Desenvolvimento de Sistemas

Segurança da informação

A segurança da informação é um tema crucial no desenvolvimento de aplicações *web*, especialmente em um mundo onde a internet é uma parte integrante do cotidiano das pessoas. Para garantir que as informações confidenciais dos usuários sejam protegidas, é necessário implementar medidas de segurança adequadas no processo de desenvolvimento de aplicativos.

Neste conhecimento, serão abordadas algumas das boas práticas de segurança da informação no desenvolvimento de aplicações *web*, incluindo XSS, SSL/HTTPS, tráfego de dados pela *web* e Spring Security.

Segurança da informação na *web*

A segurança na *web* é um conjunto de técnicas e medidas projetadas para proteger aplicativos *web* de ameaças externas. Isso inclui vulnerabilidades, como injeção de SQL, já estudada anteriormente no curso, e outras ameaças das quais os desenvolvedores devem estar cientes.

XSS (*cross-site scripting*)

O XSS é uma vulnerabilidade comum em aplicações *web*, que permite que um invasor injete *scripts* maliciosos em uma página da *web*, o que pode ser usado para roubar informações do usuário, como nomes de usuário e senhas. Existem três tipos de XSS: armazenado, refletido e DOM.

O **XSS armazenado** ocorre quando o invasor injeta o *script* malicioso em um banco de dados ou servidor. O **XSS refletido** ocorre quando o invasor “engana” o usuário para que insira informações em um formulário que contém o script malicioso. O **XSS DOM** ocorre quando o *script* malicioso é executado pelo navegador do usuário, sem a necessidade de interagir com o servidor.

Para evitar o XSS, os desenvolvedores devem validar todas as entradas de usuário e filtrar caracteres especiais. Além disso, as aplicações *web* devem usar o HTTPOnly Cookie, o que significa que os *cookies* só podem ser acessados pelo servidor e não pelo JavaScript. Outra medida preventiva é usar o Content Security Policy (CSP), que restringe quais recursos podem ser carregados em uma página da *web*.

SSL/HTTPS

O **SSL** (*secure socket layer*) e o **HTTPS** (*hypertext transfer protocol secure*) são protocolos de segurança que protegem as comunicações na *web*. O SSL é um protocolo de criptografia que garante que as informações enviadas entre o navegador do usuário e o servidor da *web* estejam criptografadas. O HTTPS é o uso do SSL em conjunto com o HTTP, que é a maneira padrão de comunicar informações na *web*.

Para garantir a segurança do SSL/HTTPS, os desenvolvedores devem implementar certificados SSL em seus servidores da *web*. Esses certificados são emitidos por autoridades de certificação e verificam a identidade do servidor da *web*. Além disso, é importante que os desenvolvedores usem a versão mais recente do SSL (atualmente TLS 1.3) e configurem o servidor da *web* corretamente para evitar vulnerabilidades, como o ataque POODLE (Padding Oracle On Downgraded Legacy Encryption).

Configurar SSL/HTTPS para uma aplicação *web* envolve a aquisição de um certificado SSL de uma autoridade de certificação confiável e a configuração do servidor *web* para usá-lo. O processo exato de configuração varia dependendo do servidor *web* específico e do sistema operacional que está sendo usado. Em seguida você terá uma visão geral do processo.

Aquisição do certificado SSL

O primeiro passo para configurar o SSL/HTTPS é adquirir um certificado SSL de uma autoridade de certificação confiável que seja válida e atualizada. Existem muitas autoridades de certificação que oferecem certificados SSL, como Let's Encrypt, DigiCert e Comodo.

Configuração do servidor *web*

Depois de adquirir o certificado SSL, é necessário configurar o servidor *web* para usá-lo. O processo exato de configuração varia dependendo do servidor *web* que está sendo usado. A seguir estão os passos gerais de configuração para alguns dos servidores *web* mais populares:

- Apache

  O Apache é um dos servidores *web* mais populares. Para configurar o SSL/HTTPS no Apache é necessário habilitar o módulo SSL e configurar o arquivo de configuração do Apache. O arquivo de configuração do Apache geralmente é chamado de "httpd.conf" ou "apache2.conf". As seguintes linhas devem ser adicionadas ao arquivo de configuração para habilitar o SSL:

```
LoadModule ssl_module modules/mod_ssl.so Listen 443
```

  Em seguida, as seguintes linhas devem ser adicionadas ao arquivo de configuração para configurar o SSL:

```
<VirtualHost *:443> SSLEngine on SSLCertificateFile /path/t
o/your_domain_name.crt SSLCertificateKeyFile /path/to/your_p
rivate.key </VirtualHost>
```

- Tomcat

  O Tomcat é um servidor *web* de código aberto usado para aplicações Java Web. Para configurar o SSL/HTTPS no Tomcat é necessário gerar um certificado e configurar o arquivo de configuração "server.xml" pelos seguintes passos:

  1. Gere um certificado usando ferramentas, como o *keytool*, ou terceiros, como o Let’s Encrypt.
  2. No arquivo "server.xml", busque o elemento **<Connector>**, que se refere ao protocolo HTTP.

