Session Management
- OWASP Mobile Top 10 2014-M5- Poor Authorization and Authentication

Referências

- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Choosing and Using Security Questions
- OWASP Cheat Sheet: Session Management
- OWASP Cheat Sheet: IOS Developer
- OWASP Testing Guide: Testing for Authentication
- NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines

Ferramentas

- Daniel Miessler: Most commonly found passwords

OWASP Proactive Controls v 3.0

C7: Aplique controles de acesso

Descrição

O Controle de Acesso (ou Autorização) é o processo de conceder ou negar solicitações específicas de um usuário, programa ou processo. O controle de acesso também envolve o ato de conceder e revogar esses privilégios.

Deve-se notar que autorização (verificação de acesso a recursos ou recursos específicos) não é equivalente a autenticação (verificação de identidade).

A funcionalidade de controle de acesso geralmente abrange muitas áreas de software, dependendo da complexidade do sistema de controle de acesso. Por exemplo, gerenciar metadados de controle de acesso ou criar cache para fins de escalabilidade geralmente são componentes adicionais em um sistema de controle de acesso que precisam ser criados ou gerenciados.

Existem vários tipos diferentes de projeto de controle de acesso que devem ser considerados.

- O controle de acesso discricionário (DAC) é um meio de restringir o acesso a objetos (por exemplo, arquivos, entidades de dados) com base na identidade e necessidade de conhecimento de assuntos (por exemplo, usuários, processos) e/ou grupos aos quais o objeto pertence .
- O Mandatory Access Control (MAC) é um meio de restringir o acesso aos recursos do sistema com base na sensibilidade (representada por um rótulo) das informações contidas no recurso do sistema e na autorização formal (ou seja, liberação) dos usuários para acessar informações de tais sensibilidade.
- O controle de acesso baseado em função (RBAC) é um modelo para controlar o acesso a recursos em que as ações permitidas nos recursos são identificadas com funções em vez de identidades de assunto individuais.
- O controle de acesso baseado em atributos (ABAC) concederá ou negará solicitações de usuários com base em atributos arbitrários do usuário e atributos arbitrários do objeto, e condições ambientais que podem ser globalmente reconhecidas e mais relevantes para as políticas em questão.

Princípios de Design de Controle de Acesso

Os seguintes requisitos "positivos" de projeto de controle de acesso devem ser considerados nos estágios iniciais do desenvolvimento de uma aplicação.

1) Projete o controle de acesso completamente na frente

Depois de escolher um padrão de design de controle de acesso específico, geralmente é difícil e demorado reprojetar o controle de acesso em sua aplicação com um novo padrão. O controle de acesso é uma das principais áreas do projeto de segurança de aplicações que deve ser totalmente projetada desde o início, especialmente ao abordar requisitos como multi-locação e controle de acesso horizontal (dependente de dados).

O projeto de controle de acesso pode começar simples, mas muitas vezes pode se transformar em um controle de segurança complexo e com muitos recursos. Ao avaliar a capacidade de controle de acesso de estruturas de software, certifique-se de que sua funcionalidade de controle de acesso permita a personalização para sua necessidade específica de recurso de controle de acesso.

2) Forçar todas as solicitações a passarem pelas verificações de controle de acesso

Certifique-se de que todas as solicitações passem por algum tipo de camada de verificação de controle de acesso. Tecnologias como filtros Java ou outros mecanismos de processamento automático de solicitações são artefatos de programação ideais que ajudarão a garantir que todas as solicitações passem por algum tipo de verificação de controle de acesso.

3) Negar por padrão

Negar por padrão é o princípio de que, se uma solicitação não for especificamente permitida, ela será negada. Há muitas maneiras pelas quais essa regra se manifestará no código da aplicação. Alguns exemplos destes são:

1. O código da aplicação pode gerar um erro ou exceção ao processar solicitações de controle de acesso. Nesses casos, o controle de acesso deve ser sempre negado.
2. Quando um administrador cria um novo usuário ou um usuário se registra em uma nova conta, essa conta deve ter acesso mínimo ou nenhum acesso por padrão até que o acesso seja configurado.
3. Quando um novo recurso é adicionado a uma aplicação, todos os usuários devem ser impedidos de usar esse recurso até que ele seja configurado corretamente.

