Padrões de Segurança e Vulnerabilidades de Software em Sistemas Web

Aula Tradicional - Experimento RiskGuard

Sobre o que é a conversa?

- Fundamentos de segurança de software
- Princípios de design de segurança de software
- Top 10 Vulnerabilidades comuns de software para web
- Padrões de Segurança

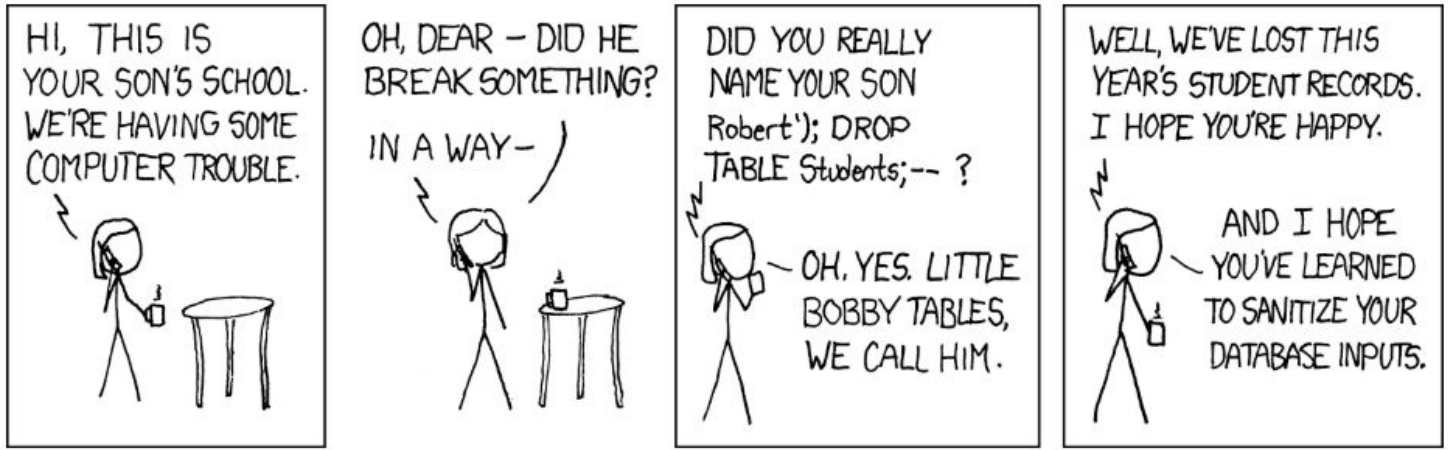
O que é segurança de software?

- Segurança de software é a ideia de projetar software para que ele continue funcionando corretamente sob ataques maliciosos.
- A segurança de software visa evitar vulnerabilidades de segurança abordando a segurança desde os estágios iniciais do ciclo de vida de desenvolvimento de software.
- **"Segurança é gerenciamento de risco."**

Por que segurança de software?

SQL Injection!!

"Bem, perdemos os registros dos alunos deste ano. Espero que esteja feliz."
"E eu espero que vocês tenham aprendido a sanitizar as entradas do banco de dados."



"Oi, aqui é da escola do seu filho. Estamos tendo alguns problemas com o computador."

"Oh, meu Deus – ele quebrou alguma coisa?"

"De certa forma..."

Você realmente chamou seu filho de Robert'); DROP TABLE Students;-- ?"

comic: http://xkcd.com/327/

"Ah, sim. Chamamos ele de Pequeno Bobby Tabelas.

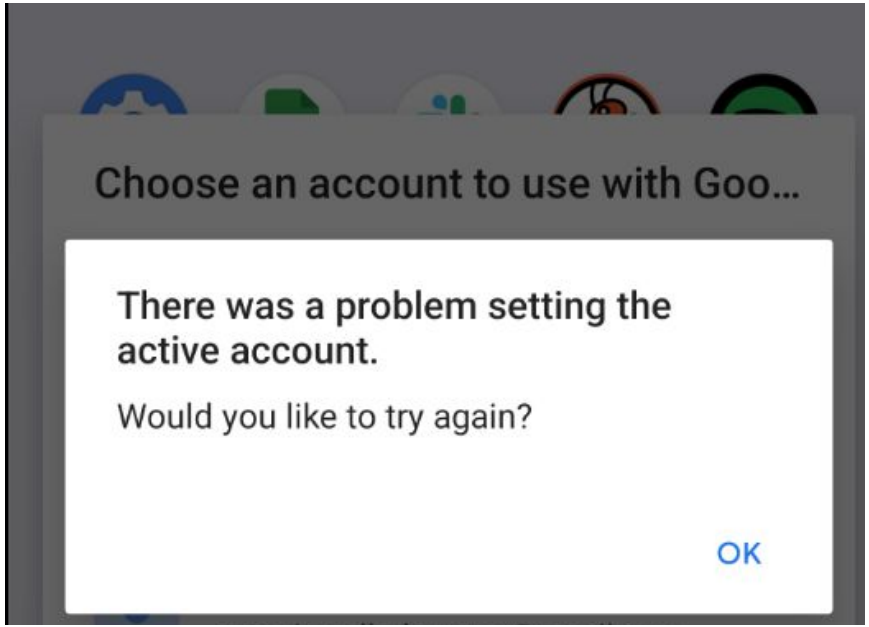
Por que segurança de software?

- A maioria dos sistemas de software hoje contém inúmeras falhas e bugs que são explorados por invasores.
- Novas ameaças surgem todos os dias.
- A conveniência supera as medidas de segurança.
- Aumento exponencial de vulnerabilidades em sistemas de software.
- A segurança de software é tarefa de todos.
- Os programadores têm um longo histórico de repetir os mesmos erros relacionados à segurança

Casos Recentes

2012 - Uma falha de segurança no Google Wallet que leva ao acesso total à sua conta do Google Wallet sem aplicativo extra ou rooting.

- Sua conta do Google Wallet está vinculada ao próprio dispositivo, mas não à conta.



Casos recentes

2009 - Uma falha de segurança no serviço Spotify, que expôs informações privadas de contas.

https://www.theguardian.com/technology/2009/mar/04/online-music-spotify-hacked

Casos recentes

Um exemplo notório ocorreu em agosto de 2024, quando a Polícia Civil do Ceará deflagrou a operação “Payroll” para desarticular um grupo criminoso que tentou invadir o sistema da Secretaria de Planejamento e Gestão (Seplag) e desviar os salários de 612 servidores.

INTELIGÊNCIA

Polícia Civil deflagra operação no CE, PB e GO para desarticular grupo criminoso

8 DE AGOSTO DE 2024 - 14:26 | #CISP #DRCC #Payroll #Seplag #Ceará #DIP #PCCE

A Polícia Civil do Estado Ceará (PCCE) deflagrou, nessa quarta-feira (07), a operação "Payroll". Com o apoio das Polícias Civis da Paraíba e de Goiás, a operação teve como objetivo desarticular um grupo criminoso responsável por ataques cibernéticos. Os envolvidos foram responsáveis por invadir um sistema da Secretaria de Planejamento e Gestão do Estado do Ceará (Seplag) na tentativa de desviar o salário de 612 servidores públicos. Os suspeitos não conseguiram desviar os salários e não houve perda para o Estado.

https://www.policiacivil.ce.gov.br/2024/08/08/policia-civil-deflagra-operacao-no-ce-pb-e-go-para-desarticular-grupo-criminoso-suspeito-de-invadir-sistema-e-tentar-desviar-salarios-de-servidores-publicos-do-ceara/

Terminologia

- **Defeitos:** são vulnerabilidades de implementação e vulnerabilidades de design.
- **Bugs** são erros de nível de implementação que podem ser detectados e removidos.
  - Exemplo: estouro de buffer.
- **Falhas de design** são problemas em um nível mais profundo. Eles são instanciados no código e presentes ou ausentes no nível de design.
  - Exemplo: problemas de tratamento de erros.
- **Falhas de implementação** são a incapacidade do software de executar sua função necessária.

Terminologia

Riscos capturam a probabilidade de que uma falha ou bug impacte o propósito do software.

- Risco = probabilidade x impacto

Vulnerabilidades são erros que um invasor pode explorar

- Ou falhas no design ou falhas na implementação.
- Vulnerabilidades em nível de design são os defeitos mais difíceis de lidar.

