Cebraspe
Centro Brasileiro de Pesquisa em Avaliação e Seleção e de Promoção de Eventos

MINISTÉRIO DO PLANEJAMENTO E ORÇAMENTO
SECRETARIA DE ORÇAMENTO FEDERAL
SUBSECRETARIA DE TECNOLOGIA E DESENVOLVIMENTO INSTITUCIONAL

CARGO 4: ANALISTA DE PLANEJAMENTO E ORÇAMENTO – ESPECIALIDADE: DESENVOLVIMENTO DE SISTEMAS ORÇAMENTÁRIOS

Prova Discursiva P_4 – Dissertação

Aplicação: 07/07/2024

PADRÃO DE RESPOSTA DEFINITIVO

1. O desenvolvimento seguro é um elemento fundamental na garantia da segurança de *software*, pois envolve a incorporação de práticas e técnicas de segurança durante todo o ciclo de vida do desenvolvimento de *software*, desde a sua concepção até sua implantação. A importância do desenvolvimento seguro reside na prevenção de vulnerabilidades e na redução do risco de exploração por ameaças cibernéticas.

2. A seguir, é apresentada uma lista de boas práticas de desenvolvimento seguro, exigindo-se do(a) candidato(a) a abordagem de pelo menos quatro delas:

   - **Conscientização sobre segurança**. É fundamental que os desenvolvedores estejam cientes das melhores práticas de segurança e das principais ameaças cibernéticas. Eles devem ser treinados regularmente sobre como identificar e remediar possíveis vulnerabilidades;
   - **Princípio da defesa em profundidade**. Devem-se implementar várias camadas de segurança para proteger o sistema como um todo, utilizando-se *firewalls*, antivírus, detecção de intrusões, criptografia e outras medidas de segurança em diferentes níveis da arquitetura do *software*;
   - **Validação de entrada**. Sempre se deve validar e filtrar todas as entradas de dados recebidas dos usuários, dispositivos externos ou outras fontes. Isso ajuda a prevenir ataques de injeção de código, como SQL *injection* e XSS (*cross-site scripting*);
   - **Tratamento seguro de erros**. Deve-se evitar exibir mensagens de erro detalhadas para os usuários, pois isso pode fornecer informações valiosas para um potencial invasor. Em vez disso, é mais apropriado registrar erros de forma adequada e apresentar mensagens genéricas aos usuários;
   - **Uso de bibliotecas e *frameworks* seguros.** Ao utilizar bibliotecas e *frameworks* de terceiros, deve-se verificar se eles são confiáveis, se estão atualizados e se possuem um histórico de segurança sólido, mantendo-os sempre atualizados para evitar vulnerabilidades conhecidas;
   - **Autenticação e autorização**. Convém implementar um sistema de autenticação seguro para verificar a identidade dos usuários. Nesse sentido, devem-se utilizar senhas fortes, autenticação de dois fatores e mecanismos adequados de controle de acesso para garantir que apenas usuários autorizados tenham acesso aos recursos apropriados;
   - **Proteção de dados sensíveis**. Devem-se utilizar técnicas adequadas de criptografia para proteger dados sensíveis em trânsito e em repouso, armazenar senhas e informações confidenciais de forma criptografada e proteger a comunicação por meio de canais seguros, como SSL/TLS;
   - **Testes de segurança**. Devem-se realizar testes regulares de segurança, como testes de penetração e análise de código, para identificar e corrigir vulnerabilidades no *software*. Isso inclui a realização de testes de segurança automatizados e a revisão manual do código em busca de possíveis falhas;
   - **Gerenciamento de patches e atualizações**. Deve-se manter o *software* atualizado com as últimas correções de segurança. Isso inclui atualizações do sistema operacional, correções de *bugs* e *patches* de segurança para bibliotecas e *frameworks* utilizados;
   - **Monitoramento e registro de atividades**. Devem-se implementar mecanismos de monitoramento e registro de atividades do sistema para detectar atividades suspeitas e anomalias. Isso ajuda a identificar possíveis violações de segurança e a tomar medidas corretivas rapidamente;

