



# Teste de Invasão de Aplicações Web

## Capítulo 8

Teste do mecanismo de autorização e da lógica de negócio

---

- **Apresentar vulnerabilidades no mecanismo de autorização e na lógica de negócio e mostrar os métodos que podem ser empregados para detectá-las e explorá-las.**

- **Autorização, acesso direto a recursos, percurso de caminho, redirecionamento não validado, condições de corrida, vulnerabilidades em lógica de negócio.**

- **Introdução**
- **Acesso direto a recursos**
- **Controle de acesso no lado cliente da aplicação**
- **Percurso de caminho**
- **Redirecionamento não validado**
- **Condições de corrida**
- **Vulnerabilidades na lógica de negócio**
- **Contramedidas**

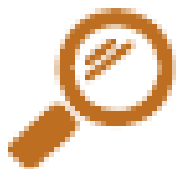
**Um dos objetivos de se autenticar um usuário no sistema consiste em fornecer meios de negar ou autorizar operações que ele deseja realizar, de acordo com os privilégios que possui.**

**Essa verificação deve ser sempre executada por um componente chamado de monitor de referências, no momento em que uma ação é iniciada, o qual deve possuir as seguintes características, para atender os requisitos que se propõe satisfazer:**

**Ser inviolável**

**Ser invocado em  
todo e qualquer  
acesso a um  
recurso**

**Ser suficientemente  
pequeno para que  
possa ser analisado  
e ter sua correção  
comprovada**



**Um problema grave resume-se em ignorar o segundo item da lista, o que acontece frequentemente em aplicações reais.**

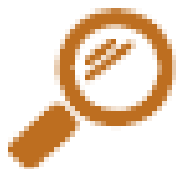
**Existem três tipos desses modelos de controle:**

**Controle de acesso  
discricionário (DAC)**

**Controle de acesso  
mandatário (MAC)**

**Controle de acesso  
baseado em papéis  
(RBAC)**





**Ataques contra o mecanismo de autorização podem ser horizontais ou verticais, dependendo do tipo de privilégio que é necessário obter.**

**Muitas aplicações web não respeitam o requisito de um monitor de referências de que seja invocado em toda tentativa de acesso.**

**Em vez disso, a verificação de privilégios é realizada em momento anterior, o que permite que o mecanismo de controle de acesso do sistema seja violado.**

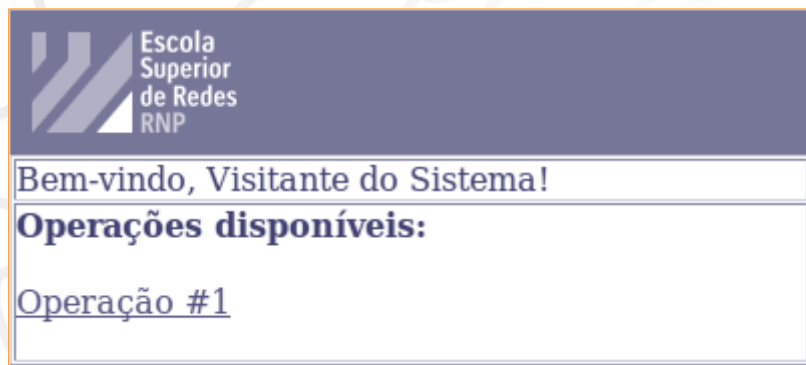
## Exemplos disso são:

**Acesso  
direto a  
páginas**

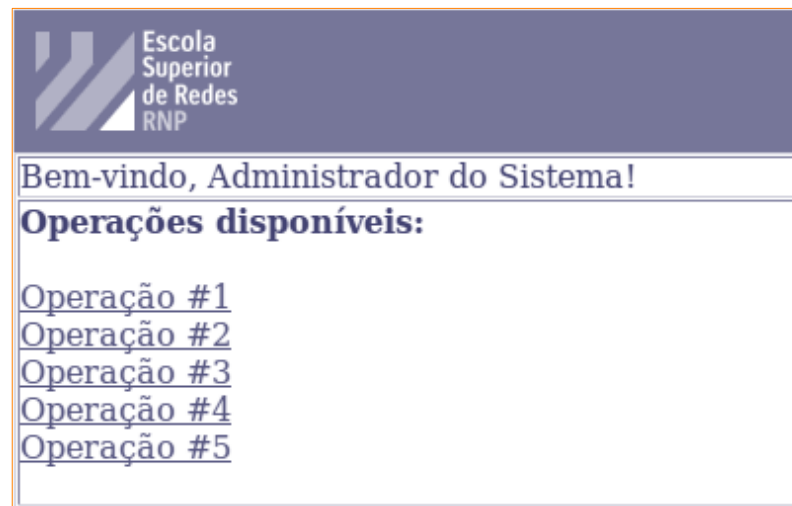
**Uso do  
cabeçalho  
HTTP  
Referer**

**Acesso  
direto a  
objetos**

**Acesso  
direto a  
recursos  
estáticos**



(a)



(b)

Figura 8.1 - Aplicação com segurança por obscuridade: (a) Operações disponibilizadas para um visitante.  
(b) Operações disponibilizadas para o administrador do sistema.

**Há diversas maneiras que um usuário malicioso pode empregar, para descobrir as URLs das operações às quais não tem acesso:**

**registros de trilhas de auditoria**

**documentação da aplicação**

**histórico de navegação**

**interação com outros usuários do sistema**

**tráfego capturado**

**inferência a partir da estrutura geral das URLs usadas nos links disponíveis**

## Roteiro de teste

**1**

**Se diversas contas de usuário, com diferentes perfis, estiverem disponíveis para o teste:**

**Autentique-se na aplicação com cada uma das contas e percorra as diversas páginas que a compõem.**

**1.1**

**1.2**

**Anote todas as funcionalidades diferentes e as URLs correspondentes que aparecem para cada uma das contas.**

**1.3**

**Com cada conta, tente acessar as funcionalidades disponíveis somente para outros usuários.**

**1.4**

**Se o acesso for permitido, o controle de acesso da aplicação é**

**vulnerável.**

## Roteiro de teste

### **2** Senão, se somente uma conta estiver disponível para o teste:

**2.1**

Autentique-se na aplicação e percorra as diversas páginas que a compõem.

**2.2**

Anote todas as URLs das diversas funcionalidades apresentadas.

**2.3**

Verifique se há algum padrão que pode ser identificado nas URLs.

**2.4**

Gere novas URLs, de acordo com o passo anterior, e tente acessá-las.

**2.5**

Se o acesso for permitido, o controle de acesso da aplicação é vulnerável.

O cabeçalho HTTP Referer permite que clientes indiquem para o servidor o endereço do recurso, a partir do qual a requisição está sendo realizada.

Algumas aplicações o utilizam inadvertidamente com fins de segurança, para impedir acesso direto a recursos. A ideia é obrigar que a requisição tenha origem em uma página na qual, necessariamente, o usuário tenha passado pelo processo de autenticação.

Contudo, a estratégia adota a premissa inválida de que o usuário não é capaz de definir o Referer como bem desejar.



## Roteiro de teste

**1** Para cada uma das URLs identificadas na fase de mapeamento:

**1.1** Digite-a na barra de endereços do navegador web e faça a requisição.

**1.2** Se a aplicação exibir uma mensagem indicando que a origem da requisição não é válida:

**1.2.1** Faça nova submissão, mas incluindo o cabeçalho Referer definido para a URL da página, a partir da qual o recurso é carregado, no fluxo normal de uso.

