



**Graduação em Análise e
Desenvolvimento de Sistemas**

**Segurança em
Desenvolvimento de Sistemas**

Prof.º: Plinio Marcos Mendes Carneiro

Introdução

- Os ataques a serviços de rede estão se tornando muito sofisticados
- A criatividade humana não para
- Mas o que faltava?
 - APLICAÇÕES!!! Mas será que é possível?
- Atualmente a maioria dos BUG's reportados estão em aplicações
- Application Security != Network Security
- Não se esqueçam: Quem provê acesso aos dados? As aplicações !!!

Introdução - Números

- Gartner
 - **75%** dos ataques acontecem no nível das aplicações
 - Em caso de falha no sistema provavelmente os desenvolvedores são três vezes mais culpados do que os administradores de sistemas
- NIST
 - **92%** das vulnerabilidades estão no nível das aplicações

Contexto

- Não há segurança de sistemas sem a conjunção de três fatores:
 - Boas praticas de codificação
 - Metodologia adequada (arquitetura e etc)
 - Testes

OSSTMM

Open-Source Security Testing Methodology Manual

Created by Pete Herzog

O que é OSSTMM?

OSSTMM é uma metodologia de testes de segurança de um ambiente, com testes meticulosos e análises minuciosas seguindo essa metodologia é possível testar se seu ambiente esta seguro.

O Manual completo está no site.

<http://www.isecom.org/osstmm/>

SQL Injection

- A idéia é injetar um comando SQL (Structured Query Language) ou comando como input dos dados em um formulário WEB.
- Todos os parâmetros passados são direcionados para o banco de dados.
- O atacante pode manipular com as tabelas e dados diretamente.



SQL Injection

Site vulnerável



Username

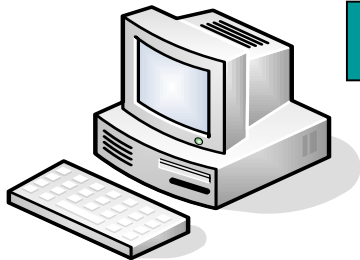
Password

Username

Password



Esperado



Usuário Malicioso

<http://target.site/login.jsp>

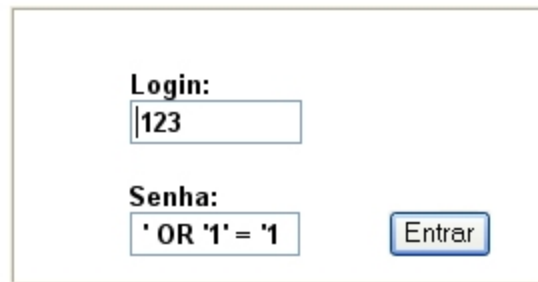
Username

Password



Não esperado

Autenticação



The image shows a login form with two input fields and a button. The first field is labeled 'Login:' and contains the text '123'. The second field is labeled 'Senha:' and contains the SQL injection payload '' OR '1' = '1'. To the right of the password field is a button labeled 'Entrar'.

O problema ocorre devido a autenticação do usuário ser validada internamente pela aplicação com a seguinte instrução, que verifica se existe um usuário com o respectivo login e senha:

```
SELECT * FROM tabela_usuarios WHERE login = 'campo_login' AND senha = 'campo_senha'
```

Normalmente, seriam digitados o login e a senha, que resultaria em uma consulta ao banco de dados para verificar se os mesmos conferem mas na figura acima foi passado parte de um comando SQL no campo reservado.

```
SELECT * FROM tabela_usuarios WHERE login = '123' AND senha = '' or '1' = '1'
```

Observe que o comando passado no campo da senha fez com que independente do login e senha informados, a condição seja sempre verdadeira, permitindo assim o acesso do usuário à aplicação sem o mesmo possuir a devida permissão.

