Curso de Extensão em Tecnologias Microsoft

INF0998 Programação segura (segurança de software)

Atividades práticas de exploração manual de vulnerabilidades no WebGoat, bWAPP e Juice Shop

A maior parte das vulnerabilidades exploradas em aplicações reais são facilmente detectadas por desenvolvedores e testadores capacitados. Pautando-se nas aplicações vulneráveis para aprendizado que foram mostradas na aula anterior, o objetivo desta aula é mostrar aos alunos como verificar manualmente (sem ferramentas automáticas) a ocorrência de vulnerabilidades simples e que não requerem técnicas sofisticadas.

Nesta aula usará principalmente as aplicações vulneráveis WebGoat 7.1, bWAPP e Juice Shop para ilustrar a realização de testes de segurança manuais.

O playbook começa mostrando a vulnerabilidade de API Broken Object Level Authorization e continua com outras vulnerabilidades exploradas: Injeção de SQL (SQL Injection - SQLi), Cross-Site Script (XSS), injeção de comando (Command Injection) e acesso de arquivos por path traversal.

Introdução ao OWASP API Security

A documentação do OWASP API Security Project pode ser acessada pelo link a seguir
https://owasp.org/www-project-api-security

Descobrindo a API da aplicação Juice Shop

Na tela/página inicial da aplicação Juice Shop no browser.

- Ferramentas do desenvolvedor → Sources
- Visualizar o arquivo main-es2015.js
    - Usar ***Pretty printing*** ou botão ***{}***
- Buscar (menu três pontos verticais → buscar) pela palavra *path.*
    - O resultado da busca é toda a API da aplicação!
- Identificar a API do dashboard de desafios e acessá-la.
    - Spoiler alert! https://try-juice-shop.herokuapp.com/#/score-board

API 1: 2019 Broken Object Level Authorization

Na Juice Shop → desafio ***View another user's shopping basket***
O objetivo é visualizar a cesta de compras de um usuário, qualquer um, sem estar logado como ele.

- Criar dois usuários. Por exemplo, user1/user1 e user2/user2
- Encher a cesta do user2 com diversos produtos do juice shop.
- Descobrir e explorar a API da cesta de compras
    - Exploração 1
        - abrir as ferramentas de desenvolvedor
        - Entrar o App Juice Shop logado como user1
        - acessar a cesta de compras do user1
        - em DevTools, em Network, observar que a API REST da cesta é muito simples
            - P.ex., https://try-juice-shop.herokuapp.com/rest/basket/7
            - O que é o número 7?
        - em DevTools, Application → Session Storage
            - aparecem dois elementos bid = 7 e ItemTotal = 0.
                - será o mesmo número 7?
        - Conclusão deduzida: bid quer dizer BasketID
        - Alterar bid para algum valor qualquer e atualizar a página.
            - por exemplo, o valor bid = 6 !
            - o que acontece?
        - Spoiler Alert!
            - Aparece a cesta de compras do user2.

Testes de SQL Injection - Aprendendo o que é o SQLi

SQLi no WebGoat 7.1

- Entrar no seu deploy do WebGoat http://localhost:8080/WebGoat/
- Logar na aplicação como usuário/senha: guest/guest

- SQL Injection em campos tipo texto
    - Selecionar a opção de menu: **Injection Flaws → String SQL Injection**
    - Testar o campo texto com “Snow” e/ou “Smith” e observar o resultado
    - Será que o campo aceita caracteres não numéricos?
        - Testar com aspas simples (‘). O que acontece? Qual o erro?
    - É possível concatenar partes de uma SQL query e modificar o comando?
    - Inserir no campo texto o seguinte: `' or 'a'='a'`. Clicar no botão Go!
    - Por que não funciona? O que deu errado?
    - Testar novamente com `' or 'a'='a' --`

- SQL Injection em campos numéricos
    - Selecione a opção de menu **Injection Flaws → Numeric SQL Injection**
    - Testar o campo da lista de seleção com as opções disponíveis. O que acontece?
    - Abrir as ferramentas de desenvolvedor do seu navegador.
    - Ativar ferramenta de seleção (ponteiro do mouse) e selecionar o campo de lista.
    - Editar o valor `101` de Columbia para `101 or 1=1`. Ativar o botão Go!
    - Qual a consulta executada? Qual o resultado da consulta?

