OWASP TOP 10
2021

A01 Broken Access Control

O que é?

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

Fatores de autenticação: Algo que eu sou, algo que eu tenho ou algo que eu sei – Dois desses tem multifactor autentication.

Quando é vulnerável?

- Violação do princípio de privilégio mínimo ou negação por padrão, onde o acesso deve ser concedido apenas para recursos, funções ou usuários específicos, mas está disponível para qualquer pessoa.
- Ignorando as verificações de controle de acesso modificando a URL (alteração de parâmetro ou navegação forçada), o estado interno do aplicativo ou a página HTML, ou usando uma ferramenta de ataque modificando solicitações de API.
- Permitir a visualização ou edição da conta de outra pessoa, fornecendo seu identificador exclusivo (referências de objetos diretos inseguros)
- Acessando API com controles de acesso ausentes para POST, PUT e DELETE.
- Elevação de privilégio. Atuar como usuário sem estar conectado ou atuar como administrador quando estiver conectado como usuário.
- Manipulação de metadados, como reproduzir ou adulterar um token de controle de acesso JSON Web Token (JWT), ou um cookie ou campo oculto manipulado para elevar privilégios ou abusar da invalidação de JWT.
- A configuração incorreta do CORS permite o acesso à API de origens não autorizadas/não confiáveis.
- Força a navegação em páginas autenticadas como usuário não autenticado ou em páginas privilegiadas como usuário padrão.

Como prevenir?

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

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

A02 Cryptographic Failures

O que é?

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

Quando acontece?

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

- As chaves criptográficas padrão estão em uso, as chaves criptográficas fracas são geradas ou reutilizadas ou o gerenciamento ou rotação de chaves está ausente? As chaves criptográficas são verificadas nos repositórios de código-fonte?
- A criptografia não é aplicada, por exemplo, há alguma diretiva de segurança de cabeçalhos HTTP (navegador) ou cabeçalhos ausentes?
- O certificado do servidor recebido e a cadeia de confiança estão devidamente validados?
- Os vetores de inicialização são ignorados, reutilizados ou não gerados suficientemente seguros para o modo de operação criptográfico? Está em uso um modo de operação inseguro, como o BCE? A criptografia é usada quando a criptografia autenticada é mais apropriada?
- As senhas estão sendo usadas como chaves criptográficas na ausência de uma função de derivação de chave de base de senha?
- A aleatoriedade é usada para fins criptográficos que não foram projetados para atender aos requisitos criptográficos? Mesmo que a função correta seja escolhida, ela precisa ser propagada pelo desenvolvedor e, caso contrário, o desenvolvedor sobrescreveu a funcionalidade de propagação forte incorporada a ela com uma semente que não possui entropia/imprevisibilidade suficiente?
- As funções de hash obsoletas, como MD5 ou SHA1, estão em uso ou as funções de hash não criptográficas são usadas quando as funções de hash criptográficas são necessárias?
- Estão em uso métodos de preenchimento criptográfico obsoletos, como PKCS número 1 v1.5?
- As mensagens de erro criptográficas ou informações de canal lateral podem ser exploradas, por exemplo, na forma de ataques oracle de preenchimento?
- Consulte ASVS Crypto (V7), Data Protection (V9) e SSL/TLS (V10)

Como prevenir

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

- Classifique os dados processados, armazenados ou transmitidos por um aplicativo. Identifique quais dados são confidenciais de acordo com as leis de privacidade, requisitos regulatórios ou necessidades de negócios.
- Não armazene dados confidenciais desnecessariamente. Descarte-o o mais rápido possível ou use tokenização compatível com PCI DSS ou até mesmo truncamento. Os dados que não são retidos não podem ser roubados.
- Certifique-se de criptografar todos os dados confidenciais em repouso.
- Garantir que algoritmos, protocolos e chaves padrão atualizados e fortes estejam em vigor; use o gerenciamento de chaves adequado.
- Criptografe todos os dados em trânsito com protocolos seguros, como TLS com cifras de sigilo de encaminhamento (FS), priorização de cifras pelo servidor e parâmetros seguros. Imponha a criptografia usando diretivas como HTTP Strict Transport Security (HSTS).
- Desabilite o armazenamento em cache para respostas que contenham dados confidenciais.

