Melhores Práticas de Codificação Segura OWASP
Guia de Referência Rápida

OWASP Secure Coding Practices - Quick Reference Guide

Copyright and License

Notas da versão

Esta versão do OWASP Secure Coding Practices foi desenvolvida como parte das atividades do capítulo Brasil da OWASP em prol da comunidade de desenvolvedores e da segurança dos sistemas desenvolvidos no Brasil. Este documento é baseado na versão 2.0 de Novembro de 2010. Participaram desta tradução:

- Leandro Resende Gomes – leandrock@gmail.com
- Sílvio Fernando Correia Vieira Filho – silviofilhosf@gmail.com
- Tarcizio Vieira Neto – tarciziovn@gmail.com

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

Sumário

Introdução

Este documento não se baseia apenas em questões tecnológicas e tem o propósito de definir um conjunto de boas práticas de segurança de codificação de aplicações. As recomendações serão apresentadas em um formato de checklist, que podem ser integradas ao ciclo de desenvolvimento de aplicações. A adoção destas práticas provavelmente vai reduzir as vulnerabilidades mais comuns em aplicações web.

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

Proteger os recursos críticos de 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% do total das tentativas de ataque observados na Internet.

Ao utilizar este guia, é recomendável que as equipes de desenvolvimento avaliem a maturidade do ciclo de vida de desenvolvimento de software e o nível de conhecimento da sua equipe de desenvolvimento. Como este guia não entra em detalhes de como implementar cada prática de codificação, os desenvolvedores precisam possuir um conhecimento prévio ou ter disponível os recursos suficientes que forneçam o direcionamento necessário. Este guia apresenta práticas de codificação que podem ser traduzidas em requisitos de codificação sem a necessidade do desenvolvedor possuir uma compreensão aprofundada das vulnerabilidades e exploits de segurança. No entanto, outros membros da equipe de desenvolvimento devem possuir responsabilidades, treinamentos adequados, ferramentas e recursos para validar se o projeto e a implementação atendem os requisitos de segurança.

Um glossário contendo termos importantes usados neste documento, incluindo títulos das seções e palavras mostradas em *itálico*, é fornecido no apêndice B.

Não faz parte do escopo deste guia fornecer orientações para implementar um framework de desenvolvimento seguro de software, no entanto, 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
  - OWASP CLASP Project
- Estabelecer padrões de codificação segura
  - OWASP Development Guide Project
- Construir uma biblioteca reutilizável ou fazer uso de uma biblioteca de segurança[1]
  - 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 as requisições das propostas, como para o contrato a ser firmado entre as partes.
  - OWASP Legal Project

1 Complemento adicionado pelo tradutor

Princípios Gerais de Segurança em Aplicações e Riscos

Construir software seguro exige ter o conhecimento básico dos princípios de segurança. Uma revisão abrangente dos princípios de segurança está fora do escopo deste guia, porém os conceitos de segurança serão abordados de forma superficial e 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. Esse objetivo é alcançado através da implementação de *controles de segurança*. Este guia se concentra nos controles técnicos específicos para *mitigar* as ocorrências de *vulnerabilidades* mais comuns de software. Como o foco principal são as aplicações web e sua infra-estrutura de apoio, boa parte deste guia pode ser usado para qualquer plataforma de desenvolvimento de software.

Para proteger o negócio contra os riscos inaceitáveis, isto é, relacionados a dependência e 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 pode ser *explorada* e causar um *impacto*. Como isto pode parecer um conceito abstrato, pense do seguinte modo: um ladrão de carros (agente de ameaça) passa por um estacionamento de carros verificando nos carros presentes (o sistema) procurando por 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. 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 partem 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. Estes 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 controles do lado cliente, como validação de entrada de dados no cliente, campos ocultos e controles de interface (combo box, radio buttons), fornecem pouco ou nenhum benefício de segurança. Neste caso, um atacante pode usar ferramentas como proxies do lado do cliente, como o OWASP WebScarab, Burp ou ferramentas de captura de pacotes de rede, como o Wireshark, para analisar o tráfego de aplicativo 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 compilados e analisados em busca de falhas.

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

- Não identificar as necessidades de segurança no início.
- Criação de arquiteturas conceituais que possuem erros de lógica.
- Usando práticas ruins de codificação que introduzem vulnerabilidades técnicas.
- Implantação do software de modo inapropriado.
- Inserção de falhas durante a manutenção ou atualização.

Além disso, é importante entender que as vulnerabilidades de software podem ter um escopo muito além do próprio software. Dependendo da natureza do software, da vulnerabilidade e da infra-estrutura de apoio, os impactos de uma exploração bem sucedida podem comprometer qualquer ou todos os seguintes aspectos:

- O software e sua informação associada.
- O sistema operacional dos servidores associados.

