

MINISTÉRIO DE MINAS E ENERGIA
Secretaria Executiva
Subsecretaria de Tecnologia e Inovação
Coordenação Geral de Inovação e Transformação Digital

Guia de Segurança em Aplicações Web

Versão 1.2

Histórico de Revisões

Data	Versão do documento	Descrição	Autor
04/04/2024	1.0	Atualização do documento	Luiz Alexandre
29/04/2024	1.2	Revisão Geral equipe Infra-MME	Claudio Augusto
03/05/2024	1.1	Revisão	Carlos Gomes

Sumário

1. Introdução 5

2. Riscos de Segurança em Aplicações 6

2.1. Controle de acesso quebrado 6

2.1.1. Descrição 6

2.1.2. Como prevenir 7

2.1.3. Cenários de Ataque de Exemplo 7

2.2. Falhas criptográficas 9

2.2.1. Descrição 9

2.2.2. Como prevenir 10

2.2.3. Cenários de Ataque de Exemplo 12

2.3. Injeção 13

2.3.1. Descrição 13

2.3.2. Como prevenir 13

2.3.3. Cenários de Ataque de Exemplo 14

2.4. Design inseguro 15

2.4.1. Descrição 15

2.4.2. Como prevenir 16

2.4.3. Cenários de Ataque de Exemplo 16

2.5. Configuração Incorreta de Segurança 17

2.5.1. Descrição 17

2.5.2. Como prevenir 18

2.5.3. Cenários de Ataque de Exemplo 18

2.6. Componentes Vulneráveis e Desatualizados 20

2.6.1. Descrição 20

2.6.2. Como prevenir 20
2.6.3. Cenários de Ataque de Exemplo 21
2.7. Falhas de Identificação e Autenticação 22
2.7.1. Descrição 22
2.7.2. Como prevenir 22
2.7.3. Cenários de Ataque de Exemplo 23
2.8. Falhas de Identificação e Autenticação 24
2.8.1. Descrição 24
2.8.2. Como Prevenir 24
2.8.3. Cenários de Ataque de Exemplo 25
2.9. Falhas de registro e monitoramento de segurança 26
2.9.1. Descrição 26
2.9.2. Como prevenir 27
2.9.3. Cenários de Ataque de Exemplo 27
2.10. Falsificação de solicitação do lado do servidor (SSRF) 28
2.10.1. Descrição 28
2.10.2. Como prevenir 28
2.10.3. Cenários de Ataque de Exemplo 29
3. Referências 30
4. Ferramenta para Análise 30

1. Introdução

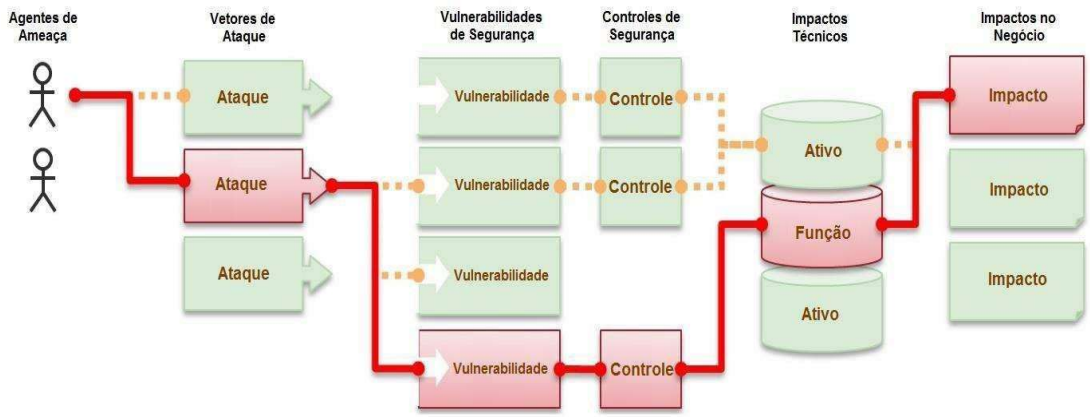
Este documento é baseado no Top 10 OWASP (*Open Web Application Security Project*), uma comunidade aberta, dedicada a capacitar as organizações a desenvolver, adquirir e manter aplicações confiáveis.

O Top 10 tem como objetivo a sensibilização sobre segurança em aplicações através da identificação de alguns dos riscos mais críticos enfrentados pelas organizações. O projeto Top 10 é referenciado por normas, livros, ferramentas e organizações, incluindo MITRE, PCI DSS, DISA, FTC, dentre outras.

O objetivo principal do OWASP Top 10 é educar desenvolvedores, projetistas, arquitetos, gestores e organizações sobre as conseqüências das mais importantes vulnerabilidades de segurança de aplicações *web*. O Top 10 fornece técnicas básicas de proteção contra essas áreas de alto risco, fornecendo também orientação quanto aos passos a seguir a partir daqui.

Os Agentes de Ameaça podem, potencialmente, usar diversos caminhos diferentes através de uma aplicação para causar danos a determinado negócio ou organização, onde cada um desses caminhos representa um risco que pode ou não ser grave o suficiente para exigir certa atenção.

Certas vezes esses caminhos são fáceis de encontrar e explorar, mas em outras são extremamente difíceis. Da mesma forma, o dano causado pode não ter conseqüências ou pode acabar com o negócio. Para determinar o risco para determinada organização, pode-se avaliar a probabilidade associada a cada agente de ameaça, vetor de ataque, vulnerabilidade de segurança e combiná-la com uma estimativa dos impactos técnicos e no negócio da empresa. Juntos, esses fatores determinam o risco total.



2. Riscos de Segurança em Aplicações

2.1. Controle de acesso quebrado

Subindo da quinta posição, 94% dos aplicativos foram testados para alguma forma de controle de acesso quebrado com taxa média de incidência de 3,81%, e tem o maior número de ocorrências no conjunto de dados contribuído com mais de 318k. As Enumerações de Fraquezas Comuns (CWEs) notáveis incluídas são CWE-200: Exposição de informações confidenciais a um agente não autorizado , CWE-201: Inserção de informações confidenciais em dados enviados e CWE-352: falsificação de solicitação entre sites .

2.1.1. Descrição

O controle de acesso impõe a política de forma que os usuários não possam agir fora de suas permissões pretendidas. As falhas geralmente levam à divulgação, modificação ou destruição de informações não autorizadas de todos os dados ou à execução de uma função comercial fora dos limites do usuário. Vulnerabilidades comuns de controle de acesso incluem:

- Violação do princípio de privilégio mínimo ou negação por padrão, onde o acesso deve ser concedido apenas para recursos, funções ou usuários específicos, mas está disponível para qualquer pessoa.