- Aplique os controles de segurança necessários de acordo com a classificação dos dados.
- Não use protocolos legados, como FTP e SMTP, para transportar dados confidenciais.
- Armazene senhas usando funções de hashing adaptáveis e salgadas fortes com um fator de trabalho (fator de atraso), como Argon2, scrypt, bcrypt ou PBKDF2.
- Os vetores de inicialização devem ser escolhidos de acordo com o modo de operação. Para muitos modos, isso significa usar um CSPRNG (gerador de números pseudo-aleatórios criptograficamente seguro). Para modos que exigem um nonce, o vetor de inicialização (IV) não precisa de um CSPRNG. Em todos os casos, o IV nunca deve ser usado duas vezes para uma chave fixa.
- Sempre use criptografia autenticada em vez de apenas criptografia.
- As chaves devem ser geradas criptograficamente aleatoriamente e armazenadas na memória como arrays de bytes. Se uma senha for usada, ela deverá ser convertida em uma chave por meio de uma função de derivação de chave de base de senha apropriada.
- Certifique-se de que a aleatoriedade criptográfica seja usada quando apropriado e que não tenha sido propagada de maneira previsível ou com baixa entropia. A maioria das APIs modernas não exige que o desenvolvedor semeie o CSPRNG para obter segurança.
- Evite funções criptográficas obsoletas e esquemas de preenchimento, como MD5, SHA1, PKCS número 1 v1.5 .
- Verifique de forma independente a eficácia da configuração e configurações.

A03 Injection

O que é?

Criminosos aproveitam-se de códigos concatenados para inserir parâmetros no código, especialmente SQL para obter retornos e fazer modificações. Pode ser utilizada sempre o usuário final passa informações pra aplicação.

Quando é vulnerável:

A aplicação é vulnerável a ataques quando:

As informações fornecidas pelo usuário não são validadas, filtradas ou sanitizadas.

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

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

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

Como diagnosticar

Utilizar programas próprios pra sanitizar o site.

Cada linguagem tem um sanitizador correspondente.

Como prevenir:

- Não utilizar strings e utilizar APIs seguras, evitar utilizar o interpretador em si e usar ferramentas ORM.
- Usar validação de entrada positiva pelo servidor (Essa não é uma defesa completa, pois muitos aplicativos exigem caracteres especiais, como áreas de texto ou APIs para aplicativos móveis.)
- Para quaisquer consultas dinâmicas residuais, escape caracteres especiais usando a sintaxe de escape específica para esse interpretador.
  Nota: Estruturas SQL, como nomes de tabelas, nomes de colunas e assim por diante, não podem ser escapadas e, portanto, os nomes de estrutura fornecidos pelo usuário são perigosos. Este é um problema comum em software de redação de relatórios.
- Use LIMIT e outros controles SQL nas consultas para evitar a divulgação em massa de registros em caso de injeção de SQL.

A04 Insecure Design

O que é?

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.

Gerenciamento de Requisitos e Recursos

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

Design Seguro

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

Ciclo de Vida de Desenvolvimento Seguro

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

Como prevenir

- Estabeleça e use um ciclo de vida de desenvolvimento seguro com profissionais da AppSec para ajudar a avaliar e projetar controles relacionados à segurança e privacidade.
- Estabeleça e use uma biblioteca de padrões de projeto seguros ou componentes prontos para uso de estradas pavimentadas.
- Use a modelagem de ameaças para autenticação crítica, controle de acesso, lógica de negócios e fluxos de chaves.

- Integre linguagem e controles de segurança em histórias de usuários.
- Integre verificações de plausibilidade em cada camada do seu aplicativo (do front-end ao back-end).
- Escreva testes de unidade e integração para validar se todos os fluxos críticos são resistentes ao modelo de ameaça.
- Compile casos de uso e casos de uso indevido para cada camada de seu aplicativo.
- Segregar camadas de camadas no sistema e nas camadas de rede, dependendo das necessidades de exposição e proteção.
- Separe os locatários de forma robusta por design em todas as camadas.
- Limitar o consumo de recursos por usuário ou serviço

A05 Security Misconfiguration

Quando está vulnerável?