- O banco de dados de backend.
- Outras aplicações em um ambiente compartilhado.
- O sistema do usuário.
- Outros softwares que o usuário interage.

Checklist de Prática de Codificação Segura

Validação de Entrada de Dados:

- □ Efetuar toda a validação dos dados em um sistema confiável, por exemplo: centralizar todo controle no servidor.
- □ Identificar todas as fontes de dados e classificar as fontes como confiável/não confiável. Em seguida, validar os dados provenientes de fontes não confiáveis (ex: banco de dados, stream de arquivos, etc).
- □ A rotina de validação de dados de entrada deve ser centralizada na aplicação.
- □ Especificar conjunto de caracteres apropriados, 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 decodificação 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, por exemplo: nomes e valores dos Cookies. Certificar-se também de incluir automaticamente mecanismos de postback[2] nos trechos de código JavaScript, Flash ou qualquer outro código incorporado.
- □ Verificar os valores de cabeçalho, tanto das requisições, como das respostas, que contém 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 comprimento dos dados.
- □ Validar, sempre que possível, todos os dados de entradas através de um método baseado em “lista branca" que utiliza uma lista de caracteres ou expressão regular que define os caracteres permitidos.
- □ Se qualquer caractere potencialmente 'perigoso' precisa ser permitido na entrada de dados da aplicação, certifique-se 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 aborda as seguintes entradas, então elas devem ser verificadas discretamente:
  - Verificar bytes nulos (%00).
  - Verificar se há caracteres de nova linha (%0d, %0a, \r, \n).
  - Verificar se há caracteres ponto-ponto barra (../ ou ..\) que alteram caminhos. Nos casos de conjunto de caracteres que usam 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[3]) ou outras formas de ataques por ofuscação.

2 http://pt.wikipedia.org/wiki/Postback
3 http://www.owasp.org/index.php/Double_Encoding

Codificação da Saída de Dados:

- □ Efetuar toda a codificação dos dados em um sistema confiável, por exemplo: centralizar todo controle no servidor.
- □ Utilizar uma rotina padrão, testada, para cada tipo de codificação de saída
- □ Realizar a codificação, baseada em contexto, de todos os dados retornados para o cliente que originam-se de ambiente fora dos limites de confiança da aplicação. A codificação da entidade 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 não confiáveis que geram comandos de sistema operacional.

Autenticação e Gerenciamento de Senhas:

- □ 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 controle no servidor.
- □ Estabelecer e utilizar serviços de autenticação padronizados e testados, sempre que possível.
- □ Utilizar uma implementação centralizada para realizar os controles de autenticação, disponibilizando bibliotecas que invocam os serviços de autenticação externos.
- □ Separar a lógica de autenticação do recurso que está sendo requisitado e usar redirecionadores dos controles de autenticação centralizados.
- □ Quando situações excepcionais ocorrerem nos controles de autenticação, executar procedimentos em caso de falha de modo a 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 gerencia um repositório de credenciais, esta deverá garantir que as senhas sejam armazenadas no banco de dados somente o resumo/hash da senha na forma de one-way salted hashes[4], e que a tabela/arquivo que armazena as senhas e que as chaves sejam manipuladas apenas pela aplicação. Obs.: não utilizar o algoritmo de hash MD5, caso seu uso puder ser evitado.
- □ A geração dos resumos (hash) das senhas devem ser executadas em um sistema confiável, por exemplo: centralizar o controle no servidor.
- □ Validar os dados de autenticação somente ao término de todas as entradas de dados, especialmente para as implementações de *autenticação sequencial*.
- □ As respostas de falhas de autenticação não devem indicar qual parte dos dados de autenticação estão incorretos. Por exemplo: em vez de exibir mensagens como "Nome de usuário incorreto" ou "Senha incorreta", apenas utilize mensagens como: "Usuário e/ou senha inválidos", para ambos os casos de erro. As respostas de erro devem ser literalmente idênticas nos dois casos.
- □ Utilize autenticação para conexão a sistemas externos que envolvem tráfego de informação sensível ou acesso a funções.
- □ As credenciais de autenticação para acessar serviços externos à aplicação devem ser criptografados e armazenados em um local protegido em um sistema confiável, por exemplo: no servidor da

