Melhores Práticas de Codificação Segura OWASP

Guia de Referência Rápida

Open Web Application Security Project (OWASP)

dezembro de 2022

Melhores Práticas de Codificação Segura OWASP

Guia de Referência Rápida

Copyright and License Copyright © 2010-2022 The OWASP Foundation.

O conteúdo deste documento é distribuído sob a licença Creative Commons Attribution ShareAlike 3.0. Para a utilização ou redistribuição, deve deixar claro os termos da licença deste trabalho.

http://creativecommons.org/licenses/by-sa/3.0/

Sumário

Notas da versão

Essa versão do OWASP *Secure Coding Practices* foi desenvolvida como parte integrante da atividade conjunta dos capítulos brasileiro e português da OWASP, em prol da comunidade de programadores e da segurança dos sistemas desenvolvidos nos países de língua portuguesa. Este documento é baseado na versão 2.0 de Novembro de 2010 e na revisão da primeira tradução, a versão 1.2, datada de Julho de 2011.

Líder do projeto de tradução:

- Tarcizio Vieira Neto – tarcizio.vieira@owasp.org – Brasil

Participaram desta tradução:

- Leandro Resende Gomes – leandrock@gmail.com – Brasil
- Sílvio Fernando Correia Vieira Filho – silviofilhosf@gmail.com – Brasil
- Paulo Alexandre Silva – me@pauloasilva.com – Portugal
- Alexandre Pupo – alexandrepupo@yahoo.com.br – Brasil
- Carlos Serrão – carlos.serrao@owasp.org – Portugal
- Rogério Vicente – rogeriopvl@gmail.com – Portugal
- Jorge Olimpia – jorgeolimpia@gmail.com – Brasil

A tradução deste documento pretende ser fiel ao texto original, tendo sido introduzidos alguns detalhes explicativos que estão dispersos no texto, em alguns casos, na forma de notas de rodapé contendo o prefixo "NT:" (nota de tradução).

Para saber mais sobre os eventos e atividades desenvolvidas pelo capítulo Brasil, acesse a página (http://www.owasp.org/index.php/Brazil) ou registre-se na lista de discussão OWASP-BR (http://lists.owasp.org/mailman/listinfo/owasp-brazilian).

Para saber mais sobre os eventos e atividades desenvolvidas pela delegação da OWASP em Portugal, acesse a página (http://www.owasp.org/index.php/Portuguese) ou registre-se na lista de discussão OWASP- PT (https://lists.owasp.org/mailman/listinfo/owasp-portuguese).

Introdução

Este documento não se baseia apenas em questões tecnológicas e tem o propósito de definir um conjunto de boas práticas de segurança no desenvolvimento de aplicações. As recomendações serão apresentadas no formato de lista de verificações, que podem ser integradas ao ciclo de desenvolvimento das aplicações. A adoção destas práticas provavelmente reduzirá as vulnerabilidades mais comuns em aplicações Web.

Geralmente, é mais barato construir software seguro do que corrigir problemas de segurança após a entrega do mesmo como um produto final ou pacote completo, sem falar nos custos que podem estar associados a uma falha de segurança.

Proteger os recursos críticos das aplicações tem se tornado cada vez mais importante, pois o foco dos atacantes mudou para a camada de aplicação. Um estudo da SANS em 2009[1] descobriu que os ataques contra aplicações Web constituem mais de 60% das tentativas de ataque observados na Internet.

Ao utilizar este guia é recomendável que equipes de desenvolvimento avaliem a maturidade do ciclo de vida de desenvolvimento de software e o nível de conhecimento da equipe. Como este guia não entra em detalhes de como implantar cada prática de programação, os programadores precisam possuir conhecimento prévio ou devem ter disponíveis os recursos que forneçam o conhecimento necessário. Este guia apresenta práticas que podem ser traduzidas em requisitos de desenvolvimento sem a necessidade do programador possuir uma compreensão aprofundada das vulnerabilidades e *exploits* de segurança. No entanto, outros membros da equipe de desenvolvimento devem possuir responsabilidades, competências adequadas, ferramentas e recursos para validar se o projeto e a implementação atendem os requisitos de segurança.

No apêndice B encontra-se um glossário dos termos importantes usados neste documento, incluindo o título das secções e palavras mostradas em *itálico*.

Não faz parte do escopo deste guia fornecer orientações para implantar um *framework* de desenvolvimento seguro de software, mas as seguintes práticas gerais e referências são recomendadas:

- Definir claramente os papéis e responsabilidades
- Fornecer às equipes de desenvolvimento pessoal com formação adequada em segurança no desenvolvimento de aplicações
- Implementar um ciclo de desenvolvimento de software seguro
- Estabelecer padrões de programação segura
  - OWASP Development Guide Project
- Construir uma biblioteca reutilizável ou fazer uso de uma biblioteca de segurança[2]
  - OWASP Enterprise Security API (ESAPI) Project
- Verificar a efetividade dos controles de segurança
  - OWASP Application Security Verification Standard (ASVS) Project
- Estabelecer práticas para garantir a segurança quando há terceirização no desenvolvimento, incluindo a definição dos requisitos de segurança e metodologias de verificação tanto para os requisitos das propostas, como para o contrato a ser firmado entre as partes

[1] NT: Complemento adicionado pelo tradutor
[2] NT: Complemento adicionado pelo tradutor

Princípios Gerais de Segurança em Aplicações e Riscos

Construir software seguro exige o conhecimento básico dos princípios de segurança. Uma revisão abrangente dos princípios de segurança está fora do escopo desse guia, porém os conceitos de segurança serão abordados de forma superficial, mas abrangente.

O objetivo da segurança em aplicações é manter a *confidencialidade*, *integridade* e *disponibilidade* dos recursos de informação a fim de permitir que as operações de negócios sejam bem sucedidas e esse objetivo é alcançado através da implementação de *controles de segurança*. Este guia concentra-se nos controles técnicos, específicos para *mitigar* as ocorrências das *vulnerabilidades* mais comuns no software e como o foco principal são as aplicações Web e a infraestrutura de apoio, boa parte desse documento pode ser usada para qualquer plataforma de desenvolvimento de software.

Para proteger o negócio contra os riscos inaceitáveis, isto é, relacionados com a dependência e com a confiança depositada na aplicação, este guia ajuda a compreender o que podemos entender por risco. Deste modo, risco é a combinação de fatores que ameaçam o sucesso do negócio. Isto pode ser descrito conceitualmente da seguinte forma: um *agente de ameaça* interage com um *sistema*, no qual pode haver uma *vulnerabilidade* latente que quando *explorada* causa um *impacto*. Como isto pode parecer um conceito abstrato, pense do seguinte modo: um ladrão de carros (agente de ameaça) passa por um estacionamento verificando nos carros presentes (o sistema) a existência de portas destrancadas (a vulnerabilidade) e quando a encontra, abre a porta (a exploração) e leva para si o que está dentro do carro (o impacto). Todos esses fatores desempenham um papel no desenvolvimento de software seguro.