Pilares da Segurança de Software

Pilar I: Gestão de Riscos

Pilar II: Touchpoints

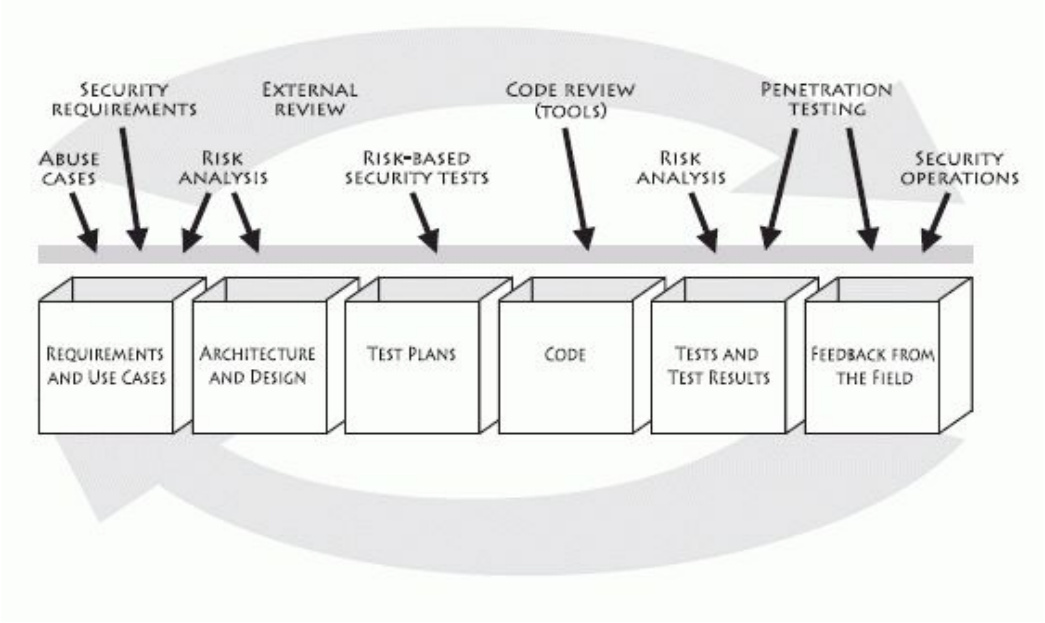
Pilar III - Conhecimento

Pilar I: Gestão de Riscos

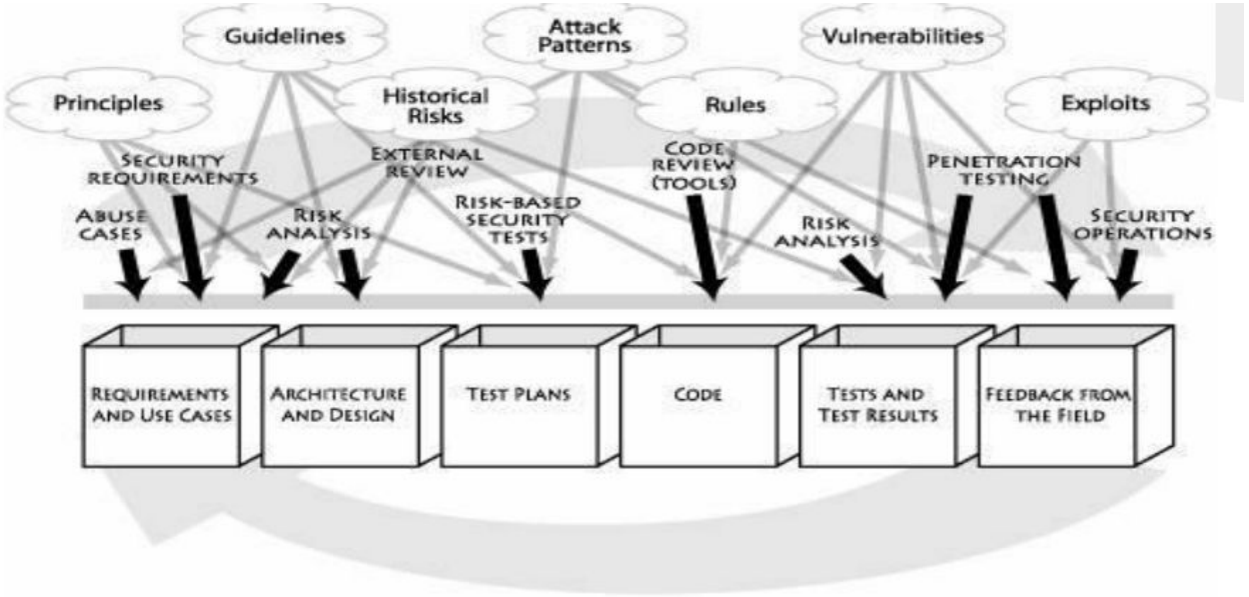
- Um processo contínuo de gerenciamento de risco é uma parte essencial da segurança de software.
- Ele identifica, classifica, rastreia e entende os riscos de segurança de software

Pilar II: Touchpoints

- Os pontos de contato de segurança são um conjunto de práticas recomendadas de segurança.



Pilar III - Conhecimento



Mapeamento de catálogos de conhecimento de segurança de software para vários artefatos de software e melhores práticas de segurança de software.

Princípios de Design de Segurança de Software

Aspectos-chave da Segurança da Informação

- **Integridade** significa garantir que somente usuários autorizados possam alterar informações, e somente de maneiras aprovadas.
    - Por exemplo, suponha que uma empresa permita que cada vendedor atualize somente seus leads, enquanto os gerentes de vendas podem atualizar todos os leads. Nesse caso, uma violação de integridade acontece se um vendedor tenta alterar os leads de outra pessoa. Isso quebra a regra de que somente o vendedor original pode atualizar seus leads.
- **Disponibilidade** significa que usuários autorizados podem acessar o sistema quando necessário.
    - Por exemplo, se o sistema de gerenciamento de leads estiver em um servidor web, restrições de IP podem ser usadas para controlar o acesso com base em endereços IP. Se todos os membros de vendas usarem o endereço IP 192.168.1.23 para acessar o sistema, bloquear todos os outros IPs garantiria que apenas usuários autorizados pudessem acessar o sistema de um local autorizado.

Aspectos-chave da Segurança da Informação

- **Confidencialidade** envolve impedir acesso não autorizado a certas informações ou ferramentas. Idealmente, aqueles sem acesso nem deveriam saber que informações ou ferramentas sensíveis existem.
    - Por exemplo, no sistema de gerenciamento de leads de vendas, os leads só podem ser alterados pelo vendedor que os criou.

Por que o Design de um Software Seguro é Importante?

- A segurança de software é uma parte crítica do processo de desenvolvimento que ajuda a proteger aplicativos de ameaças ou ataques potenciais.
- Construir aplicativos sem considerar a segurança pode levar a consequências desastrosas, como violações de dados, perda de confiança e implicações financeiras.

O custo médio global de uma violação de dados em 2023 atingiu uma alta histórica de US$ 4,45 milhões, um aumento de 2,3% em relação a 2022 e um aumento de 15,3% em relação a 2020, demonstrando o imenso risco financeiro associado ao software de design de segurança inadequado.

Fonte:https://newsroom.ibm.com/2023-07-24-IBM-Report-Half-of-Breached-Organizations-Unwilling-to-Increase-Security-Spend-Despite-Soaring-Breach-Costs

Os 10 Principais Princípios de Design de Segurança de Software

1. Minimizar superfícies de ataque
2. Estabelecer Padrões de Segurança
3. Adotar o princípio do mínimo privilégio
4. Defesa em profundidade
5. Falhar com segurança
6. Não confiar em serviços
7. Separar funções
8. Evitar segurança por obscuridade
9. Mantenha segurança simples
10. Manutenção e correção segura de problemas