- Ignorando as verificações de controle de acesso modificando a URL (alteração de parâmetro ou navegação forçada), o estado interno do aplicativo ou a página HTML, ou usando uma ferramenta de ataque modificando solicitações de API.

- Permitir a visualização ou edição da conta de outra pessoa, fornecendo seu identificador exclusivo (referências de objetos diretos inseguros)

- Acessando API com controles de acesso ausentes para POST, PUT e DELETE.

- Elevação de privilégio. Atuar como usuário sem estar conectado ou atuar como administrador quando estiver conectado como usuário.

- Manipulação de metadados, como reproduzir ou adulterar um token de controle de acesso JSON Web Token (JWT), ou um cookie ou campo oculto manipulado para elevar privilégios ou abusar da invalidação de JWT.

- A configuração incorreta do CORS permite o acesso à API de origens não autorizadas/não confiáveis.

- Força a navegação em páginas autenticadas como usuário não autenticado ou em páginas privilegiadas como usuário padrão.

2.1.2. Como prevenir

O controle de acesso só é eficaz em código do lado do servidor confiável ou API sem servidor, em que o invasor não pode modificar a verificação ou os metadados do controle de acesso.

- Exceto para recursos públicos, negar por padrão.
- Implemente mecanismos de controle de acesso uma vez e reutilize-os em todo o aplicativo, inclusive minimizando o uso do Cross-Origin Resource Sharing (CORS).
- Os controles de acesso do modelo devem impor a propriedade do registro em vez de aceitar que o usuário possa criar, ler, atualizar ou excluir qualquer registro.
- Os requisitos exclusivos de limite de negócios do aplicativo devem ser impostos por modelos de domínio.
- Desative a listagem de diretórios do servidor web e certifique-se de que os metadados do arquivo (por exemplo, .git) e os arquivos de backup não estejam presentes nas raízes da web.
- Registre falhas de controle de acesso, alerte os administradores quando apropriado (por exemplo, falhas repetidas).
- Taxa de limite de acesso à API e ao controlador para minimizar os danos das ferramentas de ataque automatizadas.
- Os identificadores de sessão com estado devem ser invalidados no servidor após o logout. Os tokens JWT sem estado devem ser de curta duração para que a janela de oportunidade para um invasor seja minimizada. Para JWTs de vida mais longa, é altamente recomendável seguir os padrões OAuth para revogar o acesso.

Os desenvolvedores e a equipe de controle de qualidade devem incluir a unidade de controle de acesso funcional e testes de integração.

2.1.3. Cenários de Ataque de Exemplo

Cenário 1: o aplicativo usa dados não verificados em uma chamada SQL que está acessando as informações da conta:

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

Um invasor simplesmente modifica o parâmetro 'acct' do navegador para enviar o número de conta que desejar. Se não for verificado corretamente, o invasor pode acessar a conta de qualquer usuário.

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

Cenário 2: um invasor simplesmente força a navegação a URLs de destino. 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 página, é uma falha. Se um não administrador puder acessar a página de administração, isso é uma falha.

2.2. Falhas criptográficas

Subindo uma posição para a posição 2, anteriormente conhecida como Exposição de Dados Sensíveis , que é mais um sintoma amplo do que uma causa raiz, o foco está nas falhas relacionadas à criptografia (ou falta dela). Que muitas vezes levam à exposição de dados confidenciais. Notáveis Enumerações de Fraquezas Comuns (CWEs) incluídas são CWE-259: Uso de Senha Codificada , CWE-327: Algoritmo de Criptografia Quebrado ou Arriscado e CWE-331 Entropia Insuficiente.

2.2.1. Descrição

A primeira coisa é determinar as necessidades de proteção dos dados em trânsito e em repouso. Por exemplo, senhas, números de cartão de crédito, registros de saúde, informações pessoais e segredos comerciais exigem proteção extra, principalmente se esses dados estiverem sob leis de privacidade, por exemplo, Regulamento Geral de Proteção de Dados da UE (GDPR) ou regulamentos, por exemplo, proteção de dados financeiros como PCI Data Security Standard (PCI DSS). Para todos esses dados:

- Algum dado é transmitido em texto simples? Isso diz respeito a protocolos como HTTP, SMTP, FTP também usando atualizações TLS como STARTTLS. O tráfego externo da Internet é perigoso. Verifique todo o tráfego interno, por exemplo, entre balanceadores de carga, servidores web ou sistemas back-end.

- Algum algoritmo ou protocolo criptográfico antigo ou fraco é usado por padrão ou em código mais antigo?

- As chaves criptográficas padrão estão em uso, as chaves criptográficas fracas são geradas ou reutilizadas ou o gerenciamento ou rotação de chaves está ausente? As chaves criptográficas são verificadas nos repositórios de código-fonte?

- A criptografia não é aplicada, por exemplo, há alguma diretiva de segurança de cabeçalhos HTTP (navegador) ou cabeçalhos ausentes?

- O certificado do servidor recebido e a cadeia de confiança estão devidamente validados?

- Os vetores de inicialização são ignorados, reutilizados ou não gerados suficientemente seguros para o modo de operação criptográfico? Está em uso um modo de operação inseguro, como o BCE? A criptografia é usada quando a criptografia autenticada é mais apropriada?

- As senhas estão sendo usadas como chaves criptográficas na ausência de uma função de derivação de chave de base de senha?

- A aleatoriedade é usada para fins criptográficos que não foram projetados para atender aos requisitos criptográficos? Mesmo que a função correta seja escolhida, ela precisa ser propagada pelo desenvolvedor e, caso contrário, o desenvolvedor sobrescreveu a funcionalidade de propagação forte incorporada a ela com uma semente que não possui entropia/imprevisibilidade suficiente?

- As funções de hash obsoletas, como MD5 ou SHA1, estão em uso ou as funções de hash não criptográficas são usadas quando as funções de hash criptográficas são necessárias?

- Estão em uso métodos de preenchimento criptográfico obsoletos, como PKCS número 1 v1.5?

- As mensagens de erro criptográficas ou informações de canal lateral podem ser exploradas, por exemplo, na forma de ataques oracle de preenchimento?

