



Requisitos Técnicos para Codificação Segura

Poder Judiciário do Estado do Rio de Janeiro

Gabinete da Presidência

Departamento de Segurança da Informação

Versão 1.0

Sumário

1. Uso de frameworks, bibliotecas e demais componentes de terceiros	3
2. Acesso seguro à base de dados.....	4
3. Validar todas as entradas de dados	4
4. Identidade digital – autenticação, gerenciamento de sessão e senhas	5
5. Aplicar controles de acesso (autorização)	7
6. Proteger os dados de ponta a ponta.....	7
7. Implementar registro de logs e monitoramento de segurança	8
8. Lidar com todos os erros e exceções	9

O presente guia, constantemente revisado, define um ponto de partida e atenção para a construção de um software seguro.

1. Uso de frameworks, bibliotecas e demais componentes de terceiros

Frameworks e bibliotecas ajudam os desenvolvedores a se proteger contra falhas de design e implementação. Escrever uma aplicação totalmente do zero, além de impactar em prazos, pode preterir as boas práticas de código e de segurança da informação.

Utilizar frameworks consolidados de desenvolvimento garante que se traga as melhores práticas de codificação e segurança ao projeto, cobrindo, inclusive, muitos dos pontos tratados adiante.

Melhores Práticas:

Ao incorporar frameworks, bibliotecas ou qualquer componente de terceiros no software, considerar as seguintes melhores práticas:

1. Utilizar as tecnologias homologadas pelo PJERJ, baixadas de fontes oficiais e ativamente mantidas;
2. Criar e manter um inventário de todos esses componentes, bem como suas versões;
3. Remover os componentes desnecessários do projeto. No caso de componentes legados, justificar sua permanência no inventário;
4. Manter todos os componentes atualizados. Quando o PJERJ não disponibilizar ferramentas específicas homologadas para tal, considerar soluções como [OWASP Dependency Check](#), [OSV Scanner](#) ou [RetireJS](#), para identificar dependências de projeto e verificar, recorrentemente, se existem vulnerabilidades publicadas no código de terceiros;

5. Reduza a superfície de ataque isolando as bibliotecas e expondo somente o comportamento requerido pelo software.

Observar tais controles previne diversas vulnerabilidades em aplicações web. Portanto, é imprescindível manter esses componentes atualizados e inventariados.

2. Acesso seguro à base de dados

Estes controles se aplicam a bases de dados relacionais e não relacionais.

- **Queries Seguras**

Para mitigar o SQL Injection, entradas não confiáveis não devem ser interpretadas e a melhor forma de tratá-las é utilizar técnicas de parametrização de query. Uma lista de exemplos de implementação pode ser encontrada neste [guia da OWASP](#).

Utilizar frameworks consolidados de mapeamento objeto-relacional (ORM), como Hibernate ou NHibernate, para reforçar esses controles.

- **Autenticação Segura**

Todo o acesso ao banco de dados deve ser propriamente autenticado e através de um canal seguro.

- **Comunicação Segura**

Todos os métodos de comunicação (*services*, APIs e afins) devem ocorrer sob canais seguros.

3. Validar todas as entradas de dados

A validação, ou sanitização, de entrada de dados garante que apenas os dados formatados corretamente possam entrar no sistema.

As linguagens de programação e a maioria dos frameworks fornecem bibliotecas ou funções de validação que devem ser aproveitadas.

Observar elementos HTML, atributos HTML, códigos JavaScript etc., para que não sejam interpretados com a entrada e persistência de dados.

Atentar-se, no mínimo, os seguintes caracteres na entrada de dados: ' (aspas simples), " (aspas duplas), > (maior que), < (menor que), & (e comercial), / (barra) e \ (barra invertida).

Requisitos:

1. Valide a entrada do usuário (incluindo todos os campos ocultos) e verifique o comprimento, tipo e intervalo adequados;
2. A validação de entrada deve ser baseada em uma abordagem de “lista branca” sempre que possível (ou seja, a aplicação deve aceitar e processar apenas a entrada esperada);
3. Sempre implemente a validação de entrada no lado do servidor; a validação feita no lado do cliente também deve ser feita, mas nunca será confiável;
4. Utilizar bibliotecas/funções de validação de entrada para seguir as boas práticas já evidenciadas e evitar retrabalhos.

Frisa-se que a validação de entrada não deve ser utilizada como principal método de prevenção a XSS, SQL Injection e outros ataques, devendo-se, também, utilizar frameworks de mapeamento objeto-relacional (ORM) e demais controles a nível de banco de dados.

4. Identidade digital – autenticação, gerenciamento de sessão e senhas

Identidade digital é a representação única do usuário, enquanto logado, que está envolvido nas atividades do sistema.