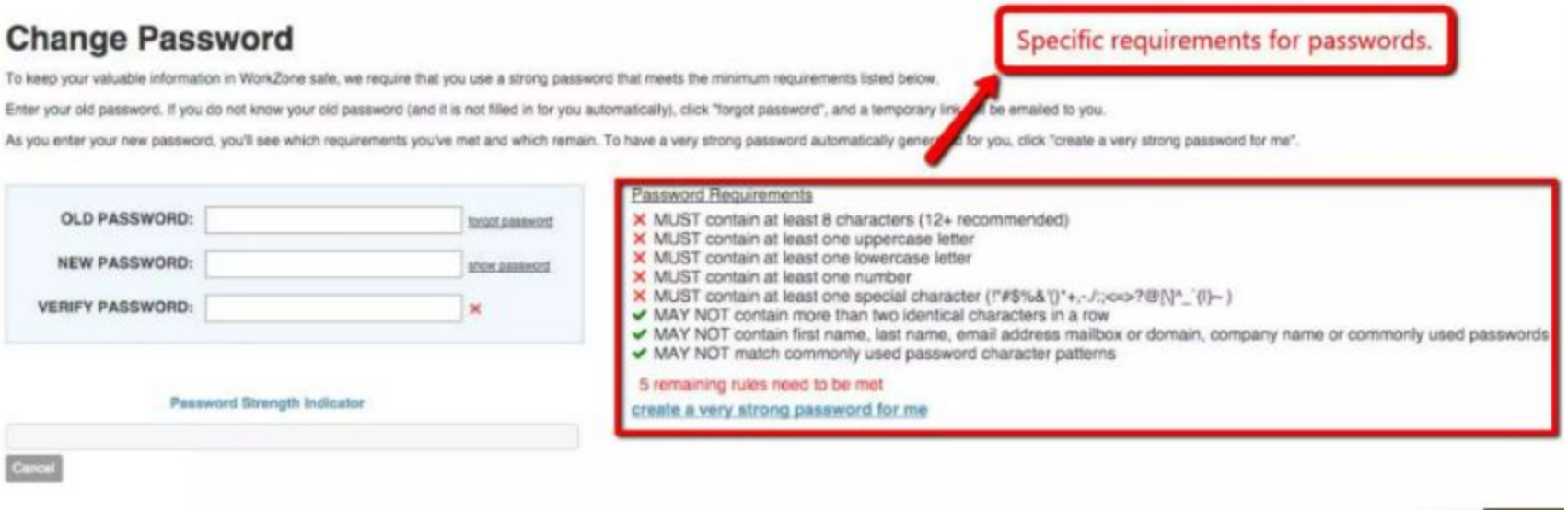
1. Minimizar Superfícies de Ataque

- Todo tempo um programador adicionou uma nova feature em sua aplicação, eles estão aumentando o risco de vulnerabilidade de segurança
- O princípio de minimizar superfícies de ataque significa **restringir as funções que um perfil pode acessar**, em outras palavras, corresponde a identificação de vulnerabilidades e vetores de ataque no aspecto digital ou físico.

2. Estabelecer Padrões de Segurança

- Estabelece que uma aplicação deve ser segura por padrão.
- Isso significa que **um novo usuário deve tomar medidas para obter privilégios** mais elevados e remover medidas de segurança adicionais (se permitido)
- Estabelecer padrões seguros significa que deve haver funções de segurança fortes para:
  - como o registro do usuário é tratado
  - com que frequência a senha deve ser atualizada
  - quão complexas as senhas devem ser e assim por diante
- Os usuários dos aplicativos podem desativar alguns desses recursos, mas eles devem ser definidos para um nível de segurança alto por padrão

2. Estabelecer Padrões de Segurança: Exemplo



3. Adotar o Princípio do Mínimo Privilégio (PLOP)

- O princípio PLOP afirma que um usuário **deve ter um conjunto mínimo de privilégios necessários para executar uma tarefa específica**
- Por exemplo, um usuário que está inscrito em um aplicativo de blog como um "autor"
  - **não deve** ter privilégios administrativos que permitam adicionar ou remover usuários
  - eles **devem** ter permissão apenas para postar artigos no aplicativo
- O PLOP pode ser aplicado a todos os aspectos de um aplicativo da web, incluindo direitos de usuário e acesso a recursos.

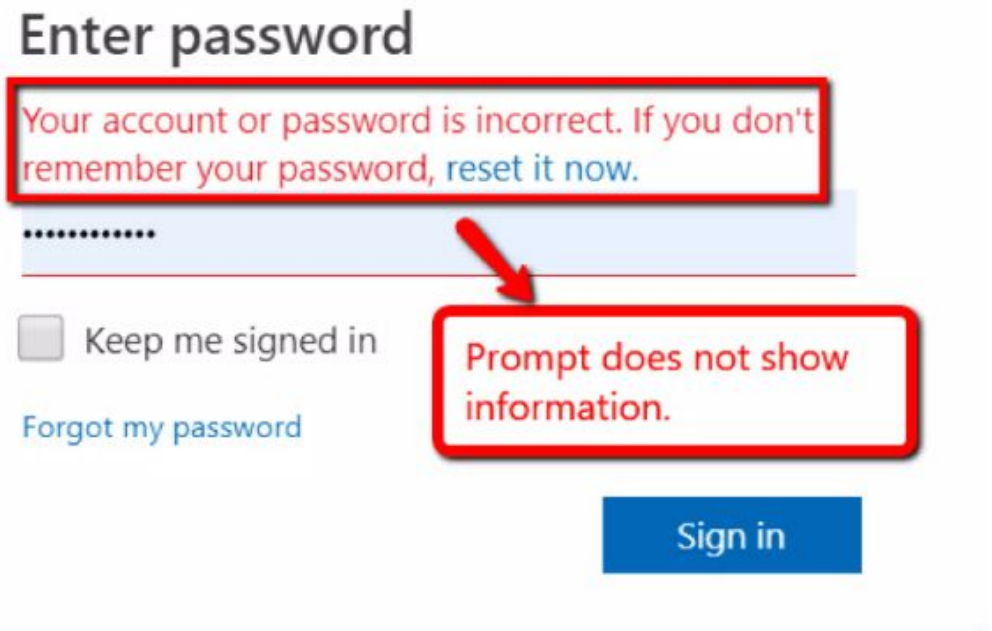
4. Defesa em Profundidade

- O princípio da defesa em profundidade afirma que **múltiplos controles de segurança que abordam o risco de diferentes maneiras** são a melhor opção para proteger um aplicativo
- Então, em vez de ter um controle de segurança para acessos de usuários, você teria **várias camadas de validação**, ferramentas de auditoria de segurança viciante e ferramentas de login.
- Por exemplo, em vez de deixar um usuário fazer login com apenas um nome de usuário e senha
    - você usaria uma verificação de IP
    - um sistema de captcha
    - registro de suas tentativas de login
    - detecção de força bruta e assim por diante

5. Falhar com Segurança

- Há muitas razões pelas quais um aplicativo f**alharia ao processar uma transação**
- Talvez u**ma conexão de banco de dados tenha falhado**, ou os **dados inseridos por um usuário estiverem incorretos**
- Este princípio afirma que os aplicativos devem falhar de forma segura
  - Failure não deve dar privilégios adicionais ao usuário
  - Não deve mostrar ao usuário informações confidenciais como consultas ou logs de banco de dados

5. Falhar com Segurança: Exemplo



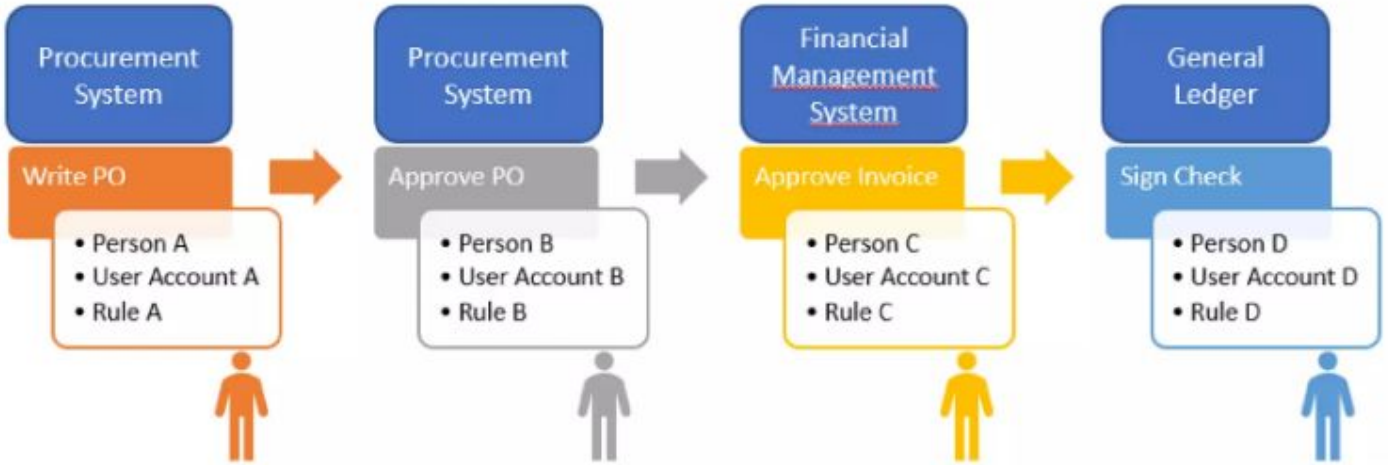
6. Não Confiar em Serviços

- Muitos aplicativos da web usam **serviços de terceiros** para acessar funcionalidades adicionais ou obter dados adicionais
- Este princípio afirma que você nunca deve confiar nesses serviços de uma perspectiva de segurança
- Isso significa que o aplicativo deve sempre **verificar a validade dos dados que os serviços de terceiros enviam** e não dar a esses serviços permissões de alto nível dentro do aplicativo.