- **Limite de privilégios**. Devem ser atribuídos aos usuários apenas os privilégios necessários para que realizem suas tarefas, evitando-se conceder privilégios excessivos, pois isso pode aumentar o risco de abuso ou exploração. Cabe implementar controle de acesso baseado em papéis e revisar regularmente as permissões atribuídas aos usuários;
- **Proteção contra ataques de negação de serviço (DoS)**. Para proteger o sistema contra ataques de negação de serviço, é preciso implementar medidas como limitação de taxa, validação de entrada e dimensionamento adequado da infraestrutura, além de monitorar o tráfego de rede e definir alertas para detectar atividades suspeitas;
- **Gerenciamento de vulnerabilidades**. É necessário realizar uma avaliação contínua de vulnerabilidades no *software* e na infraestrutura, utilizar ferramentas de varredura de vulnerabilidades e aplicar *patches* e atualizações de segurança assim que estiverem disponíveis, bem como se manter atualizado sobre as últimas vulnerabilidades divulgadas e tomar medidas imediatas para mitigar os riscos;
- **Proteção contra ataques de engenharia social**. Recomenda-se educar os usuários e a equipe de desenvolvimento sobre as técnicas de engenharia social e as ameaças associadas. Isso inclui *phishing*, *spear phishing* e outras tentativas de manipulação para obter acesso não autorizado às informações. Nesse sentido, cabe implementar políticas de segurança, treinamento de conscientização e processos de verificação para mitigar esses riscos;
- **Revisão de código e testes de segurança**. Devem-se realizar revisões de código por pares para identificar possíveis vulnerabilidades de segurança. Além disso, devem-se conduzir testes de segurança regulares, como testes de penetração e análises de código estático, para identificar falhas de segurança, e corrigir as vulnerabilidades identificadas antes de implantar o *software* em produção; ~~e~~
- **Monitoramento e resposta a incidentes**. Deve-se implementar um plano de resposta a incidentes, para lidar com violações de segurança. Isso inclui a definição de procedimentos claros para notificação, investigação e mitigação de incidentes. Também é preciso monitorar o sistema em tempo real, para detectar atividades suspeitas e responder prontamente a qualquer incidente de segurança~~.~~; e
- **Segregação de ambientes.** Deve-se implementar uma segregação dos ambientes de desenvolvimento, testes/homologação e de produção para impedir que pessoas responsáveis por cada etapa tenham acesso a outro ambiente, principalmente o de produção. Deve-se usar servidores distintos para cada ambiente, assim como o banco de dados.

3. Existem várias abordagens para realizar testes de vulnerabilidade no desenvolvimento de sistemas e aplicações, sendo as mais comuns as seguintes:

   - **Análise estática de código:** o código-fonte da aplicação é examinado minuciosamente em busca de falhas de segurança, como erros de validação, falta de tratamento adequado de entradas do usuário ou vulnerabilidades conhecidas. Ferramentas como o FindBugs, PMD e SonarQube são amplamente utilizadas para essa finalidade;
   - **Análise dinâmica:** também conhecida como teste de caixa-preta, envolve a execução de testes em tempo real no sistema ou aplicativo, para identificação de vulnerabilidades enquanto ele está em funcionamento. Isso geralmente é feito por meio de ferramentas automatizadas que enviam solicitações, fornecem entradas maliciosas e monitoram o comportamento do sistema. Exemplos de ferramentas populares são o OWASP ZAP, Burp Suite e Nessus;
   - **Varredura de vulnerabilidades:** uma ferramenta específica é usada para varrer a rede, o sistema ou o aplicativo, em busca de vulnerabilidades conhecidas. A ferramenta verifica se há versões desatualizadas de *software*, configurações incorretas, serviços não seguros e outras falhas conhecidas. Exemplos de ferramentas de varredura de vulnerabilidades são o OpenVAS, Nessus e QualysGuard; e
   - **Análise de configuração:** concentra-se na avaliação das configurações de segurança de um sistema ou rede. Isso inclui verificar se as políticas de segurança estão corretamente implementadas, se os dispositivos estão configurados de forma segura, se as permissões de acesso estão adequadas e se os mecanismos de autenticação estão configurados corretamente.
   - ***Pen-test***: o teste de intrusão, ou *pentest*, é uma técnica proativa e controlada utilizada para avaliar a segurança de sistemas, redes, aplicativos e ambientes tecnológicos como um todo. O objetivo principal de um *pentest* é simular os métodos e as técnicas que um atacante real poderia empregar para explorar vulnerabilidades e invadir sistemas. No entanto, ao contrário de um ataque real, um *pentest* é conduzido de forma ética, legal e controlada, com o objetivo de identificar falhas de segurança antes que sejam exploradas por criminosos cibernéticos.