SQL esperado	Parâmetros informados		SQL resultante	Comentário
	Campo_login	Campo_senha		
	marcos';--		SELECT * FROM tabela_usuarios WHERE login = 'marcos';--' AND senha = 'campo_senha'	Se o usuário já souber o login (no caso, login do usuário marcos) então consegue logar sem senha, já que os caracteres "--" são comentários no SQL
SELECT * FROM tabela_usuarios WHERE login = 'campo_login' AND senha = 'campo_senha'	' OR 1=1 --		SELECT * FROM tabela_usuarios WHERE login = " OR 1=1--' AND senha = 'campo_senha'	Neste caso, não é necessário saber nem o login nem a senha, pois a condição "OR 1=1" sempre vai ser satisfeita e todos os comandos posteriores são comentados pelos caracteres "--"

SQL Injection – Problemas

Esperado:

username: abc

password: teste123

Quando submetido a query será montada como:

```
select * from users where username='abc' and password = 'teste123'
```

Não esperado:

username: abc'; --

password:

A query enviada para o banco de dados será:

```
select * from users where uname='abc'; --' and password=''
```

SQL Injection – Problemas

Esperado:

Username: doug

Password: p@\$\$w0rd

```
SELECT COUNT(*)
```

```
FROM Users
```

```
WHERE username='doug' and password='p@$$w0rd'
```

Não esperado:

Username: ' OR 1=1 --

Password:

```
SELECT COUNT(*)
```

```
FROM Users
```

```
WHERE username='' OR 1=1 -- and password=''
```

Detectando a vulnerabilidade de um sistema

Para ilustrar o conceito de SQL Injection, a seguinte simulação pode ser realizada. Imaginemos que um script de validação de acesso de usuários tenha sido desenvolvido como segue:

```
1 <?php
2
3 $usuario = $_POST['usuario'];
4 $senha = $_POST['senha'];
5
6 $query_string = "SELECT * FROM usuarios
7                 WHERE codigo = '{ $usuario }' AND senha = '{ $senha }' ";
8
9 ?>
10
```

Nas linhas 3 e 4, as variáveis **\$usuario** e **\$senha**, respectivamente, recebem o conteúdo submetido por um formulário através do método POST. Eis a fonte do problema.

Suponha que a seguinte entrada tenha sido informada no campo usuário no formulário chamador do script de validação.

The diagram shows a login form with two input fields: 'Usuário:' and 'Senha:'. The 'Usuário:' field is empty. The 'Senha:' field contains ten dots, representing a password. Below the fields is a button labeled 'Efetuar login'. A yellow callout box with a pointer to the password field contains the text '' or 1='1', indicating a SQL injection payload.

Logo, a query string resultante será:



```
SELECT * FROM usuarios WHERE codigo = '' AND senha = '' or 1='1'
```

entrada do usuário
(\$senha)

- Se nenhuma outra validação for realizada, o usuário mal intencionado terá efetuado login no sistema, sem ao menos informar um usuário contido na tabela.
- Isto foi possível pois o valor de entrada informado não recebeu o tratamento devido, sendo adicionado à instrução para ser executado.
- Vale ressaltar que as validações apresentadas no exemplo são apenas ilustrativas, havendo a necessidade de checagens mais eficazes para um script de validação de acesso.

PEN TEST

← → ↻ ⓘ Não seguro | testphp.vulnweb.com/login.php

TEST and Demonstration site for [Acunetix Web Vulnerability Scanner](#)

[home](#) | [categories](#) | [artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#)

search art

go

[Browse categories](#)

[Browse artists](#)

[Your cart](#)

[Signup](#)

[Your profile](#)

[Our guestbook](#)

[AJAX Demo](#)


Links

[Security art](#)

[PHP scanner](#)

[PHP vuln help](#)

[Fractal Explorer](#)



If you are already registered please enter your login information below:

Username :

Password :

Username:

Password: 'OR'1'='1

You can also [signup here](#).

Signup disabled. Please use the username **test** and the password **test**.

[About Us](#) | [Privacy Policy](#) | [Contact Us](#) | ©2019 Acunetix Ltd

Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL Injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more.