7. Separar funções

- A separação de funções, como princípio de segurança, tem como **objetivo principal a prevenção de fraudes e erros**
- Este objetivo é alcançado distribuindo as tarefas e privilégios associados para um processo de negócios específico entre vários usuários
- Por exemplo, um usuário de um site de comércio eletrônico não deve ser promovido para também ser um administrador, pois ele poderá alterar pedidos e fornecer produtos para si mesmo.
- O inverso também é verdadeiro - e o administrador não deve ter a capacidade de fazer coisas que os clientes fazem, como pedir itens do front-end do site.

7. Separar Funções: Exemplo



- Definição de regras de segregação de funções aplicáveis ao ambiente
- Criação de matriz de riscos
- Análise de riscos para identificação de violações da segregação de funções
- Análise de atividades conflitantes executadas por usuários alternativos
- Resolução de conflitos que apresentem alto risco

8. Evitar Segurança por Obscuridade

- Em engenharia de segurança, **segurança através da obscuridade** (ou **segurança pela obscuridade**) é a **confiança no sigilo do design ou implementação** como o principal método de fornecer segurança para um sistema ou componente de um sistema
- Um sistema ou componente que depende da obscuridade pode ter vulnerabilidades de segurança teóricas ou reais, mas seus proprietários ou designers acreditam que se as falhas não forem conhecidas, isso será suficiente para evitar um ataque bem-sucedido
- Especialistas em segurança rejeitam essa visão e afirmam que a obscuridade não deve ser o único mecanismo de segurança
- se seu aplicativo requer que sua URL de administração seja oculta para que ele possa permanecer seguro, então ele não é seguro de forma alguma
- deve haver controles de segurança suficientes para manter seu aplicativo seguro sem ocultar a funcionalidade principal ou o código-fonte

8. Evitar segurança por obscuridade: Exemplo

- Algumas pessoas gostam de tornar seu javascript difícil de ler (e, portanto, hackear) usando **ofuscação.**
- O Google está entre os usuários dessa técnica. No nível mais simples, eles mudam os nomes das variáveis e métodos para uma única letra misteriosa, esta primeira variável é chamada de "a", a segunda é chamada de "b" e assim por diante.
    - Isso consegue tornar o javascript extremamente difícil de ler e seguir
    - e adiciona alguma proteção à propriedade intelectual contida no código javascript, que deve ser baixado para o navegador do usuário para ser utilizável, tornando-o acessível a todos.

8. Evitar segurança por obscuridade: Exemplo

Original Source Code Before Rename Obfuscation

```
private void
CalculatePayroll(SpecialList
employeeGroup) {
    while(employeeGroup.HasMore()) {
        employee =
employeeGroup.GetNext(true);
        employee.UpdateSalary();
        DistributeCheck(employee);
    }
}
```

Reverse-Engineered Source Code After Rename Obfuscation

```
private void a(a b) {
    while (b.a()) {
        a = b.a(true);
        a.a();
        a(a);
    }
}
```

9. Mantenha a Segurança Simples

Os desenvolvedores devem evitar o uso de arquitetura muito sofisticada ao desenvolver controles de segurança para seus aplicativos

Ter mecanismos muito complexos pode aumentar o risco de erros. **Na prática:**

- Reutilize componentes que você sabe que são confiáveis
- Evite arquitetura e codificação complexas
- centralize suas abordagens tornando os fundamentos parte do seu design
- Integre suas ferramentas de segurança dentro dos ambientes já familiares dos desenvolvedores, incluindo IDEs, sistemas de rastreamento de bugs, etc.

10. Manutenção e Correção Segura de Problemas

- Se um problema de segurança for identificado em um aplicativo, os **desenvolvedores devem**:
  - **determinar a causa raiz do problema**
  - eles devem então repará-lo
  - testar os reparos rapidamente (testes de regressão)
- Se o aplicativo usa padrões de design, é provável que o **erro esteja presente em vários sistemas**
- Os programadores devem ter cuidado para identificar todos os sistemas afetados

Vulnerabilidades Comuns de Software para Aplicações Web

Padrão de conscientização para desenvolvedores e

segurança de aplicações Web

*Representa um amplo consenso sobre os **riscos de segurança** mais críticos para aplicações Web*

Fonte: OWASP - Open Web Application Security Project. The OWASP Top Ten.Owasptopten.org. [s.d]. Disponível em: <https://www.owasptopten.org/>.

Evolução do Projeto

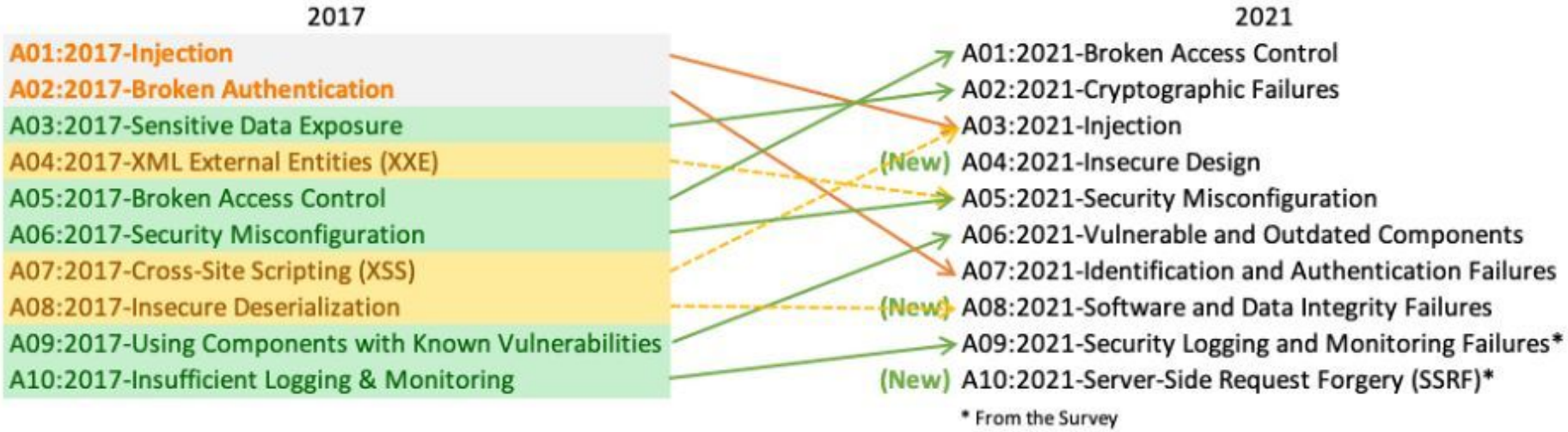


Imagem: OWASP - Open Web Application Security Project. OWASP Top 10:2021. Owasp.org/Top10/. [s.d]. Disponível em: <https://owasp.org/Top10/>.

Top 10 → riscos

CVE → aplicações vulneráveis

CWE → vulnerabilidades

CAPEC
ATT&CK
} Táticas, Técnicas e Procedimentos (TTPs)

OWASP Top 10

A1:2021. Violação/Quebra do Controle de Acesso

A2:2021. Falhas Criptográficas

A3:2021. Injeção

A4:2021. Projeto Inseguro

A5:2021. Configuração Incorreta de Segurança

A6:2021. Componentes Vulneráveis ou Desatualizados

A7:2021. Falhas de Identificação e Autenticação

A8:2021. Falhas de Integridade de Software e Dados

A9:2021. Falhas no Registro e Monitoramento de Segurança

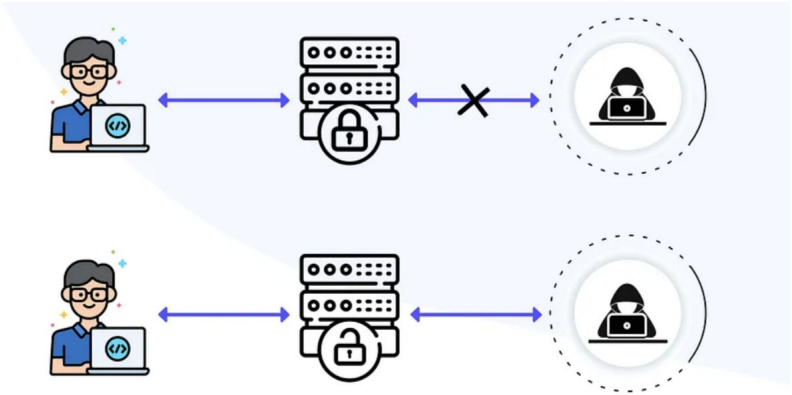
A10:2021. Falsificação de Requisição do Lado do Servidor

A01:2021 – Quebra de Controle de Acesso

Exemplo: Um usuário comum acessa uma página de administração apenas alterando a URL para /admin.