Existe uma diferença fundamental entre a abordagem adotada por uma equipe de desenvolvimento e a abordagem que é adotada por alguém que está interessado em atacar uma aplicação. Uma equipe de desenvolvimento normalmente desenvolve uma aplicação com base naquilo que ela pretende fazer e isso significa criar uma aplicação para executar tarefas específicas, baseadas em requisitos funcionais e casos de uso documentados. Um atacante, por outro lado, está mais interessado no que a aplicação pode ser levada a fazer e parte do princípio que "qualquer ação não expressamente proibida, é permitida". Para resolver isso, alguns elementos adicionais precisam ser integrados nas fases iniciais do ciclo de vida do software e esses novos elementos são *requisitos de segurança* e *casos de abuso*. Este guia foi construído para ajudar a identificar os requisitos de segurança de alto nível e abordar vários cenários de ataques.

É importante que as equipes de desenvolvimento de aplicações Web entendam que os mecanismos de controle do lado cliente, como validação de dados de entrada no cliente, campos ocultos e controles de interface (combo box, radio buttons), fornecem pouco ou nenhum benefício de segurança. Nesse caso, um atacante pode usar ferramentas como *proxies* do lado cliente, como o OWASP WebScarab, Burp ou ferramentas de captura de pacotes de rede, como o Wireshark, para analisar o tráfego da aplicação e enviar requisições manipuladas, burlando todas as interfaces. Além disso, o Flash, os Applets Java e demais objetos que trabalham no lado cliente podem ser alvo de engenharia reversa e analisados em busca de falhas.

As falhas de segurança de software podem ser introduzidas em qualquer fase do ciclo de desenvolvimento, inclusive:

- No início, ao não identificar as necessidades de segurança
- Na criação de arquiteturas conceituais que possuam erros de lógica
- No uso de más práticas de programação que introduzam vulnerabilidades técnicas
- Na implementação do software de modo inapropriado
- Na inserção de falhas durante a manutenção ou a atualização

Além disso, é importante entender que as vulnerabilidades de software podem ter um escopo muito maior do que o do próprio software. Dependendo da natureza do software, da vulnerabilidade e da infraestrutura de apoio, o impacto de uma exploração bem sucedida pode comprometer qualquer um, ou mesmo todos os seguintes aspectos:

- O software e sua informação associada

- O sistema operacional dos servidores associados
- A base de dados do *backend*
- Outras aplicações em um ambiente compartilhado
- O sistema do usuário
- Outros softwares com os quais o usuário interage

Lista de Verificação de Práticas de Programação Segura

Validação dos Dados de Entrada

- ☐ Efetuar toda a validação dos dados em um sistema confiável. Por exemplo, centralizar todo o processo no servidor
- ☐ Identificar todas as fontes de dados e classificá-las como sendo confiáveis ou não. Em seguida, validar os dados provenientes de fontes nas quais não se possa confiar (ex: base de dados, stream de arquivos etc.)
- ☐ A rotina de validação de dados de entrada deve ser centralizada na aplicação
- ☐ Especificar o conjunto de caracteres apropriado, como UTF-8, para todas as fontes de entrada de dados
- ☐ Codificar os dados para um conjunto de caracteres comuns antes da validação (*Canonicalize*)
- ☐ Quando há falha de validação, a aplicação deve rejeitar os dados fornecidos
- ☐ Determinar se o sistema suporta conjuntos de caracteres estendidos UTF-8 e, em caso afirmativo, validar após efetuar a descodificação UTF-8
- ☐ Validar todos os dados provenientes dos clientes antes do processamento, incluindo todos os parâmetros, campos de formulário, conteúdos das URLs e cabeçalhos HTTP, como, por exemplo, os nomes e os valores dos Cookies. Certificar-se, também, de incluir mecanismos automáticos de *postback*[3] nos blocos de código JavaScript, Flash ou qualquer outro código embutido
- ☐ Verificar se os valores de cabeçalho, tanto das requisições, como das respostas, contêm apenas caracteres ASCII
- ☐ Validar dados provenientes de redirecionamentos. Os atacantes podem incluir conteúdo malicioso diretamente para o alvo do mecanismo de redirecionamento, podendo assim contornar a lógica da aplicação e qualquer validação executada antes do redirecionamento
- ☐ Validar tipos de dados esperados
- ☐ Validar intervalo de dados
- ☐ Validar o tamanho dos dados
- ☐ Validar, sempre que possível, todos os dados de entrada através de um método baseado em “listas brancas" que utilizem uma lista de caracteres ou expressões regulares para definirem os caracteres permitidos
- ☐ Se qualquer caractere potencialmente perigoso precisa ser permitido na entrada de dados da aplicação, certificar-se de que foram implementados controles adicionais como codificação dos dados de saída, APIs especificas que fornecem tarefas seguras e trilhas de auditoria no uso dos dados pela aplicação. A seguir, como exemplo de caracteres “potencialmente perigosos”, temos: <, >, ", ', %, (,), &, +, \, \', \"
- ☐ Se a rotina de validação padrão não abordar as seguintes entradas, então elas devem ser verificadas:
  a. Verificar bytes nulos (%00)
  b. Verificar se há caracteres de nova linha (%0d, %0a, \r, \n)
  c. Verificar se há caracteres “ponto-ponto barra” (../ ou ..\) que alteram caminhos. Nos casos de conjunto de caracteres que usem a extensão UTF-8, o sistema deve utilizar representações alternativas como: %c0%ae%c0%ae/. A *canonicalização* deve ser utilizada para resolver problemas de codificação dupla (double encoding[4] ou outras formas de ataques por ofuscação

[3]NT: [http://pt.wikipedia.org/wiki/Postback]
[4]NT: [http://www.owasp.org/index.php/Double_Encoding]

Codificação de Dados de Saída

- ☐ Efetuar toda a codificação dos dados em um sistema confiável, por exemplo, centralizar todo o processo no servidor
- ☐ Utilizar uma rotina padrão, testada, para cada tipo de codificação de saída
- ☐ Realizar a codificação, baseada em contexto, de todos os dados enviados para o cliente que têm origem em um ambiente fora dos limites de confiança da aplicação. A codificação das entidades HTML é um exemplo, mas nem sempre funciona para todos os casos
- ☐ Codificar todos os caracteres, a menos que sejam conhecidos por serem seguros para o interpretador de destino
- ☐ Realizar o tratamento (*sanitização*), baseado em contexto, de todos os dados provenientes de fontes não confiáveis usados para construir consultas SQL, XML, e LDAP
- ☐ Tratar todos os dados provenientes de fontes que não sejam confiáveis que gerem comandos para o sistema operacional