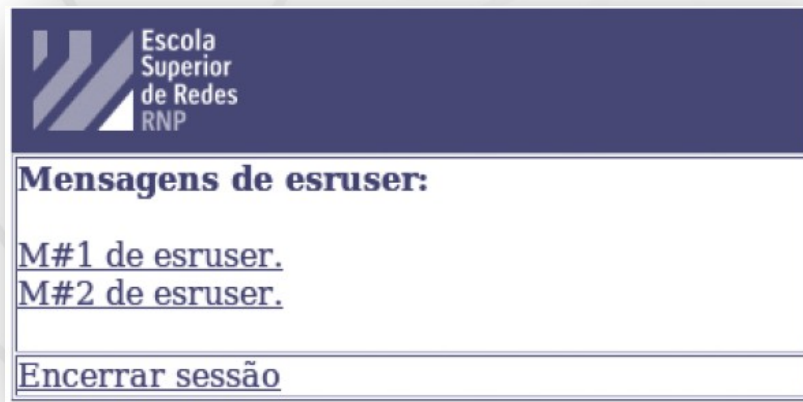
**1.2.1** Se a página for carregada, a aplicação emprega o cabeçalho Referer para controlar o acesso e é, portanto, vulnerável.

- Se uma página contiver os seguintes links, que tipo de teste pode ser realizado?

<http://bssac.esr.rnp.br/view.php?mid=1>

<http://bssac.esr.rnp.br/view.php?mid=2>

(a)



(b)

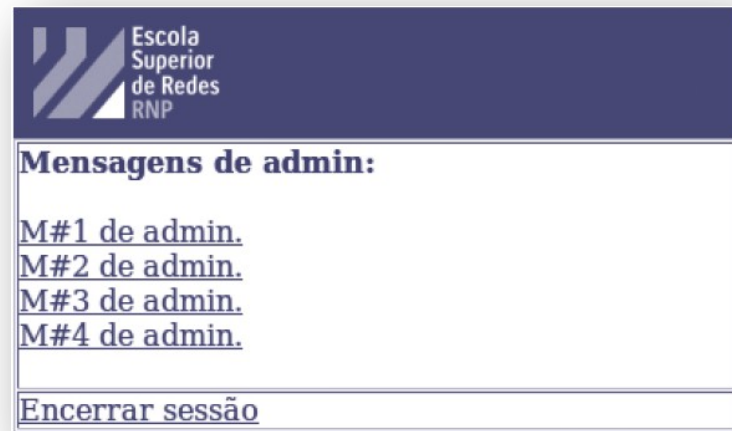


Figura 8.3 - Exemplo de aplicação vulnerável a acesso direto a objetos: (a) Mensagens de “esruser”.  
(b) Mensagens de “admin”.

## Roteiro de teste

**1**

**Se diversas contas de usuário, com diferentes perfis, estiverem disponíveis para o teste:**

**1.1**

**Autentique-se na aplicação com cada uma das contas e percorra as diversas páginas que a compõem.**

**1.2**

**Anote todos os itens de entrada que pareçam ser chaves primárias de registros ou identificadores de objetos e os respectivos valores.**

**1.3**

**Com cada conta, tente acessar os objetos de outra, alterando o valor do parâmetro, de acordo com o encontrado no passo anterior.**

**1.4**

**Se o acesso for permitido, o controle de acesso da aplicação é vulnerável.**

## Roteiro de teste

**2** Senão, se somente uma conta estiver disponível para o teste:

**2.1** Autentique-se na aplicação e percorra as diversas páginas que a compõem.

**2.2** Anote todos os itens de entrada que pareçam ser chaves primárias de registros ou identificadores de objetos e os respectivos valores.

**2.3** Verifique se há algum padrão que pode ser identificado nos valores de cada parâmetro encontrado no passo anterior.

**2.4** Gere novos valores, de acordo com o Passo 2.3, e os utilize nos parâmetros associados, para realizar requisições à aplicação.