Explicação:

- O controle de acesso impede que usuários não autorizados acessem recursos protegidos.
- Quando há falhas, usuários podem acessar áreas que não deveriam, como dados de outros usuários ou funções administrativas, simplesmente manipulando a URL, parâmetros ou usando funções de desenvolvedor no navegador.



A01:2021 – Quebra de Controle de Acesso: Exemplo

Passagem de parâmetros sem verificação

- Código Vulnerável:

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

- Ataque:

```
https://example.com/app/accountInfo?acct=<número-de-conta>
```

A01:2021 – Quebra de Controle de Acesso: Exemplo

Requisições não verificadas a recursos

- O atacante obtém acesso a partes restritas da aplicação

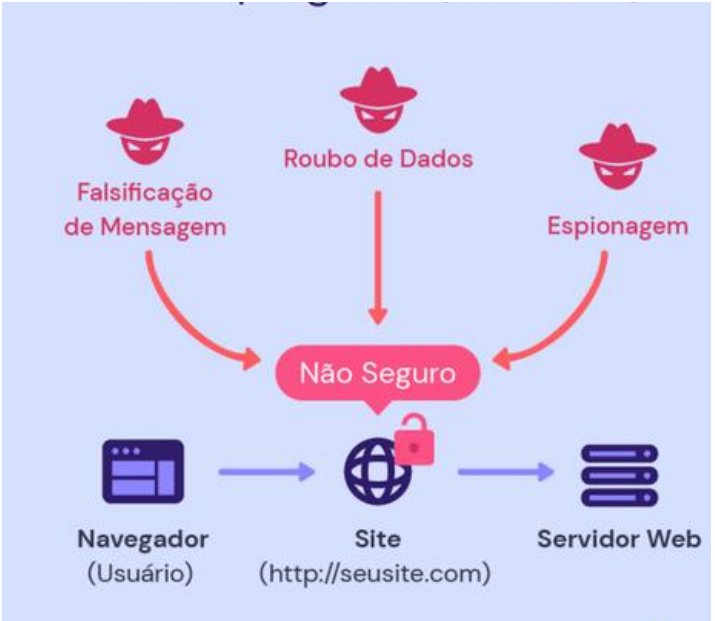
```
https://example.com/app/getappInfo

https://example.com/app/admin_getappInfo
```

A02:2021 – Falhas Criptográficas

Exemplo: Armazenar senhas em **texto simples** ou usar **algoritmos fracos** como MD5 ou DES para criptografar dados sensíveis.

Explicação:

- Refere-se à falha em proteger dados sensíveis por meio de **criptografia inadequada**.
- Se dados como senhas, números de cartão de crédito ou dados pessoais não forem devidamente protegidos, um invasor pode interceptá-los ou acessá-los facilmente.



A02:2021 - Ataques Criptográficos: Exemplos

Tipos (Alguns):

- Somente cifra
- Texto plano conhecido (parte)
- Texto plano escolhido (todo)
- Força bruta

Resultados

- Ruptura total – descoberta da chave
- Dedução global – algoritmo equivalente
- Dedução local – novas cifras ou textos plano
- Dedução de informação

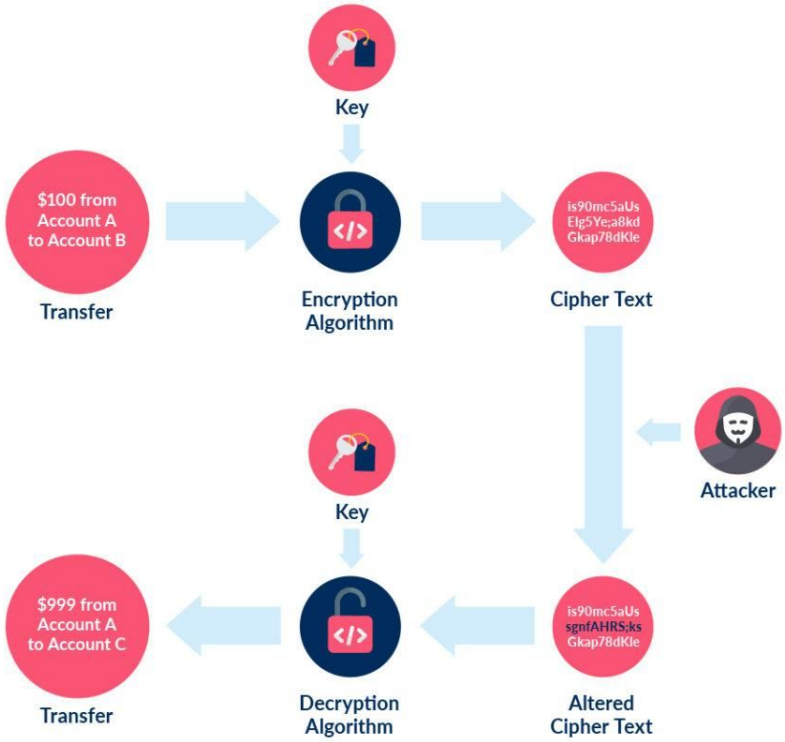


Imagem: SCHNEIR, G. Authenticated Encryption: An Explainer. Ubiqsecurity.com, 12 jan. 2021

A02:2021 – Falhas Criptográficas

Vulnerabilidades:

- Senhas escritas (hard-coded) em códigos ou arquivos – CWE-259
  - https://cwe.mitre.org/data/definitions/259.html
- Algoritmos criptográficos inseguros – CWE-327
  - https://cwe.mitre.org/data/definitions/327.html
- Entropia insuficiente – CWE-331
  - https://cwe.mitre.org/data/definitions/331.html

A03:2021 – Injeção (Injection)

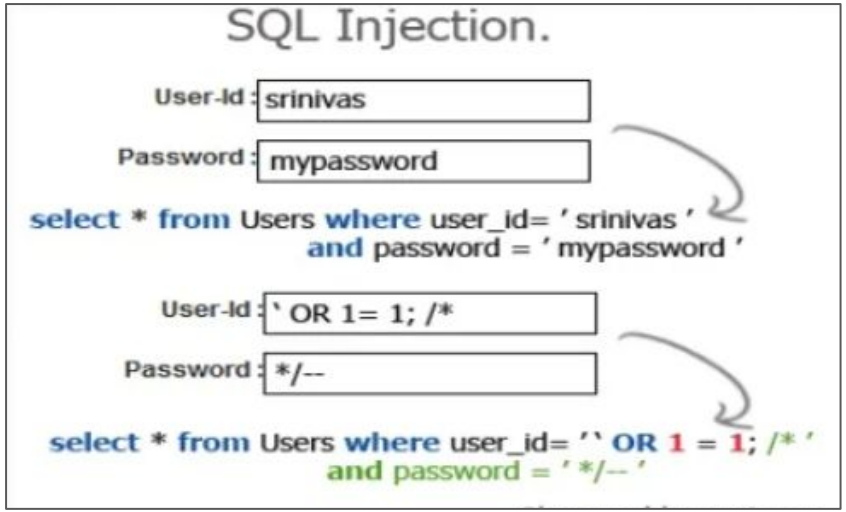
Ocorrem quando a aplicação repassa dados não confiáveis para um interpretador

A03:2021 – Injeção (Injection)

Exemplo: Inserir um comando SQL malicioso em um campo de login, como `" OR 1=1; --`, que permite ao atacante contornar a autenticação e acessar dados do banco de dados.

Explicação:

- A injeção ocorre quando um atacante envia dados maliciosos em uma entrada de sistema, como um formulário, que o aplicativo executa de maneira imprópria. O **SQL Injection** é o tipo mais comum, mas também pode ocorrer em **comandos de shell**, **XML** e **LDAP**.



A03:2021 – Injeção (Injection)

Não confiar nas entradas recebidas do cliente!