Autenticação e Gerenciamento de Credenciais

- ☐ Requerer autenticação para todas as páginas e recursos, exceto para aqueles que são intencionalmente públicos
- ☐ Os controles de autenticação devem ser executados em um sistema confiável, por exemplo, centralizar todo o processo no servidor
- ☐ Sempre que possível, estabelecer e utilizar serviços de autenticação padronizados e testados
- ☐ Utilizar uma implementação centralizada para realizar os procedimentos de autenticação, disponibilizando bibliotecas que invoquem os serviços externos de autenticação
- ☐ Separar a lógica de autenticação do recurso que está a ser requisitado e usar redirecionadores dos controladores de autenticação centralizados
- ☐ Quando ocorrerem situações excepcionais nos controles de autenticação, executar procedimentos em caso de falha com o propósito de manter o sistema seguro
- ☐ Todas as funções administrativas e de gerenciamento de contas devem ser tão seguras quanto o mecanismo de autenticação principal
- ☐ Se a aplicação gerenciar um repositório de credenciais, esta deverá garantir que as senhas são armazenadas na base de dados somente sob a forma de resumo/*hash* da senha na forma de *one-way salted hashes*[5] e que a tabela/arquivo que armazena as senhas e as próprias chaves são manipuladas apenas pela aplicação. Não utilizar o algoritmo de hash MD5, sempre que esse puder ser evitado
- ☐ A geração dos resumos (*hash*) das senhas deve ser executada em um sistema confiável, por exemplo, centralizar o controle no servidor
- ☐ Validar os dados de autenticação somente no final de todas as entradas de dados, especialmente para as implementações de *autenticação sequencial*
- ☐ As mensagens de falha na autenticação não devem indicar qual parte dos dados de autenticação está incorreta. Por exemplo, em vez de exibir mensagens como "Nome de usuário incorreto" ou "Senha incorreta", utilize apenas mensagens como: "Usuário e/ou senha inválidos", para ambos os casos de erro. As respostas de erro devem ser idênticas nos dois casos
- ☐ Utilizar autenticação para conexão a sistemas externos que envolvam tráfego de informação sensível ou acesso a funções

[5] NT: one-way salted hash é um algoritmo de hash gerado com o auxílio de valores aleatórios ou pré-definidos que compõem o parâmetro da função de geração da hash e dificulta o processo de quebra da hash através de ataques de dicionário. Mais informações sobre o assunto em: [http://en.wikipedia.org/wiki/Salt_(cryptography)]

- ☐ As credenciais de autenticação para acessar serviços externos à aplicação devem ser cifradas e armazenadas em um local protegido de um sistema confiável, por exemplo, no servidor da aplicação. Obs.: o código-fonte não é considerado um local seguro
- ☐ Utilizar apenas requisições POST para transmitir credenciais de autenticação
- ☐ Somente trafegar senhas (não temporárias) através de uma conexão protegida (SSL/TLS) ou no formato de dado cifrado, como no caso de envio de e-mail cifrado. Senhas temporárias enviadas por e-mail podem ser um caso de exceção aceitável
- ☐ Exigir que os requisitos de complexidade de senha estabelecidos pela política ou regulamento sejam cumpridos. As credenciais de autenticação devem resistir a ataques que, tipicamente, ameaçam o ambiente de produção. Um exemplo pode ser a exigência do uso simultâneo de caracteres alfabéticos, numérico e/ou caracteres especiais
- ☐ Exigir que os requisitos de comprimento de senha estabelecidos pela política ou regulamento sejam cumpridos. O uso de oito caracteres é o mais comum, porém usar 16 é mais recomendado. Considere, ainda, o uso de senhas que contenham várias palavras (uma frase)
- ☐ A entrada da senha deve ser ocultada na tela do usuário. Em HTML, utilizar o campo do tipo "password"
- ☐ Desativar a conta após um número pré-definido de tentativas inválidas de autenticação (ex: cinco tentativas é o mais comum). A conta deve ser desativada por um período de tempo suficientemente longo para desencorajar a dedução das credenciais pelo método de força bruta, mas não tão longo ao ponto de permitir um ataque de negação de serviço
- ☐ Os processos de redefinição de senhas e operações de mudanças devem exigir os mesmos níveis de controle previstos para a criação de contas e autenticação
- ☐ Esquemas de pergunta/resposta (pré-definidas) usadas para a redefinição da senha devem evitar ataques que lancem respostas aleatórias. Por exemplo, “livro favorito” é uma questão fraca, pois “A Bíblia” é uma resposta muito comum
- ☐ Se optar por usar redefinição de senha baseada em e-mail, envie um e-mail somente para o endereço pré-definido contendo um *link* ou senha de acesso temporário que permitam ao usuário redefinir a senha
- ☐ O tempo de validade das senhas e dos *links* temporários deve ser curto
- ☐ Exigir a mudança de senhas temporárias na próxima vez que o usuário realizar a autenticação no sistema
- ☐ Notificar o usuário quando a senha for reiniciada (*reset*)
- ☐ Prevenir a reutilização de senhas
- ☐ As senhas devem ter, pelo menos, um dia de duração antes de poderem ser alteradas, a fim de evitar ataques de reutilização de senhas
- ☐ Garantir que a troca de senhas está em conformidade com os requisitos estabelecidos na política ou regulamento. Sistemas críticos podem exigir alterações mais frequentes nas credenciais de segurança. O tempo entre as trocas de senhas deve ser controlado administrativamente
- ☐ Desativar a funcionalidade de lembrar a senha nos campos de senha do navegador
- ☐ A data/hora da última utilização (bem ou mal sucedida) de uma conta de usuário deve ser comunicada no próximo acesso ao sistema
- ☐ Realizar monitoramento para identificar ataques contra várias contas de usuários, utilizando a mesma senha. Esse padrão de ataque é utilizado para explorar o uso de senhas padrão
- ☐ Modificar todas as senhas que, por padrão, são definidas pelos fornecedores, bem como os identificadores de usuários (IDs) ou desativar as contas associadas

- ☐ Exigir nova autenticação dos usuários antes da realização de operações críticas
- ☐ Utilizar *autenticação de múltiplos fatores* (utilizando simultaneamente token, senha, biometria etc.[6]) para contas altamente sensíveis ou de alto valor transacional
- ☐ Caso utilize código de terceiros para realizar a autenticação, inspecione-o cuidadosamente para garantir que o mesmo não é afetado por qualquer código malicioso