3. Adicione um novo elemento **<Connector>** com os atributos **protocol** (geralmente **HTTP/1.1**), **port** (geralmente **443**), **scheme** (geralmente **https**) e o caminho para o certificado e a chave privada.
4. Certifique-se de que o atributo **secure** esteja definido como **true** para o novo elemento **<Connector>**.
5. Salve as alterações no arquivo "server.xml" e reinicie o servidor

Para garantir que seus aplicativos *web* funcionem corretamente com SSL/HTTPS, é importante verificar se eles estão usando URLs seguras (https://) em vez de URLs não seguras (http://) para acessar recursos protegidos. Além disso, se seus aplicativos web usam *cookies*, é necessário definir o atributo **secure** como **true** para garantir que os *cookies* sejam enviados apenas por conexões SSL/HTTPS seguras.

Também é importante assegurar-se de que o certificado usado seja confiável e válido para garantir a segurança das conexões SSL/HTTPS.

- GlassFish

O GlassFish também é usado para Java Web e, para configurar o SSL/HTTPS nesse servidor *web*, é necessário gerar um certificado e configurar seu arquivo de configuração.

Depois de gerar o certificado, a configuração do SSL/HTTPS no GlassFish pode ser feita por meio da interface de administração (acessando https://localhost:4848) ou por meio do arquivo de configuração "domain.xml".

Para configurar o SSL/HTTPS por meio do arquivo de configuração "domain.xml", siga estas etapas:

1. Abra o arquivo "domain.xml" em um editor de texto.
2. Procure o elemento **<network-listeners>**.
3. Adicione um novo elemento **<network-listeners>** com os atributos **protocol** (**HTTPS**), **port** (geralmente **443**) e o caminho para o certificado e a chave privada.

Depois de configurar o SSL/HTTPS no GlassFish, reinicie o servidor para que as alterações entrem em vigor.

Renovação do certificado SSL

Os certificados SSL têm uma validade limitada e precisam ser renovados regularmente. A maioria das autoridades de certificação oferece certificados com validade de um ano ou mais. É importante renovar o certificado SSL antes de expirar para garantir que a segurança da aplicação web não seja comprometida.

Tráfego de dados pela *web*

Os dados enviados e recebidos pela *web* podem ser interceptados por invasores. Para garantir a segurança desses dados, os desenvolvedores devem criptografá-los antes de enviá-los. A criptografia é o processo de transformar dados legíveis em um formato ilegível que só pode ser decodificado por uma chave de criptografia.

Existem duas maneiras de criptografar os dados: criptografia simétrica e criptografia assimétrica. A criptografia simétrica usa uma única chave para criptografar e descriptografar os dados. A criptografia assimétrica usa uma chave pública para criptografar os dados e uma chave privada para descriptografá-los.

É preferível usar a criptografia assimétrica, também conhecida como criptografia de chave pública, em que a chave pública pode ser compartilhada com qualquer pessoa, enquanto a chave privada deve ser mantida em segredo pelo proprietário. Esse método de criptografia é amplamente utilizado em projetos de segurança da informação, incluindo projetos Java Web.

Para implementar a criptografia assimétrica em projetos Java Web, é necessário seguir os seguintes passos:

Gerar um par de chaves pública/privada

O primeiro passo é gerar um par de chaves pública/privada. Isso pode ser feito usando a biblioteca de criptografia Java KeyStore (JKS) e a ferramenta de linha de comando *keytool*. O seguinte comando pode ser usado para gerar um novo par de chaves:

```
keytool -genkeypair -alias mykey -keyalg RSA -keysize 2048 -keystore mykeystore.jks
```

Esse comando gera um novo par de chaves RSA com tamanho de 2048 *bits* e armazena-o no arquivo mykeystore.jks. A chave privada é armazenada no arquivo de *keystore*, enquanto a chave pública pode ser exportada para uso posterior.