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

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 criptografada (SSL/TLS) ou como dado criptografado, como no caso de envio de e-mail criptografado. 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 ser suficientes para resistir 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 16 é melhor ou então considere o uso de senhas que contém várias palavras (uma frase).
- □ A entrada de senha deve ser ocultada na tela do usuário. Em HTML, utilize o campo tipo "password".
- □ Desativar a conta após um número pré-definido de tentativas inválidas de login (ex: cinco tentativas é o mais comum). A conta deve ser desativada por um período de tempo suficiente para desencorajar a dedução das credenciais pelo método de força bruta, mas nem 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 de senha devem evitar ataques que lançam respostas aleatórias, ex “livro favorito” é uma questão ruim, pois “A Bíblia” é uma resposta muito comum.
- □ Se for usar redefinição de senha baseada em e-mail, somente envie um e-mail para um endereço pré-definido contendo um link ou senha de acesso temporário que permitem ao usuário redefinir a senha.
- □ O tempo de validade das senhas e dos links temporários devem ser curtos.
- □ Exigir a mudança de senhas temporárias na próxima vez que o usuário realizar a autenticação no sistema.
- □ Notificar o usuário quando a sua senha for reiniciada (reset).
- □ Prevenir a reutilização de senhas.
- □ As senhas devem ter pelo menos um dia de duração antes de poderem ser alteradas para evitar ataques de reuso de senhas.
- □ Garantir que a troca de senhas estejam em conformidade com os requisitos estabelecidos na política ou regulamento. Sistemas críticos podem exigir alterações mais freqüentes nas credenciais de segurança. O tempo entre as trocas de senhas devem ser controladas administrativamente.
- □ Desabilitar 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 ao usuário no seu próximo login.
- □ Realizar monitoramento para identificar ataques contra várias contas de usuário, utilizando a mesma senha. Este 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 desabilite as contas associadas.
- □ Exigir uma re-autenticação dos usuários antes da realização de operações críticas.
- □ Utilizar *autenticação de múltiplos fatores* (utilizando simultaneamente token, senha, biometria, etc[5]) para contas altamente sensíveis ou de alto valor transacional.
- □ Caso for utilizar código de terceiros para realizar a autenticação, inspecione cuidadosamente o

5 Complementação explicativa provida pelo tradutor.

código para garantir se o mesmo não é afetado por qualquer código malicioso.

Gerenciamento de Sessão:

- □ Utilize controles de gerenciamento de sessão baseados no servidor ou em framework. A aplicação deve reconhecer apenas os identificadores de sessão como válidos.
- □ A criação dos identificadores de sessão devem ser sempre realizados em um sistema confiável, por exemplo: centralizar todo controle no servidor.
- □ Usar algoritmos bem controlados que garantam a aleatoriedade dos identificadores de sessão.
- □ Defina o domínio e o caminho para os cookies que contém identificadores de sessão autenticados para um valor devidamente restrito para o site.
- □ A funcionalidade de logout deve encerrar completamente a sessão ou conexão associada.
- □ A funcionalidade de 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 mais do que algumas horas.
- □ Não permitir logins persistentes (sem prazo para expirar sessão) e realizar o encerramento da sessão periodicamente, mesmo quando a sessão estiver ativa. Especialmente para aplicações que suportam várias conexões de rede ou que se conectam a sistemas críticos. O tempo de encerramento deve apoiar os requisitos de negócio, enquanto o usuário deve receber notificação suficientes para atenuar os impactos negativos destas medidas.
- □ Se uma sessão estava estabelecida antes do login, então esta sessão deve ser encerrada para que uma nova sessão seja estabelecida após o login.
- □ Gerar um novo identificador de sessão quando houver alguma 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 serem localizados no cabeçalho do cookie HTTP. Por exemplo, não trafegar os identificadores de sessão na 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 controle de acesso apropriado no servidor.
- □ Gerar um novo identificador de sessão e desativar o antigo identificador periodicamente. Isto pode mitigar certos cenários de ataques de sequestro de sessão (session hijacking), quando o identificador de sessão original é 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 para HTTPS.
- □ Utilize mecanismos complementares ao mecanismo de gerenciamento de sessão padrão para operações sensíveis do lado do servidor, como é o caso de operações de gerenciamento de contas, através da utilização de tokens aleatórios ou parâmetros associados à sessão. Este método pode ser usado para prevenir-se de ataques do tipo Cross Site Request Forgery.
- □ Utilize mecanismos complementares ao gerenciamento de sessão para operações altamente sensíveis ou críticas utilizando tokens aleatórios ou parâmetros em cada requisição.
- □ Utilizar somente identificadores de sessão gerados pelo sistema para gerenciamento de sessão do lado cliente. Evite usar parâmetros ou outros dados fornecidos pelos clientes para o gerenciamento do estado.
- □ 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 cookies 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 decisões de autorização de acesso.
- □ Utilize um único componente em todo o site para realizar o processo de verificação de autorização de acesso. Isso inclui bibliotecas que invocam os serviços externos de autorização.
- □ Quando ocorrer alguma falha no controle de acesso elas devem ocorrer de modo seguro.
- □ Negar todos os acessos caso a aplicação não consiga ter acesso as informações contidas na configuração de segurança.
- □ Garantir o controles de autorização em todas requisições, inclusive em scripts do lado do servidor, "includes" e requisições provenientes de tecnologias do lado cliente, como AJAX e Flash.
- □ Separar os trechos de código que contém a lógica privilegiada da aplicação do restante do código da aplicação.
- □ Restringir o acesso aos arquivos e outros recursos, incluindo aqueles que estão fora do controle direto da aplicação, somente a 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 para usuários autorizados.
- □ As regras de controle de acesso representados pela camada de apresentação devem coincidir com as regras presentes no lado servidor.
- □ Se o *estado dos dados* devem ser armazenados no lado cliente, utilize mecanismos de criptografia e verificação de integridade no lado servidor para detectar possíveis adulterações no *estado dos dados*.
- □ Garantir que os fluxos lógicos da aplicação respeitem as regras de negócio.
- □ Limitar o número de transações que um único usuário ou dispositivo podem executar em um determinado período de tempo. As transações por período de tempo devem estar acima da necessidade real do negócio, mas abaixo o suficiente para impedir ataques automatizados.
- □ Use o campo “referer” do cabeçalho somente como forma de verificação suplementar. Ele não deve ser usado sozinho como forma de checagem de autorização, pois o valor deste campo pode ser adulterado.
- □ Se é permitido a permanência de sessões autenticadas por longos períodos de tempo, faça revalidação periódica da autorização do usuário para garantir que seus privilégios não foram modificados e caso forem, realize o registro em log do usuário e exija nova autenticação.
- □ Implementar a auditoria das contas de usuário e assegure a desativação de contas não utilizadas. Por exemplo: após não mais do que 30 dias após expirar a senha da conta, a mesma deve ser desativada.
- □ A aplicação deve dar suporte a desativação de contas e encerramento das sessões quando encerrar a autorização do usuário, por exemplo: quando ocorrem alterações de mudança de papéis de 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 de acesso para que os acessos possam ser