- Falta de proteção de segurança apropriada em qualquer parte da pilha de aplicativos ou permissões configuradas incorretamente em serviços de nuvem.
- Recursos desnecessários são ativados ou instalados (por exemplo, portas, serviços, páginas, contas ou privilégios desnecessários).
- As contas padrão e suas senhas ainda estão habilitadas e inalteradas.
- O tratamento de erros revela rastreamentos de pilha ou outras mensagens de erro excessivamente informativas aos usuários.
- Para sistemas atualizados, os recursos de segurança mais recentes são desabilitados ou não configurados com segurança.
- As configurações de segurança nos servidores de aplicativos, estruturas de aplicativos (por exemplo, Struts, Spring, ASP.NET), bibliotecas, bancos de dados etc., não são definidas para valores seguros.
- O servidor não envia cabeçalhos ou diretivas de segurança ou eles não estão configurados para valores seguros.
- O software está desatualizado ou vulnerável (consulte A06:2021-Componentes vulneráveis e desatualizados).
- Sem um processo de configuração de segurança de aplicativos concertado e repetível, os sistemas correm um risco maior.

Como prevenir

Os processos de instalação segura devem ser implementados, incluindo:

- Um processo de proteção repetível torna rápido e fácil a implantação de outro ambiente devidamente bloqueado. Os ambientes de desenvolvimento, controle de

qualidade e produção devem ser configurados de forma idêntica, com credenciais diferentes usadas em cada ambiente. Esse processo deve ser automatizado para minimizar o esforço necessário para configurar um novo ambiente seguro.
- Uma plataforma mínima sem recursos, componentes, documentação e amostras desnecessários. Remova ou não instale recursos e estruturas não utilizados.
- Uma tarefa para revisar e atualizar as configurações apropriadas para todas as notas de segurança, atualizações e patches como parte do processo de gerenciamento de patches (consulte A06:2021-Componentes vulneráveis e desatualizados). Revise as permissões de armazenamento em nuvem (por exemplo, permissões de bucket do S3).
- Uma arquitetura de aplicativo segmentada fornece separação eficaz e segura entre componentes ou locatários, com segmentação, conteinerização ou grupos de segurança de nuvem (ACLs).
- Envio de diretivas de segurança para clientes, por exemplo, cabeçalhos de segurança.
- Um processo automatizado para verificar a eficácia das configurações e ajustes em todos os ambientes.

A06 Vulnerable and Outdated Components

Quando está vulnerável?

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

Como prevenir?

- Remova dependências não utilizadas, recursos, componentes, arquivos e documentação desnecessários.
- Faça um inventário contínuo das versões de componentes do lado do cliente e do lado do servidor (por exemplo, estruturas, bibliotecas) e suas dependências usando ferramentas como versões, verificação de dependência OWASP, retire.js etc. Monitore continuamente fontes como Vulnerabilidade e exposições comuns (CVE) e National Vulnerability Database (NVD) para vulnerabilidades nos componentes. Use ferramentas de análise de composição de software para automatizar o processo. Assine alertas por e-mail para vulnerabilidades de segurança relacionadas aos componentes que você usa.
- Obtenha apenas componentes de fontes oficiais em links seguros. Prefira pacotes assinados para reduzir a chance de incluir um componente mal-intencionado modificado (consulte A08:2021-Falhas de integridade de software e dados).
- Monitore bibliotecas e componentes que não são mantidos ou não criam patches de segurança para versões mais antigas. Se a aplicação de patches não for possível, considere a implantação de um patch virtual para monitorar, detectar ou proteger contra o problema descoberto.
- Toda organização deve garantir um plano contínuo de monitoramento, triagem e aplicação de atualizações ou alterações de configuração durante a vida útil do aplicativo ou portfólio.

A07 Identification and Authentication Failures

O que é?

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

Quando é vulnerável?

- Permite ataques automatizados, como preenchimento de credenciais, em que o invasor possui uma lista de nomes de usuários e senhas válidos.
- Permite força bruta ou outros ataques automatizados.
- Permite senhas padrão, fracas ou conhecidas, como "Password1" ou "admin/admin".
- Usa recuperação de credenciais fraca ou ineficaz e processos de esquecimento de senha, como "respostas baseadas em conhecimento", que não podem ser seguras.