Gerenciamento de Sessões

- ☐ Utilizar controles de gerenciamento de sessão baseados no servidor ou em framework. A aplicação deve reconhecer apenas esses identificadores de sessão como válidos
- ☐ A criação dos identificadores de sessão deve ser sempre realizada em um sistema confiável, por exemplo, centralizado no servidor
- ☐ O controle de gestão de sessão deve usar algoritmos conhecidos, padronizados e bem testados que garantam a aleatoriedade dos identificadores de sessão
- ☐ Definir o domínio e o caminho para os cookies que contenham identificadores de sessão autenticados, para um valor devidamente restrito ao site
- ☐ A funcionalidade de saída (*logout*) deve encerrar completamente a sessão ou conexão associada
- ☐ A funcionalidade de saída (*logout*) deve estar disponível em todas as páginas que requerem autenticação
- ☐ Estabelecer um tempo de expiração da sessão que seja o mais curto possível, baseado no balanceamento dos riscos e requisitos funcionais do negócio. Na maioria dos casos não deve ser maior do que algumas horas
- ☐ Não permitir logins persistentes (sem prazo de expiração) e realizar o encerramento da sessão periodicamente, mesmo quando ela estiver ativa. Isso deve ser feito, especialmente, em aplicações que suportam várias conexões de rede ou que se conectam a sistemas críticos. O tempo de encerramento deve estar de acordo com os requisitos do negócio e o usuário deve receber notificações suficientes para atenuar os impactos negativos dessa medida
- ☐ Se uma sessão estava estabelecida antes do login ela deve ser encerrada para que uma nova seja estabelecida após o login
- ☐ Gerar um novo identificador de sessão quando houver uma nova autenticação
- ☐ Não permitir conexões simultâneas com o mesmo identificador de usuário
- ☐ Não expor os identificadores de sessão em URLs, mensagens de erro ou logs. Os identificadores de sessão devem apenas ser encontrados no cabeçalho do cookie HTTP. Por exemplo, não trafegar os identificadores de sessão sob a forma de parâmetros GET
- ☐ Proteger os dados de sessão do lado servidor contra acessos não autorizados por outros usuários do servidor, através da implementação de controles de acesso apropriados no servidor
- ☐ Gerar um novo identificador de sessão e desativar o antigo periodicamente. Isso pode mitigar certos cenários de ataques de sequestro de sessão (*session hijacking*), quando o identificador de sessão original for comprometido
- ☐ Gerar um novo identificador de sessão caso a segurança da conexão mude de HTTP para HTTPS, como pode ocorrer durante a autenticação. Internamente à aplicação, é recomendável utilizar HTTPS de forma constante em vez de alternar entre HTTP e HTTPS
- ☐ Utilizar mecanismos complementares ao mecanismo padrão de gerenciamento de sessões para operações sensíveis do lado servidor – como no caso de operações de gerenciamento de contas – através da utilização de tokens aleatórios ou parâmetros associados à sessão. Esse método pode ser usado para prevenir ataques do tipo Cross Site Request Forgery (CSRF)

[6] NT: Complementação explicativa fornecida pelo tradutor.

- ☐ Utilizar mecanismos complementares ao gerenciamento de sessões para operações altamente sensíveis ou críticas, utilizando tokens aleatórios ou parâmetros em cada requisição em vez de basear-se apenas na sessão
- ☐ Configurar o atributo "secure" para cookies transmitidos através de uma conexão TLS
- ☐ Configurar os cookies com o atributo “HttpOnly”, a menos que seja explicitamente necessário ler ou definir os valores dos mesmos através de scripts do lado cliente da aplicação

Controle de Acessos

- ☐ Utilizar apenas objetos do sistema que sejam confiáveis, como ocorre com os objetos de sessão do servidor, para realizar a tomada de decisão sobre a autorização de acesso
- ☐ Utilizar um único componente em toda a aplicação Web para realizar o processo de verificação de autorização de acesso. Isto inclui bibliotecas que invocam os serviços externos de autorização
- ☐ Quando ocorrer alguma falha no controle de acesso, ela deve ocorrer de modo seguro
- ☐ Negar todos os acessos, caso a aplicação não consiga ter acesso às informações contidas na configuração de segurança
- ☐ Garantir o controle de autorização em todas as requisições, inclusive em scripts do lado servidor, "includes" e requisições provenientes de tecnologias do lado cliente, como AJAX e Flash
- ☐ Isolar do código da aplicação os trechos de código que contêm lógica priviligiada
- ☐ Restringir o acesso aos arquivos e outros recursos, incluindo aqueles que estão fora do controle direto da aplicação, somente aos usuários autorizados
- ☐ Restringir o acesso às URLs protegidas somente aos usuários autorizados
- ☐ Restringir o acesso às funções protegidas somente aos usuários autorizados
- ☐ Restringir o acesso às referências diretas aos objetos somente aos usuários autorizados
- ☐ Restringir o acesso aos serviços somente aos usuários autorizados
- ☐ Restringir o acesso aos dados da aplicação somente aos usuários autorizados
- ☐ Restringir o acesso aos atributos e dados dos usuários, bem como informações das políticas usadas pelos mecanismos de controle de acesso
- ☐ Restringir o acesso às configurações de segurança relevantes apenas aos usuários autorizados
- ☐ As regras de controle de acesso representadas pela camada de apresentação devem coincidir com as regras presentes no lado servidor
- ☐ Se o *estado dos dados* deve ser armazenado no lado cliente, utilizar mecanismos de criptografia e verificação de integridade no lado servidor para detectar possíveis adulterações
- ☐ Garantir que os fluxos lógicos da aplicação obedecem as regras de negócio
- ☐ Limitar o número de transações que um único usuário ou dispositivo pode executar em determinado período de tempo. As transações por período de tempo devem estar acima das necessidades reais do negócio, mas abaixo o suficiente para impedirem ataques automatizados
- ☐ Utilizar o campo “referer” do cabeçalho somente como forma de verificação suplementar. O mesmo não deve ser usado sozinho como forma de validação de autorização porque ele pode ter o valor adulterado
- ☐ Se for permitida a existência de sessões autenticadas por longos períodos de tempo, fazer a revalidação periódica da autorização do usuário para garantir que os privilégios não foram modificados e, caso tenham sido, realizar o registro em log do usuário e exigir nova autenticação

- ☐ Implementar a auditoria das contas de usuário e assegurar a desativação de contas não utilizadas. Por exemplo, a conta deve ser desativada não mais do que 30 dias após a expiração da senha
- ☐ A aplicação deve dar suporte à desativação de contas e ao encerramento das sessões quando terminar a autorização do usuário. Por exemplo, quando ocorrer alguma alteração dos dados do usuário, situação profissional, processos de negócio etc.
- ☐ As contas de serviço ou contas de suporte a conexões provenientes ou destinadas a serviços externos devem possuir o menor privilégio possível
- ☐ Criar uma Política de Controle de Acesso para documentar as regras de negócio da aplicação, tipos de dados e critérios ou processos de autorização para que os acessos possam ser devidamente concedidos e controlados. Isto inclui identificar requisitos de acessos, tanto para os dados, como para os recursos do sistema

