Relatório Técnico de Pentesting – Parte 2

SQL Injection – Ataque e Prevenção

1. Implementação do Ataque

Para testar a vulnerabilidade SQL Injection no endpoint '/users/login', utilizei a seguinte carga maliciosa no campo de email:

' OR '1'='1

Essa entrada maliciosa força a condição da cláusula WHERE a ser sempre verdadeira, permitindo acesso indevido ao sistema.

Passos:

1. Iniciar o servidor com `node app.js`

2. Utilizar um cliente HTTP (como Postman ou curl) para enviar uma requisição POST para http://localhost:3000/users/login

3. Enviar os seguintes dados no body:

{ "email": "' OR '1'='1", "password": "qualquer" }

4. Observar que o login foi aceito sem autenticação válida.

2. Ações para Correção e Prevenção

• Usar consultas parametrizadas: evitar concatenar diretamente os dados do usuário na string SQL.

• Validar entradas: garantir que o formato dos dados (email, senha) esteja correto.

• Limitar permissões: o usuário do banco de dados usado pela aplicação deve ter apenas os acessos necessários.

Cross-Site Scripting (XSS) – Ataque e Prevenção

1. Implementação do Ataque

1. Em um campo de formulário de entrada de dados (ex: comentários), injete o seguinte código:

<script>alert("XSS")</script>

2. Ao enviar o formulário, o navegador executará o código JavaScript, mostrando um alerta.

Esse tipo de ataque pode ser usado para roubar cookies, redirecionar usuários, etc.

2. Ações para Correção e Prevenção

• Sanitize: remova ou escape tags HTML perigosas das entradas do usuário.

• Validar: certifique-se que campos só aceitem textos válidos (sem tags).

• Cabeçalhos de segurança: implemente Content Security Policy (CSP).

• Cookies com HTTPOnly: para evitar acesso via JavaScript.

Cross-Site Request Forgery (CSRF) – Ataque e Prevenção

1. Implementação do Ataque

1. Criar uma página HTML com um formulário oculto que envia uma requisição POST para o backend da aplicação:

```
<form action="http://localhost:3000/users/1" method="POST">
  <input type="hidden" name="email" value="hacker@exemplo.com">
  <input type="hidden" name="name" value="Hacker">
  <input type="submit" value="Enviar">
</form>
<script>document.forms[0].submit();</script>
```

2. Hospedar essa página maliciosa.

3. Convencer um usuário autenticado a acessá-la.

4. O navegador enviará automaticamente os cookies de sessão e executará a ação sem o consentimento do usuário.

2. Ações para Correção e Prevenção

• Usar tokens CSRF: gere um token exclusivo por sessão e exija que ele seja enviado com requisições POST/PUT/DELETE.

• Exemplo de proteção com pacote 'csurf' no Express:

```
const csrf = require('csurf');
app.use(csrf({ cookie: true }));
// No formulário HTML inclua o token:
<input type="hidden" name="_csrf" value="{{csrfToken}}">
```

• Verificar esse token no servidor antes de aceitar a requisição.

• Exigir autenticação em ações críticas e revalidação de senha para mudanças sensíveis.

Conclusão

Este relatório demonstrou, de forma prática e técnica, como explorar e corrigir três das principais vulnerabilidades web: SQL Injection, XSS e CSRF. Com a implementação de boas práticas de segurança e bibliotecas adequadas, é possível proteger aplicações Node.js contra ataques comuns e fortalecer a segurança geral da aplicação.