SQL Injection

O que é SQL Injection?

- O SQL Injection é a primeira mais comum vulnerabilidade em aplicações web, de acordo com o Open Web Application Security Project.

- É um ataque contra um banco de dados de uma empresa via website.

- Nesse ataque, os crackers executam comandos não autorizados de SQL ao aproveitar sistemas inseguros que estão conectados na Internet.

O que faz um cracker?

- Um cracker injeta a query SQL na aplicação usando um navegador web comum.
- O objetivo é injetar linguagem SQL maliciosa dentro do que a aplicação usa para fazer query no banco de dados.
- Tudo o que o criminoso precisa é de “um navegador web, conhecimento em queries SQL e um trabalho criativo de adivinhação para saber nomes de tabelas e de campos”.

Quando o banco de dados é invadido...

- ... o cracker pode ler dados sensíveis do banco de dados, modificá-los.

- Além disso, ele pode realizar operações como o fechamento do BD e potencialmente enviar comandos diretamente para o sistema operacional.

Funcionamento

- Para exemplificar o funcionamento da injeção de SQL, consideremos uma instrução SQL comum:

- `SELECT id, nome, sobrenome FROM autores;`

- Uma consulta na base de dados, retorna todos os registros das colunas "id", "nome" e "sobrenome" da tabela "autores".

Funcionamento

- Com base nesta instrução, é fácil supor que "josé" e "silva" são strings, cujo conteúdo será preenchido pela entrada feita por algum usuário que estiver fazendo uso da aplicação.

Funcionamento

- Portanto, supondo que a aplicação não faça o tratamento apropriado do conteúdo inserido pelo usuário, o mesmo pode fazer o uso acidental do caractere de aspas simples.

- Mas, gerando a entrada:
  - `nome = jo'sé`
  - `sobrenome = silva`

Funcionamento

- E fazendo com que a aplicação gere o código:

- ```
  SELECT id, nome, sobrenome FROM
  autores WHERE nome = 'jo'sé' AND
  sobrenome = 'silva';
  ```
  ```

Funcionamento

- De acordo com a especificação da linguagem SQL, existe um erro de sintaxe nessa instrução, uma vez que a string passada para o campo **nome** é a apenas palavra "jo", pois a adição das aspas simples quebrou a delimitação das aspas simples originais da consulta.

Funcionamento

- O interpretador do SQL espera que o restante da instrução seja outros comandos SQL válidos que complementem a instrução principal.

- No entanto, como "sé" não é um identificador válido, essa instrução não será executada e retornará um erro.

O Ataque

- Com base neste problema, um possível atacante pode manipular os dados de entrada a fim de gerar um comportamento não esperado na base de dados.

O Ataque

- Para exemplificar este conceito, consideremos na mesma consulta apresentada, a entrada dos seguintes dados pela aplicação:

```
nome = jo'; DROP TABLE autores ; --
sobrenome = silva
```

O Ataque

- Fazendo com que a aplicação gere o código:

```
SELECT id, nome, sobrenome FROM
autores WHERE nome = 'jo'; DROP
TABLE autores ; --' AND sobrenome =
'silva';
```

O Ataque

- Neste caso, a instrução será executada normalmente, pois não há um erro de sintaxe, no entanto, com a adição do caractere ponto-e-vírgula, a instrução foi dada como finalizada de modo prematuro dando espaço para uma nova instrução.

O Ataque

- Essa nova instrução, que poderia ser qualquer uma escolhida pelo atacante:

  1) pode ser a responsável por retornar dados confidenciais armazenados na base de dados ou;

  2) de executar instruções que comprometam o sistema, como a remoção de dados e/ou tabelas.

As consequências do ataque

- Dados críticos podem ser modificados ou enviados para outros computadores longe da empresa.

- Os crackers também podem usar as SQL Injection para conectar nos sistemas corporativos como se fosse um usuário autorizado, driblando a necessidade de uma senha.

Exemplo:

```
<body>
  <form action='HelloInjection'>
  <input type='text' name='emailAddress' value=''/>
  <input type='submit' value='Submit'/>
</form>
</body>
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
  PrintWriter out = response.getWriter();
  Connection conn = null;
  try {
    Class.forName ("oracle.jdbc.driver.OracleDriver");
    String url = "jdbc:oracle:thin:@//db_server:1521/service_name";
    String userName = "XXXX";
    String password = "XXXX";
    conn = DriverManager.getConnection(url, userName, password);
    String email = request.getParameter("emailAddress");
    Statement st = conn.createStatement();
    String query= "select * from email_subscriptions where email_address = '" + email + "'";
    out.println("Query : " + query + "<br/>");
    ResultSet res = st.executeQuery(query);
     out.println("Results:" + "<br/>");
    while (res.next()) {
   String s = res.getString("email_address");
  out.println(s + "<br/>");
    }    } catch (Exception e) {   e.printStackTrace();   }   }
```

Exemplo:

- A cláusula SQL:

String query= "select * from usuarios where email_address = '" + email + "'";

Transforma tudo que o usuário colocar em String.

Exemplo: se colocarmos ilorivero@live.com, a cláusula fica:

select * from usuarios where email_address = ‘ilorivero@live.com'

Exemplo:

- Podemos colocar qualquer coisa entre as aspas.
- Permite que adicionemos qualquer cláusula adicional simplesmente fechando as aspas.
- É possível adicionar o seguinte: ' or 1='1
- O que transforma a query em:

```
select * from usuarios where email_address = '' or 1='1'
```

Exemplo:

- É possível juntar queries através de Union…

- E se não souber os nomes das tabelas? Use:

' union all select table_name from all_tables where 1 = '1

- E a query se transforma em:

```
select * from usuarios where email_address = ''
      union all
      select table_name from all_tables where 1 = '1
```

Exemplo:

- Usando PreparedStatement
  - Para evitar SQL Injection, altere o Statement para um PreparedStatement.
  - Em seguida, modifique a cadeia de consulta para ter um ponto de interrogação (?), ao invés de concatenar o endereço de email para ela.
  - Por fim, defina o valor da variável para o endereço de e-mail fornecido.

Exemplo:

```
String query= "select * from usuarios where email_address = ?";
PreparedStatement st = conn.prepareStatement(query);
 st.setString(1, email);
ResultSet res = st.executeQuery();
```

- Independentemente do que o usuário digitar no campo do formulário, a string será sempre:

```
select * from email_subscriptions where email_address = ?
```

- Tentativas de hacking como ' or 1='1 não irão retornar resultados.

Tentando sair do problema

- Aparentemente um método para prevenir esse problema seria a remoção de aspas simples dos campos de inserção da aplicação, ou simplesmente não executando a *query* nestas situações.

Tentando sair do problema

- Isso é verdade, mas existem várias dificuldades com esse método tanto quanto soluções.

- Primeiro, nem todos os usuários inserem dados em forma de strings.

Tentando sair do problema

- Se o usuário puder selecionar um autor pelo 'id' (presumivelmente um número) por exemplo, nossa *query* aparecerá como abaixo:

Tentando sair do problema

- **SELECT** id, forename, surname **FROM** authors **WHERE** id=1234

- Nesta situação, o atacante pode simplesmente adicionar uma instrução SQL no fim do 'input' numérico.

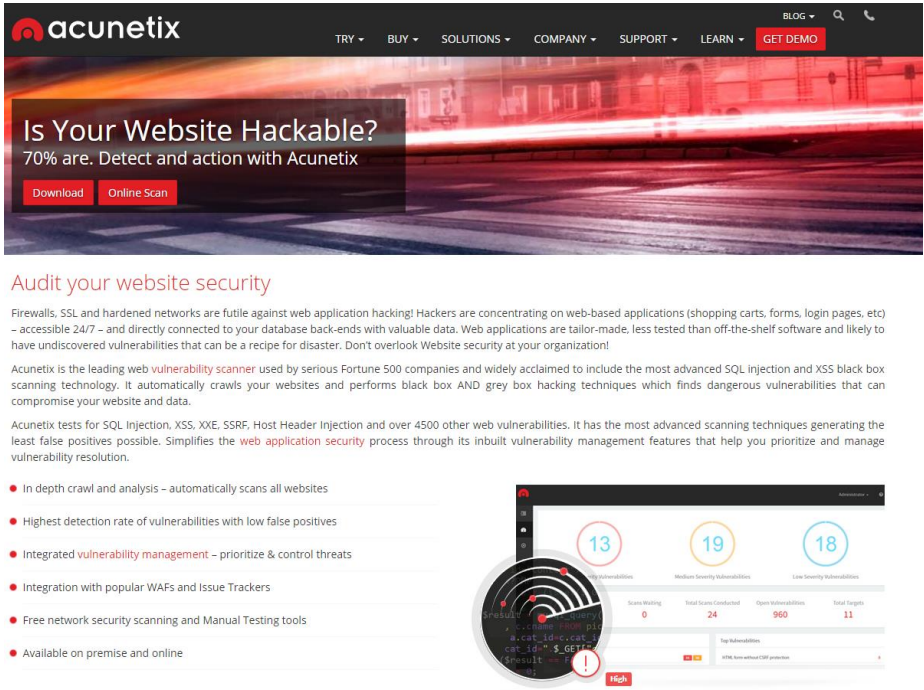
Tentando sair do problema

- Verificando os dialetos de SQL, vários delimitadores podem ser usados no Microsoft Jet DBMS engine, por exemplo, datas podem ser delimitadas com o caracter sustenido.

Ferramentas de Auditoria

- Acunetix: http://www.acunetix.com/

- PARA ANALISAR O SEU SITE
  - Injeção de SQL
  - Cross-Site Scripting
  - Outras vulnerabilidades



ACUNETIX

- GERAR RELATÓRIOS
  - Dados **PCI DSS** (a norma de segurança de transações eletrônicas**) Payment Card Industry (PCI) – Data Security standard (DSS)**
  - **OWASP** (**Open Web Application Security Project**) Top 10 Vulnerabilidades

Outras ferramentas de auditoria BD

- **DB Audit Free Edition** – ferramenta de auditoria e análise de segurança para bancos de dados *Oracle*, *Sybase*, *DB2*, *MySQL* e *Microsoft SQL Server*. http://www.softtreetech.com

- **SQL Map** – ferramenta automática em linha de comando para testes de *sql-injection*. http://sqlmap.sourceforge.net

- **Wapiti** - *Wapiti* permite realizar auditoria de segurança de aplicações web. http://wapiti.sourceforge.net