2.2.2. Como prevenir

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

- Classifique os dados processados, armazenados ou transmitidos por um aplicativo. Identifique quais dados são confidenciais de acordo com as leis de privacidade, requisitos regulatórios ou necessidades de negócios.

- Não armazene dados confidenciais desnecessariamente. Descarte-o o mais rápido possível ou use tokenização compatível com PCI DSS ou até mesmo truncamento. Os dados que não são retidos não podem ser roubados.

- Certifique-se de criptografar todos os dados confidenciais em repouso.

- Garantir que algoritmos, protocolos e chaves padrão atualizados e fortes estejam em vigor; use o gerenciamento de chaves adequado.

- Criptografe todos os dados em trânsito com protocolos seguros, como TLS com cifras de sigilo de encaminhamento (FS), priorização de cifras pelo servidor e parâmetros seguros. Imponha a criptografia usando diretivas como HTTP Strict Transport Security (HSTS).

- Desabilite o armazenamento em cache para respostas que contenham dados confidenciais.

- Aplique os controles de segurança necessários de acordo com a classificação dos dados.

- Não use protocolos legados, como FTP e SMTP, para transportar dados confidenciais.

- Armazene senhas usando funções de hashing adaptáveis e salgadas fortes com um fator de trabalho (fator de atraso), como Argon2, scrypt, bcrypt ou PBKDF2.

- Os vetores de inicialização devem ser escolhidos de acordo com o modo de operação. Para muitos modos, isso significa usar um CSPRNG (gerador de números pseudo-aleatórios criptograficamente seguro). Para modos que exigem um nonce, o vetor de inicialização (IV) não precisa de um CSPRNG. Em todos os casos, o IV nunca deve ser usado duas vezes para uma chave fixa.

- Sempre use criptografia autenticada em vez de apenas criptografia.

- As chaves devem ser geradas criptograficamente aleatoriamente e armazenadas na memória como arrays de bytes. Se uma senha for usada, ela deverá ser convertida em uma chave por meio de uma função de derivação de chave de base de senha apropriada.

- Certifique-se de que a aleatoriedade criptográfica seja usada quando apropriado e que não tenha sido propagada de maneira previsível ou com baixa entropia. A maioria das APIs modernas não exige que o desenvolvedor semeie o CSPRNG para obter segurança.

- Evite funções criptográficas obsoletas e esquemas de preenchimento, como MD5, SHA1, PKCS número 1 v1.5 .

- Verifique de forma independente a eficácia da configuração e configurações.

2.2.3. Cenários de Ataque de Exemplo

Cenário 1 : Um aplicativo criptografa números de cartão de crédito em um banco de dados usando a criptografia automática do banco de dados. No entanto, esses dados são descriptografados automaticamente quando recuperados, permitindo que uma falha de injeção de SQL recupere números de cartão de crédito em texto não criptografado.

Cenário 2 : um site não usa ou impõe TLS para todas as páginas ou oferece suporte a criptografia fraca. Um invasor monitora o tráfego de rede (por exemplo, em uma rede sem fio insegura), faz downgrade de conexões de HTTPS para HTTP, intercepta solicitações e rouba o cookie de sessão do usuário. O invasor então reproduz esse cookie e sequestra a sessão (autenticada) do usuário, acessando ou modificando os dados privados do usuário. Em vez do acima, eles 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 simples ou sem sal para armazenar as senhas de todos. Uma falha de upload de arquivo permite que um invasor recupere o banco de dados de senhas. Todos os hashes sem sal podem ser expostos com uma tabela arco-íris de hashes pré-calculados. Hashes gerados por funções de hash simples ou rápidas podem ser quebrados por GPUs, mesmo que tenham sido salgados.

2.3. Injeção

A injeção desliza para a terceira posição. 94% dos aplicativos foram testados para alguma forma de injeção com uma taxa de incidência máxima de 19%, uma taxa de incidência média de 3% e 274 mil ocorrências. As CWEs (Common Weakness Enumerations) notáveis incluídas são CWE-79: Cross-site Scripting , CWE-89: SQL Injection e CWE-73: External Control of File Name or Path .

2.3.1. Descrição

Um aplicativo é vulnerável a ataques quando:

- Os dados fornecidos pelo usuário não são validados, filtrados ou higienizados pelo aplicativo.

- Consultas dinâmicas ou chamadas não parametrizadas sem escape sensível ao contexto são usadas diretamente no interpretador.

- Dados hostis são usados em parâmetros de pesquisa de mapeamento relacional de objeto (ORM) para extrair registros confidenciais adicionais.

- Dados hostis são usados diretamente ou concatenados. O SQL ou comando contém a estrutura e os dados maliciosos em consultas dinâmicas, comandos ou procedimentos armazenados.

Algumas das injeções mais comuns são SQL, NoSQL, comando do SO, Mapeamento Relacional de Objetos (ORM), LDAP e Linguagem de Expressão (EL) ou injeção de Biblioteca de Navegação de Gráficos de Objetos (OGNL). O conceito é idêntico entre todos os intérpretes. A revisão do código-fonte é o melhor método para detectar se os aplicativos são vulneráveis a injeções. O teste automatizado de todos os parâmetros, cabeçalhos, URL, cookies, JSON, SOAP e entradas de dados XML é fortemente recomendado. As organizações podem incluir ferramentas de teste de segurança de aplicativos estáticas (SAST), dinâmicas (DAST) e interativas (IAST) no pipeline de CI/CD para identificar falhas de injeção introduzidas antes da implantação de produção.

2.3.2. Como prevenir

Prevenir a injeção requer manter os dados separados de comandos e consultas:

- A opção preferencial é usar uma API segura, que evite totalmente o uso do interpretador, forneça uma interface parametrizada ou migre para Object Relational Mapping Tools (ORMs). Nota: Mesmo quando parametrizados, os procedimentos armazenados ainda podem introduzir injeção de SQL se PL/SQL ou T-SQL concatenar consultas e dados ou executar dados hostis com EXECUTE IMMEDIATE ou exec().

- Use validação de entrada positiva do lado do servidor. Essa não é uma defesa completa, pois muitos aplicativos exigem caracteres especiais, como áreas de texto ou APIs para aplicativos móveis.

- Para quaisquer consultas dinâmicas residuais, escape caracteres especiais usando a sintaxe de escape específica para esse interpretador. Nota: Estruturas SQL, como nomes de tabelas, nomes de colunas e assim por diante, não podem ser escapadas e, portanto, os nomes de estrutura fornecidos pelo usuário são perigosos. Este é um problema comum em software de redação de relatórios.