devidamente concedidos e controlados. Isso inclui identificar requisitos de acessos, tanto para os dados, como para os recursos do sistema.

Práticas de Criptografia:

- □ Todos as funções de criptografia utilizados para proteger dados sensíveis dos usuários da aplicação devem ser implementados em um sistema confiável (neste caso o servidor).
- □ A senha mestre deve ser protegida contra acessos não autorizados.
- □ Quando ocorrer alguma falha dos módulos de criptografia, permitir que as falhas ocorram de modo seguro.
- □ Todos os números aleatórios, nomes de arquivos aleatórios, GUIDs aleatórios, e strings aleatórias devem ser geradas usando um módulo criptográfico com gerador de números aleatórios aprovado 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 padrão equivalente (http://csrc.nist.gov/groups/STM/cmvp/validation.html)
- □ Estabelecer e utilizar uma política e processo que define como é realizado o gerenciamento das chaves de criptografia.

Tratamento de Erros e Log:

- □ Não exponha informações sensíveis nas repostas de erros, inclusive detalhes de sistema, identificadores de sessão ou informação da conta do usuário.
- □ Use mecanismos de tratamento de erros que não exibam informações de debug ou informações da pilha de exceção.
- □ Implemente mensagens de erro genéricas e páginas de erro personalizadas.
- □ A aplicação deve tratar seus erros sem confiar 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 controles de segurança devem por padrão negar o acesso.
- □ Todos os controles de log devem ser implementados em um sistema confiável (neste caso o servidor).
- □ Os controles de log devem dar suporte tanto para os casos de sucesso ou falha relacionados a eventos de segurança específicos.
- □ Garantir que os logs armazenam eventos importantes.
- □ Garantir que as entradas de log que incluem dados não confiáveis não sejam executadas como um código na interface de visualização de logs.
- □ Restringir o acesso aos logs apenas para pessoal autorizado.
- □ Utilizar uma rotina centralizada para realizar todas 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 falhas de autenticação.
- □ Registrar em log todas as falhas de controle de acesso.
- □ Registrar em log todos os eventos aparentes de adulterações, 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 política de privilégio mínimo, restringindo os usuários apenas às funcionalidades, dados e informações do sistema que são necessárias para executar suas tarefas.
- □ Proteger todas as cópias temporárias ou registradas em cache que contenham dados sensíveis e estejam armazenados no servidor contra acesso não autorizado e realizar a remoção destes arquivos tão logo não sejam mais necessários.
- □ Criptografar informações altamente sensíveis quando armazenadas, como dados de verificação de autenticação, mesmo que estejam no lado servidor. Sempre usar algoritmos bem controlados. Consulte a seção que trata sobre "Práticas de Criptografia" para orientações adicionais.
- □ Proteger o código-fonte presente no servidor para que não sejam baixados por algum usuário.
- □ Não armazenar senhas, strings de conexão ou outras informações confidenciais em texto claro ou em qualquer forma criptograficamente insegura no lado cliente. Isso vale também quando há incorporação de formatos inseguros como: MS viewstate, Adobe Flash ou código compilado que roda no lado cliente.
- □ Remover comentários do código de produção que são acessíveis 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.
- □ Desabilitar a funcionalidade de auto completar nos formulários que contenham informações sensíveis, inclusive no formulário de autenticação.
- □ Desabilitar o cache realizado 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 cabeçalhos HTTP **"Pragma: no-cache"**, que é menos efetivo, mas é compatível com HTTP/1.0.
- □ A aplicação deve dar suporte a remoção de dados sensíveis quando os mesmos não forem mais necessários. Por exemplo: informação pessoal ou determinados 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 por criptografia de arquivos que contem dados sensíveis ou conexões que não usam o protocolo HTTP.
- □ Os certificados TLS devem ser válidos, possuir o nome de domínio correto, não estarem expirados e serem instalados com certificados intermediários, quando necessário.