4. A lista OWASP Top Ten é baseada em uma combinação de análise de dados fornecidos por usuários e uma pesquisa com profissionais do setor. A OWASP desenvolveu uma série de recursos que descrevem as vulnerabilidades mais comuns existentes em vários sistemas, incluindo aplicativos *web*, API e dispositivos móveis. O OWASP Top Ten descreve as dez vulnerabilidades mais comuns e impactantes que aparecem em aplicativos *web* em produção. Essa lista é atualizada com base em uma combinação de dados de testes de segurança e pesquisas com profissionais do setor. A versão mais recente da lista OWASP Top Ten foi lançada em 2021. Esse recurso fornece informações sobre as vulnerabilidades mais comuns, exemplos de cada tipo, práticas recomendadas para preveni-las e descrições de como a vulnerabilidade pode ser explorada.

5. A versão mais recente do OWASP Top Ten promoveu várias alterações em relação à versão anterior. A lista de 2021 inclui as seguintes vulnerabilidades:

   - **Controle de acesso quebrado**. Os sistemas de controle de acesso têm como objetivo garantir que apenas usuários legítimos tenham acesso aos dados ou a funcionalidades. A vulnerabilidade na categoria de controle de acesso quebrado inclui qualquer problema que permita a um invasor contornar os controles de acesso ou que não consiga implementar o princípio do menor privilégio. Por exemplo, um aplicativo *web* pode permitir que um usuário acesse a conta de outro usuário modificando a URL fornecida;