SQLi no bWAPP

SQLi para fazer buscas em uma base de dados.

- Selecionar da lista de vulnerabilidades o bug “***SQL Injection (Search/GET)***”.
- No campo Search for a movie, digitar duas aspas separadas por espaço (`' '`).
- Clicar no botão search. O conteúdo completo da base de dados é mostrado.
- A URL de teste é http://localhost/bWAPP/sqli_1.php?title='+'&action=search
- Testar com `iron'+or+1=1#`. O que acontece?
    - http://localhost/bWAPP/sqli_1.php?title=iron'+or+1=1#&action=search

Neste exercício, o SQLi é usado para Injeção escalação de privilégio e acesso não autorizado, quando um usuário ilegítimo consegue logar na aplicação como administrador:

- Selecionar da lista de vulnerabilidades o bug “***SQL Injection (Login Form)***”.
- Verificar se a página aceita o caracter de aspas simples “ ' “ no campo de login.

- Testar a string de injeção `' or 1=1#` no campo de login. O que acontece?
- Testar com '`alice`' no campo de login e `' or 1=1#` no campo de senha. O que acontece?

SQLi no Juice Shop

Neste exercício, o SQLi é usado para Injeção escalação de privilégio e acesso não autorizado, quando um usuário ilegítimo consegue logar na aplicação como administrador:

- Acessar a página de login da aplicação
- Verificar se a página aceita o caracter de aspas simples " ' "
- Testar a string de injeção `' or 'a' = 'a' --`.

Testes de Cross-Site Script (XSS) - Aprendendo o que é XSS

XSS no WebGoat 7.1

- Entrar no seu deploy do WebGoat http://localhost:8080/WebGoat/
- Logar na aplicação como usuário/senha: guest/guest

Há dois tipos de XSS, o refletido e o armazenado. No XSS refletido, o script injetado vai até o servidor e volta para o mesmo usuário. Já no XSS Armazenado, o script injetado fica armazenado no servidor e, no futuro, pode voltar para outro usuário diferente ou o mesmo usuário.

XSS Refletido

- Selecionar a opção de menu: **Cross-Site Scripting (XSS) → Reflected XSS Attacks**
- Explorar a funcionalidade de Shopping Cart.
- Qual campo devolve mensagem de erro para o usuário se contiver valor inválido?
    - Dica: um campo com restrição de formato ou tamanho de dados
- O campo com o código 111 aceita javascript? Testar com: **`<script>alert(1)<\scritp>`**
- O que aconteceu com o script injetado?
- Funciona com esse payload malicioso `<a onmouseover="alert(1)" href="#">read this!</a>` ?

XSS Armazenado

- Selecionar a opção de menu: **Cross-Site Scripting (XSS) → Stored XSS Attacks**.
- Explorar a funcionalidade do quadro de avisos e o formulário de mensagens.
- Qual campo da mensagem aceita caracteres especiais?
- O campo de mensagem aceita tags HTML? Testar com: **`<h1>`**
- O campo de mensagem aceita javascript? Testar com: **`<script>alert(1)<\scritp>`**
- Como o script injetado é ativado?

XSS no Juice Shop

Um passo-a-passo dessa exploração pode ser o seguinte:

- Entrar na aplicação Juice Shop
- Logar na aplicação com usuário e senha (criar usuário, se necessário)
- Desafio: descobrir a URL que lista todos os desafios de hacking da aplicação
    - Dica: ver os nomes das páginas da aplicação e testar por variações de “Score Board”
- `Escolher o desafio DOM XSS da lista de desafios simples (uma estrela ⭐)`
    - Este desafio explora vulnerabilidades no campo de busca 🔍
    - Fazer uma busca por um nome de fruta (em inglês), O que acontece?
    - O campo de busca exibe de volta para o usuário as palavras buscadas!
- O campo de busca aceita tags HTML? Testar com: **`<h1> orange`**
- O campo de busca aceita javascript? Testar com: **`<script>alert(1)<\scritp>`**
- A busca aceita DOM? Testar com: **`<iframe src="javascript:alert(`xss`)">`**
- Quais outros objetos DOM podem ser incluídos?