4) Princípio do Mínimo Privilégio

Certifique-se de que todos os usuários, programas ou processos recebam apenas o mínimo ou o mínimo de acesso necessário possível. Desconfie de sistemas que não fornecem recursos de configuração de controle de acesso granular.

5) Não codifique funções

Muitas estruturas de aplicações usam como padrão o controle de acesso baseado em função. É comum encontrar código de aplicação que é preenchido com verificações dessa natureza.

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {
      deleteAccount();
}
```

Tenha cuidado com esse tipo de programação baseada em função no código. Tem as seguintes limitações ou perigos.

- A programação baseada em papéis dessa natureza é frágil. É fácil criar verificações de função incorretas ou ausentes no código.
- A programação baseada em função não permite multi-locação. Medidas extremas, como bifurcação do código ou verificações adicionais para cada cliente, serão necessárias para permitir que sistemas baseados em funções tenham regras diferentes para clientes diferentes.
- A programação baseada em função não permite regras de controle de acesso horizontais ou específicas de dados.
- Bases de código grandes com muitas verificações de controle de acesso podem ser difíceis de auditar ou verificar a política geral de controle de acesso da aplicação.

Em vez disso, considere a seguinte metodologia de programação de controle de acesso:

```
if (user.hasAccess("DELETE_ACCOUNT")) {
      deleteAccount();
}
```

As verificações de controle de acesso baseadas em atributos ou recursos dessa natureza são o ponto de partida para a construção de sistemas de controle de acesso bem projetados e ricos em recursos. Esse tipo de programação também permite maior capacidade de personalização do controle de acesso ao longo do tempo.

6) Registrar todos os eventos de controle de acesso

Todas as falhas de controle de acesso devem ser registradas, pois podem ser indicativos de um usuário mal-intencionado sondando a aplicação em busca de vulnerabilidades.

Vulnerabilidades evitadas

- OWASP Top 10 2017-A5-Broken Access Control
- OWASP Mobile Top 10 2014-M5 Poor Authorization and Authentication

Referências

- OWASP Cheat Sheet: Access Control
- OWASP Cheat Sheet: iOS Developer - Poor Authorization and Authentication
- OWASP Testing Guide: Testing for Authorization

Ferramentas

- OWASP ZAP com o uso opcional do add-on Access Control Testing

OWASP Proactive Controls v 3.0

C8: Proteja os dados em todos os lugares

Descrição

Dados confidenciais, como senhas, números de cartão de crédito, registros de saúde, informações pessoais e segredos comerciais exigem proteção extra, especialmente se esses dados estiverem sob leis de privacidade (Regulamento Geral de Proteção de Dados da UE GDPR), regras de proteção de dados financeiros, como PCI Data Security Standard (PCI DSS) ou outros regulamentos.

Os invasores podem roubar dados de aplicações Web e de serviços da Web de várias maneiras. Por exemplo, se informações confidenciais forem enviadas pela Internet sem segurança de comunicação, um invasor em uma conexão sem fio compartilhada poderá ver e roubar os dados de outro usuário. Além disso, um invasor pode usar SQL Injection para roubar senhas e outras credenciais de um banco de dados de aplicações e expor essas informações ao público.

Classificação de dados

É fundamental classificar os dados em seu sistema e determinar a qual nível de sensibilidade cada parte dos dados pertence. Cada categoria de dados pode então ser mapeada para as regras de proteção necessárias para cada nível de sensibilidade. Por exemplo, informações de marketing público que não são confidenciais podem ser categorizadas como dados públicos que podem ser colocados no site público. Os números de cartão de crédito podem ser classificados como dados privados do usuário que podem precisar ser criptografados enquanto armazenados ou em trânsito.

Criptografando dados em trânsito

Ao transmitir dados confidenciais em qualquer rede, algum tipo de segurança de comunicação de ponta a ponta (ou criptografia em trânsito) deve ser considerada. O TLS é de longe o protocolo criptográfico mais comum e amplamente suportado para segurança das comunicações. Ele é usado por muitos tipos de aplicações (web, webservice, mobile) para se comunicar em uma rede de maneira segura. O TLS deve ser configurado adequadamente de várias maneiras para defender adequadamente as comunicações seguras.