Impossibilitando o uso de SQL Injection

- Para que se esteja livre da utilização da SQL Injection, certas providências devem ser tomadas. Algumas das ações serão realizadas no servidor de banco de dados, outras devem ser garantidas pelo código fonte.
- Deve-se tomar cuidado com a configuração do usuário que estabelece a conexão com o banco de dados. O ideal é que as permissões de acesso deste usuário estejam restritamente limitadas às funções que irá realizar, ou seja, para a exibição de um relatório, a conexão com o banco de dados deve ser realizada por um usuário com permissões de leitura e acesso somente às tabelas necessárias para sua operação.
- Todos os valores originados da coleta de dados externos, devem ser validados e tratados a fim de impedir a execução de eventuais instruções destrutivas ou operações que não sejam as esperadas.

Impossibilitando o uso de SQL Injection

- Um tratamento básico para a execução de queries com variáveis contendo valores informados pelo usuário:

```
1 <?php
2
3     $usuario = $_POST['usuario'];
4     $senha = $_POST['senha'];
5     $usuario_escape = addslashes($usuario);
6     $senha_escape = addslashes($senha);
7
8     $query_string = "SELECT * FROM usuarios
9                     WHERE codigo = '{$usuario_escape}'
10                     AND senha = '{$senha_escape}'";
11
12 ?>
```

- Com a utilização da função addslashes() será adicionada uma barra invertida antes de cada aspa simples e aspa dupla encontrada, processo conhecido como escape.
- Se a diretiva de configuração do PHP magic_quotes_gpc estiver ativada, o escape é realizado automaticamente sobre os dados de COOKIES e dados recebidos através dos métodos GET e POST.

Impossibilitando o uso de SQL Injection

- A função `get_magic_quotes_gpc()`, disponível nas versões do PHP a partir da 3.0.6, retorna a configuração atual da diretiva `magic_quotes_gpc`.
- Abaixo, a query string resultante da aplicação do tratamento mencionado:

```
SELECT * FROM usuarios WHERE codigo = '' AND senha = '\\' or 1=\\' 1'
```

entrada do usuário
após tratamento (`$senha`)

Google Hacking



O que é Google Hacking?

- Google Hacking é a atividade de usar recursos de busca do site, visando atacar ou proteger melhor as informações de uma empresa.
- As informações disponíveis nos servidores web da empresa provavelmente estarão nas bases de dados do Google.
- Um servidor mal configurado pode expor diversas informações da empresa no Google. Não é difícil conseguir acesso a arquivos de base de dados de sites através do Google.

Comandos Avançados do Google

intitle, allintitle

- link

inurl, allinurl

- inanchor

filetype

- daterange

allintext

- cache

site

- info

- related

Levantamento de Informações

- O Google é a principal ferramenta para o levantamento de informações de nosso alvo.
- Por isso, vamos definir um alvo específico, e buscar toda informação possível de conseguir através do Google sobre o mesmo. VAMOS À PRÁTICA!
- Dica: tente a busca "currículo + identidade + cpf", sem as aspas. Você já ouviu falar de "laranjas"? Então...
- Google Hacking Database:
 - <http://johnny.ihackstuff.com/ghdb/>
 - Vamos testar algumas tags do GHD e ver que tipo de informação conseguimos.

Mais prática...

site:gov.br ext:sql

inurl:"powered by" site:sistema.com.br

inurl:e-mail filetype:mdb

intitle:VNC inurl:5800 intitle:VNC

"Active Webcam Page" inurl:8080

intitle:"toshiba network camera - User Login"

intitle:"Flash Operator Panel" -ext:php -wiki -cms -inurl:as

"Microsoft-IIS/6.0" intitle:index.of

Contramedidas

- Possuir uma boa política referente à publicações de informações na internet.
- Não deixar configurações padrão em servidores web, para que os mesmos não consigam ser identificados facilmente.
- Sempre analisar as informações disponíveis sobre a empresa em sites de busca.
- Alertar e treinar os funcionários da empresa com relação a maneira com que um ataque de engenharia social pode acontecer, e as possíveis informações que o atacante poderá usar nesse ataque.