- Use LIMIT e outros controles SQL nas consultas para evitar a divulgação em massa de registros em caso de injeção de SQL.

2.3.3. Cenários de Ataque de Exemplo

Cenário 1: Um aplicativo usa dados não confiáveis na construção da seguinte chamada SQL vulnerável:

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

Cenário 2: Da mesma forma, a confiança cega de um aplicativo em estruturas pode resultar em consultas que ainda são vulneráveis (por exemplo, Hibernate Query Language (HQL)):

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

Em ambos os casos, o invasor modifica o valor do parâmetro 'id' em seu navegador para enviar: ' ou '1'='1. Por exemplo:

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

Isso altera o significado de ambas as consultas para retornar todos os registros da tabela de contas. Ataques mais perigosos podem modificar ou excluir dados ou até mesmo invocar procedimentos armazenados.

2.4. Design inseguro

Uma nova categoria para 2021 se concentra nos riscos relacionados a falhas de design e arquitetura, com um apelo por mais uso de modelagem de ameaças, padrões de design seguros e arquiteturas de referência. Como comunidade, precisamos ir além do "shift-left" no espaço de codificação para atividades de pré-codificação que são críticas para os princípios do Secure by Design. As Enumerações de Fraquezas Comuns (CWEs) notáveis incluem CWE-209: Geração de mensagem de erro contendo informações confidenciais , CWE-256: Armazenamento desprotegido de credenciais , CWE-501: Violação de limite de confiança e CWE-522: Credenciais protegidas insuficientemente.

2.4.1. Descrição

O design inseguro é uma categoria ampla que representa diferentes pontos fracos, expressos como "design de controle ausente ou ineficaz". O design inseguro não é a fonte de todas as outras 10 categorias de risco. Há uma diferença entre design inseguro e implementação insegura. Diferenciamos entre falhas de design e defeitos de implementação por um motivo, eles têm diferentes causas-raiz e remediação. Um design seguro ainda pode ter defeitos de implementação levando a vulnerabilidades que podem ser exploradas. Um design inseguro não pode ser corrigido por uma implementação perfeita, pois, por definição, os controles de segurança necessários nunca foram criados para se defender contra ataques específicos. Um dos fatores que contribuem para o design inseguro é a falta de perfil de risco de negócio inerente ao software ou sistema que está sendo desenvolvido.

Gerenciamento de Requisitos e Recursos

Colete e negocie os requisitos de negócios de um aplicativo com os negócios, incluindo os requisitos de proteção relativos a confidencialidade, integridade, disponibilidade e autenticidade de todos os ativos de dados e a lógica de negócios esperada. Leve em consideração a exposição do seu aplicativo e se você precisa de segregação de locatários (além de controle de acesso). Compilar os requisitos técnicos, incluindo requisitos de segurança funcionais e não funcionais. Planeje e negocie o orçamento cobrindo todo o projeto, construção, teste e operação, incluindo atividades de segurança.

Design Seguro

O design seguro é uma cultura e metodologia que avalia constantemente as ameaças e garante que o código seja projetado e testado de forma robusta para evitar métodos de ataque conhecidos. A modelagem de ameaças deve ser integrada em sessões de refinamento (ou atividades similares); procure mudanças nos fluxos de dados e controle de acesso ou outros controles de segurança. No desenvolvimento da história do usuário, determine o fluxo correto e os estados de falha, certifique-se de que eles sejam bem compreendidos e acordados pelas partes responsáveis e afetadas. Analise suposições e condições para fluxos esperados e de falha, certifique-se de que eles ainda sejam precisos e desejáveis. Determine como validar as suposições e impor condições necessárias para comportamentos adequados. Certifique-se de que os resultados sejam documentados na história do usuário. Aprenda com os erros e ofereça incentivos positivos para promover melhorias.

Ciclo de Vida de Desenvolvimento Seguro

O software seguro requer um ciclo de vida de desenvolvimento seguro, alguma forma de padrão de projeto seguro, metodologia de estradas pavimentadas, biblioteca de componentes seguros, ferramentas e modelagem de ameaças. Procure seus especialistas em segurança no início de um projeto de software durante todo o projeto e manutenção de seu software. Considere aproveitar o modelo de maturidade de garantia de software OWASP (SAMM) para ajudar a estruturar seus esforços de desenvolvimento de software seguro.

2.4.2. Como prevenir

- Estabeleça e use um ciclo de vida de desenvolvimento seguro com profissionais da AppSec para ajudar a avaliar e projetar controles relacionados à segurança e privacidade
- Estabeleça e use uma biblioteca de padrões de projeto seguros ou componentes prontos para uso de estradas pavimentadas
- Use a modelagem de ameaças para autenticação crítica, controle de acesso, lógica de negócios e fluxos de chaves
- Integre linguagem e controles de segurança em histórias de usuários
- Integre verificações de plausibilidade em cada camada do seu aplicativo (do front-end ao back-end)
- Escreva testes de unidade e integração para validar se todos os fluxos críticos são resistentes ao modelo de ameaça. Compile casos de uso e casos de uso indevido para cada camada de seu aplicativo.
- Segregar camadas de camadas no sistema e nas camadas de rede, dependendo das necessidades de exposição e proteção
- Separe os locatários de forma robusta por design em todas as camadas
- Limitar o consumo de recursos por usuário ou serviço.

2.4.3. Cenários de Ataque de Exemplo

Cenário 1: Um fluxo de trabalho de recuperação de credenciais pode incluir “perguntas e respostas”, o que é proibido pelo NIST 800-63b, OWASP ASVS e OWASP Top 10. Perguntas e respostas não podem ser confiáveis como evidência de identidade como mais de uma pessoa podem saber as respostas, razão pela qual são proibidas. Esse código deve ser removido e substituído por um design mais seguro.

Cenário 2: Uma cadeia de cinemas permite descontos em reservas de grupos e tem um máximo de quinze participantes antes de exigir um depósito. Os invasores poderiam modelar esse fluxo e testar se poderiam reservar seiscentos assentos e todos os cinemas de uma só vez em poucas solicitações, causando uma enorme perda de receita.

Cenário 3: O site de comércio eletrônico de uma rede de varejo não tem proteção contra bots executados por cambistas que compram placas de vídeo de alta qualidade para revender sites de leilões. Isso cria uma publicidade terrível para os fabricantes de placas de vídeo e donos de cadeias de varejo e para os entusiastas que não podem obter essas placas a qualquer preço. O design antibot e as regras de lógica de domínio cuidadosas, como compras feitas dentro de alguns segundos de disponibilidade, podem identificar compras não autênticas e rejeitar essas transações.