Práticas de Criptografia

- ☐ Todas as funções de criptografia utilizadas para proteger dados sensíveis dos usuários da aplicação, devem ser implantadas em um sistema confiável (neste caso o servidor)
- ☐ A senha mestre deve ser protegida contra acessos não autorizados
- ☐ Quando ocorrer alguma falha nos módulos de criptografia, permitir que as mesmas ocorram de modo seguro
- ☐ Todos os números, nomes de arquivos, GUIDs e strings aleatórias devem ser gerados usando um módulo criptográfico com gerador de números aleatórios, somente se os valores aleatórios gerados forem impossíveis de serem deduzidos
- ☐ Os módulos de criptografia usados pela aplicação devem ser compatíveis com a FIPS 140-2 ou com um padrão equivalente (NIST STM group)
- ☐ Estabelecer e utilizar uma política e processo que defina como é realizado o gerenciamento das chaves criptográficas

Tratamento de Erros e Log

- ☐ Não expor informações sensíveis nas repostas de erros, inclusive detalhes de sistema, identificadores de sessão ou informação da conta do usuário
- ☐ Usar mecanismos de tratamento de erros que não mostrem informações de depuração (*debug*) ou informações da pilha de exceção
- ☐ Usar mensagens de erro genéricas e páginas de erro personalizadas
- ☐ A aplicação deve tratar os erros sem se basear nas configurações do servidor
- ☐ A memória alocada deve ser liberada de modo apropriado quando ocorrerem condições de erro
- ☐ O tratamento de erros lógicos associados com os controles de segurança devem, por padrão, negar o acesso
- ☐ Todos os controles de log devem ser implementados em um sistema confiável, por exemplo, centralizar todo o processo no servidor
- ☐ Os controles de log devem dar suporte tanto para os casos de sucesso como os de falha relacionados com os eventos de segurança
- ☐ Garantir que os logs armazenam eventos importantes
- ☐ Garantir que as entradas de log que incluam dados nos quais não se confia não sejam executadas como código-fonte na interface de visualização de logs
- ☐ Restringir o acesso aos logs apenas para pessoal autorizado

- ☐ Utilizar uma rotina centralizada para realizar todas as operações de log
- ☐ Não armazenar informações sensíveis nos registros de logs, como detalhes desnecessários do sistema, identificadores de sessão e senhas
- ☐ Garantir o uso de algum mecanismo que conduza (ou facilite) o processo de análise de logs
- ☐ Registrar em log todas as falhas de validação de entrada de dados
- ☐ Registrar em log todas as tentativas de autenticação, especialmente as que falharam por algum motivo
- ☐ Registrar em log todas as falhas de controle de acesso
- ☐ Registrar em log todos os eventos suspeitos de adulteração, inclusive alterações inesperadas no estado dos dados
- ☐ Registrar em log as tentativas de conexão com tokens de sessão inválidos ou expirados
- ☐ Registrar em log todas as exceções lançadas pelo sistema
- ☐ Registrar em log todas as funções administrativas, inclusive as mudanças realizadas nas configurações de segurança
- ☐ Registrar em log todas as falhas de conexão TLS com o backend
- ☐ Registrar em log todas as falhas que ocorreram nos módulos de criptografia
- ☐ Utilizar uma função de hash criptográfica para validar a integridade dos registros de log

Proteção de Dados

- ☐ Implementar uma política de privilégio mínimo, restringindo os usuários apenas às funcionalidades, dados e informações do sistema que são necessárias para executarem suas tarefas
- ☐ Proteger contra acesso não autorizado todas as cópias temporárias ou registradas em cache que contenham dados sensíveis e estejam armazenadas no servidor e excluir esses arquivos logo que não forem mais necessários
- ☐ Criptografar informações altamente sensíveis quando armazenadas – como dados de verificação de autenticação – mesmo que estejam no lado servidor, usando sempre algoritmos conhecidos, padronizados e bem testados. Consulte a seção que trata sobre "Práticas de Criptografia" para orientações adicionais
- ☐ Proteger o código-fonte presente no servidor para que não sejam acessados por algum usuário
- ☐ Não armazenar senhas, strings de conexão ou outras informações confidenciais em texto claro/legível ou em qualquer forma criptograficamente insegura no lado cliente. Isso é válido também quando há utilização de formatos inseguros como: MS viewstate, Adobe Flash ou código compilado que é executado no lado cliente
- ☐ Remover comentários do código de produção que podem ser acessados pelos usuários e podem revelar detalhes internos do sistema ou outras informações sensíveis
- ☐ Remover aplicações desnecessárias e documentação do sistema que possam revelar informações importantes para os atacantes
- ☐ Não incluir informações sensíveis nos parâmetros de requisição HTTP GET
- ☐ Desativar a funcionalidade de auto completar nos formulários que contenham informações sensíveis, inclusive no formulário de autenticação
- ☐ Desativar a cache realizada no lado cliente das páginas que contenham informações sensíveis. O parâmetro **Cache-Control: no-store**, pode ser usado em conjunto com o controle definido no cabeçalho HTTP **"Pragma: no-cache"**, que é menos efetivo, porém compatível com HTTP/1.0

- ☐ A aplicação deve dar suporte à remoção de dados sensíveis quando estes não forem mais necessários. Por exemplo, informação pessoal ou dados financeiros
- ☐ Implementar mecanismos de controle de acesso apropriados para dados sensíveis armazenados no servidor. Isto inclui dados em cache, arquivos temporários e dados que devem ser acessíveis somente por usuários específicos do sistema

Segurança nas comunicações

- ☐ Utilizar criptografia na transmissão de todas as informações sensíveis. Isto deve incluir TLS para proteger a conexão e deve ser complementado com criptografia de arquivos que contém dados sensíveis ou conexões que não usam o protocolo HTTP
- ☐ Os certificados TLS devem ser válidos, possuírem o nome de domínio correto, não estarem expirados e serem instalados com certificados intermediários, quando necessário
- ☐ Quando ocorrer alguma falha nas conexões TLS, o sistema não deve fornecer uma conexão insegura
- ☐ Utilizar conexões TLS para todo o conteúdo que requerer acesso autenticado ou que contenha informação sensível
- ☐ Utilizar TLS para conexões com sistemas externos que envolvam funções ou informações sensíveis
- ☐ Utilizar um padrão único de implementação TLS configurado de modo apropriado
- ☐ Especificar a codificação dos caracteres para todas as conexões
- ☐ Filtrar os parâmetros que contenham informações sensíveis, provenientes do "HTTP referer", nos links para sites externos