XSS no bWAPP

XSS ocorre sempre que a aplicação devolve sem filtrar o que o usuário digita.

- Selecionar da lista de vulnerabilidades o bug “***XSS - Reflected (GET)***”.
- No campo First name, digitar <script>alert(1).
- No campo Last Name, digitar </script>. Clicar no botão Go!
- O conteúdo completo do XSS pode ser visto na url.
- `http://localhost/bWAPP/xss_get.php?firstname=<script>alert(1)&lastname=</script>&form=submit`
- Testar com outras maneiras de entrar o mesmo script `<script>alert(1)</script>`

Testes de Command Injection

No WebGoat 7.1

- Entrar no seu deploy do WebGoat http://localhost:8080/WebGoat/
- Logar na aplicação como usuário/senha: guest/guest

- Selecionar a opção de menu **Injection Flaws → Command Injection**
- Testar a funcionalidade de seleção e visualização de arquivos
    - Seleção de um arquivo em uma lista de opções para visualização (botão *View*)
- Como o arquivo é carregado?
- Qual comando é executado pelo sistema operacional?
- Como executar comandos dir, ou ls, ou ipconfig, ou ifconfig ou netstat?
    - Dica: anexar mais comandos ao final da linha de comando do cmd.exe
- Abrir as ferramentas de desenvolvedor do seu navegador.
- Ativar ferramenta de seleção (ponteiro do mouse) e selecionar a lista de seleção.
- Editar o primeiro item da lista `AccessControlMatrix.help`
- Fazer diversos testes com injeção de vários comandos:
    - Anexar `" & dir .` e ativar o botão View.
    - Anexar `" & ipconfig` e ativar o botão View.
    - Anexar `" & netstat -a` e ativar o botão View.
    - Anexar `" & netstat -a & ipconfig` e ativar o botão View.
- O que aconteceu em cada caso?
- O que acontece se outro comando diferente for digitado?

- Selecionar a opção de menu **Ajax Security → Dangerous Use of Eval**
- Explorar a funcionalidade de Shopping Cart.
- Qual campo devolve mensagem de erro para o usuário se contiver valor inválido?
    - Dica: um campo com restrição de formato ou tamanho de dados
- O campo com o código 123 aceita comandos?
- Testar com: `<script>alert(1)<\scritp>.` O que aconteceu?
- Testar com: `123'); alert(1);(' .` O que aconteceu?

No bWAPP

Injeção de comando do sistema operacional pelo EVAL.

- Selecionar da lista de vulnerabilidades o bug "***PHP Eval Function***".
- Testar com http://localhost/bWAPP/php_eval.php?eval=echo shell_exec("ls -l");
- Testar http://localhost/bWAPP/php_eval.php?eval=echo shell_exec("cat /etc/passwd");

Testes de acesso de arquivo não autorizado

Esta vulnerabilidade é bastante comum e já aconteceu na Invillia recentemente!

No Juice Shop

Um passo-a-passo dessa exploração pode ser o seguinte:

- Acessar a página “About us”.
- Clicar no link dos termos de uso.
- Como o arquivo é acessado? Por uma pasta de ftp!
- Acessar a pasta ftp diretamente e tentar acessar os arquivos.

No bWAPP

Directory Traversal de pasta e de arquivos no PHP.

- Selecionar da lista de vulnerabilidades o bug “***Directory Traversal - Directories***”.
- Clicar em um nome de arquivo da lista apresentada.
- Na aba de visualização do arquivo, observar a pasta “documents” na url.
    - http://localhost/bWAPP/documents
    - **(Se não for locahost, usar o IP do seu bWAPP na url !!!!!!!!!)**
- Acessar a pasta “documents” diretamente pelo navegador. O que acontece?

- Selecionar da lista de vulnerabilidades o bug “***Directory Traversal - Files***”.
- Observar o nome de arquivo na url
    - http://localhost/bWAPP/directory_traversal_1.php?page=message.txt
    - **(Se não for locahost, usar o IP do seu bWAPP na url !!!!!!!!!)**
- Digitar o nome de outros arquivos com o caminho relativo.
- http://localhost/bWAPP/directory_traversal_1.php?page=../../../../../../../../../etc/passwd
- O que acontece?