2.5. Configuração Incorreta de Segurança

Subindo do nº 6 na edição anterior, 90% dos aplicativos foram testados para algum tipo de configuração incorreta, com uma taxa de incidência média de 4,% e mais de 208 mil ocorrências de uma Enumeração de Fraqueza Comum (CWE) nesta categoria de risco. Com mais mudanças em software altamente configurável, não é surpreendente ver essa categoria subir. CWEs notáveis incluídos são Configuração CWE-16 e CWE-611 Imprópria Restrição de Referência de Entidade Externa XML.

2.5.1. Descrição

O aplicativo pode estar vulnerável se o aplicativo for:

- Falta de proteção de segurança apropriada em qualquer parte da pilha de aplicativos ou permissões configuradas incorretamente em serviços de nuvem.
- Recursos desnecessários são ativados ou instalados (por exemplo, portas, serviços, páginas, contas ou privilégios desnecessários).
- As contas padrão e suas senhas ainda estão habilitadas e inalteradas.
- O tratamento de erros revela rastreamentos de pilha ou outras mensagens de erro excessivamente informativas aos usuários.
- Para sistemas atualizados, os recursos de segurança mais recentes são desabilitados ou não configurados com segurança.
- As configurações de segurança nos servidores de aplicativos, estruturas de aplicativos (por exemplo, Struts, Spring, ASP.NET), bibliotecas, bancos de dados etc., não são definidas para valores seguros.

- O servidor não envia cabeçalhos ou diretivas de segurança ou eles não estão configurados para valores seguros.

- O software está desatualizado ou vulnerável (consulte A06:2021-Componentes vulneráveis e desatualizados).

Sem um processo de configuração de segurança de aplicativos concertado e repetível, os sistemas correm um risco maior..

2.5.2. Como prevenir

Os processos de instalação segura devem ser implementados, incluindo:

- Um processo de proteção repetível torna rápido e fácil a implantação de outro ambiente devidamente bloqueado. Os ambientes de desenvolvimento, controle de qualidade e produção devem ser configurados de forma idêntica, com credenciais diferentes usadas em cada ambiente. Esse 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 estruturas 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 (consulte A06:2021-Componentes vulneráveis e desatualizados). Revise as permissões de armazenamento em nuvem (por exemplo, permissões de bucket do S3).

- Uma arquitetura de aplicativo segmentada fornece separação eficaz e segura entre componentes ou locatários, com segmentação, conteinerização ou grupos de segurança de nuvem (ACLs).

- Envio de diretivas de segurança para clientes, por exemplo, Cabeçalhos de Segurança.

- Um processo automatizado para verificar a eficácia das configurações e ajustes em todos os ambientes.

2.5.3. Cenários de Ataque de Exemplo

As recomendações para evitar esse tipo de ataque são:

Cenário 1: O servidor de aplicativos vem com aplicativos de amostra não removidos do servidor de produção. Esses aplicativos de exemplo têm falhas de segurança conhecidas que os invasores usam para comprometer o servidor. Suponha que um desses aplicativos seja o console de administração e as contas padrão não tenham sido alteradas. Nesse caso, o invasor efetua login com senhas padrão e assume o controle.

Cenário 2: A listagem de diretórios não está desabilitada no servidor. Um invasor descobre que pode simplesmente listar diretórios. O invasor encontra e baixa as classes Java compiladas, que ele descompila e faz engenharia reversa para visualizar o código. O invasor encontra uma falha grave de controle de acesso no aplicativo.

Cenário 3: A configuração do servidor de aplicativos permite que mensagens de erro detalhadas, por exemplo, rastreamentos de pilha, sejam retornadas aos usuários. Isso potencialmente expõe informações confidenciais ou falhas subjacentes, como versões de componentes que são conhecidas por serem vulneráveis.

Cenário 4: Um provedor de serviços de nuvem (CSP) tem permissões de compartilhamento padrão abertas para a Internet por outros usuários de CSP. Isso permite que dados confidenciais armazenados no armazenamento em nuvem sejam acessados.

2.6. Componentes Vulneráveis e Desatualizados

Foi o número 2 da pesquisa da comunidade Top 10, mas também tinha dados suficientes para entrar no Top 10 por meio de dados. Componentes vulneráveis são um problema conhecido que lutamos para testar e avaliar o risco e é a única categoria a não ter nenhuma Vulnerabilidade e Exposições Comuns (CVEs) mapeadas para os CWEs incluídos, portanto, um peso padrão de exploits/impacto de 5,0 é usado. Os CWEs notáveis incluídos são CWE-1104: Uso de componentes de terceiros não mantidos e os dois CWEs do Top 10 2013 e 2017.

2.6.1. Descrição

Você provavelmente está vulnerável:

- Se você não conhece as versões de todos os componentes que usa (tanto do lado do cliente quanto do lado do servidor). Isso inclui componentes que você usa diretamente, bem como dependências aninhadas.
- Se o software for vulnerável, sem suporte ou desatualizado. Isso inclui o sistema operacional, servidor de aplicativos/web, sistema de gerenciamento de banco de dados (DBMS), aplicativos, APIs e todos os componentes, ambientes de tempo de execução e bibliotecas.
- Se você não verificar vulnerabilidades regularmente e assinar boletins de segurança relacionados aos componentes que usa.
- Se você não corrigir ou atualizar a plataforma, estruturas e dependências subjacentes de maneira oportuna e baseada em riscos. Isso geralmente acontece em ambientes em que a correção é uma tarefa mensal ou trimestral sob controle de alterações, deixando as organizações abertas a dias ou meses de exposição desnecessária a vulnerabilidades corrigidas.
- Se os desenvolvedores de software não testarem a compatibilidade de bibliotecas atualizadas, atualizadas ou corrigidas.
- Se você não proteger as configurações dos componentes (consulte A05:2021-Configuração incorreta de segurança).

2.6.2. Como prevenir

Deve haver um processo de gerenciamento de patches para:

- Remova dependências não utilizadas, recursos, componentes, arquivos e documentação desnecessários.