Configuração do Sistema

- ☐ Garantir que os servidores, frameworks e componentes do sistema estão executando a última versão aprovada
- ☐ Garantir que os servidores, frameworks e componentes do sistema possuem as atualizações mais recentes aplicadas para a versão em uso
- ☐ Desativar a listagem de diretórios
- ☐ Restringir, para o mínimo possível, os privilégios do servidor Web, dos processos e das contas de serviços
- ☐ Quando ocorrerem exceções no sistema, garantir que as falhas ocorram de modo seguro
- ☐ Remover todas as funcionalidades e arquivos desnecessários
- ☐ Remover código de teste ou qualquer funcionalidade desnecessária para o ambiente de produção, antes de instalar o sistema no servidor de produção
- ☐ Prevenir a divulgação da estrutura de diretórios impedindo que robôs[7] de busca façam indexação de arquivos sensíveis, através da configuração do arquivo "robots.txt"[8]. Os diretórios que não devem ser acessados por estes indexadores devem ser colocados em um diretório isolado. Assim, apenas é necessário negar o acesso ao diretório pai definido no arquivo "robots.txt", evitando ter que negar o acesso a cada diretório individualmente
- ☐ Definir quais métodos HTTP, GET ou POST, a aplicação irá suportar e se serão tratados de modo diferenciado nas diversas páginas da aplicação

[7]NT: Robôs são programas de computador que percorrem automaticamente as páginas da Internet em busca de documentos, com o propósito de indexá-los, validá-los ou monitorar alterações de conteúdo. [http://pt.wikipedia.org/wiki/Robots.txt]

[8]NT: O link abaixo mostra exemplos de como configurar o arquivo robots.txt. [http://www.mundoseo.com.br/seo-tecnico/robotstxt-configuracao-seu-site/]

- ☐ Desativar as extensões HTTP desnecessárias como, por exemplo, o WebDAV. Caso seja necessário o uso de alguma extensão HTTP com o propósito de suportar a manipulação de arquivos, utilize um mecanismo de autenticação conhecido, padronizado e bem testado
- ☐ Se o servidor processa tanto requisições HTTP 1.0 como HTTP 1.1, certificar-se de que ambos são configurados de modo semelhante ou assegure que qualquer diferença existente seja compreendida, como, por exemplo, o manuseio de métodos HTTP estendidos
- ☐ Remover informações desnecessárias presentes nos cabeçalhos de resposta HTTP e que podem estar relacionadas com o sistema operacional, versão do servidor web e frameworks de aplicação
- ☐ O armazenamento da configuração de segurança para a aplicação deve ser capaz de ser produzida de forma legível para dar suporte à auditoria
- ☐ Implementar um sistema de gestão de ativos para manter o registro dos componentes e programas
- ☐ Isolar o ambiente de desenvolvimento da rede de produção e conceder acesso somente para grupos de desenvolvimento e testes. É comum os ambientes de desenvolvimento serem configurados de modo menos seguro do que os ambientes de produção. Deste modo, os atacantes podem usar essa diferença para descobrir vulnerabilidades comuns ou encontrar formas de exploração das mesmas
- ☐ Implementar um sistema de controle de mudanças para gerenciar e registrar as alterações no código, tanto de desenvolvimento, como dos sistemas em produção

Segurança em Banco de Dados

- ☐ Usar *consultas parametrizadas* fortemente tipadas
- ☐ Utilizar validação de entrada e codificação de saída e assegurar a abordagem de meta caracteres. Se houver falha, o comando não deverá ser executado no banco de dados
- ☐ Certificar-se de que as variáveis são fortemente tipadas
- ☐ Realizar a codificação (escaping) de meta caracteres em instruções SQL[9]
- ☐ A aplicação deve usar o menor nível possível de privilégios ao acessar o banco de dados
- ☐ Usar credenciais seguras para acessar o banco de dados
- ☐ Não incluir strings de conexão na aplicação. As strings de conexão devem estar em um arquivo de configuração separado, armazenado em um sistema confiável e as informações devem ser criptografadas
- ☐ Usar procedimentos armazenados (*stored procedures*) para abstrair o acesso aos dados e permitir a remoção de permissões das tabelas no banco de dados
- ☐ Encerrar a conexão assim que possível
- ☐ Remover ou modificar senhas padrão de contas administrativas. Utilizar senhas robustas (pouco comuns ou difíceis de deduzir) ou implementar autenticação de múltiplos fatores. Desativar qualquer funcionalidade desnecessária no banco de dados, como "stored procedures" ou serviços não utilizados. Instalar o conjunto mínimo de componentes ou de opções necessárias (redução da superfície de ataque)
- ☐ Eliminar o conteúdo desnecessário incluído por padrão pelo fornecedor como esquemas e bancos de dados de exemplo
- ☐ Desativar todas as contas criadas por padrão e que não sejam necessárias para suportar os requisitos de negócio
- ☐ A aplicação deve conectar-se ao banco de dados com diferentes credenciais de segurança para cada tipo de necessidade como, por exemplo, usuário, somente leitura, convidado ou administrador

[9] NT: SQL_Injection_Prevention_Cheat_Sheet

Gerenciamento de Arquivos

- ☐ Não repassar dados fornecidos pelos usuários diretamente a uma função de inclusão dinâmica
- ☐ Solicitar autenticação antes de permitir que seja feito o carregamento de arquivos
- ☐ Limitar os tipos de arquivos que podem ser enviados para aceitar somente os necessários ao propósito do negócio
- ☐ Validar se os arquivos enviados são do tipo esperado, através da validação dos cabeçalhos, pois, realizar a verificação apenas pela extensão não é suficiente
- ☐ Não salvar arquivos no mesmo diretório de contexto da aplicação Web. Os arquivos devem ser armazenados no servidor de conteúdos ou no banco de dados
- ☐ Prevenir ou restringir o carregamento de qualquer arquivo que possa ser interpretado ou executado pelo servidor Web
- ☐ Desativar privilégios de execução nos diretórios de armazenamento de arquivos
- ☐ Implantar o carregamento seguro nos ambientes UNIX por meio da montagem do diretório de destino como uma unidade lógica, usando o caminho associado ou o ambiente de "chroot"
- ☐ Ao referenciar arquivos, usar uma lista branca (*whitelist*) de nomes e de tipos de arquivos permitidos. Realizar a validação do valor do parâmetro passado e caso ele não corresponda ao que é esperado, rejeitar a entrada ou utilizar um valor padrão
- ☐ Não transmitir, sem nenhum tipo de tratamento, os dados informados pelo usuário a redirecionamentos dinâmicos. Se isso for necessário, o redirecionamento deverá aceitar apenas URLs relativas e validadas
- ☐ Não passar caminhos de diretórios ou de arquivos em requisições. Usar algum mecanismo de mapeamento desses recursos para índices definidos em uma lista pré-definida de caminhos
- ☐ Nunca enviar o caminho absoluto do arquivo para o lado cliente de uma aplicação ou para o usuário
- ☐ Certificar-se de que os arquivos da aplicação e os recursos estão definidos somente com o atributo de leitura
- ☐ Verificar os arquivos que os usuários submeterem através do mecanismo de carregamento em busca de vírus e malwares