O principal benefício da segurança da camada de transporte é a proteção de dados de aplicações da Web contra divulgação e modificação não autorizadas quando são transmitidos entre clientes (navegadores da Web) e o servidor de aplicações da Web, e entre o servidor de aplicações Web e o back-end e outras aplicações não baseados em navegador. componentes empresariais.

Criptografando dados em repouso

A primeira regra do gerenciamento de dados confidenciais é evitar o armazenamento de dados confidenciais sempre que possível. Se você precisar armazenar dados confidenciais, verifique se eles estão protegidos criptograficamente de alguma forma para evitar divulgação e modificação não autorizadas.

A criptografia (ou criptografia) é um dos tópicos mais avançados da segurança da informação, e aquele cuja compreensão requer mais escolaridade e experiência. É difícil acertar porque há muitas abordagens para criptografia, cada uma com vantagens e desvantagens que precisam ser totalmente compreendidas pelos arquitetos e desenvolvedores de soluções da Web. Além disso, a pesquisa séria em criptografia é tipicamente baseada em matemática avançada e teoria dos números, fornecendo uma séria barreira à entrada.

Em vez de criar capacidade criptográfica do zero, é altamente recomendável que soluções abertas e revisadas por pares sejam usadas, como o projeto Tink, Libsodium e capacidade de armazenamento seguro incorporado em muitas estruturas de software e serviços em nuvem.

Aplicativo móvel: armazenamento local seguro

Os aplicativos móveis correm um risco particular de vazamento de dados porque os dispositivos móveis são perdidos ou roubados regularmente, mas contêm dados confidenciais.

Como regra geral, apenas os dados mínimos necessários devem ser armazenados no dispositivo móvel. Mas se você deve armazenar dados confidenciais em um dispositivo móvel, os dados confidenciais devem ser armazenados em cada diretório de armazenamento de dados específico do sistema operacional móvel. No Android, este será o keystore do Android e no iOS, este será o do iOS keychain.

Ciclo de vida da chave

As chaves secretas são usadas em inúmeras funções sigilosas dentro das aplicações. Por exemplo, as chaves secretas podem ser usadas para assinar JWTs, proteger cartões de crédito, fornecer várias formas de autenticação, bem como facilitar outros recursos de segurança confidenciais. No gerenciamento de chaves, várias regras devem ser seguidas, incluindo:

- Certifique-se de que qualquer chave secreta esteja protegida contra acesso não autorizado
- Armazene as chaves em um cofre de segredos adequado, conforme descrito abaixo
- Use chaves independentes quando forem necessárias várias chaves
- Crie suporte para alterar algoritmos e chaves quando necessário
- Crie recursos de aplicações para lidar com uma rotação de chave

Gerenciamento de segredos da aplicação

As aplicações contêm vários "segredos" necessários para operações de segurança. Isso inclui certificados, senhas de conexão SQL, credenciais de contas de serviços de terceiros, senhas, chaves SSH, chaves de criptografia e muito mais. A divulgação ou modificação não autorizada desses segredos pode levar ao comprometimento total do sistema. Ao gerenciar segredos de aplicações, considere o seguinte.

- Não armazene segredos em código, arquivos de configuração ou os passe por variáveis de ambiente. Use ferramentas como GitRob ou TruffleHog para escanear repositórios de código em busca de segredos.
- Mantenha as chaves e seus outros segredos no nível da aplicação em um cofre de segredos como o KeyWhiz ou o projeto Hashicorp's Vault ou o Amazon KMS para fornecer armazenamento seguro e acesso a segredos no nível da aplicação em tempo de execução.

Vulnerabilidades evitadas

- OWASP Top 10 2017 - A3: Sensitive Data Exposure
- OWASP Mobile Top 10 2014-M2 Insecure Data Storage

Referências

- OWASP Cheat Sheet: Transport Layer Protection
- Ivan Ristic: SSL/TLS Deployment Best Practices
- OWASP Cheat Sheet: HSTS
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: IOS Developer - Insecure Data Storage
- OWASP Testing Guide: Testing for TLS

Ferramentas

- SSLyze - Biblioteca de verificação de configuração SSL e ferramenta CLI
- SSLLabs - Serviço gratuito para digitalizar e verificar a configuração TLS/SSL
- OWASP O-Saft TLS Tool - Ferramenta de teste de conexão TLS