Exportar a chave pública

Para usar a chave pública em uma aplicação Java Web, é necessário exportá-la em um formato apropriado. Isso pode ser feito usando a ferramenta *keytool* da seguinte forma.

```
keytool -exportcert -alias mykey -keystore mykeystore.jks -file mykey.cer
```

Esse comando exporta a chave pública para o arquivo mykey.cer no formato X.509, que pode ser facilmente lido por aplicativos Java.

Importar a chave pública

Depois de exportar a chave pública, ela pode ser importada para o aplicativo Java Web. Isso pode ser feito usando a classe **java.security.KeyStore**, que permite acessar o arquivo de *keystore* e recuperar a chave pública.

```
KeyStore keystore = KeyStore.getInstance("JKS");
keystore.load(new FileInputStream("mykeystore.jks"),
"mystorepassword".toCharArray());
PublicKey publicKey = keystore.getCertificate("mykey").getPublicKey();
```

Esse código carrega o arquivo de *keystore* mykeystore.jks e obtém a chave pública com o alias *mykey*. A chave pública pode ser usada para criptografar dados que só podem ser descriptografados pela chave privada correspondente.

Criptografar dados com a chave pública

Para criptografar dados com a chave pública, a classe **javax.crypto.Cipher** pode ser usada. Essa classe permite criptografar dados com a chave pública e descriptografá-los com a chave privada correspondente.

```
byte[] data = "Dados a serem criptografados".getBytes("UTF-8");
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.ENCRYPT_MODE, publicKey);
byte[] encryptedData = cipher.doFinal(data);
```

Esse código criptografa a *string* "Dados a serem criptografados" com a chave pública e armazena o resultado em **encryptedData**.

Descriptografar dados com a chave privada

Para descriptografar os dados criptografados com a chave pública, a classe **javax.crypto.Cipher** pode ser usada novamente, dessa vez no modo de descriptografia.

```
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.DECRYPT_MODE, privateKey);
byte[] decryptedData = cipher.doFinal(encryptedData);
String originalData = new String(decryptedData, "UTF-8");
```

Esse código descriptografa os dados criptografados em **encryptedData** usando a chave privada e armazena o resultado na variável *string* **originalData**.

Armazenar a chave privada de forma segura

A chave privada deve ser armazenada de forma segura, de preferência em um arquivo protegido por senha. A classe **java.security.KeyStore** pode ser usada para criar um arquivo de *keystore* protegido por senha e armazenar a chave privada nele.

```
KeyStore keystore = KeyStore.getInstance("JKS");
keystore.load(null, null);
PrivateKey privateKey = /* obter a chave privada */;
Certificate[] chain = /* obter a cadeia de certificados */;
keystore.setKeyEntry("mykey", privateKey, "mypassword".toCharArray(), chain);
keystore.store(new FileOutputStream("mykeystore.jks"), "mystorepassword".toCharArray());
```

Esse código cria um novo arquivo de *keystore* e armazena a chave privada com o alias *mykey* e a senha **mypassword**. A cadeia de certificados é armazenada junto à chave privada para permitir a verificação da identidade do proprietário da chave.

Em resumo, a criptografia assimétrica é um método seguro para proteger dados em projetos Java Web. Ao gerar um par de chaves pública/privada, exportar a chave pública, importá-la no aplicativo, criptografar e descriptografar dados e armazenar a chave privada de forma segura, os desenvolvedores podem garantir que os dados sensíveis estejam protegidos contra acesso não autorizado.

Spring Security

O Spring Security é um *framework* de segurança de código aberto que fornece uma ampla gama de recursos de segurança para aplicações Java. Ele fornece uma camada de segurança em nível de aplicação que protege as aplicações contra ameaças externas, como XSS e CSRF.

O Spring Security fornece recursos, como autenticação e autorização de usuários, gerenciamento de sessões de usuário e prevenção de ataques de força bruta. Ele também suporta a integração com outros *frameworks*, como Spring MVC e Spring Boot.

Ao usar o Spring Security, os desenvolvedores devem seguir as melhores práticas de segurança, como o uso de senhas fortes e a criptografia de dados confidenciais. Também devem monitorar regularmente os *logs* de segurança para detectar atividades suspeitas.

Spring Security em aplicações Java

O Spring Security é um *framework* de segurança amplamente utilizado para proteger aplicações Java. Ele oferece uma ampla gama de recursos e funcionalidades para autenticação (verificação de identidade) e autorização (controle de acesso).