- □ Quando ocorre falha nas conexões TLS, o sistema não deve retornar uma conexão insegura.
- □ Utilizar conexões TLS para todo conteúdo que requer acesso autenticado ou manutenção da confidencialidade das informações sensíveis.
- □ Utilizar um padrão único de implementação TLS que é configurado de modo apropriado.
- □ Especificar a codificação dos caracteres para todas as conexões.
- □ Filtrar os parâmetros que contenham informações sensíveis, provenientes do HTTP referer, quando realizar apontamentos 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 possuam os patches mais recentes aplicados para a versão em uso.
- □ Desabilitar a listagem de diretórios.
- □ Restringir os privilégios do servidor web, dos processos e das contas de serviços para o mínimo possível.
- □ Quando exceções ocorrem no sistema, permitir que as falhas ocorram de modo seguro.
- □ Remover todas as funcionalidades e arquivos desnecessários.
- □ Remover o código de teste ou qualquer funcionalidade desnecessária para o ambiente de produção, antes que seja realizada a implantação do sistema.
- □ Prevenir a divulgação da estrutura de diretórios impedindo que robôs[6] de busca façam indexação de arquivos sensíveis, através da correta configuração[7] do arquivo robots.txt, definindo diretórios que devem ser inacessíveis a estes indexadores em um diretório subjacente isolado. Assim, o acesso ao diretório pai definido no arquivo robots.txt deve estar desabilitado em vez de desabilitar 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.
- □ Desativar os métodos HTTP desnecessários, como extensões WebDAV. Caso for necessário o uso de algum método HTTP estendido para suportar manipulação de arquivos, então utilize algum mecanismo de autenticação bem controlado.
- □ Se o servidor processa tanto requisições HTTP 1.0 e 1.1, certificar-se de que ambos são configurados de modo semelhante ou assegurar que qualquer diferença que possa existir sejam compreendidas (ex. manuseio de métodos HTTP extendidos).
- □ Remover informações desnecessárias presentes nos cabeçalhos de resposta HTTP que podem estar relacionadas ao 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 nele.
- □ Isolar o ambiente de desenvolvimento da rede de produção e prover acesso somente para grupos de desenvolvimento e testes. Os ambientes de desenvolvimento comumente são configurados de modo menos seguro do que os ambientes de produção. Assim, os atacantes podem usar esse diferencial para descobrir vulnerabilidades compartilhadas ou encontrar caminhos para explorar as vulnerabilidades.

6 Robôs são programas de computador que percorrem automaticamente as páginas da Internet em busca de documentos, a fim de indexá-los, validá-los ou monitorar alterações de conteúdo. http://pt.wikipedia.org/wiki/Robots.txt

7 http://www.mundoseo.com.br/seo-tecnico/robotstxt-configuracao-seu-site/

- □ Implementar um sistema de controle de mudanças para gerenciar e registrar as alterações no código, tanto do desenvolvimento, como dos sistemas em produção.