A03:2021 – Injeção (Injection)

Vulnerabilidades:

- SQL Injection – CWE-89
  - https://cwe.mitre.org/data/definitions/89.html
- ● Cross-Site Scripting – XSS – CWE-79
  - https://cwe.mitre.org/data/definitions/79.html
- Controle externo de nomes de arquivos ou caminhos - CWE-73
  - https://cwe.mitre.org/data/definitions/73.html

A03:2021 – Injeção (Injection)

Impactos:

- Ler dados sensíveis do BD
- Modificar dados no BD (inserir/atualizar/excluir)
- Executar operações de administração no BD (como desligar o DBMS)
- Recuperar o conteúdo de um determinado arquivo existente no sistema de arquivos do DBMS
- Gravar arquivos no sistema de arquivos
- Lançar comandos para o SO

A04:2021 – Projeto Inseguro - Exposição de Dados Sensíveis

Projeto inseguro vs Implementação insegura

- Um projeto seguro pode ter uma implementação insegura levando a vulnerabilidades que podem ser exploradas.

  Um projeto inseguro não pode ser corrigido por uma implementação “perfeita”, uma vez que os controles de segurança necessários não foram criados.

A04:2021 – Projeto Inseguro - Exposição de Dados Sensíveis

Exemplo: Armazenar dados bancários sem criptografia ou usar algoritmos de criptografia fracos.

Explicação:

- A falta de segurança no design de um sistema pode levar à **exposição de dados sensíveis**. Isso pode acontecer se dados como **números de cartão de crédito**, **informações pessoais** ou **senhas** forem mal gerenciados ou não criptografados corretamente.

A04:2021 – Projeto Inseguro - Exposição de Dados Sensíveis

Vulnerabilidades:

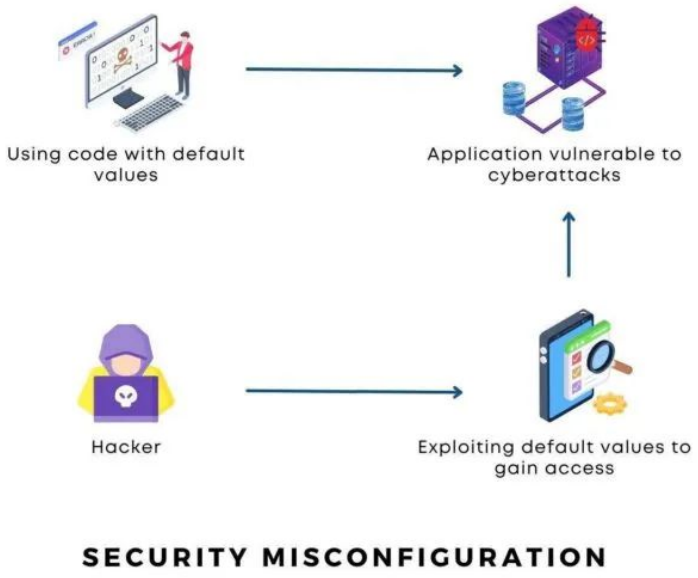
- Mensagens de erro contendo dados sensíveis – CWE-209
  - https://cwe.mitre.org/data/definitions/209.html
- Armazenamento desprotegido de credenciais – CWE-256
  - https://cwe.mitre.org/data/definitions/256.html
- Violação de Fronteira de Confiança – CWE-501
  - https://cwe.mitre.org/data/definitions/501.html
- Credenciais insuficientemente protegidas – CWE-522
  - https://cwe.mitre.org/data/definitions/522.html

A05:2021 – Configuração Incorreta de Segurança

Exemplo: Expor dados de configuração do banco de dados em um ambiente de produção ou não aplicar as correções de segurança necessárias em servidores.

Explicação:

- Configurações inadequadas, tanto em servidores como em sistemas, podem **expor dados ou funcionalidades sensíveis**. Isso inclui permissões excessivas, **exposição de arquivos de configuração**, ou servidores em **ambientes de produção mal configurados**.



A05:2021 – Configuração Incorreta de Segurança

Ocorre quando… (alguns cenários)

- Falta de mecanismos de segurança apropriados em qualquer parte da pilha de aplicações ou permissões configuradas incorretamente em serviços de nuvem
- **Recursos desnecessários** são ativados ou instalados
- As contas padrão e suas senhas ainda estão habilitadas e inalteradas
- O **tratamento de erros** revela stacktraces ou outras mensagens de erro
- excessivamente informativas aos usuários

A05:2021 – Configuração Incorreta de Segurança: Exemplo

```
HTTP Status 500 - For input string: "null" type Exception report

message For input string: "null"

description The server encountered an internal error that prevented it from fulfilling this request. exception

java.lang.NumberFormatException: For input string: "null"

java.lang.NumberFormatException.forInputString(NumberFormatException.java:65) java.lang.Integer.parseInt(Integer.java:492)
java.lang.Integer.parseInt(Integer.java:527)

sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
java.lang.reflect.Method.invoke(Method.java:606)
com.opensymphony.xwork2.DefaultActionInvocation.invokeAction(DefaultActionInvocation.java:450)
com.opensymphony.xwork2.DefaultActionInvocation.invokeActionOnly(DefaultActionInvocation.java:289)
com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:252)
org.apache.struts2.interceptor.debugging.DebuggingInterceptor.intercept(DebuggingInterceptor.java:256)
com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:246)

...

note: The full stack trace of the root cause is available in the Apache Tomcat/7.0.56 logs.
```

A05:2021 - Mensagens de Erro: Consequências

- Informações sobre APIs utilizadas
- Mapeamento das estruturas internas no lado do servidor
- Versões e tipos de aplicações utilizadas
- Derrubada do servidor
    - condições de deadlock, race conditions, etc
- Desvio de controles quando exceções não são tratadas adequadamente

A06:2021 – Componentes Vulneráveis ou Desatualizados

Ocorre quando …

- Falta do conhecimento sobre versões de componentes utilizados
  - (tanto do lado do cliente quanto do lado do servidor)
  - Componentes de uso direto, bem como dependências aninhadas
- Software vulnerável, sem suporte ou desatualizado.
  - Sistema operacional, servidor web, sistema de gerenciamento de banco de dados (DBMS), aplicações, APIs e todos os componentes, ambientes de tempo de execução e bibliotecas.
- Ausência de verificação de vulnerabilidades regularmente e não acompanhamento de boletins de segurança relacionados aos componentes utilizados.

A06:2021 – Componentes Vulneráveis ou Desatualizados

Ocorre quando …

- Ausência de correção ou atualização da plataforma, estruturas e dependências subjacentes de maneira oportuna e **baseada em riscos**
    - 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
- Os desenvolvedores de software não testam a compatibilidade de bibliotecas atualizadas, aprimoradas ou corrigidas
- Ausência de proteção das configurações dos componentes

A07:2021. Falhas de Identificação e Autenticação

Ocorre quando …

A aplicação permite:

- Ataques automatizados
  - Ex.: Credential Stuffing → em que o invasor possui uma lista de nomes de usuários e senhas válidos
- Força bruta ou outros ataques automatizados
- Senhas padrão, fracas ou conhecidas, como "Password1" ou "admin/admin"
- Processo de recuperação de senhas fraco ou ineficiente
  - Ex.: sistema de recuperação através de “respostas baseadas em conhecimento”

A07:2021. Falhas de Identificação e Autenticação

Ocorre quando …

- Armazenamento inseguro de senhas
  - Senhas em texto plano, senhas encriptadas ou algoritmos de hash fracos
- Autenticação multifator ausente ou ineficaz
- Exposição do identificador de sessão na URL
- Reutilização do 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

Credential Stuffing

- Uso de listas de credenciais conhecidas em domínios/sites/aplicações diferentes
- A aplicação pode ser usada como um “oráculo de senhas” para determinar se as credenciais são válidas

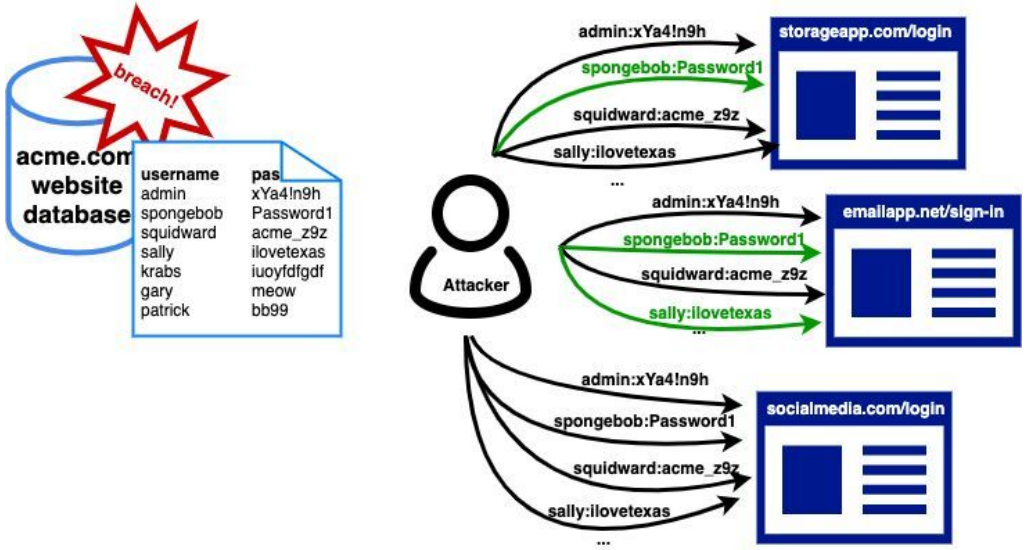


Imagem: MUELLER, N. et al. Credential stuffing. Owasp.org, [s.d.].
Disponível em: < https://owasp.org/www-community/attacks/Credential_stuffing>.