- Usa armazenamentos de dados de senhas de texto simples, criptografados ou com hash fraco (consulte A02:2021-Falhas criptográficas).
- Tem autenticação multifator ausente ou ineficaz.
- Expõe o identificador de sessão na URL.
- Reutilize o identificador de sessão após o login bem-sucedido.
- Não invalida corretamente os IDs de sessão. Sessões de usuário ou tokens de autenticação (principalmente tokens de logon único (SSO)) não são invalidados corretamente durante o logout ou um período de inatividade.

Como prevenir?

- Sempre que possível, implemente a autenticação multifator para evitar ataques automatizados de preenchimento de credenciais, força bruta e reutilização de credenciais roubadas.
- Não envie ou implante com credenciais padrão, principalmente para usuários administradores.
- Implemente verificações de senhas fracas, como testar senhas novas ou alteradas na lista das 10.000 piores senhas.
- Alinhe as políticas de comprimento, complexidade e rotação de senha com as diretrizes do Instituto Nacional de Padrões e Tecnologia (NIST) 800-63b na seção 5.1.1 para Segredos Memorizados ou outras políticas de senha modernas baseadas em evidências.
- Garanta que os caminhos de registro, recuperação de credenciais e API sejam protegidos contra ataques de enumeração de conta usando as mesmas mensagens para todos os resultados.
- Limite ou retarde cada vez mais as tentativas de login com falha, mas tome cuidado para não criar um cenário de negação de serviço. Registre todas as falhas e alerte os administradores quando forem detectados preenchimento de credenciais, força bruta ou outros ataques.
- Use um gerenciador de sessão integrado, seguro e do lado do servidor que gera um novo ID de sessão aleatório com alta entropia após o login. O identificador de sessão não deve estar no URL, ser armazenado com segurança e invalidado após o logout, inatividade e tempos limites absolutos.

A08 Software and Data Integrity Failures

O que é?

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.

Como prevenir?

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

A09 Security Logging and Monitoring Failures

Quando se está vulnerável?

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

Como prevenir

- Deve haver um processo de gerenciamento de patches para: Remova dependências não utilizadas, recursos, componentes, arquivos e documentação desnecessários.
- Faça um inventário contínuo das versões de componentes do lado do cliente e do lado do servidor (por exemplo, estruturas, bibliotecas) e suas dependências usando ferramentas como versões, verificação de dependência OWASP, retire.js etc. Monitore continuamente fontes como Vulnerabilidade e exposições comuns (CVE) e National Vulnerability Database (NVD) para vulnerabilidades nos componentes. Use ferramentas de análise de composição de software para automatizar o processo. Assine alertas por e-mail para vulnerabilidades de segurança relacionadas aos componentes que você usa.
- Obtenha apenas componentes de fontes oficiais em links seguros. Prefira pacotes assinados para reduzir a chance de incluir um componente mal-intencionado modificado (consulte A08:2021-Falhas de integridade de software e dados).
- Monitore bibliotecas e componentes que não são mantidos ou não criam patches de segurança para versões mais antigas. Se a aplicação de patches não for possível, considere a implantação de um patch virtual para monitorar, detectar ou proteger contra o problema descoberto.
- Toda organização deve garantir um plano contínuo de monitoramento, triagem e aplicação de atualizações ou alterações de configuração durante a vida útil do aplicativo ou portfólio.

A10 Server Side Request Forgery (SSRF)

Quando está vulnerável?

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.

Como prevenir

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

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

Dicas:

Estabeleça uma propriedade e um ciclo de vida para regras de firewall baseadas em aplicativos.

Registre todos os fluxos de rede aceitos e bloqueados em firewalls (consulte A09:2021-Registro de segurança e falhas de monitoramento).

- Da camada de aplicação:
    - Higienize e valide todos os dados de entrada fornecidos pelo cliente.
    - Aplique o esquema de URL, a porta e o destino com uma lista de permissões positiva.
    - Não envie respostas brutas aos clientes
    - Desabilitar redirecionamentos HTTP
    - Esteja ciente da consistência do URL para evitar ataques como religação de DNS e condições de corrida "tempo de verificação, tempo de uso" (TOCTOU)
    - Não reduza o SSRF por meio do uso de uma lista de negação ou expressão regular. Os invasores têm listas de carga útil, ferramentas e habilidades para contornar as listas de negação.

- Medidas adicionais a considerar:

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