**2.5** Se o acesso for permitido, o controle de acesso da aplicação é vulnerável.

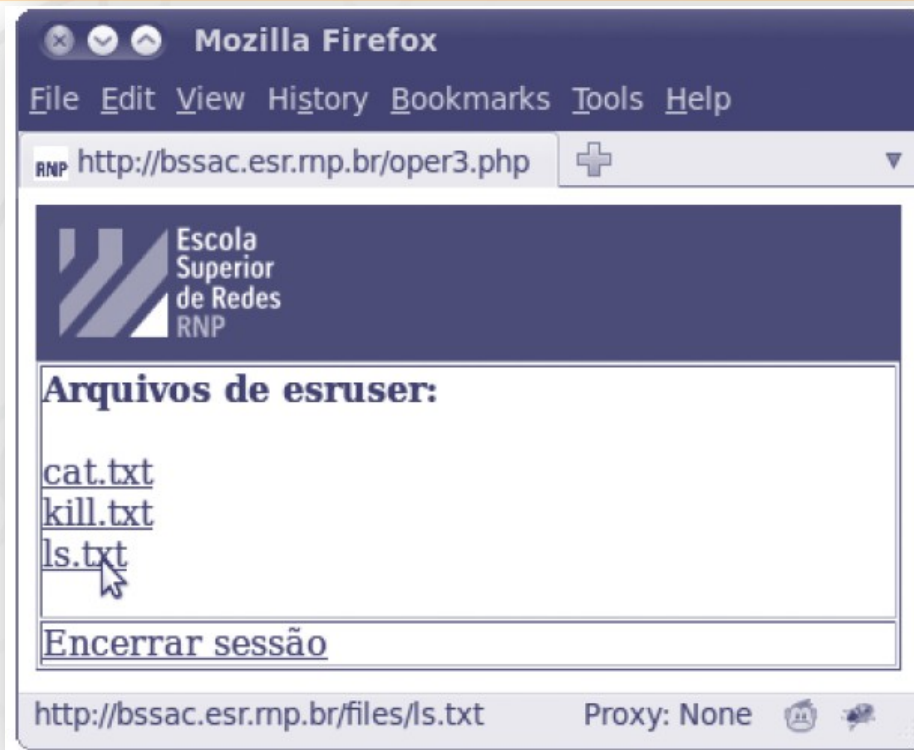


Figura 8.4 - Aplicação que permite acesso direto a recursos estáticos.

**Controles de segurança que são executados no lado cliente da aplicação são ineficazes, pois o usuário tem poder total sobre o que ocorre em seu ambiente. Apesar desse fato, diversas aplicações reais acabam sendo desenvolvidas com essa abordagem, o que as torna completamente vulneráveis a usuários maliciosos.**

## Exemplos dessa vulnerabilidade incluem:

**Autorização  
no lado  
cliente da  
aplicação**

**Manutenção  
de perfil no  
lado cliente  
da aplicação**

**Proteção de  
referências  
a objetos**

**Esse tipo de vulnerabilidade é normalmente encontrado, em um teste de invasão, na etapa de mapeamento, durante a análise dos diversos scripts fornecidos pela aplicação. Qualquer item que manipule links ou que exiba mensagens de erro referentes a controle de acesso deve ser inspecionado com mais atenção, pelo analista de segurança.**



Com o intuito de evitar ataques de acesso direto a objetos, diversos mecanismos foram criados e adotados no passado, porém, alguns deles continuaram sendo vulneráveis, porque não deixaram de expor a real identidade do objeto, de uma maneira ou de outra.

A premissa adotada, de modo geral, era de que a causa raiz do problema se encontrava na exposição direta de uma chave ou nome de um elemento interno da aplicação.

Embora isto estivesse correto, a solução de somente ofuscar o identificador e seguir enviando o resultado para o usuário estava errada, pois o atacante ainda conseguia inferir as referências para recursos alheios.

- Original:

`http://r.rnp.br/view.php?f=fielding.1999.txt&t=0`

- Primeira tentativa:

`http://r.rnp.br/view.php?f=txt.9991.sknaarf&t=1`

- Segunda tentativa:

`http://r.rnp.br/view.php?f=a2x1bnNpbi4yMDAxLnR4dA==&t=2`

Parâmetro decodificado: `klensin.2001.txt`.