A08:2021 - Falha de Integridade de Software e Dados

Ocorre quando…

- Aplicações que dependem de complementos de origens (repositórios ou CDNs) não confiáveis
- Atualizações ou instalações onde a verificação de integridade é insuficiente
- ou inexistente
- [De]Serialização de dados

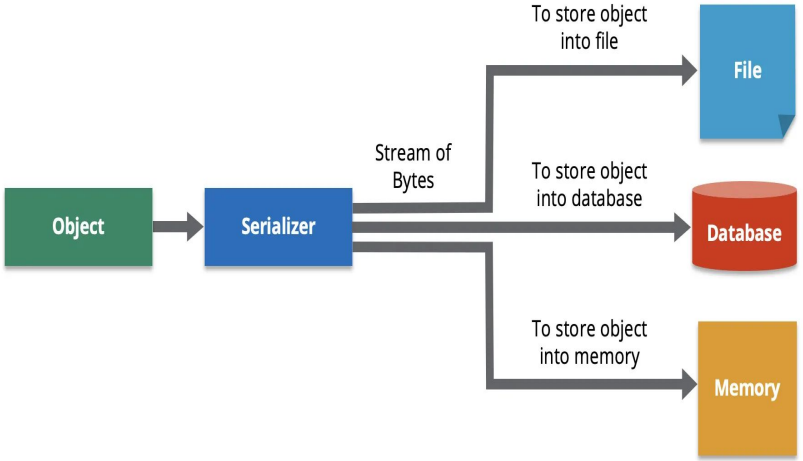
A08:2021 - Falha de Integridade de Software e Dados

Potencial para

- Acesso não autorizado
- Malware
- Comprometimento do sistema

A08:2021 - Falha de Integridade de Software e Dados: Exemplo

Exemplo: Um atacante envia um objeto serializado malicioso para um servidor, que é deserializado sem verificação, permitindo que o código do atacante seja executado no sistema.



Explicação:

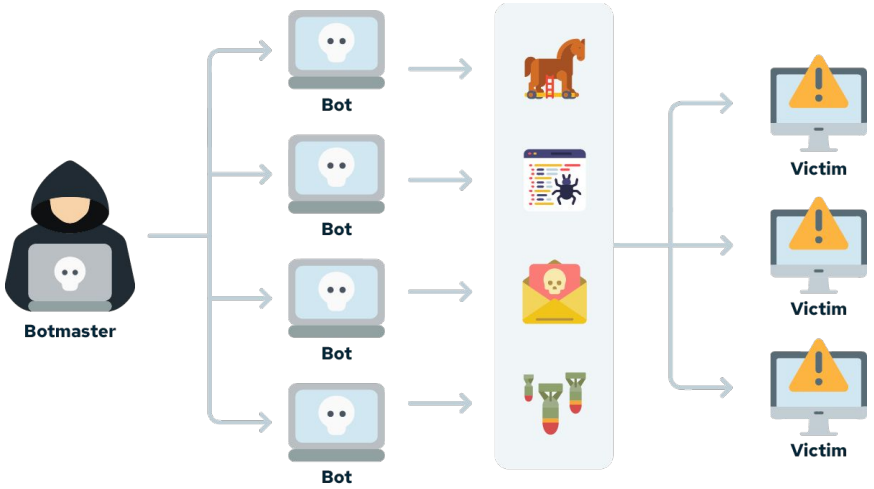
- Quando os dados do usuário são serializados e desserializados de maneira insegura, um atacante pode manipular esses dados e executar código malicioso. Isso ocorre quando um aplicativo não valida corretamente os dados antes de processá-los.

A09:2021 - Falhas no Registro e Monitoramento de Segurança

Exemplo: Não registrar falhas de autenticação ou tentativas de acesso não autorizado, dificultando a detecção de um ataque de **força bruta**.

Explicação:

- A falta de **registro adequado de atividades** e **monitoramento contínuo** torna difícil detectar ataques em tempo real, o que pode atrasar a resposta a incidentes de segurança. Isso também dificulta a **investigação forense** após um ataque.



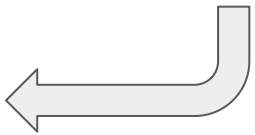
A9:2021 - Falhas no Registro e Monitoramento de Segurança: Exemplo

Omissão de Informações Relevantes à Segurança – CWE 223

- Ausência informações relevantes para a identificação de ameaças nos logs

```
function login($userName,$password){
      if(authenticate($userName,$password)){
            return True;
      }
      else{ incrementLoginAttempts($userName);
            if(recentLoginAttempts($userName) > 5){
                  writeLog("Failed login attempt by User: " . $userName . " at " + date('r')); }
      }
}
```

Não registra tentativas mal sucedidas abaixo dessa quantidade



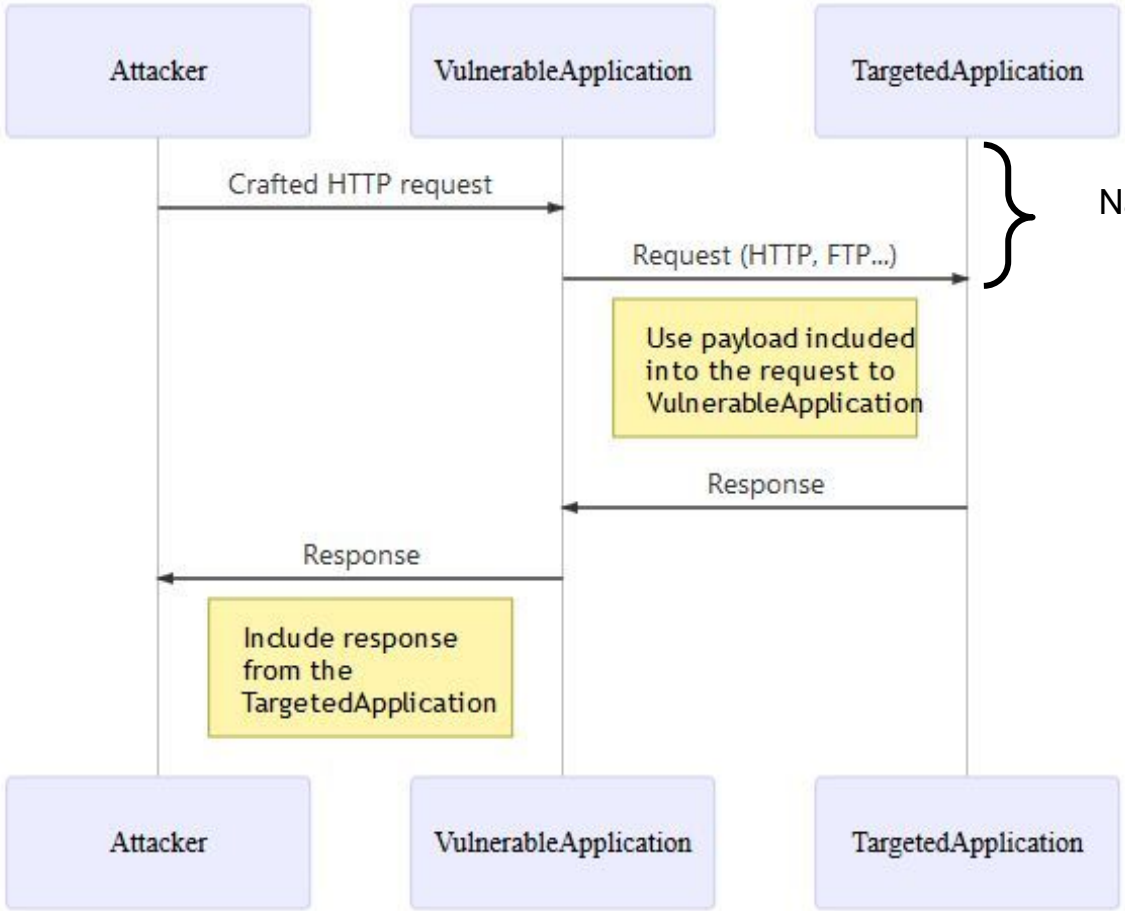
O que NÃO registrar?

