VOLTAR

Auditar aplicações WEB

Mostrar a importância de auditar aplicações WEB e um exemplo de implementação de um tipo de criptografia.

NESTE TÓPICO

Marcar tópico

Olá alunos,

Vamos verificar a importância de auditorias em WEB, já que estamos vivenciando o quarto paradigma da computação: o mundo ligado em rede. E hoje a internet, que antes só podia ser acessada através de um computador, pode ser acessada por qualquer dispositivo móvel.

A ideia da auditoria em WEB é um pouco diferente de outros tipos de auditorias, ela está mais ligada a parte da segurança da informação. É descobrir possíveis vulnerabilidades que possam permitir acesso indevido aos dados.

A auditoria representa um conjunto de ações para se evitar ou pelo menos dificultar a ação de terceiros sobre dados confidenciais. Este conjunto de ações podem ser relacionados a entrada ou a saída de dados. Mas, muitos especialistas recomendam uma maior atenção para a entrada de dados. Por exemplo, um invasor pode injetar algum tipo de comando de entrada de dados e conseguir, então extrair os dados que deseja, esta ação é conhecida como **EXPLOIT**. Os comandos mais comuns são do tipo SQL (Structured Query Language) e são conhecidos como SQLi (SQL injection). Assim, a ideia é se você evitar a intrusão, a entrada de dados, automaticamente evitará a saída de dados.

Vamos relembrar os comandos SQL:

OS COMANDOS SQL

Os comandos SQL estão divididos em categorias: **DDL (Data Definition Language), DML (Data Manipulation Language), DQL (Data Query Language) e DCL (Data Control Language)**.

Basicamente, as categorias são formadas pelos seguintes comandos:

- **DDL: CREATE, ALTER, TRUNCATE e DROP.**
- **DML: INSERT, UPDATE e DELETE.**
- **DCL: GRANT e REVOKE.**
- **DQL: SELECT.**

O comando SELECT é de longe o mais utilizado com frequência quando se fala em banco de dados e é o único comando do SQL que pode retornar nenhum dado, um registro ou várias linhas de registros. É o comando também que permite a elaboração e a utilização de várias funções e clausulas. Vamos ver um exemplo: vamos supor que eu queira fazer uma consulta em uma tabela chamada Produto:

SELECT * FROM Produto WHERE preco >= 200.00;

Lembrando que depois do comando SELECT listamos os campos (colunas) do nosso relatório e o asterisco (*) é um coringa que representa todos os campos da tabela e como utilizamos a clausula WHERE, filtramos os registros. Neste exemplo, pegamos todos campos da tabela Produto, mas só dos produtos com preços iguais ou maiores que 200.00.

Agora, vamos ver um outro exemplo: suponha que existe uma tabela **Usuario** com os campos: **nome** e **senha**, onde o nome seria o login do usuário e a senha dele. Esta tabela armazena os dados de login e senha dos usuários. Vamos supor que alguém queira invadir e tentar descobrir estes dados. Se o invasor conseguir passar pela segurança da infraestrutura, ele poderá injetar ou enviar, um tipo de dado em um comando SELECT, por exemplo:

SELECT * FROM Usuario WHERE senha = **' ' OR '1' = '1'**;

Este poderia ser um SELECT da aplicação para retornar ou até mesmo autenticar o usuário e a senha, pode ser até este tipo de SELECT:

SELECT * FROM Usuario WHERE nome = 'vNome' AND senha = **' ' OR '1' = '1'**;

Neste caso, qualquer um dos dois comandos SELECT retornariam **todos os dados** da tabela **Usuario**, vamos analisar o porquê:

O usuário vai digitar o seu login: nome e senha pela aplicação, por uma tela de login, estes dados serão armazenados em variáveis e o comando SELECT irá comparar através do WHERE o nome e a senha de acordo com os valores armazenados: **WHERE nome = 'vNome' AND senha = 'vSenha'**. Neste caso,

mesmo que o invasor não tenha privilégio para realizar um SELECT direto na tabela, mas ele pode mandar para a senha (pode ser também para o nome) estes dados: **' ' OR '1' = '1'**. O que isto significa?