Segurança em Base de Dados:

- □ Usar *consultas parametrizadas* fortemente tipadas.
- □ Utilizar validação de entrada e codificação de saída e assegure a abordagem de meta caracteres. Se houver falha, o comando no banco de dados não deve ser executado.
- □ Certificar-se de que as variáveis são fortemente tipadas.
- □ Realizar a codificação (escaping) de meta caracteres em instruções SQL[8].
- □ 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.
- □ As strings de conexão não devem ser codificadas na aplicação. A string de conexão deve ser armazenada em um arquivo de configuração separado 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 das permissões das tabelas no banco de dados.
- □ Encerrar a conexão tão logo seja possível.
- □ Remover ou modificar todas as senhas padrão de contas administrativas. Utilizar senhas robustas (incomuns ou difíceis de deduzir) ou implementar autenticação de múltiplos fatores. Desabilitar qualquer funcionalidade desnecessária no banco de dados, como stored procedures ou serviços desnecessários. Instale o mínimo conjunto de componentes ou opções necessárias (método de redução da área de superfície).
- □ Eliminar o conteúdo desnecessário incluído por padrão pelo fornecedor, ex: esquemas de exemplo.
- □ Desabilitar todas as contas criadas por padrão e que não são necessárias para suportar os requisitos de negócio.
- □ A aplicação deve se conectar ao banco de dados com diferentes credenciais de segurança para cada tipo de necessidade, como: usuário, somente leitura, convidado, administrador, etc.

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 upload de um arquivo.
- □ Limitar os tipos de arquivos que podem ser enviados para aceitar somente os tipos que são necessários para os propósitos do negócio.
- □ Validar se os arquivos enviados são do tipo esperado através da checagem dos cabeçalhos. Realizar a verificação de tipo de arquivo 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 na base de dados.
- □ Prevenir ou restringir upload de qualquer arquivo que possa ser interpretado pelo servidor web.
- □ Desabilitar privilégios de execução nos diretórios de upload de arquivos.
- □ Implementar o upload seguro nos ambientes UNIX por meio da montagem do diretório de destino como um unidade lógica, usando o caminho associado ou o ambiente de chroot.
- □ No referenciamento de arquivos existentes, use uma lista branca (white list) de nomes e tipos de arquivos permitidos. Realize a validação do valor do parâmetro passado e caso não corresponda ao

8 http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

que é esperado, rejeite a entrada ou utilize um valor de arquivo especificado por padrão pela aplicação.

- □ Não transmitir sem nenhum tratamento os dados informados pelo usuário a redirecionamentos dinâmicos. Se isto deve ser permitido, então o redirecionamento deve aceitar somente URLs relativas e validadas.
- □ Não passar parâmetros de caminhos de diretórios ou arquivos nas requisições. Utilize algum mecanismo de mapeamento dos caminhos em disco para índices que são repassados para os usuários e servem para serem mapeados em uma lista pré-definida de caminhos dos arquivos.
- □ Nunca enviar o caminho absoluto do arquivo para o cliente.
- □ Certificar-se de que os arquivos da aplicação e os recursos são do tipo somente leitura.
- □ Escanear arquivos que os usuários submeteram por mecanismo de upload em busca de vírus e malwares.

Gerenciamento de Memória:

- □ Utilize controle de entrada/saída para dados não confiáveis.
- □ Verificar se o buffer é tão grande quanto o especificado.
- □ Ao usar funções que aceitam um 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 seqüência de caracteres com valor nulo (null).
- □ Verificar os limites do buffer caso as chamadas a função são realizadas em um loop e verificar se não há nenhum perigo de escrever além do espaço alocado.
- □ Truncar todas as strings de entrada para um tamanho razoável antes de passá-las para as funções de cópia e concatenação.
- □ Encerre os recursos de modo específico, sem contar com o garbage collector na liberação dos recursos alocados para objetos de conexão, identificadores de arquivo, etc.
- □ Usar pilhas não-executáveis, quando disponíveis.
- □ Evitar o uso de funções reconhecidas por serem vulneráveis, por exemplo: 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:

- □ Utilizar sempre código testado, gerenciado e aprovado em vez de criar código novo, não gerenciado, para tarefas comuns.
- □ Utilizar APIs que embutem tarefas específicas para realizar tarefas do sistema operacional. Não permitir que a aplicação execute comandos diretamente no sistema operacional, especialmente através da utilização de shells de comando iniciados pela aplicação.
- □ Utilize mecanismo de verificação de integridade por checksum ou hash para verificar a integridade do código interpretado, bibliotecas, arquivos executáveis e arquivos de configuração.
- □ Utilize mecanismos de lock para evitar requisições simultâneas para a aplicação ou utilize um mecanismo de sincronização para evitar condições de disputa (race conditions).
- □ Proteja as variáveis compartilhadas e recursos contra acessos concorrentes inapropriados.
- □ Inicialize explicitamente todas as variáveis e outros dados persistidos, durante a declaração ou antes do primeiro uso da variável.
- □ Nos casos em que a aplicação deve ser executada com privilégios elevados, aumente os privilégios o mais tarde possível e revogue os privilégios tão logo seja possível.