Requisitos:

1. As senhas devem cumprir os padrões definidos pelo PJERJ, **conforme disposto no Ato Normativo nº 27/2020 e suas atualizações, que trata da Gestão de Acessos;**

2. Implementar mecanismo de recuperação de senha segura:
 - a. Utilizar elementos de autenticação multifator para recuperar a senha (ex.: token gerado por algum dispositivo ou enviado por e-mail, pergunta de segurança etc.).
3. Implementar armazenamento seguro de senha
 - a. Aplicar controles criptográficos com *hash* e técnicas de “salt”, de forma que se uma credencial for comprometida o invasor não consiga acesso imediato a essas informações.
4. Quanto ao gerenciamento de sessão:
 - a. Do lado do cliente, o usuário só precisa manter seu identificador de sessão (ID);
 - b. O ID de sessão deve ser longo, único e aleatório;
 - c. A aplicação deve gerar uma nova sessão, ou pelo menos alternar o ID da sessão, durante a autenticação e a reautenticação;
 - d. A aplicação deve implementar um limite de inatividade, conforme definido pelo PJERJ no Ato Normativo nº 10/2019 e suas atualizações, que trata da Gestão de Ativos de Segurança da Informação, encerrando a sessão do usuário após o período de inatividade.
 - e. Cookies:
 - i. *Cookies* devem ser acessíveis a um conjunto mínimo de domínios. Seus caminhos devem ser marcados para expirar junto com a sessão;
 - ii. A flag “secure” deve ser definida para garantir que transferência seja feita somente via canal seguro (TLS);
 - iii. A flag “HttpOnly” deve ser definida para evitar que o *cookie* seja acessado via JavaScript;
 - iv. Adicionar atributos “SameSite = Strict” aos *cookies* para mitigar vazamento de informações e falsificações.
 - f. JWT (JSON Web Tokens)
 - i. Garantir que os tokens enviados ao cliente também sejam salvos no lado do servidor para garantir a integridade e verificar, posteriormente, se são válidos e que não foram adulterados desde a criação.

5. Aplicar controles de acesso (autorização)

Autorização: envolve a verificação de acesso a recursos ou recursos específicos. Não confundir com Autenticação, que diz respeito a verificação de identidade.

Requisitos:

1. Certificar de que todas as solicitações passem por algum tipo de camada de verificação de controle de acesso;
2. Seguir o princípio da “negação por padrão”: **se uma solicitação não for especificamente permitida, ela deverá ser negada.**
 - a. Quando o administrador criar um novo usuário, ou o usuário puder se registrar em uma nova conta, essa conta deve ter acesso mínimo ou nenhum acesso por padrão até que o acesso seja configurado;
 - b. Quando um novo recurso é adicionado a uma aplicação, todos os usuários devem ser impedidos de usar esse recurso até que ele seja configurado corretamente.
3. Seguir o princípio do privilégio mínimo:
 - a. Todos os usuários, programas ou processos devem receber apenas o mínimo de acesso necessário possível.

6. Proteger os dados de ponta a ponta

Requisitos:

1. Não armazenar credenciais ou quaisquer outras chaves de acesso em código-fonte ou arquivos de configuração. Utilizar apenas as ferramentas homologadas pelo PJERJ para tal inspeção.
2. Criptografar dados em trânsito:
 - a. O TLS deve ser configurado adequadamente para proteger as comunicações seguras e prover a segurança na camada de transporte.
3. Criptografar dados “em repouso”:

- a. Evitar armazenar dados confidenciais. Quando necessário armazenar dados confidenciais, verificar se estão protegidos criptograficamente, de modo a evitar divulgação e modificação não autorizadas.
4. Utilizar soluções consolidadas, abertas e atualizadas, quando for necessário tratar de criptografia;
5. Quando do desenvolvimento de aplicações móveis, certificar de manter o armazenamento local seguro. Armazenar no dispositivo móvel apenas os dados mínimos necessários. Se for necessário armazenar dados confidenciais, utilizar os diretórios de armazenamento de dados específicos do sistema operacional móvel (no Android será o diretório “keystore” e no iOS o “keychain”).

7. Implementar registro de logs e monitoramento de segurança

Requisitos:

- Registrar data/hora, navegador (User-Agent), método, URL, incluindo o IP de origem e o ID do usuário;
- **Não registrar dados privados ou confidenciais**, como senhas, ID de sessão, entre outros identificadores;
- Que haja sincronização de tempo no registro de logs para garantir a consistência dos dados;
- Validar contra caracteres indesejáveis antes de gravar os registros, de modo a evitar ataques de injeção ou falsificação de logs;
- Registros que indiquem tentativas de intrusão ou comportamentos anômalos, identificando-os como de “alta gravidade”, incluindo dados como:
 - Dados enviados diferentes dos permitidos na “lista branca” ou nulos;
 - Dados enviados fora de um intervalo numérico esperado;
 - Dados enviados através de campos que não devem ser modificados (ex.: lista de seleção, caixa de seleção ou outro componente de entrada limitada);
 - Solicitações que violam as regras de controle de acesso do lado do servidor;

- Implementar controles a nível de aplicação para que, assim que ocorra o evento anormal, a aplicação responda ao possível ataque invalidando a sessão do usuário ou bloqueando sua conta.

É importante manter esses registros para:

- Análise e investigações forenses;
- Cumprir requisitos de conformidade e legislação.

8. Lidar com todos os erros e exceções

Requisitos:

1. Certificar de que todo o comportamento inesperado seja tratado corretamente dentro da aplicação;
2. Certificar de que as mensagens de erro exibidas aos usuários não vazem dados críticos, mas que ainda sejam detalhadas o suficiente para permitir a resposta adequada ao usuário;
3. Certificar de que as exceções sejam registradas, fornecendo informações suficientes para que as equipes de suporte, controle de qualidade, análise forense ou de resposta a incidentes entendam o problema;
4. Testar e verificar continuamente o código de tratamento de erros.