- Faça um inventário contínuo das versões de componentes do lado do cliente e do lado do servidor (por exemplo, estruturas, bibliotecas) e suas dependências usando ferramentas como versões, verificação de dependência OWASP, retire.js etc. Monitore continuamente fontes como Vulnerabilidade e exposições comuns (CVE) e National Vulnerability Database (NVD) para vulnerabilidades nos componentes. Use ferramentas de análise de composição de software para automatizar o processo. Assine alertas por e-mail para vulnerabilidades de segurança relacionadas aos componentes que você usa.

- Obtenha apenas componentes de fontes oficiais em links seguros. Prefira pacotes assinados para reduzir a chance de incluir um componente mal-intencionado modificado (consulte A08:2021-Falhas de integridade de software e dados).
-
- Monitore bibliotecas e componentes que não são mantidos ou não criam patches de segurança para versões mais antigas. Se a aplicação de patches não for possível, considere a implantação de um patch virtual para monitorar, detectar ou proteger contra o problema descoberto.

Toda organização deve garantir um plano contínuo de monitoramento, triagem e aplicação de atualizações ou alterações de configuração durante a vida útil do aplicativo ou portfólio.

2.6.3. Cenários de Ataque de Exemplo

Cenário 1: Os componentes normalmente são executados com os mesmos privilégios que o próprio aplicativo, portanto, falhas em qualquer componente podem resultar em sérios impactos. Tais falhas podem ser acidentais (por exemplo, erro de codificação) ou intencionais (por exemplo, um backdoor em um componente). Alguns exemplos de vulnerabilidades de componentes exploráveis descobertas são:

- A CVE-2017-5638, uma vulnerabilidade de execução remota de código do Struts 2 que permite a execução de código arbitrário no servidor, foi responsabilizada por violações significativas.
- Embora a Internet das Coisas (IoT) seja frequentemente difícil ou impossível de corrigir, a importância de corrigi-las pode ser grande (por exemplo, dispositivos biomédicos).

Existem ferramentas automatizadas para ajudar os invasores a encontrar sistemas sem patches ou configurados incorretamente. Por exemplo, o mecanismo de pesquisa Shodan IoT pode ajudá-lo a encontrar dispositivos que ainda sofrem com a vulnerabilidade Heartbleed corrigida em abril de 2014.

2.7. Falhas de Identificação e Autenticação

Anteriormente conhecida como autenticação quebrada , esta categoria desceu da segunda posição e agora inclui Enumerações de Fraqueza Comum (CWEs) relacionadas a falhas de identificação. CWEs notáveis incluídos são CWE-297: Validação Imprópria de Certificado com Incompatibilidade de Host , CWE-287: Autenticação Imprópria e CWE-384: Fixação de Sessão.

2.7.1. Descrição

A confirmação da identidade do usuário, autenticação e gerenciamento de sessão é fundamental para proteger contra ataques relacionados à autenticação. Pode haver deficiências de autenticação se o aplicativo:

- Permite ataques automatizados, como preenchimento de credenciais, em que o invasor possui uma lista de nomes de usuários e senhas válidos.
- Permite força bruta ou outros ataques automatizados.
- Permite senhas padrão, fracas ou conhecidas, como "Password1" ou "admin/admin".
- Usa recuperação de credenciais fraca ou ineficaz e processos de esquecimento de senha, como "respostas baseadas em conhecimento", que não podem ser seguras.
- Usa armazenamentos de dados de senhas de texto simples, criptografados ou com hash fraco (consulte A02:2021-Falhas de criptografia).
- Tem autenticação multi-fator ausente ou ineficaz.
- Expõe o identificador de sessão na URL.
- Reutilize o identificador de sessão após o login bem-sucedido.
- Não invalida corretamente os IDs de sessão. Sessões de usuário ou tokens de autenticação (principalmente tokens de logon único (SSO)) não são invalidados corretamente durante o logout ou um período de inatividade.

2.7.2. Como prevenir

- Sempre que possível, implemente a autenticação multifator para evitar ataques automatizados de preenchimento de credenciais, força bruta e reutilização de credenciais roubadas.
- Não envie ou implante com credenciais padrão, principalmente para usuários administradores.
- Implemente verificações de senhas fracas, como testar senhas novas ou alteradas na lista das 10.000 piores senhas.
- Alinhe as políticas de comprimento, complexidade e rotação de senha com as diretrizes do Instituto Nacional de Padrões e Tecnologia (NIST) 800-63b na seção 5.1.1 para Segredos Memorizados ou outras políticas de senha modernas baseadas em evidências.
- Garanta que os caminhos de registro, recuperação de credenciais e API sejam protegidos contra ataques de enumeração de conta usando as mesmas mensagens para todos os resultados.
- Limite ou retarde cada vez mais as tentativas de login com falha, mas tome cuidado para não

criar um cenário de negação de serviço. Registre todas as falhas e alerte os administradores quando forem detectados preenchimento de credenciais, força bruta ou outros ataques.

- Use um gerenciador de sessão integrado, seguro e do lado do servidor que gera um novo ID de sessão aleatório com alta entropia após o login. O identificador de sessão não deve estar no URL, ser armazenado com segurança e invalidado após o logout, inatividade e tempos limites absolutos.

2.7.3. Cenários de Ataque de Exemplo

Cenário 1: O preenchimento de credenciais, o uso de listas de senhas conhecidas, é um ataque comum. Suponha que um aplicativo não implemente proteção automatizada contra ameaças ou preenchimento de credenciais. Nesse caso, o aplicativo pode ser usado como um oráculo de senha para determinar se as credenciais são válidas.

Cenário 2: A maioria dos ataques de autenticação ocorre devido ao uso contínuo de senhas como único fator. Uma vez consideradas as melhores práticas, a rotação de senhas e os requisitos de complexidade incentivam os usuários a usar e reutilizar senhas fracas. Recomenda-se que as organizações interrompam essas práticas de acordo com o NIST 800-63 e usem a autenticação multifator.

Cenário 3: os tempos limite da sessão do aplicativo não estão definidos corretamente. Um usuário usa um computador público para acessar um aplicativo. Em vez de selecionar "logout", o usuário simplesmente fecha a guia do navegador e vai embora. Um invasor usa o mesmo navegador uma hora depois e o usuário ainda está autenticado.

2.8. Falhas de Identificação e Autenticação

Uma nova categoria incluida em 2021 se concentra em fazer suposições relacionadas a atualizações de software, dados críticos e pipelines de CI/CD sem verificar a integridade. Um dos impactos mais ponderados dos dados de Vulnerabilidade Comum e Exposições/Sistema de Pontuação de Vulnerabilidade Comum (CVE/CVSS). Notáveis Common Weakness Enumerations (CWEs) incluem CWE-829: Inclusão de Funcionalidade de Untrusted Control Sphere , CWE-494: Download de Código Sem Verificação de Integridade e CWE-502: Deserialization of Untrusted Data.