- □ Evitar erros de cálculo decorrentes da falta de entendimento da representação interna da linguagem de programação usada e como é realizada a interação com os aspectos de cálculo numérico. Assim, deve-se prestar bastante atenção para as discrepâncias de tamanho de byte, precisão, distinções de sinal (signed/unsigned), truncamento, conversão e casting entre os tipos, cálculos que devolvem erros do tipo "not-a-number", e também como a linguagem de programação trata números muito grandes ou muito pequenos para sua representação interna.
- □ Não repassar diretamente dados fornecidos pelo usuário para qualquer função de execução dinâmica, sem antes realizar o *tratamento dos dados* de modo adequado.
- □ Restringir os usuários de gerar um novo código ou alterar o código existente.
- □ 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 deve realizar atualizações automáticas, então utilize mecanismos de assinatura digital para garantir a integridade do código e garanta que os clientes façam a verificação da assinatura após o download. Use canais criptografados para transferir o código a partir do host do servidor.

Apêndice A (Recursos externos e referências)

Referências Externas:

1. Referência citada

   Sans and TippingPoint "The Top Cyber Security Risks"

   http://www.sans.org/top-cyber-security-risks/

- Web Application Security Consortium
  http://www.webappsec.org/
- Common Weakness Enumeration (CWE)
  http://cwe.mitre.org/
- Department of Homeland Security
  Build Security In Portal
  https://buildsecurityin.us-cert.gov/daisy/bsi/home.html
- CERT Secure Coding
  http://www.cert.org/secure-coding/
- MSDN Security Developer Center
  http://msdn.microsoft.com/en-us/security/default.aspx
- SQL Injection Cheat Sheet

  http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- Cross Site Scripting (XSS) Cheat Sheet

  http://ha.ckers.org/xss.html

Sites de Avisos de Segurança:

Estes links podem ser úteis para verificar e garantir se a infra-estrutura de apoio e os frameworks não possuem vulnerabilidades conhecidas que afetam a infraestrutura de suporte e frameworks.

- Secunia Citrix Vulnerability List:
  http://secunia.com/advisories/search/?search=citrix
- Security Focus Vulnerability Search:
  http://www.securityfocus.com/vulnerabilities
- Open Source Vulnerability Database (OSVDB):
  http://osvdb.org/search/web_vuln_search
- Common Vulnerability Enumeration:
  http://www.cve.mitre.org/

Apêndice B (Glossário)

Agente de Ameaça: Qualquer entidade que pode ter um impacto negativo sobre o sistema. Este pode ser tanto um usuário mal-intencionado que quer comprometer os controles de segurança do sistema, como um desvio acidental do sistema ou uma ameaça física como incêndios ou inundações.

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

Autenticação Seqüencial: Ocorre quando os dados de autenticação são solicitados em sucessivas páginas, em vez de serem solicitados em uma única página.

Autenticação de múltiplos fatores: É um processo de autenticação que requer vários tipos de credenciais produzidas pelo usuário. Normalmente é baseado em algo que eles possuem (ex.: cartão inteligente), algo que eles sabem (uma ex.: senha), ou algo que eles são (ex.: dados provenientes de um leitor biométrico).

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

Caracteres Maliciosos: São caracteres ou representação codificada de um caractere que podem produzir efeitos indesejáveis sobre a operação normal da aplicação ou dos sistemas associados quando são interpretados, por terem significado especial. Estes caracteres podem ser usados para:

- Modificar a estrutura de código existente
- Inserir novo código não intencional
- Modificar caminhos
- Causar saídas inesperadas das funções ou rotinas
- Causar condições de erro
- Ter qualquer dos efeitos anteriores nas aplicações subjacentes

Casos de Abuso: Descreve o mau uso intencional (ou não) do software. Os casos de abuso desafiam os pressupostos do projeto do sistema.

Codificação de entidade HTML: Compreende ao processo de substituição de determinados caracteres ASCII com seus equivalentes em entidade HTML. Por exemplo, a codificação poderia substituir o caractere “<” com o equivalente em HTML "<". As entidades HTML são inertes na maioria dos interpretadores, especialmente navegadores, que podem atenuar os ataques do lado do cliente.

Codificação de saída baseada em contexto: Ocorre quando a codificação de dados de saída é realizada com base em como ela será utilizada pela aplicação. Os métodos específicos variam dependendo da forma como os dados são usados. Se os dados de saída estiverem incluídos na resposta ao cliente, é levado em conta como cenários de inclusão, situações como: o corpo de um documento HTML, um atributo de HTML, codificação JavaScript, codificação CSS ou em uma URL. Deve também ser levado em conta 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 que a saída de dados gerada pela aplicação seja segura.

