Grupo de Resposta a Incidentes de Segurança

SQL INJECTION

por:
George Lucas L. Da Silva
Jarcy de Azevedo Junior

gris@gris.dcc.ufrj.br
george@gris.dcc.ufrj.br
jarcy@gris.dcc.ufrj.br

Grupo de Resposta a Incidentes de Segurança
Departamento de Ciência da Computação
Instituto de Matemática
Universidade Federal do Rio de Janeiro

Índice

O que é SQL?

SQL – Structured Query Language

Linguagem amplamente utilizada para administrar bancos de dados

Sintaxe de fácil aprendizado

Sistemas de banco de dados que usam SQL

Exemplos de diferentes versões de sistemas de bancos de dados:

MySQL

Microsoft SQL Server

Oracle

PostgreSQL

(...)

Usaremos como base a sintaxe do MySQL

Exemplo de tabela

Nº	Nome	Identidade	CPF	Email	MD5(Senha)
001	Amelia	xx.xxx.xxx-x	A	amelia @provedor.com	81dc9bdb52d04dc20036dbd8313ed055
002	Ronaldo	yy.yyy.yyy-y	B	ronaldo @provedor.com	6f1ed002ab5595859014ebf0951522d9
003	Batman	zz.zzz.zzz-z	C	batman @provedor.com	d8578edf8458ce06fbc5bb76a58c5ca4

Exemplo 1: Tabela **usuarios**

Principais Instruções SQL

Declarações:

SELECT *coluna(s)* FROM *tabela*

INSERT INTO *tabela* VALUES (argumentos)

UPDATE *tabela* SET (argumentos)

DELETE FROM *tabela*

Principais Instruções SQL

Cláusulas: WHERE, ORDER BY

Operadores lógicos: AND, OR, NOT

Curingas: *, _, %

Caracteres relevantes:

' “ ; --, #

SQL Injection na prática

SQL Injection costuma ser feito em aplicações (como PHP, Java ou ASP), e o objetivo do ataque é extrair informação relevante da base de dados.

O ataque explora vulnerabilidades diretamente na aplicação, baseando-se no grande problema de administradores: validar toda forma de input, assumindo sempre que ele pode ser malicioso

Acessando a base de dados

Pegando input – forma padrão:

SELECT * FROM ***usuarios*** WHERE Nome = '**Amelia**' AND Senha = '**1234**'

Note as aspas simples sendo adicionadas no início e no fim do input antes de serem enviados para a base de dados. Funcionaria para aspas duplas também

Acessando a base de dados

No login escrevemos a seguinte string:

' OR 1=1 #

Produzindo a seguinte query:

SELECT * FROM *usuarios* WHERE Nome = '' OR 1=1 **#' AND Senha = ''**

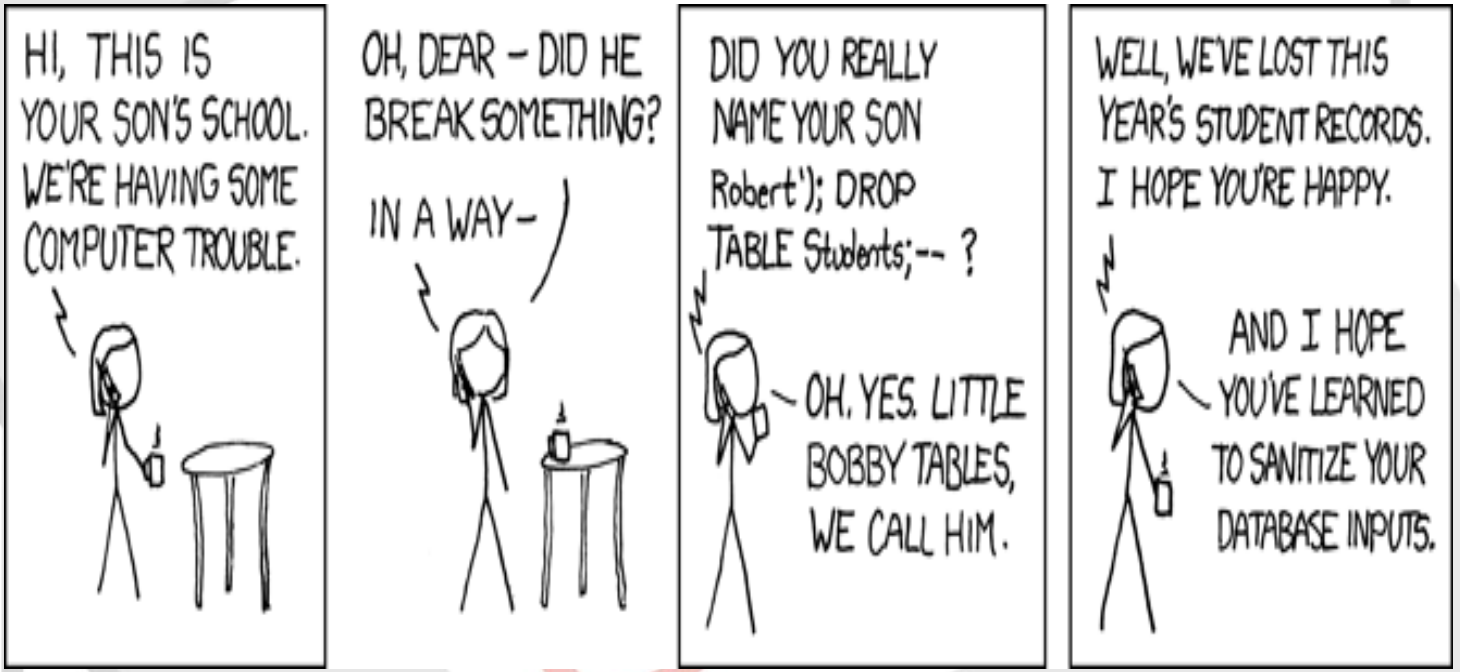
Algumas notas

Não importa a complexidade da senha – o simbolo de comentário (#) irá fazer a query ignorar tudo o que vem depois do login

Como 1 sempre é igual a 1, ganharemos acesso ao sistema de forma fácil e rápida!

Além de conseguir acesso, é possível também inserir outras instruções SQL no login, usando o que se chama de *Stacking Queries*

Para descontrair



Exemplo 2 – *Stacking queries*

Acessando a base de dados

Saída natural: escapar os caracteres problemáticos, como aspas simples e aspas duplas?

Acessando a base de dados

Método eficaz para a maior parte dos casos, mas tem pequenos problemas

O uso de codificações que interpretam caracteres com mais de um byte pode ser fatal

A melhor solução é usar o que se chama de “declarações preparadas” para separar o input, tratá-lo individualmente e só depois adicioná-lo a query em questão

Ataques pela URL

Forma de ataque que visa obter informações das tabelas do banco de dados

Baseado em mensagens de erro do servidor

Utiliza a declaração “UNION ALL SELECT”, o qual exige que ambas as queries chamadas pelos SELECTs tenham o mesmo número de colunas e o mesmo tipo de dados

Ataques pela URL

Formas de obter informação:

Utilize queries com sintaxe propositalmente ruim para forçar mensagens de erro e obter informações delas – essa é a forma de ataque!

Exemplos:

Onde se tem “(...)userid=200”, tente colocar “(...)userid=200' ” ou “ (...)userid='boo' ”

Ataques pela URL

Primeiramente, vamos descobrir quantas colunas tem a tabela em questão...

Adicionamos a seguinte string na query:

ORDER BY 1 #

E vamos aumentando o valor numérico de 1 em 1 até obtermos uma mensagem de erro

Ataques pela URL

Por quê funciona?

A Cláusula ORDER BY ordena os resultados da query de acordo com o nome ou número da coluna que for especificado nela.

Ataques pela URL

SELECT * FROM *usuarios* ORDER BY 1,2,..,6

Nº	Nome	Identidade	CPF	Email	MD5(Senha)
001	Amelia	xx.xxx.xxx-x	A	amelia @provedor.com	81dc9bdb52d04dc20036dbd8313ed055
002	Ronaldo	yy.yyy.yyy-y	B	ronaldo @provedor.com	6f1ed002ab5595859014ebf0951522d9
003	Batman	zz.zzz.zzz-z	C	batman @provedor.com	d8578edf8458ce06fbc5bb76a58c5ca4

Atenção: existem tipos de dados não-ordenáveis!

Ataques pela URL

Agora vamos descobrir o tipo de dados que cada coluna armazena!

Isso é feito adicionando-se a seguinte string na query:

UNION ALL SELECT NULL, NULL, ...(n vezes) #

n é o número de colunas

Caso o parâmetro seja uma string, precisamos adicionar uma aspas simples ou dupla antes do “ # ”

Ataques pela URL

Por quê funciona?

UNION ALL SELECT combina o resultado de duas ou mais queries. A ideia aqui é, sabendo-se que o tipo NULL pode ser combinado com qualquer outro tipo, iremos fazer testes a fim de descobrir os tipos de dados das colunas.

“WHERE 0=1” no fim da query pode ser necessário para evitar possíveis problemas da aplicação em lidar com o tipo NULL.

Ataques pela URL

Testes (para n = 4):

(…) UNION SELECT **1**, NULL, NULL, NULL, NULL #

Ok.

(…) UNION SELECT **1, 2**, NULL, NULL, NULL #

Erro – a coluna não é de tipo numérica.

(…) UNION SELECT **1, '2'**, NULL, NULL, NULL #

Ok.