- Terceira tentativa:

`http://r.rnp.br/view.php?  
f=706f7374656c2e313938352e747874&t=3`

Parâmetro decodificado: `postel.1985.txt`.

**Com esse ataque, é possível acessar qualquer arquivo do ambiente, para o qual a conta de sistema operacional da aplicação possua privilégio de leitura.**

**A causa raiz deste problema consiste em permitir que o usuário, além do nome do arquivo, também informe o local em que ele se encontra, em vez de assumir que estão em um determinado diretório.**

- Exemplo de código vulnerável:

```
$arquivo = '/var/arquivos/' . $nome;
```

- Entrada maliciosa:

```
../../etc/passwd
```

- Caminho resultante:

```
/var/arquivos/../../etc/passwd
```

Muitas vezes, não é possível saber em que ponto da árvore de diretórios encontra-se a pasta base da aplicação e, assim, fica difícil estabelecer quantas ocorrências de “..” devem ser empregadas no ataque.

Nesses casos, uma técnica que pode ser usada para evitar que a descoberta seja por tentativa e erro, resultando na submissão de múltiplas requisições, resume-se em enviar uma grande sequência de “../” seguida do nome de arquivo desejado.

**Exemplo: ../../../../../../../../etc/passwd**

## Adição de extensão

### ▲ Código vulnerável:

```
$f=$_GET['f'];  
  
$file = 'files/.'.$f.'.txt';  
  
$out = popen('cat '.$file, "r");  
while (!feof($out)) {  
    echo (fgets($out) . '<br>');  
}  
pclose($out);
```

### ▲ Entrada maliciosa:

```
../../../../../../etc/passwd%00
```

## Roteiro de teste

**1** Para cada item de entrada identificado na fase de mapeamento:

**1.1** Selecione aqueles que parecem fornecer um nome de arquivo.

**1.2** Se o nome completo de arquivo é especificado, incluindo a extensão:

**1.2.1** Submeta o nome de um arquivo que se sabe existir no ambiente, precedido de uma longa sequência de “../”.

**1.2.1** Se o arquivo não for exibido, tente outros candidatos, dependendo da mensagem de erro fornecida.

**1.2.3** Senão, a aplicação é vulnerável a percurso de caminho.

## Roteiro de teste

**1** Para cada item de entrada identificado na fase de mapeamento:

**1.3** Senão:

**1.3.1** Submeta o nome de um arquivo que se sabe existir no ambiente, precedido de uma longa sequência de “../” e finalizado com “%00”.

**1.3.1** Se o arquivo não for exibido, tente outros candidatos, dependendo da mensagem de erro fornecida. Pode ser que a evasão com “%00” não seja aplicável e o teste deva ser encerrado.

**1.3.3** Senão, a aplicação é vulnerável a percurso de caminho.



**Redirecionamento é uma técnica que permite instruir o navegador web a carregar outra página, após a originalmente requisitada.**

## Métodos:

Devolver ao navegador web uma mensagem com código de estado 3XX, seguido de um cabeçalho “Location:”, especificando a nova URL. Pode ser realizado programaticamente ou por meio do arquivo .htaccess.

Uso do “meta-tag” “Refresh”, disponível na linguagem HTML.

Utilizar a função “location.replace()”.

**Embora, muitas vezes, o redirecionamento ocorra dentro de um mesmo domínio, também é possível realizar o processo para domínios diferentes.**

**Isso pode ser usado por atacantes, para carregar páginas maliciosas, a partir de um domínio confiável, de modo a conseguir mais facilmente que vítimas as visitem, do que se fosse necessário clicar em links suspeitos.**

## ▲ Código vulnerável (redir.php):

```
<?php  
header('HTTP/1.1 302 Found');  
header('Location: ' . $_GET['url']);  
?>
```

## ▲ Link malicioso:

```
<a href="http://esr.rnp.br/redir.php?  
url=http%3A%2F%2Fwww.evil.org%2F">  
Clique e concorra a prêmios!!!</a>
```