O invasor simplesmente utilizou a álgebra booleana: neste caso, mesmo que eu tenha um operador booleano **AND**, onde os dois lados da comparação têm que ser verdadeiros para retornar algo. Do outro lado temos: **senha = ' ' OR '1' = '1'**. O invasor passou uma comparação de um valor em branco ou (**OR) 1 = 1**. Como no caso do operador OR, se qualquer uma das comparações for verdadeira, ele retornará algo. E como 1 é igual a 1 (verdadeiro) e se o nome for verdadeiro, então neste caso, a senha será considerada qualquer valor e todos os dados da tabela Usuario serão retornados. Simplesmente, este tipo de SELECT é equivalente a SELECT * FROM Usuario. O invasor vai ter todos os nomes e senhas.

Vamos ver como poderemos dificultar, se um invasor conseguir passar por várias barreiras de segurança e conseguir injetar algum dado malicioso para conseguir obter algo.

A FUNÇÃO HASH

A função hash tem vários tipos de aplicações, no caso da computação, geralmente faz parte da criptografia, no caso seria transformar um valor absoluto em um **valor hash**. A função internamente vai estabelecer um link entre estes dois valores. Por exemplo, o usuário vai digitar os seus dados, mas estes estarão armazenados com um **valor hash** na tabela e a função hash conseguirá comparar estes dois valores.

Em Python, temos a função hash como nativa, basta importarmos:

```
>>> import hashlib
>>> var = hashlib.md5(b'abc123')
>>> print(var.hexdigest())
e99a18c428cb38d5f260853678922e03
>>>
```

Importamos a função hash com **import hashlib**, passamos um valor: abc123 para uma variável e na sequência, imprimimos, mostrando o valor em hexadecimal.

Em Python 3, principalmente, podemos trabalhar com vários algoritmos ou **digest** de condensação do hash criptográfico: **md5, sha1, sha224, sha256, sha384 e sha512**:

Hashlib.md5(b' ') em Python 3 é obrigatório, neste caso, utilizar o parâmetro **b** (bytes) antes do valor a ser criptografado.

Vamos ver o mesmo exemplo com outro parâmetro:

```
>>> var = hashlib.sha256(b'abc123')
>>> print(var.hexdigest())
6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090
```

Utilizamos o algoritmo **sha256** e em comparação com o **md5**, o md5 apresenta um valor mais condensado.

Vamos construir um exemplo na prática, utilizando um banco de dados que já vem nativo no Python: o **SQLite**. Lembre-se que quando trabalhamos com variáveis, estes dados ficam na memória principal e quando trabalhamos com banco de dados (SQL) estes dados ficam em disco.

Então, a primeira coisa a ser feita, é construirmos o banco de dados e a tabela. Para isso vamos criar um **script**:

```
1.  # script para criar o banco de dados e a tabela
2.  import sqlite3
3.  conn = sqlite3.connect('empresa.db')
4.  cursor = conn.cursor()
5.  cursor.execute(
6.      'CREATE TABLE usuario(nome TEXT NOT NULL,senha TEXT NOT NULL);'
7.      )
```

Na linha **2**, importamos o banco de dados SQLite.

Na linha **3**, criamos a conexão e ao mesmo tempo o banco de dados com o nome empresa e a extensão **db**. É necessário criar um banco de dados com um nome e este banco de dados é o local onde ficarão as tabelas.

Na linha **4**, criamos um cursor, que é uma espécie de ponteiro. Todo o comando SQL daqui para frente, utilizará este cursor. Se utilizarmos os comandos SQL diretamente numa IDE do banco de dados, não é necessário criar este cursor.

Na linha **6**, utilizamos o cursor com o comando SQL: **CREATE TABLE**, para criar a tabela usuário com dois campos: nome e senha, com os tipos TEXT e a restrição NOT NULL.

Depois de criarmos a tabela, na sequência, vamos criar um script para cadastrar os dados:

```
1.  # script para inserir dados do login
2.  import sqlite3
3.  import hashlib
4.  conn = sqlite3.connect('empresa.db')
5.  cursor = conn.cursor()
6.  vnome = input('Digite o nome para o login: ')
7.  vsenha = input('Digite uma senha para o login: ')
8.  d = hashlib.md5()
9.  d.update(vsenha.encode('utf-8'))
10. cursor.execute('insert into usuario values ("%s", "%s")' % (vnome, d))
11. conn.commit()
12. print("senha hash gerada: ",d.hexdigest())
13. input("Pressione ENTER para sair...")
```

Na linha **3**, importamos a biblioteca **hashlib**.

Nas linhas **4 e 5**, fizemos a conexão com o banco de dados e criamos um cursor.

Nas linhas **6 e 7**, criamos variáveis para receber o nome e a senha.

Na linha **8**, criamos uma variável **d**, para receber o hash com o algoritmo **md5**.