- **Falhas criptográficas**. Os algoritmos criptográficos são inestimáveis para proteger a privacidade e a segurança dos dados, entretanto esses algoritmos podem ser muito sensíveis a erros de implementação ou configuração. As falhas criptográficas incluem a falha no uso da criptografia, configurações incorretas de algoritmos criptográficos e gerenciamento inseguro de chaves. Por exemplo, uma organização pode usar um algoritmo *hash* inseguro para armazenamento de senhas, falhar ao salvar senhas ou usar o mesmo valor para todas as senhas de usuários armazenadas;
- **Injeção**. A vulnerabilidade de injeção é possível devido a uma falha na higienização adequada da entrada do usuário antes de processá-la. Isso pode ser especialmente problemático em linguagens como SQL, na qual dados e comandos são misturados, de modo que dados maliciosos e malformados fornecidos pelo usuário possam ser interpretados como parte de um comando. Por exemplo, em um código SQL, normalmente se usam aspas simples ou duplas para delinear os dados do usuário em uma consulta, portanto a entrada do usuário que contém esses caracteres pode ser capaz de alterar o comando que está sendo processado;
- ***Design* inseguro**. A vulnerabilidade pode ser introduzida no *software* durante o processo de desenvolvimento, de duas maneiras diferentes. Embora muitas das vulnerabilidades na lista Top Ten do OWASP tratem de erros de implementação, essa vulnerabilidade descreve falhas de *design* que prejudicam a segurança do sistema. Por exemplo, se o *design* de um aplicativo que armazena e processa dados confidenciais não inclui um sistema de autenticação, então uma implementação perfeita do *software* conforme projetado ainda será insegura e não protegerá adequadamente esses dados confidenciais;
- **Configuração incorreta de segurança**. A segurança de um aplicativo é determinada não apenas por seu *design* e sua implementação, mas também pela forma como ele é configurado. Um fabricante de *software* terá configurações-padrão para seu aplicativo, e os usuários também poderão ativar ou desativar diversas configurações, o que pode melhorar ou prejudicar a segurança do sistema. Exemplos de configurações incorretas de segurança incluem a ativação de aplicativos ou portas desnecessárias, deixar contas e senhas-padrão ativas e inalteradas, ou configurar mensagens de erro para expor muitas informações a um usuário;
- **Componentes vulneráveis e desatualizados**. A vulnerabilidade da cadeia de fornecimento surgiu como uma grande preocupação nos últimos anos, especialmente porque os agentes de ameaças tentaram inserir códigos maliciosos ou vulneráveis em bibliotecas comumente usadas e dependências de terceiros. Se uma organização não tiver visibilidade do código externo usado em seu aplicativo — incluídas dependências aninhadas — e não conseguir verificar se há dependências, ela poderá ficar vulnerável à exploração. Além disso, a falha na aplicação imediata de atualizações de segurança a essas dependências pode deixar uma vulnerabilidade explorável aberta a ataques. Por exemplo, um aplicativo pode importar uma biblioteca de terceiros que tenha as próprias dependências que possam conter vulnerabilidades exploráveis conhecidas;
- **Falhas de identificação e autenticação**. Muitos aplicativos e sistemas exigem alguma forma de identificação e autenticação, como um usuário que comprove sua identidade a um aplicativo ou um servidor que forneça um certificado digital que verifica sua identidade a um usuário ao configurar uma conexão criptografada por TLS. Falhas de identificação e autenticação ocorrem quando um aplicativo depende de processos de autenticação fracos ou não consegue validar adequadamente as informações de autenticação. Por exemplo, um aplicativo que não possui autenticação multifatorial pode ser vulnerável a um ataque de preenchimento de credenciais, no qual um invasor tenta automaticamente combinações de nome de usuário e senha a partir de uma lista de credenciais fracas, comuns, padrão ou comprometidas;
- **Falhas de *software* e integridade de dados**. A vulnerabilidade de falhas de integridade de *software* e dados na lista dos dez principais do OWASP aborda pontos fracos na segurança do *pipeline* DevOps de uma organização e nos processos de atualização de *software* semelhantes àqueles que tornaram possível o *hack* da SolarWinds. Essa classe de vulnerabilidade inclui depender de código de terceiros de fontes ou repositórios não confiáveis, não proteger o acesso ao *pipeline* de CI/CD e não validar adequadamente a integridade das atualizações aplicadas automaticamente. Por exemplo, se um invasor puder substituir um módulo ou uma dependência confiável por uma versão modificada ou maliciosa, os aplicativos criados com essa dependência poderão executar código malicioso ou ficar vulneráveis à exploração;
- **Falhas de registro e monitoramento de segurança**. As falhas de registro e monitoramento de segurança são a primeira das vulnerabilidades derivadas das respostas da pesquisa. Muitos incidentes de segurança são ativados ou exacerbados pelo fato de um aplicativo não conseguir registrar eventos de segurança significativos ou de esses arquivos de *log* não serem monitorados e manipulados adequadamente. Por exemplo, um aplicativo pode não gerar arquivos de *log*, pode gerar *logs* de segurança sem informações críticas ou esses arquivos de *log* podem estar disponíveis apenas localmente em um computador, tornando-os úteis apenas para investigação após a detecção de um incidente. Todas essas falhas degradam a capacidade de uma organização detectar rapidamente um potencial incidente de segurança e responder em tempo real; e
- **Falsificação de solicitação do lado do servidor**. A falsificação de solicitação do lado do servidor (SSRF) é incomum entre as vulnerabilidades listadas pelo OWASP, porque descreve uma vulnerabilidade ou ataque muito específico, em vez de uma categoria geral. A vulnerabilidade dos SSRF é relativamente rara, no entanto tem um impacto significativo se forem identificados e explorados por um invasor. O *hack* do Capital One é um exemplo de incidente de segurança recente e de alto impacto que se aproveitou de uma vulnerabilidade SSRF. A vulnerabilidade SSRF pode existir quando um aplicativo *web* não valida adequadamente uma URL fornecida por um usuário ao buscar um recurso remoto localizado nessa URL. Se for esse o caso, então um invasor que explora a vulnerabilidade pode usar o aplicativo *web* vulnerável para enviar uma solicitação elaborada pelo invasor para a URL indicada. Isso permite que o invasor contorne os controles de acesso, como um *firewall*, que bloquearia conexões diretas do invasor com a URL de destino, mas está configurado para fornecer acesso ao aplicativo *web* vulnerável.