2.8.1. Descrição

As falhas de integridade de software e dados estão relacionadas a código e infraestrutura que não protegem contra violações de integridade. Um exemplo disso é quando um aplicativo depende de plugins, bibliotecas ou módulos de fontes não confiáveis, repositórios e redes de entrega de conteúdo (CDNs). Um pipeline de CI/CD inseguro pode introduzir o potencial de acesso não autorizado, código malicioso ou comprometimento do sistema. Por fim, muitos aplicativos agora incluem a funcionalidade de atualização automática, em que as atualizações são baixadas sem verificação de integridade suficiente e aplicadas ao aplicativo anteriormente confiável. Os invasores podem fazer upload de suas próprias atualizações para serem distribuídas e executadas em todas as instalações. Outro exemplo é quando objetos ou dados são codificados ou serializados em uma estrutura que um invasor pode ver e modificar é vulnerável à desserialização insegura..

2.8.2. Como Prevenir

- Use assinaturas digitais ou mecanismos semelhantes para verificar se o software ou os dados são da fonte esperada e não foram alterados.
- Garanta que bibliotecas e dependências, como npm ou Maven, estejam consumindo repositórios confiáveis. Se você tiver um perfil de risco mais alto, considere hospedar um repositório interno em boas condições que seja verificado.
- Certifique-se de que uma ferramenta de segurança da cadeia de suprimentos de software, como OWASP Dependency Check ou OWASP CycloneDX, seja usada para verificar se os componentes não contêm vulnerabilidades conhecidas
- Certifique-se de que haja um processo de revisão para alterações de código e configuração para minimizar a chance de que código ou configuração mal-intencionados possam ser introduzidos em seu pipeline de software.
- Certifique-se de que seu pipeline de CI/CD tenha segregação, configuração e controle de acesso adequados para garantir a integridade do código que flui pelos processos de compilação e implantação.
- Certifique-se de que os dados serializados não assinados ou não criptografados não sejam enviados para clientes não confiáveis sem alguma forma de verificação de integridade ou assinatura digital para detectar adulteração ou repetição dos dados serializados.

2.8.3. Cenários de Ataque de Exemplo

Cenário 1 - Atualização sem assinatura: muitos roteadores domésticos, decodificadores, firmware de dispositivo e outros não verificam atualizações por meio de firmware assinado. Firmware não assinado é um alvo crescente para invasores e espera-se que só piore. Esta é uma grande preocupação, pois muitas vezes não há mecanismo para corrigir além de corrigir em uma versão futura e esperar que as versões anteriores expirem.

Cenário 2 - Atualização maliciosa do SolarWinds: Sabe-se que os estados-nação atacam os mecanismos de atualização, com um ataque notável recente sendo o ataque do SolarWinds Orion. A empresa que desenvolve o software tinha processos seguros de integridade de compilação e atualização. Ainda assim, eles puderam ser subvertidos e, por vários meses, a empresa distribuiu uma atualização maliciosa altamente direcionada para mais de 18.000 organizações, das quais cerca de 100 foram afetadas. Esta é uma das violações mais abrangentes e significativas dessa natureza na história.

Cenário 3 - Desserialização insegura: um aplicativo React chama um conjunto de microsserviços Spring Boot. Sendo programadores funcionais, eles tentaram garantir que seu código fosse imutável. A solução que eles encontraram foi serializar o estado do usuário e passá-lo para frente e para trás a cada solicitação. Um invasor percebe a assinatura do objeto Java "rO0" (em base64) e usa a ferramenta Java Serial Killer para obter a execução remota de código no servidor de aplicativos.

2.9. Falhas de registro e monitoramento de segurança

O registro e o monitoramento de segurança vieram da pesquisa da comunidade Top 10 (nº 3), ligeiramente acima da décima posição no OWASP Top 10 2017. O registro e o monitoramento podem ser difíceis de testar, geralmente envolvendo entrevistas ou perguntando se os ataques foram detectados durante uma penetração teste. Não há muitos dados CVE/CVSS para essa categoria, mas detectar e responder a violações é fundamental. Ainda assim, pode ser muito impactante para responsabilidade, visibilidade, alerta de incidentes e análise forense. Esta categoria se expande além do CWE-778 Registro Insuficiente para incluir CWE-117 Neutralização de Saída Imprópria para Registros , CWE-223 Omissão de Informações Relevantes à Segurança e CWE-532 Inserção de Informações Confidenciais no Arquivo de Registro.

2.9.1. Descrição

Esta categoria é para ajudar a detectar, escalar e responder a violações ativas. Sem registro e monitoramento, as violações não podem ser detectadas. Registro, detecção, monitoramento e resposta ativa insuficientes ocorrem a qualquer momento:

- Eventos auditáveis, como logins, logins com falha e transações de alto valor, não são registrados.
- Avisos e erros geram mensagens de log inexistentes, inadequadas ou pouco claras.
- Os logs de aplicativos e APIs não são monitorados quanto a atividades suspeitas.
- Os logs são armazenados apenas localmente.
- Limites de alerta apropriados e processos de escalação de resposta não estão em vigor ou não são eficazes.
- Testes de penetração e varreduras por ferramentas de teste de segurança de aplicativos dinâmicos (DAST) (como OWASP ZAP) não acionam alertas.
- O aplicativo não pode detectar, escalar ou alertar para ataques ativos em tempo real ou quase em tempo real.

Você está vulnerável ao vazamento de informações ao tornar os eventos de registro e alerta visíveis para um usuário ou invasor (consulte A01:2021-Broken Access Control).

2.9.2. Como prevenir

Os desenvolvedores devem implementar alguns ou todos os controles a seguir, dependendo do risco do aplicativo:

- Garanta que todas as falhas de login, controle de acesso e validação de entrada do lado do servidor possam ser registradas com contexto de usuário suficiente para identificar contas suspeitas ou maliciosas e mantidas por tempo suficiente para permitir análises forenses atrasadas.

- Certifique-se de que os logs sejam gerados em um formato que as soluções de gerenciamento de log possam consumir facilmente.

- Certifique-se de que os dados de log sejam codificados corretamente para evitar injeções ou ataques nos sistemas de log ou monitoramento.