Na linha **9**, aplicamos o hash na variável **vsenha**.

Na linha **10**, com o cursor utilizamos o comando SQL: **INSERT**, para cadastrar o nome e a senha, passando os valores das variáveis: **vnome** e **d**.

Na linha **11,** utilizamos o comando do SQL: commit para gravar os dados no disco.

Na lnha **12**, mostramos o valor em hash gerado.

Vamos ver um exemplo do resultado do cadastro de cinco usuários:

```
Digite o nome para o login: denilson
Digite uma senha para o login: dd457
senha hash gerada:  39f8e34ed16c8a5aad61340cbb626664
Pressione ENTER para sair...
>>>
Digite o nome para o login: cida
Digite uma senha para o login: yyt87r
senha hash gerada:  82e1d7a0c6490560b9dddaf32d2d81ce
Pressione ENTER para sair...
>>>
Digite o nome para o login: mary
Digite uma senha para o login: Afp76
senha hash gerada:  82ff7acc17160345dbb723c688c6922f
Pressione ENTER para sair...
>>>
Digite o nome para o login: analie
Digite uma senha para o login: 9d6te
senha hash gerada:  9a5fedd99272cca7318f1a85788b96cf
Pressione ENTER para sair...
>>>
Digite o nome para o login: rapha
Digite uma senha para o login: g42m22
senha hash gerada:  4ba8d246e6189f442581743d22f9e4f0
Pressione ENTER para sair...
```

Agora vamos fazer uma consulta nos registros armazenados, para isto vamos criar um script de consulta:

```
# script para listar os logins dos usuários
import sqlite3
conn = sqlite3.connect('empresa.db')
cursor = conn.cursor()
cursor.execute('SELECT * FROM usuario;' )
for i in cursor.fetchall():
        print(i)
input('Pressione ENTER para sair...')
```

Na linha **5**, executamos o comando SQL: **SELECT**.

Na linha **6**, criamos um loop com a função **fetchall()**, o comando fetch pega a linha (registro) da tabela. E com o loop, varremos toda a tabela.

Ao rodar o scirpt, este será o resultado mostrado:

```
('denilson', '<md5 HASH object @ 0x01517608>')
('cida', '<md5 HASH object @ 0x015D7608>')
('mary', '<md5 HASH object @ 0x011A7608>')
('analie', '<md5 HASH object @ 0x03157608>')
('rapha', '<md5 HASH object @ 0x031A7608>')
Pressione ENTER para sair...
```

Neste caso, as senhas com o valor original não são mostradas. Internamente a função hash criou uma espécie de objeto para o valor que não está em caractere e sim em bytes. E outro detalhe, não há como reverter, ou seja, se o valor está em hash, não tem como descobrir o seu valor em caracteres. Desta forma, se o invasor conseguir acessar os dados, seria retornado algo como no exemplo acima.

Atualmente existem outras funções e formas de implementar criptografias. Estes exemplos são praticamente considerados de criptografia básica, mas já poderia dificultar a ação de um invasor.

SAIBA MAIS...

Dê uma olhada nos links abaixo para saber mais sobre a linguagem Python:

https://www.python.org/doc/ (https://www.python.org/doc/)

https://wiki.python.org/moin/PythonBooks (https://wiki.python.org/moin/PythonBooks)

Neste tópico vimos a importância de proteger dados que vêm por alguma aplicação da WEB.

Apresentamos um exemplo de criptografia, através da implantação da função hash.

Quiz

Exercício Final

Auditar aplicações WEB

INICIAR

Referências

SUMMERFIELD, M. *Programação em Python 3*: Uma introdução completa à linguagem Python. Rio de Janeiro Alta Books, 2012. 495 p.

MENEZES, N. N. C. *Introdução à programação com Python:* algoritmos e lógica de programação para iniciantes. 2. ed. São Paulo: Novatec, 2014. 328 p.

SWEIGART, AL. *Automatize tarefas maçantes com Python:* programação prática para verdadeiros iniciantes. São Paulo: Novatec, 2015. 568 p.

PYTHON, doc. Disponível em: <https://www.python.org/doc/>. Acesso em: Junho/2018.

PYTHON, books. Disponível em: <https://wiki.python.org/moin/PythonBooks>. Acesso em: Junho/2018.

Avalie este tópico

ANTERIOR
Criando script de cavalo de tróia para logging em teclas

Índice

PRÓXIMO
Criação crawlers web e exploits em Python

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site

Ajuda? (https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)