QUESITOS AVALIADOS

QUESITO 2.1
Conceito 0 – Não abordou o quesito ou o fez de forma totalmente equivocada.
Conceito 1 – Mencionou o quesito, porém não o desenvolveu ou o fez de forma inadequada.
Conceito 2 – Desenvolveu o quesito parcialmente, de forma insuficiente ou com inconsistências.
Conceito 3 – Desenvolveu o quesito adequadamente, de forma completa e correta.

QUESITO 2.2
Conceito 0 – Não citou nem descreveu nenhuma boa prática que deve ser observada para o desenvolvimento seguro de *software*.
Conceito 1 – Limitou-se a citar até duas boas práticas que devem ser observadas para o desenvolvimento seguro de *software*.
Conceito 2 – Limitou-se a citar três ou quatro boas práticas que devem ser observadas para o desenvolvimento seguro de *software*.
Conceito 3 – Citou e descreveu corretamente apenas uma boa prática que deve ser observada para o desenvolvimento seguro de *software*.
Conceito 4 – Citou e descreveu corretamente apenas duas boas práticas que devem ser observadas para o desenvolvimento seguro de *software*.
Conceito 5 – Citou e descreveu corretamente apenas três boas práticas que devem ser observadas para o desenvolvimento seguro de *software*.
Conceito 6 – Citou e descreveu corretamente quatro boas práticas que devem ser observadas para o desenvolvimento seguro de *software*.

QUESITO 2.3
Conceito 0 – Não definiu nenhuma abordagem utilizada para realizar testes de vulnerabilidade no desenvolvimento de sistemas e aplicações.
Conceito 1 – Mencionou uma abordagem utilizada para realizar testes de vulnerabilidade no desenvolvimento de sistemas e aplicações, porém não apresentou sua definição ou a definiu incorretamente.
Conceito 2 – Mencionou duas abordagens utilizadas para realizar testes de vulnerabilidade no desenvolvimento de sistemas e aplicações, porém não apresentou sua definição ou as definiu incorretamente.
Conceito 3 – Definiu corretamente apenas uma abordagem utilizada para realizar testes de vulnerabilidade no desenvolvimento de sistemas e aplicações.
Conceito 4 – Definiu corretamente duas abordagens utilizadas para realizar testes de vulnerabilidade no desenvolvimento de sistemas e aplicações.

QUESITO 2.4
Conceito 0 – Não abordou a origem das informações nem o que o OWASP Top Ten fornece.
Conceito 1 – Abordou corretamente apenas a origem das informações do OWASP Top Ten ou o que ele fornece.
Conceito 2 – Abordou corretamente a origem das informações contidas no OWASP Top Ten e o que ele fornece.

QUESITO 2.5
Conceito 0 – Não citou nem conceituou nenhuma das vulnerabilidades listadas no OWASP Top Ten.
Conceito 1 – Limitou-se a citar até duas vulnerabilidades listadas no OWASP Top Ten.
Conceito 2 – Limitou-se a citar três ou quatro vulnerabilidades listadas no OWASP Top Ten.
Conceito 3 – Citou e conceituou, corretamente, apenas uma das vulnerabilidades listadas no OWASP Top Ten.
Conceito 4 – Citou e conceituou, corretamente, apenas duas das vulnerabilidades listadas no OWASP Top Ten.
Conceito 5 – Citou e conceituou, corretamente, apenas três das vulnerabilidades listadas no OWASP Top Ten.
Conceito 6 – Citou e conceituou, corretamente, quatro vulnerabilidades listadas no OWASP Top Ten.