Gerenciamento de Memória

- ☐ Utilizar controle de entrada/saída para os dados que não sejam confiáveis
- ☐ Verificar se o buffer é tão grande quanto o especificado
- ☐ Ao usar funções que aceitem determinado número de bytes para realizar cópias, como strncpy(), esteja ciente de que se o tamanho do buffer de destino for igual ao tamanho do buffer de origem, ele não pode encerrar a sequência de caracteres com valor nulo (null)
- ☐ Verificar os limites do buffer caso as chamadas à função sejam realizadas em ciclos e verificar se não há nenhum risco de ocorrer gravação de dados além do espaço reservado
- ☐ Truncar todas as strings de entrada para um tamanho razoável antes de passá-las para as funções de cópia e concatenação
- ☐ Na liberação de recursos alocados para objetos de conexão, identificadores de arquivo etc., não contar com o "*garbage collector*" e realizar a tarefa explicitamente
- ☐ Usar pilhas não executáveis, quando disponíveis
- ☐ Evitar o uso de funções reconhecidamente vulneráveis como printf(), strcat(), strcpy() etc.
- ☐ Liberar a memória alocada de modo apropriado após concluir a sub-rotina (função/método) e em todos pontos de saída

Práticas Gerais de Codificação

- ☐ Para tarefas comuns, utilizar sempre código testado, gerenciado e aprovado ao invés de criar código novo e não gerenciado
- ☐ Utilizar APIs que executem tarefas específicas para realizar operações do sistema operacional. Não permitir que a aplicação execute comandos diretamente no sistema operacional, especialmente através da utilização de "shells" de comando iniciadas pela aplicação
- ☐ Utilizar mecanismos de verificação de integridade por "checksum" ou "hash" para verificar a integridade do código interpretado, bibliotecas, arquivos executáveis e arquivos de configuração
- ☐ Utilizar mecanismos de bloqueio para evitar requisições simultâneas para a aplicação ou utilizar um mecanismo de sincronização para evitar condições de concorrência (*race conditions*)
- ☐ Proteger as variáveis compartilhadas e os recursos contra acessos concorrentes inapropriados
- ☐ Instanciar explicitamente todas as variáveis e dados persistentes durante a declaração, ou antes da primeira utilização
- ☐ Quando a aplicação tiver que ser executada com privilégios elevados, aumentar os privilégios o mais tarde possível e revogá-los logo que seja possível
- ☐ Evitar erros de cálculo decorrentes da falta de entendimento da representação interna da linguagem de programação usada e de como é realizada a interação com os aspectos de cálculo numérico. Prestar bastante atenção nas discrepâncias de tamanho de byte, precisão, distinções de sinal (signed/unsigned), truncamento, conversão e "casting" entre os tipos, cálculos que devolvam erros do tipo "not-a-number" e, também, como a linguagem de programação trata a representação interna de números muito grandes ou muito pequenos
- ☐ Não transferir, diretamente, dados fornecidos pelo usuário para qualquer função de execução dinâmica sem realizar o *tratamento dos dados* de modo adequado
- ☐ Restringir a geração e a alteração de código por parte dos usuários
- ☐ Revisar todas as aplicações secundárias, códigos e bibliotecas de terceiros para determinar a necessidade do negócio e validar as funcionalidades de segurança, uma vez que estas podem introduzir novas vulnerabilidades
- ☐ Implementar atualizações de modo seguro. Se a aplicação precisar realizar atualizações automáticas, utilizar mecanismos de assinatura digital para garantir a integridade do código e garantir que os clientes façam a verificação da assinatura após descarregarem as atualizações. Usar canais criptografados para transferir o código a partir do host do servidor

Apêndice A Referências Externas

Referência citada

- SANS CIS Controls version 8

  https://www.sans.org/blog/cis-controls-v8/

- Web Application Security Consortium

  http://www.webappsec.org/

- Common Weakness Enumeration (CWE)

  https://cwe.mitre.org/

- Department of Homeland Security: Build Security In Portal

  https://www.cisa.gov/uscert/bsi

- CERT Secure Coding

  http://www.cert.org/secure-coding/

- MSDN Security Developer Center

  http://msdn.microsoft.com/en-us/security/default.aspx

Sites de Avisos de Segurança

Estes links podem ser úteis para verificar e garantir que a infraestrutura de apoio e os frameworks não possuem vulnerabilidades conhecidas:

Secunia Citrix Vulnerability List:

- https://www.flexera.com/products/software-vulnerability-research/secunia-research

Security Focus Vulnerability Search (archived):

- https://bugtraq.securityfocus.com/archive

Common Vulnerability Enumeration:

- https://www.cve.org/

Apêndice B (Glossário)

Agente de Ameaça: Qualquer entidade que pode causar um impacto negativo em um sistema. Pode ser tanto um usuário mal-intencionado querendo comprometer os controles de segurança do sistema, quanto um desvio acidental do sistema ou uma ameaça física como incêndios ou inundações.

Autenticação: É um conjunto de controles usados para verificar a identidade de um usuário, ou outra entidade que interage com o software.

Autenticação de Múltiplos Fatores: É um processo de autenticação que requer vários tipos de credenciais do usuário. Normalmente é baseado em algo que ele possui (ex.: cartão inteligente), algo que ele sabe (uma ex.: senha), ou em algo que ele é (ex.: dados provenientes de um leitor biométrico).

Autenticação Sequencial: Ocorre quando os dados de autenticação são solicitados em sucessivas páginas, ao invés de serem solicitados em uma única página.

Canonicalização: É uma operação realizada para reduzir várias codificações e representações de dados em uma única forma simplificada.

Caracteres Maliciosos: Quaisquer caracteres ou representações codificadas de caracteres que podem produzir efeitos indesejáveis sobre a operação normal de aplicações ou dos sistemas associados quando são interpretados, por terem significado especial. Estes caracteres podem ser usados para:

- Modificar a estrutura de código ou de declarações
- Inserir código indesejado
- Modificar caminhos
- Causar saídas inesperadas das funções ou rotinas dos programas
- Causar condições de erro
- Causar qualquer dos efeitos anteriores em aplicações subjacentes

Casos de Abuso: Descrevem o mau uso, intencional ou não, do software. Os casos de abuso devem desafiar os pressupostos do projeto do sistema.

Codificação de Entidade HTML: Processo de substituição de determinados caracteres ASCII pelas entidades HTML equivalentes. Por exemplo, a codificação poderia substituir o caractere "<" pela entidade HTML equivalente "<". Essas entidades são "inertes" na maioria dos interpretadores – especialmente navegadores – e podem atenuar os ataques do lado cliente.