## Correção vulnerável

### ▲ Código vulnerável:

```
<?php  
header('HTTP/1.1 302 Found');  
header('Location: http://esr.rnp.br' . $_GET['target']);  
?>
```

### ▲ Entrada maliciosa:

`.evil.org`

### ▲ Resultado:

`esr.rnp.br.evil.org`

## Roteiro de teste

**1** Para cada item de entrada identificado na fase de mapeamento:

**1.1** Verifique aqueles que são usados em redirecionamento, tanto no lado cliente, como no servidor. Neste último caso, isso pode ser feito, observando-se se a resposta contém um código de estado 3XX.

**1.2** Substitua o valor dos itens encontrados, um por vez, pelo nome de um recurso do servidor que se saiba existir. Se o redirecionamento ocorrer para o elemento escolhido, a aplicação sofre da vulnerabilidade.

**1.3** Repita o passo anterior, mas especificando uma URL absoluta para outro domínio. Se a página especificada for carregada, a aplicação também pode ser usada para redirecionamentos externos.

**Condições de corrida são conhecidas há muito tempo e ocorrem quando processos que são executados concorrentemente manipulam recursos compartilhados, sem o uso de mecanismos de sincronização.**

**Como resultado direto deste problema, programas podem apresentar comportamento inesperado, o qual, na maioria das vezes, é difícil de ser reproduzido e depurado.**

**Alguns impactos à segurança decorrentes de condições de corrida, que podem ser citados, incluem a violação de autenticação, a adulteração da lógica de negócio do sistema e a quebra do mecanismo de autorização e de outros controles.**

## Código vulnerável

```
x = [saldo]BD;  
if (x >= val) {  
    x = x - val;  
    [saldo]BD = x;  
}
```

fatia de tempo	P#1			P#2			[saldo]BD
	# linha	x	val	# linha	x	v	
01	01	100	70				100
	02	100	70				100
	03	30	70				100
02				01	100	50	100
				02	100	50	100
				03	50	50	100
03	04	30	70				30
04				04	50	50	50

Figura 8.13- Resultado da execução concorrente de duas instâncias do código da Figura 8.12.



**A camada de lógica de negócio da aplicação é responsável por ligar as camadas de apresentação e de dados, impondo as regras de negócio aplicáveis.**

**Erros de codificação da regra ou na lógica estabelecida podem resultar em vulnerabilidades, que podem ser exploradas com os mais diversos objetivos.**

**Problemas desse tipo decorrem da adoção de algumas premissas equivocadas e são difíceis de serem detectados.**

## Exemplos

### TV interativa

---

telespectadores podiam mandar comentários, para serem apresentados como texto em um programa de televisão. Revisão acontecia somente na primeira vez.

## Exemplos

### Mac World 2007 Expo

---

códigos prioritários de cinco caracteres, utilizados para se conseguir isenção da taxa de inscrição, eram mantidos no lado cliente, protegidos por funções de hash criptográficas.

## Exemplos

### Competição simulada de bolsa de valores

---

aplicação permitia tomar decisão de compra de lote reservado, após o fechamento do mercado, com o preço do momento da reserva.

## Exemplos

### Vinte e um

---

em uma versão online do jogo, havia um atraso perceptível na oferta de seguro, sempre que a carta fechada era favorável à mesa.

## Transferência de valor negativo

- ▲ Aplicação bancária que permite transferir valores entre contas:
- ▲ Código vulnerável:

```
if ($amount > $balanceA) {  
    $msg = $prefix."Valor solicitado  
        &eacute; maior que o saldo."  
    goto error;  
}  
  
$balanceA = $balanceA - $amount;  
$balanceB = $balanceB + $amount;
```

- ▲ Como o código pode ser explorado?