SQL Injection – Mais exemplos

Identificando campos de uma tabela

```
SELECT fieldlist  
FROM table WHERE  
field = 'x' AND email IS NULL; --';
```

Verificando se a tabela existe

```
SELECT email, passwd, login_id, full_name  
FROM table  
WHERE email = 'x' AND 1=(SELECT COUNT(*) FROM tablename); --';
```


SQL Injection – Mais exemplos

Procurando por usuários

```
SELECT email, passwd, login_id, full_name  
FROM members  
WHERE email = 'x' OR full_name LIKE '%Maria%';
```

Ataques de força bruta

```
SELECT email, passwd, login_id, full_name  
FROM members  
WHERE email = 'usuario@provedor.com' AND passwd = 'senha123';
```

SQL Injection – Mais exemplos

Verificando as permissões no Database

```
SELECT email, passwd, login_id, full_name  
FROM members  
WHERE email = 'x'; DROP TABLE members; --';
```

Criando um usuário

```
SELECT email, passwd, login_id, full_name  
FROM members  
WHERE email = 'x';  
INSERT INTO members ('email','passwd','login_id',  
    'full_name')  
VALUES ('usuario@provedor.com','senha',  
    'user','User da Internet');--';
```

SQL Injection – Mais exemplos

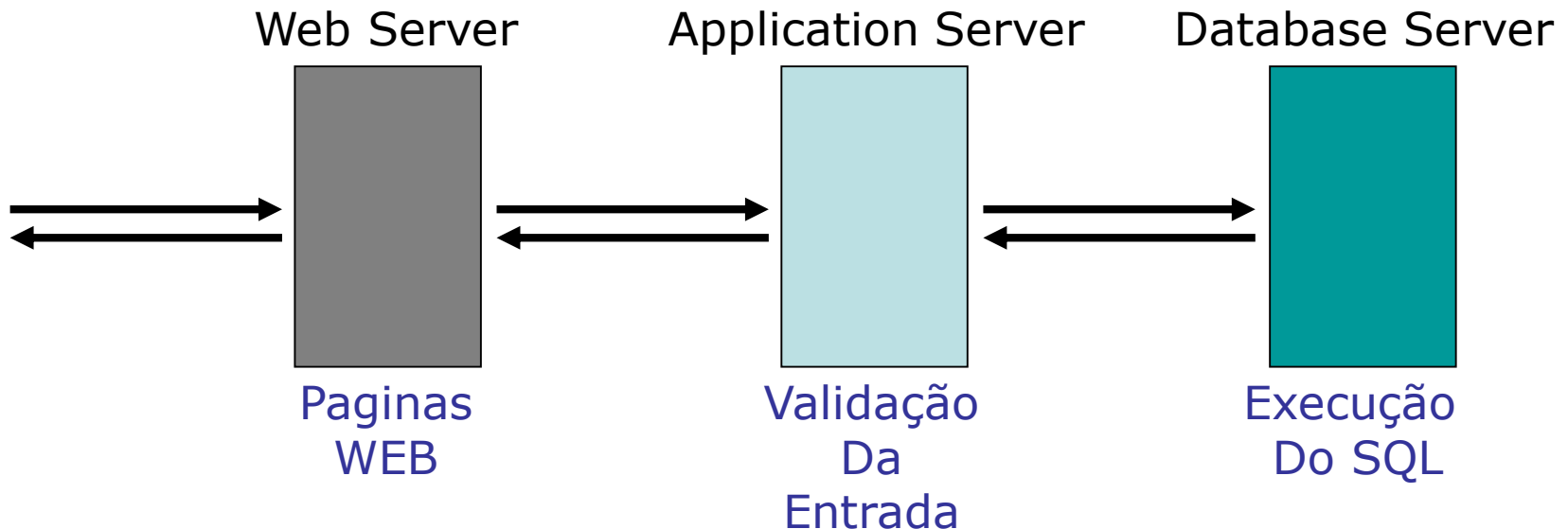
Alterando o mail de comunicação

```
SELECT email, passwd, login_id, full_name  
FROM members  
WHERE email = 'x';
```

```
UPDATE members SET email = 'usuario@provedor.com'  
WHERE email = 'usuario@hackprovedor.com';
```

Arquitetura mais comum

- Sempre tenha em mente
- O SQL normalmente é executado em um outro servidor
- O servidor de BD normalmente não tem acesso a Internet e é protegido



• Solução

- Sempre valide a sessão com login e os parâmetros críticos