A implementação do Spring Security em aplicações Java envolve a configuração de vários componentes-chave:

Configuração de dependências: antes de tudo, como qualquer outra dependência, é necessário incluir o Spring Security como dependência em um projeto por meio do Maven ou Gradle.

Configuração do contexto de segurança: com Spring Security, é possível criar uma classe (geralmente herdeira de **WebSecurityConfigurereAdapter** e com anotação **@EnableWebSecurity**), na qual se configuram aspectos como regras de autenticação, autorização, gerenciamento de sessões e configurações específicas do aplicativo.

Autenticação: o Spring Security fornece várias opções para autenticação de usuários, como autenticação baseada em banco de dados, autenticação baseada em formulário, autenticação com OAuth (um protocolo de autorização que permite que os usuários concedam acesso a recursos protegidos a aplicativos de terceiros sem compartilhar suas credenciais de *login*), entre outras. Você pode configurar um provedor de autenticação personalizado ou usar um provedor fornecido pelo Spring Security.

Autorização: depois que um usuário é autenticado, o Spring Security permite que você defina regras de autorização para controlar quais recursos e funcionalidades ele pode acessar. Você pode configurar permissões e papéis (*roles*) para diferentes partes do aplicativo, como URLs, APIs, métodos de serviço etc. Isso é feito usando anotações, expressões de segurança ou configurações específicas.

Proteção contra ataques: o Spring Security também oferece recursos de proteção contra ataques comuns, como ataques de falsificação de solicitação entre *sites* (CSRF), ataques de injeção (SQL, XSS), ataques de força bruta e muito mais. Ele inclui mecanismos de segurança integrados para lidar com esses ataques e ajuda a fortalecer a segurança da sua aplicação.

Esses são apenas alguns dos aspectos básicos da implementação do Spring Security em aplicações Java. O Spring Security oferece uma ampla gama de recursos adicionais, como autenticação de dois fatores, auditoria, proteção de API, integração com provedores

de identidade externos e muito mais. A documentação oficial do Spring Security é uma ótima fonte de referência para explorar esses recursos com mais detalhes.

Boas práticas no desenvolvimento de aplicações *web*

Para garantir a segurança da aplicação *web*, deve-se seguir algumas boas práticas, tais como:

Utilizar *frameworks* e bibliotecas seguras: ao utilizar *frameworks* e bibliotecas de terceiros, é importante escolher aquelas que são seguras e estão em conformidade com os padrões de segurança. Além disso, é necessário manter as versões atualizadas, pois, muitas vezes, as atualizações corrigem falhas de segurança.
Validar dados de entrada: é importante validar todos os dados de entrada fornecidos pelos usuários, como formulários de *login* e cadastro. Isso ajuda a evitar ataques de injeção de SQL e outras vulnerabilidades.
Utilizar autenticação e autorização: a autenticação e autorização ajudam a garantir que apenas usuários autorizados tenham acesso a recursos protegidos. É importante utilizar métodos de autenticação seguros, como o OAuth, para evitar a divulgação de senhas e de outros dados confidenciais.
Implementar medidas de segurança em camadas: é importante implementar medidas de segurança em camadas, como *firewalls* e sistemas de detecção de intrusos, para garantir que a aplicação esteja protegida em todas as etapas do processo.
Utilizar certificado HTTPS: como mencionado anteriormente, o certificado HTTPS é fundamental para garantir a segurança das informações transmitidas entre o navegador do usuário e o servidor da aplicação. É importante certificar-se de que o certificado está válido e configurado corretamente.

Seguindo essas boas práticas, os desenvolvedores podem garantir que as aplicações *web* sejam seguras e confiáveis. A adoção de certificado HTTPS é uma das medidas mais importantes que podem ser tomadas para proteger a privacidade e a segurança dos usuários. Por isso, é fundamental que os desenvolvedores sejam proativos em relação à segurança de suas aplicações *web* e utilizem as melhores práticas disponíveis.

Encerramento

As boas práticas de segurança mencionadas neste texto são fundamentais para garantir que as aplicações *web* sejam seguras e confiáveis. A adoção de certificado HTTPS é uma das medidas mais importantes que podem ser tomadas para proteger a privacidade e a segurança dos usuários. Além disso, é importante validar os dados de entrada, utilizar autenticação e autorização, implementar medidas de segurança em camadas e utilizar

frameworks e bibliotecas seguras. Também é fundamental lembrar, no entanto, que a segurança da informação é um processo contínuo e deve ser tratado como tal tanto por desenvolvedores quanto por usuários, que precisam estar cientes de ações relativas à segurança da informação e ao bom uso de sistemas *web*.