- GitRob - Ferramenta de linha de comando para encontrar informações confidenciais em arquivos disponíveis publicamente no GitHub
- TruffleHog - Procura segredos cometidos acidentalmente
- KeyWhiz - Gerenciador de senhas
- Hashicorp Vault - Gerenciador de senhas
- Amazon KMS - Gerenciar chaves na Amazon AWS

OWASP Proactive Controls v 3.0

C9: Implementar registro e monitoramento de segurança

Descrição

Logging é um conceito que a maioria dos desenvolvedores já usa para fins de depuração e diagnóstico. O log de segurança é um conceito igualmente básico: registrar informações de segurança durante a operação de tempo de execução de uma aplicação. O monitoramento é a revisão ao vivo de logs de aplicações e segurança usando várias formas de automação. As mesmas ferramentas e padrões podem ser usados para fins de operações, depuração e segurança.

Benefícios do registro de segurança

O registro de segurança pode ser usado para:

1) Sistemas de detecção de intrusão de alimentação
2) Análise e investigações forenses
3) Satisfazendo os requisitos de conformidade regulatória

Implementação de registro de segurança

Veja a seguir uma lista de práticas recomendadas de implementação de log de segurança.

- Siga um formato de registro comum e uma abordagem dentro do sistema e entre os sistemas de uma organização. Um exemplo de uma estrutura de log comum é o Apache Logging Services, que ajuda a fornecer consistência de log entre aplicações Java, PHP, .NET e C++.
- Não registre muito ou pouco. Por exemplo, certifique-se de sempre registrar o carimbo de data/hora e as informações de identificação, incluindo o IP de origem e o ID do usuário, mas tome cuidado para não registrar dados privados ou confidenciais.
- Preste muita atenção à sincronização de tempo entre nós para garantir que os carimbos de data e hora sejam consistentes.

Registro para detecção e resposta de intrusão

Use o log para identificar a atividade que indica que um usuário está se comportando maliciosamente. Atividades potencialmente maliciosas para registrar incluem:

- Dados enviados que estão fora de um intervalo numérico esperado.
- Dados enviados que envolvem alterações em dados que não devem ser modificáveis (lista de seleção, caixa de seleção ou outro componente de entrada limitada).
- Solicitações que violam as regras de controle de acesso do lado do servidor.
- Uma lista mais abrangente de possíveis pontos de detecção está disponível aqui.

Quando sua aplicação encontra essa atividade, sua aplicação deve, no mínimo, registrar a atividade e marcá-la como um problema de alta gravidade. Idealmente, sua aplicação também deve responder a um possível ataque identificado, por exemplo, invalidando a sessão do usuário e bloqueando a conta do usuário. Os mecanismos de resposta permitem que o software reaja em tempo real a possíveis ataques identificados.

Design de registro seguro

As soluções de registro devem ser criadas e gerenciadas de maneira segura. O design do Secure Logging pode incluir o seguinte:

- Codifique e valide quaisquer caracteres perigosos antes de registrar para evitar ataques de injeção de log ou falsificação de log.
- Não registre informações confidenciais. Por exemplo, não registre senha, ID de sessão, cartões de crédito ou números de previdência social.
- Proteja a integridade do log. Um invasor pode tentar adulterar os logs. Portanto, a permissão de arquivos de log e auditoria de alterações de log deve ser considerada.
- Encaminhe logs de sistemas distribuídos para um serviço de log central e seguro. Isso garantirá que os dados de log não possam ser perdidos se um nó for comprometido. Isso também permite o monitoramento centralizado.

Referências

- OWASP AppSensor Detection Points - Pontos de detecção usados para identificar um usuário mal-intencionado investigando vulnerabilidades ou pontos fracos na aplicação.
- OWASP Log injection
- OWASP Log forging
- OWASP Cheat Sheet: Logging - Como implementar corretamente o log em uma aplicação
- OWASP Development Guide: Logging
- OWASP Code Review Guide: Reviewing Code for Logging Issues

Ferramentas

- OWASP Security Logging Project
- Apache Logging Services

OWASP Proactive Controls v 3.0

C10: Lidar com todos os erros e exceções

Descrição