- Garanta que as transações de alto valor tenham uma trilha de auditoria com controles de integridade para evitar adulteração ou exclusão, como tabelas de banco de dados somente anexadas ou similares.

- As equipes de DevSecOps devem estabelecer monitoramento e alertas eficazes para que atividades suspeitas sejam detectadas e respondidas rapidamente.

- Estabeleça ou adote um plano de resposta e recuperação de incidentes, como o Instituto Nacional de Padrões e Tecnologia (NIST) 800-61r2 ou posterior.

Existem estruturas de proteção de aplicativos comerciais e de código aberto, como o OWASP ModSecurity Core Rule Set, e software de correlação de log de código aberto, como a pilha Elasticsearch, Logstash, Kibana (ELK), que apresenta painéis e alertas personalizados.

2.9.3. Cenários de Ataque de Exemplo

Cenário 1: O operador do site de um provedor de plano de saúde infantil não conseguiu detectar uma violação devido à falta de monitoramento e registro. Uma parte externa informou ao provedor do plano de saúde que um invasor havia acessado e modificado milhares de registros de saúde confidenciais de mais de 3,5 milhões de crianças. Uma revisão pós-incidente descobriu que os desenvolvedores do site não abordaram vulnerabilidades significativas. Como não houve registro ou monitoramento do sistema, o vazamento de dados poderia estar em andamento desde 2013, período de mais de sete anos.

Cenário 2: Uma grande companhia aérea indiana teve uma violação de dados envolvendo mais de dez anos de dados pessoais de milhões de passageiros, incluindo dados de passaporte e cartão de crédito. A violação de dados ocorreu em um provedor de hospedagem em nuvem terceirizado, que notificou a empresa aérea sobre a violação após algum tempo.

Cenário 3: Uma grande companhia aérea europeia sofreu uma violação reportável do GDPR. A violação foi causada por vulnerabilidades de segurança de aplicativos de pagamento exploradas por invasores, que coletaram mais de 400.000 registros de pagamento de clientes. A companhia aérea foi multada em 20 milhões de libras como resultado do regulador de privacidade.

2.10. Falsificação de solicitação do lado do servidor (SSRF)

Esta categoria foi adicionada a partir da pesquisa da comunidade Top 10 (#1). Os dados mostram uma taxa de incidência relativamente baixa com cobertura de teste acima da média e classificações de potencial de exploração e impacto acima da média. Como as novas entradas provavelmente serão um único ou pequeno grupo de Enumerações de Fraquezas Comuns (CWEs) para atenção e conscientização, a esperança é que elas estejam sujeitas a foco e possam ser agrupadas em uma categoria maior em uma edição futura..

2.10.1. Descrição

As falhas do SSRF ocorrem sempre que um aplicativo da Web está buscando um recurso remoto sem validar a URL fornecida pelo usuário. Ele permite que um invasor force o aplicativo a enviar uma solicitação criada para um destino inesperado, mesmo quando protegido por um firewall, VPN ou outro tipo de lista de controle de acesso à rede (ACL).

Como os aplicativos da Web modernos fornecem aos usuários finais recursos convenientes, a busca de uma URL se torna um cenário comum. Como resultado, a incidência de SSRF está aumentando. Além disso, a gravidade do SSRF está se tornando maior devido aos serviços em nuvem e à complexidade das arquiteturas.

2.10.2. Como prevenir

Os desenvolvedores podem impedir o SSRF implementando alguns ou todos os seguintes controles de defesa em profundidade:

Da camada de rede

- Segmente a funcionalidade de acesso remoto a recursos em redes separadas para reduzir o impacto do SSRF

- Aplique políticas de firewall “negar por padrão” ou regras de controle de acesso à rede para bloquear todo o tráfego de intranet, exceto o essencial.

Dicas:

- Estabeleça uma propriedade e um ciclo de vida para regras de firewall baseadas em aplicativos.
- Registre todos os fluxos de rede aceitos e bloqueados em firewalls (consulte A09:2021-Registro de segurança e falhas de monitoramento).

Da camada de aplicação

- Higienize e valide todos os dados de entrada fornecidos pelo cliente
- Aplique o esquema de URL, a porta e o destino com uma lista de permissões positiva
- Não envie respostas brutas aos clientes
- Desabilitar redirecionamentos HTTP
- Esteja ciente da consistência do URL para evitar ataques como religação de DNS e condições de corrida "tempo de verificação, tempo de uso" (TOCTOU)

Não reduza o SSRF por meio do uso de uma lista de negação ou expressão regular. Os invasores têm listas de carga útil, ferramentas e habilidades para contornar as listas de negação.

Medidas adicionais a considerar

- Não implante outros serviços relevantes de segurança em sistemas frontais (por exemplo, OpenID). Controle o tráfego local nesses sistemas (por exemplo, localhost).
- Para frontends com grupos de usuários dedicados e gerenciáveis, use criptografia de rede (por exemplo, VPNs) em sistemas independentes para considerar necessidades de proteção muito altas.

2.10.3. Cenários de Ataque de Exemplo

Os invasores podem usar o SSRF para atacar sistemas protegidos por firewalls de aplicativos da Web, firewalls ou ACLs de rede, usando cenários como:

Cenário 1: Servidores internos de varredura de portas – Se a arquitetura de rede não for segmentada, os invasores podem mapear redes internas e determinar se as portas estão abertas ou fechadas em servidores internos a partir dos resultados da conexão ou do tempo decorrido para conectar ou rejeitar conexões de carga útil do SSRF.

Cenário 2: Exposição de dados confidenciais – Os invasores podem acessar arquivos locais ou serviços internos para obter informações confidenciais, como `file:///etc/passwd` e `http://localhost:28017/`.

Cenário 3: Acesso ao armazenamento de metadados de serviços em nuvem – A maioria dos provedores de nuvem tem armazenamento de metadados, como `http://169.254.169.254/`. Um invasor pode ler os metadados para obter informações confidenciais.

Cenário 4: Comprometer os serviços internos – O invasor pode abusar dos serviços internos para realizar outros ataques, como Execução Remota de Código (RCE) ou Negação de Serviço (DoS)..

3. Referências

1. OWASP: *The Open Web Application Security Project* [Internet]. [acesso em: 2024 Abril 04]. Disponível em: https://owasp.org/www-project-top-ten/

4. Ferramenta para Análise

O MME utiliza o software SONARQUBE para análise de vulnerabilidade.