Codificação de Saída Baseada em Contexto: É a codificação de dados da saída realizada usando como referência o modo como os dados serão utilizados pela aplicação. Se os dados da saída estiverem incluídos na resposta ao cliente, deve-se levar em consideração situações como: o corpo de um documento HTML, um atributo de HTML, codificação JavaScript, codificação dentro de um CSS ou de uma URL. Devem também ser levados em consideração outros casos como consultas SQL, XML e LDAP.

Codificação de Saída de Dados: É um conjunto de controles que abordam o uso de codificação para garantir uma saída de dados segura gerada pela aplicação.

Confidencialidade: É o ato de garantir que as informações são divulgadas apenas para as partes autorizadas.

Configuração do Sistema: Conjunto de controles que ajudam a garantir que os componentes de infraestrutura de apoio ao software são implantados de forma segura.

Consultas Parametrizadas (prepared statements): Mantém a consulta e os dados separados através do uso de espaços reservados. A estrutura de consulta é definida por caracteres especiais que representam os parâmetros a serem substituídos. A consulta parametrizada é enviada para o banco de dados, preparada para receber os parâmetros e, em seguida, é combinada com os valores dos mesmos. Isto impede que a consulta seja alterada porque os valores dos parâmetros são combinados com a declaração compilada e não concatenados diretamente na sequência de caracteres que compõem a consulta SQL.

Controle de Acesso: É um conjunto de controles que libera ou nega o acesso a um recurso do sistema a um usuário ou outra entidade qualquer. Normalmente é baseado em regras hierárquicas e privilégios individuais associados a papéis, porém também inclui interações entre sistemas.

Controles de Segurança: São ações que mitigam uma vulnerabilidade potencial e ajudam a garantir que o software se comporta conforme o esperado.

Cross Site Request Forgery (CSRF): Ocorre quando uma aplicação ou site web externos força o navegador do cliente a realizar uma requisição involuntária para uma aplicação em que o cliente possua uma sessão ativa. As aplicações são vulneráveis ao CSRF quando usam URLs e parâmetros conhecidos ou previsíveis ou quando o navegador transmite todas as informações da sessão para a aplicação vulnerável de forma automática em cada solicitação. Esse é apenas um dos ataques discutidos nesse documento e só está incluído porque a vulnerabilidade associada é muito comum, porém mal compreendida.

Disponibilidade: É a propriedade de estar acessível e utilizável quando demandado por uma entidade autorizada.

Estado dos Dados: São dados ou parâmetros usados pela aplicação ou pelo servidor para emular uma conexão persistente ou controlar o estado (*status*) de um cliente através de um processo de múltiplas requisições ou transações.

Exploit: É a ação de aproveitar-se da existência de uma vulnerabilidade. Normalmente é uma ação intencional e tem como objetivo comprometer os controles de segurança do software.

Gerência de Arquivo: É um conjunto de controles que resguardam a interação entre o código da aplicação e os arquivos do sistema.

> **Gerenciamento de Memória:** É um conjunto de controles que dizem respeito ao uso de memória e do buffer.

Gerenciamento de Sessão: É um conjunto de controles que tratam da manipulação de sessões HTTP de forma segura por aplicações Web.

Impacto: É o efeito negativo perceptível para o negócio, resultante da ocorrência de um evento indesejável, que por sua vez é o resultado da exploração de vulnerabilidades.

Integridade: É a garantia de que as informações são precisas, completas e válidas, e que não foram alteradas por uma ação não autorizada.

Limites de Confiança: Normalmente, um limite de confiança é constituído pelos componentes do sistema sobre os quais se tem controle direto. Todas as conexões e dados do sistema fora desse controle direto – incluindo todos os clientes e sistemas gerenciados por terceiros – devem ser considerados como não confiáveis e necessitam de validação na fronteira, antes de receberem permissões para realizarem interações com o sistema.

Mitigar: São medidas tomadas para reduzir o grau de severidade de uma vulnerabilidade. Essas medidas incluem a remoção de uma vulnerabilidade, seja ao torná-la mais difícil de ser explorada ou ao reduzir o impacto negativo de uma exploração bem sucedida.

Práticas de Criptografia: Conjunto de controles para garantir que as operações de criptografia dentro da aplicação sejam executadas de modo seguro.

Práticas Gerais de Codificação: Conjunto de controles abrangendo práticas de codificação que não se encaixam facilmente em outras categorias.

Proteção dos Dados: Conjunto de controles para ajudar a garantir que o software trata o armazenamento das informações de modo seguro.

Realizar Log dos Eventos: Esta operação deve incluir os seguintes requisitos:

1. Utilizar um timestamp[10] proveniente de um sistema confiável

[10]NT: Conjunto de caracteres que representa, de forma não ambígua, o momento do registro da ocorrência de um evento em um sistema. Para mais informações, ver a RFC 3339 e a norma ISO 8601, Data elements and interchange formats — Information interchange — Representation of dates and times

2. Classificar a severidade para cada evento
3. Destacar eventos de segurança relevantes, caso eles sejam misturados com outros registros de log
4. Registrar o identificador da conta ou usuário que causou o evento
5. Registrar o endereço IP de origem que realizou a requisição
6. Registrar o resultado dos eventos (sucesso ou falha)
7. Registrar a descrição do evento

Requisitos de Segurança: Conjunto de requisitos funcionais e de projeto para ajudar a garantir que o software seja construído e implantado de forma segura.

Segurança das Comunicações: Conjunto de controles para ajudar a garantir o envio e o recebimento das informações de modo seguro.

Segurança de Banco de Dados: Conjunto de controles para garantir que o software interaja com os bancos de dados de forma segura e que os bancos de dados estejam configurados de forma segura.

Sistema: É um termo genérico que abrange sistemas operacionais, servidores web, frameworks de aplicações e infraestrutura relacionada.

Tratamento de Erros e Log: Conjunto de práticas para garantir que a aplicação realize o tratamento dos erros de modo seguro e, também realize o registro de log dos eventos de modo apropriado.

Tratamento dos Dados: É o processo de tornar seguros os dados potencialmente prejudiciais através do processo de remoção, substituição, codificação ou *escaping*[11] dos caracteres.

Validação de Entrada de Dados: Conjunto de controles para verificar se as propriedades de todas as entradas de dados correspondem ao que é esperado pela aplicação, como, por exemplo, tipo dos dados, tamanho, intervalos, conjunto de caracteres aceitáveis e ausência de caracteres maliciosos.

Vulnerabilidade: É uma fragilidade que torna um sistema suscetível a um ataque ou a um dano.

[11] NT: Nesse contexto é a representação de caracteres especiais por conjuntos específicos de caracteres. Como exemplo, tem-se a substituição do caractere "&" pela entidade HTML equivalente "&"