O tratamento de exceção é um conceito de programação que permite que uma aplicação responda a diferentes estados de erro (como rede inativa ou falha na conexão do banco de dados, etc.) de várias maneiras. O tratamento correto de exceções e erros é fundamental para tornar seu código confiável e seguro.

O tratamento de erros e exceções ocorre em todas as áreas de uma aplicação, incluindo lógica de negócios crítica, bem como recursos de segurança e código de estrutura.

O tratamento de erros também é importante do ponto de vista da detecção de intrusão. Certos ataques contra sua aplicação podem desencadear erros que podem ajudar a detectar ataques em andamento.

Erros de tratamento de erros

Pesquisadores da Universidade de Toronto descobriram que mesmo pequenos erros no tratamento de erros ou esquecimento de lidar com erros podem levar a falhas catastróficas em sistemas distribuídos.

Erros no tratamento de erros podem levar a diferentes tipos de vulnerabilidades de segurança.

- **Vazamento de informações:** o vazamento de informações confidenciais em mensagens de erro pode ajudar involuntariamente os invasores. Por exemplo, um erro que retorna um rastreamento de pilha ou outros detalhes de erros internos pode fornecer a um invasor informações sobre seu ambiente. Mesmo pequenas diferenças no tratamento de diferentes condições de erro (por exemplo, retornar "usuário inválido" ou "senha inválida" em erros de autenticação) podem fornecer pistas valiosas para os invasores. Conforme descrito acima, certifique-se de registrar os detalhes do erro para fins de análise forense e de depuração, mas não exponha essas informações, especialmente para um cliente externo.
- **Desvio de TLS:** O "bug de falha" da Apple foi um erro de fluxo de controle no código de tratamento de erros que levou a um comprometimento completo das conexões TLS nos sistemas da Apple.
- **DOS:** A falta de tratamento básico de erros pode levar ao desligamento do sistema. Esta é geralmente uma vulnerabilidade bastante fácil para os invasores explorarem. Outros problemas de tratamento de erros podem levar ao aumento do uso da CPU ou do disco de forma a degradar o sistema.

Conselho Positivo

- Gerencie exceções de maneira centralizada para evitar blocos try/catch duplicados no código. Certifique-se de que todo comportamento inesperado seja tratado corretamente dentro da aplicação.
- Certifique-se de que as mensagens de erro exibidas aos usuários não vazem dados críticos, mas ainda sejam detalhadas o suficiente para permitir a resposta adequada do usuário.
- Certifique-se de que as exceções sejam registradas de forma a fornecer informações suficientes para que as equipes de suporte, controle de qualidade, análise forense ou de resposta a incidentes entendam o problema.
- Teste e verifique cuidadosamente o código de tratamento de erros.

Referências

- OWASP Code Review Guide: Error Handling
- OWASP Testing Guide: Testing for Error Handling
- OWASP Improper Error Handling
- CWE 209: Information Exposure Through an Error Message
- CWE 391: Unchecked Error Condition

Ferramentas

- Error Prone - Uma ferramenta de análise estática do Google para detectar erros comuns no tratamento de erros para desenvolvedores Java
- Uma das ferramentas automatizadas mais famosas para encontrar erros em tempo de execução é o Chaos Monkey da Netflix, que desativa intencionalmente as instâncias do sistema para garantir que o serviço geral seja recuperado corretamente.

Palavra final

Este documento deve ser visto como um ponto de partida e não como um conjunto abrangente de técnicas e práticas. Queremos enfatizar novamente que este documento tem como objetivo fornecer uma conscientização inicial sobre a criação de software seguro.

As próximas etapas para ajudar a criar um programa de segurança de aplicações incluem:

1) Para entender alguns dos riscos na segurança de aplicações Web, consulte o OWASP Top Ten e o OWASP Mobile Top Ten.
2) De acordo com o Controle Proativo nº 1, um programa de desenvolvimento seguro deve incluir uma lista abrangente de requisitos de segurança com base em um padrão como o OWASP (Web) ASVS e o OWASP (Mobile) MASVS.
3) Para entender os principais blocos de construção de um programa de software seguro de um ponto de vista mais macro, revise o projeto OWASP OpenSAMM.

Se você tiver alguma dúvida para a equipe de liderança do projeto, inscreva-se em nossa lista de e-mails em https://lists.owasp.org/mailman/listinfo/owasp_proactive_controls.

OWASP
Open Web Application
Security Project

FOR DEVELOPERS