E assim vai. Para o caso de o input ser filtrado para aspas simples (ou dupla) como já vimos, existe uma forma de driblar essa defesa!

Ataques pela URL

Como sabemos, toda string possui uma representação em hexadecimal. Se conseguirmos converter a string e as aspas para um código numérico, poderemos fazer a Injection!

Exemplo: '2' em hexadecimal vira 0x273227.

Ataques pela URL

Vamos refazer a última query:

(…) UNION SELECT 1, **'2'**, NULL, NULL, NULL #

Vira:

(…) UNION SELECT 1,**CONCAT(CHAR(27),CHAR(32),CHAR(27))**,
NULL, NULL, NULL #

Uma alternativa seria fazer diretamente:

(...),CHAR(27)+CHAR(32)+CHAR(27),(...)

Ataques pela URL - notas

De posse dos tipos de dados das colunas, o atacante agora sabe por qual delas extrair o tipo de dado desejado – violando a confidencialidade do servidor

Tenha em mente que, sabendo a aplicação que você está utilizando, o atacante pode descobrir os nomes-padrão de outras tabelas suas usando um site de busca e extrair dados delas também!

Ataques pela URL - notas

O passo natural agora é coletar informações de arquivos do servidor e até mesmo de outras tabelas, especificando o nome e a coluna desejada desta na hora de fazer o UNION ALL SELECT

Assim, temos várias possibilidades de ataque novas: podemos usar a função LOAD_FILE() para ver os conteúdos de algum arquivo importante rodando no servidor, ou mesmo tentar ver o hash da senha de administrador para quebrá-lo!

Ataques pela URL - notas

Dicas gerais:

Existe uma constante chamada “ @@version “ em qualquer banco de dados SQL que mostra a versão dele – isso ajuda a confirmar com qual base de dados estamos lidando

Busque burlar os filtros! Um bom exemplo é se existe uma filtragem dos espaços em branco: basta trocá-los por ' /**/ ' (sem aspas)

Blind SQL Injection

Mesmo quando o administrador do banco de dados é cuidadoso com a higienização do input e fortalece suas defesas de diversos jeitos, ele ainda pode estar vulnerável a SQL Injection

Caso de estudo:

Ataques ao site do Kaspersky e da Symantec em fevereiro desse ano

Blind SQL Injection

Descobrindo silenciosamente se um site é vulnerável a esse tipo de Injection:

Veja se a URL aceita receber operações lógicas (AND, OR) e matemáticas. Exemplos:

(...)userid=199-1

(...)userid=200 AND 1=1

(...)user='b'+'oo'

Formas de ataque

Pode ser realizado de duas formas:

Primeira forma: perguntando à base de dados, registro por registro nas tabelas, cada letra das palavras.

Ou seja, perguntamos à base de dados se o primeiro caractere da primeira coluna possui valor ASCII maior que “x”. Analisamos a resposta, e já preparamos outra pergunta até descobrirmos esse primeiro caracter.

Formas de ataque

Segunda forma: fazer consultas à base de dados tentando descobrir, por exemplo, nome de tabelas e colunas por força bruta, fazendo uso de funções de delay.

A ideia é usar algo como a declaração IF para fazer essas perguntas, e caso a condição seja satisfeita, forçar o servidor a esperar um tempo e só então retornar alguma coisa. No caso do MySQL usa-se a função BENCHMARK(numero de vezes, ação) para o serviço.

Blind SQL Injection - notas

As duas formas de ataque podem ser combinadas: podemos ir varrendo os registros de uma tabela letra a letra e, caso não tenhamos respostas claras na primeira forma de ataque, chamamos funções de delay para obtermos nossas respostas

Existem diversas ferramentas que fazem Blind SQL Injection de forma extremamente efetiva: essa é uma ameaça real!

Dicas gerais

PREOCUPE-SE COM TODA FORMA DE ENTRADA!
Todo input é malicioso até que se prove o contrário, mas ele deve ser validado

Atente aos privilégios dos usuários – várias formas de ataque não abordadas aqui podem ser facilmente evitadas dessa forma

Com **SQL Injection** é possível até mesmo atacar o próprio **sistema operacional** em ataques mais complexos, então pesquise e se informe sobre as formas de defesas mais completas existentes

Referências

http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/ - "SQL Injection Cheat Sheet" por Ferruh Mavituna

http://dev.mysql.com/tech-resources/articles/guide-to-php-security-ch3.pdf - "Guide to PHP Security: Chapter 3. SQL Injection" por Ilia Alshanetsky

http://www.webappsec.org/projects/articles/091007.shtml - “The Unexpected SQL Injection” por Alexander Andonov

http://www.imperva.com/docs/Blindfolded_SQL_Injection.pdf - por Imperva

http://xkcd.com/327/

http://www.google.com

Fim!

Dúvidas?