- Trechos de código, por exemplo, queries ao BD
- Identificadores de sessão
- Senhas de acesso
- Informações sensíveis dos usuários
- Informações relacionadas à infraestrutura, por exemplo, caminhos na árvore de diretórios
- Nomes ou endereços de serviços/servidores internos

A10:2021 - Falsificação de Registro do Lado do Servidor

Ocorre quando…

- **S**erver **S**ide **R**equest **F**orgery
  - Vetor de ataque que explora a interação da aplicação com recursos na rede interna ou externa, ou até na mesma máquina
- Durante a busca por um recurso remoto (javascript, imagem, css, etc), sem a validação da URL fornecida, ou controlada, pelo cliente
  - Permite que o atacante force a aplicação a fazer requisições para destinos desconhecidos/não permitidos, ainda que por trás de um FW ou dentro de uma VPN



Não se restringe ao protocolo HTTP

- O alvo do payload pode ser qualquer outro protocolo ou esquema
  - ftp://
  - file://
  - data://
  - dict://
  - phar://
  - etc

Imagem: OWASP. Server-side request forgery prevention cheat sheet. Cheatsheetseries.owasp.org, [s.d.]. Disponível em: <https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html>.

Padrões de Segurança

Controle de Acesso Seguro *(Mitiga A01:2021 – Quebra de Controle de Acesso)*

- **Explicação:**

O controle de acesso impede que usuários não autorizados acessem dados ou funções administrativas.

- **Exemplo:**

Um usuário comum tenta acessar a página de administração alterando a URL de `/perfil` para `/admin`. Se o sistema não verificar permissões corretamente, ele poderá acessar funções administrativas.

- **Solução:**

  ✔ Implementar verificações no **lado do servidor**.

  ✔ Usar **tokens seguros (JWT, OAuth)** para controle de acesso.

  ✔ Aplicar **RBAC (Role-Based Access Control)** para definir permissões corretamente.

Boas Práticas de Criptografia *(Mitiga A02:2021 – Criptografia Quebrada)*

- **Explicação:**

Protege dados sensíveis contra acessos não autorizados, garantindo que apenas usuários legítimos possam acessá-los.

- **Exemplo:**

Uma aplicação armazena senhas em texto puro no banco de dados. Se um atacante obtiver acesso, poderá visualizar todas as credenciais.

- **Solução:**

  ✔ Usar **bcrypt, Argon2 ou PBKDF2** para armazenar senhas.

  ✔ **Criptografar dados sensíveis** (como números de cartão) com AES-256.

  ✔ Habilitar **TLS** para proteger dados em trânsito.

Prevenção de Injeção (SQL Injection, XSS) *(Mitiga A03:2021 – Injeção e A07:2021 – XSS)*

🔹 **Explicação:**
Protege contra ataques onde entradas maliciosas são executadas pelo sistema, como injeção de código SQL ou scripts JavaScript.
🔹 **Exemplo (SQL Injection):**
Um atacante insere a seguinte entrada em um campo de login:

```
' OR '1'='1'; --
```

Se o sistema não sanitiza essa entrada, ele pode burlar a autenticação e acessar o banco de dados.
🔹 **Solução:**
✔ **Utilizar consultas preparadas (Prepared Statements)** para evitar SQL Injection.
✔ **Sanitizar e escapar entradas** antes de renderizá-las no navegador (para evitar XSS).
✔ Aplicar **Content Security Policy (CSP)** para impedir execução de scripts não autorizados.

Segurança no Design do Sistema (Mitiga A04:2021 – Exposição de Dados Sensíveis)

- Explicação:

Prevenir vulnerabilidades desde a fase de design reduz riscos estruturais, evitando a exposição de dados sensíveis.

- Exemplo:

Um aplicativo financeiro exibe números de cartão de crédito sem mascaramento. Se um invasor acessar a conta de outro usuário, ele poderá visualizar todas as informações.

- Solução:

✔ Aplicar princípios de segurança desde a fase de design.

✔ Mascarar informações sensíveis (exibir ****-****-****-1234 em vez do número completo).

✔ Implementar segurança em múltiplas camadas, combinando criptografia, controle de acesso e monitoramento.

Melhoria na Autenticação (MFA, Senhas Fortes) (Mitiga A05:2021 – Quebra de Autenticação)

- **Explicação:**

Garante que apenas usuários legítimos possam acessar o sistema, impedindo ataques como roubo de credenciais e ataques de força bruta.

- **Exemplo:**

Um usuário reutiliza uma senha fraca, como "123456", e um atacante consegue acessá-la através de um vazamento de dados.

- **Solução:**

✔ Implementar **autenticação multifator (MFA)** para acessos sensíveis.

✔ Exigir **senhas fortes** e utilizar hash seguro (bcrypt).

✔ Implementar **bloqueio de conta temporário** após várias tentativas falhas.

Configuração Segura de Sistemas (Mitiga A06:2021 – Configuração de Segurança Incorreta)

- **Explicação:**

Evita falhas de segurança causadas por configurações incorretas em servidores, bancos de dados e aplicações.

- **Exemplo:**

Uma aplicação expõe informações sensíveis devido a um arquivo `.env` mal configurado, permitindo que atacantes acessem credenciais do banco de dados.

- **Solução:**

✔ **Desativar informações de depuração** em produção.
✔ Usar **privilégios mínimos** para serviços e bancos de dados.
✔ Implementar **segurança por padrão**, evitando permissões excessivas.

Validação de Entrada do Usuário (Mitiga A08:2021 – Deserialização Insegura)

- **Explicação:**

Evita que dados maliciosos sejam processados sem validação, prevenindo ataques via deserialização insegura.

- **Exemplo:**

Um sistema recebe um objeto serializado de um usuário e o processa sem validação. Se o atacante modificar esse objeto, poderá executar comandos no servidor.

- **Solução:**

✔ **Validar e sanitizar dados** antes da desserialização.

✔ **Evitar desserialização automática** de objetos desconhecidos.

✔ **Utilizar formatos seguros**, como JSON em vez de binários inseguros.

Gestão de Dependências e Atualizações (Mitiga A09:2021 – Uso de Componentes com Vulnerabilidades Conhecidas)

🔹 **Explicação:**

Manter bibliotecas, frameworks e dependências atualizadas reduz o risco de exploração de vulnerabilidades conhecidas.

🔹 **Exemplo:**

Uma aplicação utiliza uma versão desatualizada do Apache Struts, que contém uma vulnerabilidade explorável para execução remota de código (RCE).

🔹 **Solução:**

✔ **Monitorar vulnerabilidades conhecidas** em dependências com ferramentas como Dependabot, Snyk ou OWASP Dependency Check.

✔ **Atualizar regularmente** bibliotecas e frameworks.

✔ **Remover dependências desnecessárias** para reduzir a superfície de ataque.

Monitoramento e Auditoria de Segurança *(Mitiga A10:2021 – Insuficiente Registro e Monitoramento)*

- **Explicação:**

Ajuda a detectar atividades suspeitas e ataques antes que causem danos.

- **Exemplo:**

Uma empresa não registra tentativas de login falhas. Um atacante realiza um ataque de força bruta por semanas sem ser detectado.

- **Solução:**

✔ Habilitar **logs detalhados** para falhas de autenticação, acessos suspeitos e alterações de permissão.

✔ Configurar **alertas automáticos** para padrões de ataque.

✔ Usar um **SIEM (Security Information and Event Management)** para análise e resposta rápida.

Implementação de Políticas de Segurança e Treinamento (Mitiga diversas vulnerabilidades do OWASP Top 10)

- **Explicação:**

A conscientização dos desenvolvedores e usuários sobre segurança cibernética reduz a probabilidade de falhas humanas e ataques sociais.

- **Exemplo:**

Funcionários reutilizam senhas entre contas pessoais e corporativas. Se um vazamento expuser essas senhas, um atacante pode acessá-las.

- **Solução:**

✔ Treinar equipes para reconhecer e evitar ataques de engenharia social.

✔ Implementar **políticas de segurança bem definidas** (ex.: regras para senhas, controle de acesso, boas práticas de codificação segura).

✔ Realizar **testes regulares de segurança**, como pentests e auditorias.

Obrigado!!!!