Cebraspe
Centro Brasileiro de Pesquisa em Avaliação e Seleção e de Promoção de Eventos

MINISTÉRIO DO PLANEJAMENTO E ORÇAMENTO
SECRETARIA DE ORÇAMENTO FEDERAL
SUBSECRETARIA DE TECNOLOGIA E DESENVOLVIMENTO INSTITUCIONAL

CARGO 4: ANALISTA DE PLANEJAMENTO E ORÇAMENTO – ESPECIALIDADE: DESENVOLVIMENTO DE SISTEMAS ORÇAMENTÁRIOS

Prova Discursiva P_4 – Questão

Aplicação: 07/07/2024

PADRÃO DE RESPOSTA DEFINITIVO

O modelo de fundação assemelha-se a uma grande enciclopédia digital que foi lida e compreendida por uma inteligência artificial. Esses modelos são treinados com uma quantidade enorme de dados, abrangendo uma variedade de tópicos muito grande. Eles aprendem padrões e informações gerais que podem ser aplicados a uma ampla gama de tarefas sem a necessidade de grandes ajustes. Isso os torna extremamente flexíveis e poderosos, capazes de entender e gerar linguagem, resolver problemas e até criar arte de modo semelhante ao trabalho humano.

RAG (*retrieval-augmented generation*) é como um assistente inteligente que, ao receber uma pergunta, rapidamente consulta uma biblioteca imensa para buscar a informação mais relevante antes de responder. Ele combina a capacidade de geração de respostas do modelo de fundação com um mecanismo de busca, trazendo informações precisas e atualizadas. Isso é particularmente útil em tarefas que exigem respostas detalhadas e baseadas em evidências, como responder perguntas complexas ou fornecer recomendações detalhadas.

O modelo de fundação customizado é uma adaptação mais personalizada desses grandes modelos enciclopédicos. Caso uma empresa tenha necessidades muito específicas, como entender jargões técnicos de uma área específica como engenharia, tecnologia da informação, medicina, direito, entre outras áreas, ou responder a perguntas sobre leis de patentes, um modelo de fundação customizado pode ser ajustado para se especializar nesses tópicos, proporcionando resultados mais precisos e relevantes para a empresa. Ele é customizado para absorver e refletir o conhecimento e as necessidades específicas de seu treinamento, oferecendo uma ferramenta poderosa e personalizada para tarefas específicas.

Esses modelos não são apenas ferramentas, sendo também considerados recursos de busca contínua para expansão das capacidades humanas por meio da tecnologia.

QUESITOS AVALIADOS

QUESITO 2.1
Conceito 0 – Não abordou o quesito ou o fez de forma totalmente equivocada.
Conceito 1 – Descreveu o modelo de forma apenas superficial, sem desenvolvimento.
Conceito 2 – Desenvolveu uma descrição do modelo de forma parcialmente correta ou insuficiente.
Conceito 3 – Desenvolveu uma descrição do modelo de forma correta e completa.

QUESITO 2.2
Conceito 0 – Não abordou o quesito ou o fez de forma totalmente equivocada.
Conceito 1 – Descreveu o modelo de forma apenas superficial, sem desenvolvimento.
Conceito 2 – Desenvolveu uma descrição do modelo de forma parcialmente correta ou insuficiente.
Conceito 3 – Desenvolveu uma descrição do modelo de forma correta e completa.

QUESITO 2.3
Conceito 0 – Não abordou o quesito ou o fez de forma totalmente equivocada.
Conceito 1 – Descreveu o modelo de forma apenas superficial, sem desenvolvimento.
Conceito 2 – Desenvolveu uma descrição do modelo de forma parcialmente correta ou insuficiente.
Conceito 3 – Desenvolveu uma descrição do modelo de forma correta e completa.