## Empréstimo acima do limite

### ▲ Representação de números inteiros:

- ▲ O esquema mais utilizado para isso é chamado de complemento de dois, o qual, considerando-se números de  $n$  bits, calcula um valor negativo  $x$ , por meio da fórmula  $2^n - x$ .
- ▲ Números de  $n$  bits podem assumir valores de  $-2^{n-1}$  a  $2^{n-1}-1$ .

## Empréstimo acima do limite

### ▲ Representação de números inteiros:

- ▲ Um fato imediato que pode ser constatado é que nem sempre o resultado de algumas operações caberá na mesma quantidade de bits.
- ▲ Exemplos, considerando números de 8 bits:
  - ▲  $100 + 30$ .
  - ▲  $-100 - 30$ .
  - ▲  $-(-128)$ .



## Empréstimo acima do limite

$$\begin{array}{r} \boxed{01100100} \text{ (100)} \\ + \boxed{00011110} \text{ (30)} \\ \hline \boxed{10000010} \text{ (-126)} \end{array}$$

(a)

$$\begin{array}{r} \boxed{01100100} \text{ (100)} \\ * \boxed{00000010} \text{ (2)} \\ \hline \boxed{11001000} \text{ (-56)} \end{array}$$

(b)

$$\begin{array}{r} \boxed{01100100} \text{ (100)} \\ * \boxed{00001000} \text{ (8)} \\ \hline 11 \boxed{00100000} \text{ (32)} \end{array}$$

(c)

Figura 8.14 - Extravasamento de inteiro: (a)  $100 + 30$ . (b)  $100 * 2$ . (c)  $100 * 8$ .

## Empréstimo acima do limite

- ▲ Código vulnerável:

```
if (limiteUsado + valorEmprestimo <= limiteMaximo)
{
    // Concede o empréstimo
    valorConta = valorConta + valorEmprestimo;
} else {
    // Exibe mensagem de erro
}
```

- ▲ Como o código pode ser explorado?



## Exercício de Fixação

### Ferramentas automatizadas

---

1. Ferramentas automatizadas são capazes de detectar falhas na lógica de negócio? Justifique sua resposta.

**As seguintes contramedidas devem ser adotadas, para evitar a ocorrência das vulnerabilidades que afetam o mecanismo de autorização e a lógica de negócio da aplicação:**



O mecanismo de autorização deve ser invocado, imediatamente antes de uma operação ser realizada.



Nunca use o cabeçalho HTTP Referer para controlar o acesso às páginas da aplicação.



Identificadores internos de objetos, como chaves primárias de tabelas em bancos de dados, por exemplo, não devem ser expostos aos usuários.

**As seguintes contramedidas devem ser adotadas, para evitar a ocorrência das vulnerabilidades que afetam o mecanismo de autorização e a lógica de negócio da aplicação:**



Não use o servidor web para fornecer, diretamente, recursos estáticos que sejam sensíveis.



Tome as decisões relacionadas a controle de acesso sempre no lado servidor da aplicação.

**Quando a aplicação precisar manipular arquivos especificados por usuários:**

Valide o nome de arquivo fornecido pelo usuário, por meio de técnicas de lista de permissão.

Garanta que todo arquivo manipulado esteja dentro do diretório alocado para este propósito.

Considere executar a aplicação dentro de um chroot.

**Utilize casos de abuso para identificar vulnerabilidades na lógica de negócio da aplicação.**

**Use mecanismos de sincronização, quando manipular dados compartilhados, como arquivos ou registros de bancos de dados, por exemplo.**

**Sempre que possível, utilize links diretos para os diversos recursos, em vez de uma página de redirecionamento.**

## Quando redirecionamento for necessário:

Verifique se a URL do recurso de destino pertence a uma lista de valores permitidos.

Não defina a URL do recurso de destino com base em dados que sejam controlados por usuários.

Se em algum caso muito especial for necessário usar dados fornecidos por usuários, para definição da URL do recurso de destino, adicione no início dela o protocolo e o nome de domínio esperados, com uma barra (“/”) no final.



# Perguntas