Confidencialidade: Para garantir que as informações sejam divulgadas apenas para as partes autorizadas.

Configuração do Sistema: É um conjunto de controles que ajudam a garantir se os componentes de infraestrutura de apoio ao software são implementados 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 com caracteres especiais que representam os parâmetros da consulta a serem substituídos. A instrução SQL é enviada para o banco de dados e preparada para receber os parâmetros. Em seguida, a consulta parametrizada (prepared statement) é combinada com os valores dos parâmetros. Isto impede que a consulta seja alterada, pois, os valores dos parâmetros são combinados com a declaração compilada, e não concatenados diretamente a uma seqüência SQL.

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

Controles de Segurança: Uma ação que elimina uma vulnerabilidade potencial e ajuda a garantir que o software se comporte conforme o esperado.

Cross Site Request Forgery (CSRF): Uma aplicação ou site web externos forçam o navegador do cliente a realizar uma requisição involuntária para um aplicativo que o cliente possua uma sessão ativa. As aplicações são vulneráveis quando usam URLs e parâmetros conhecidas ou previsíveis, ou quando o navegador automaticamente transmite todas informações solicitadas pela sessão em cada solicitação para a aplicação vulnerável. Isto é apenas um dos ataques discutidos de modo especifico neste documento e só está incluído porque a vulnerabilidade associada é muito comum e mal compreendida.

Disponibilidade: Propriedade de estar acessível e utilizável quando demandado por uma entidade autorizada.

Estado dos Dados: Ocorre quando os dados ou parâmetros são usados, através da aplicação ou servidor, para emular uma conexão persistente ou controlar o status de um cliente através de um processo de múltiplas requisições ou transações.

Exploit: É uma ação que tira proveito de uma vulnerabilidade. Normalmente é uma ação intencional destinada a comprometer os controles de segurança do software, aproveitando a existência de uma vulnerabilidade.

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

Gerenciamento de memória: É um conjunto de controles que tratam do uso de memória e do buffer.

Gerenciamento de Sessão: É um conjunto de controles que ajudam a garantir que as aplicações web lidem com sessões http de um modo seguro.

Impacto: É o perceptível efeito negativo para o negócio, resultante da ocorrência de um evento indesejável, que por sua vez é o resultado de uma, ou mais, vulnerabilidade(s) explorada(s).

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 constitui nos componentes do sistema sob seu controle direto. Todas as conexões e dados do sistema fora de seu controle direto, incluindo todos os clientes e sistemas gerenciados por terceiros, devem ser considerados como não-confiáveis e que necessitam ser validados na fronteira, antes de permitir uma maior interação com o sistema.

Mitigar: São medidas tomadas para reduzir o impacto de uma vulnerabilidade. Estas medidas incluem a remoção de uma vulnerabilidade, seja ao tornar uma vulnerabilidade mais difícil de ser explorada, ou ao reduzir o impacto negativo de uma exploração bem sucedida.

Práticas de Codificação Geral: É um conjunto de controles que abrangem práticas de codificação que não se encaixam facilmente em outras categorias.

Práticas de Criptografia: É um conjunto de controles que garantem que as operações de criptografia dentro da aplicação sejam executadas de modo seguro.

Proteção dos Dados: É um conjunto de controles que ajuda 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 o timestamp proveniente de um sistema confiável
2. Classificar a severidade para cada evento
3. Marcar eventos de segurança relevantes, caso forem misturados com outros registros de log
4. Registrar o identificador da conta ou usuário que causou o evento
5. Registrar o 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: É um conjunto de requisitos funcionais e de projeto que ajudam a garantir que o software seja construído e implantado de forma segura.

Segurança das Comunicações: É um conjunto de controles que ajudam a garantir o envio e o recebimento das informações de modo seguro.

Segurança de Banco de Dados: É um conjunto de controles que garantem que o software interage com o banco de dados de forma segura e garantem também que o banco de dados esteja configurado de forma segura.

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

Tratamento de erros e log: É um conjunto de práticas que garantem que a aplicação realiza o tratamento dos erros de modo seguro, como também realiza de modo apropriado o registro de log dos eventos.

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

Validação de Entrada de Dados: É o conjunto de controles que verificam se as propriedades de todas as entradas de dados correspondem ao que é esperado pela aplicação, como tipo dos dados, tamanho, intervalos, conjunto de caracteres aceitáveis que não contenham caracteres maliciosos.

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