Security in Web Services

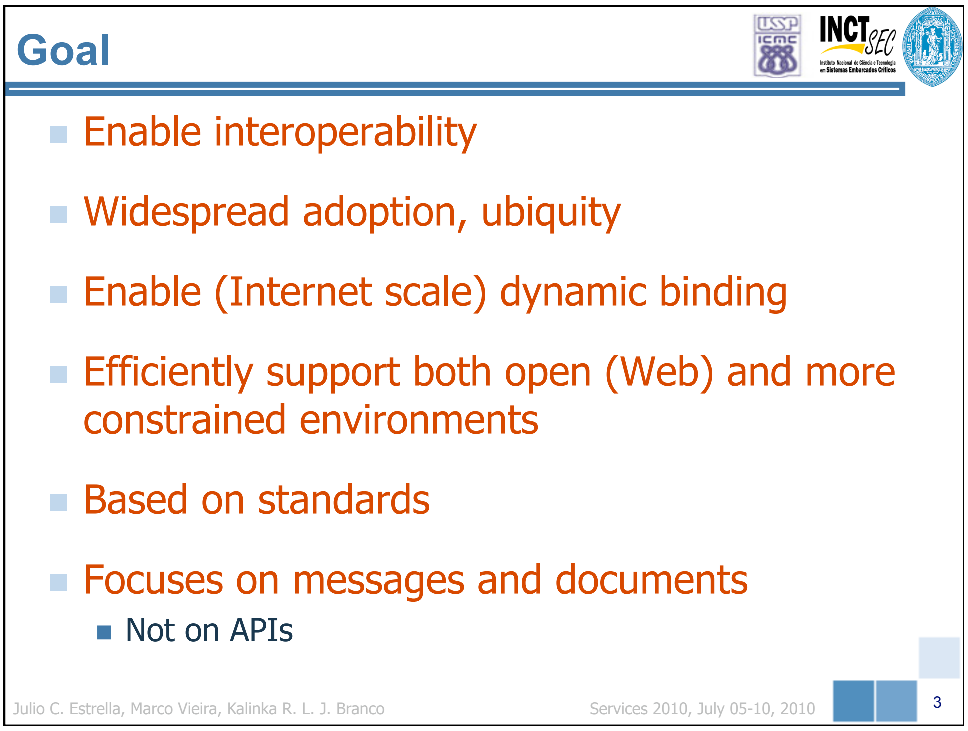
State-of-the-art and Research Opportunities

ICMC-USP
and
Univ. of Coimbra

Júlio Cezar Estrella – jcezar@icmc.usp.br
Marco Vieira - mvieira@dei.uc.pt
Kalinka R. L. J. C Branco – kalinka@icmc.usp.br

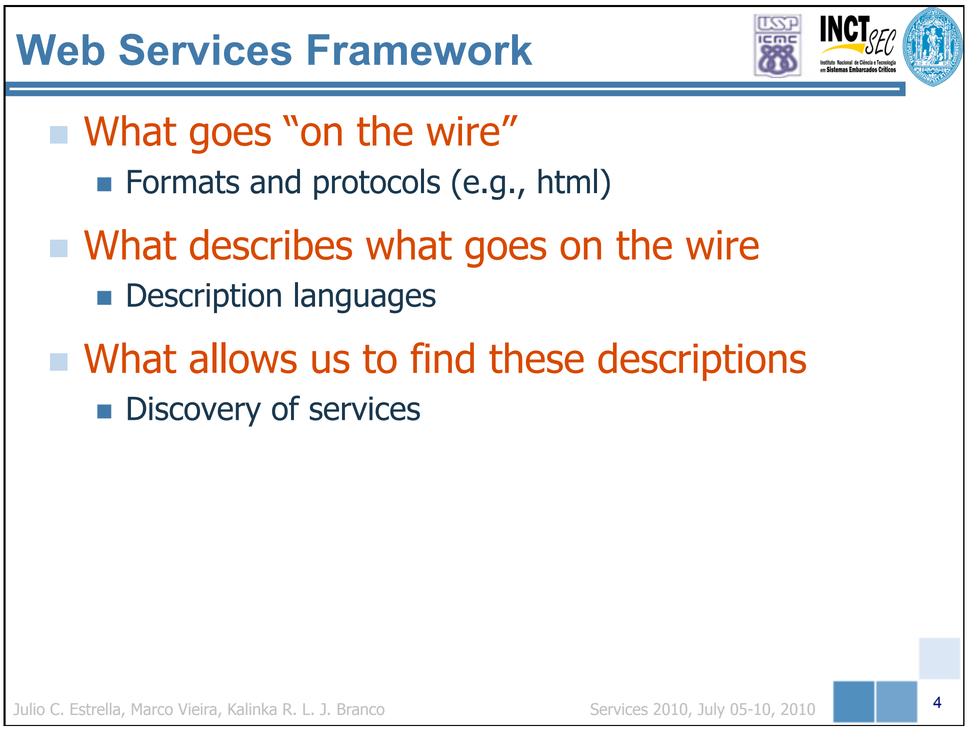
Web Services

- Strategic component in a wide range of organizations
- Components that can be remotely invoked
  - Well defined interface
- Dynamic, evolving, and supported by unreliable networks
- Deployment implies complex SW infrastructure
- Must be:
  - Robust, Secure, Reliable, Available, …



Goal

- Enable interoperability
- Widespread adoption, ubiquity
- Enable (Internet scale) dynamic binding
- Efficiently support both open (Web) and more constrained environments
- Based on standards
- Focuses on messages and documents
  - Not on APIs



Web Services Framework

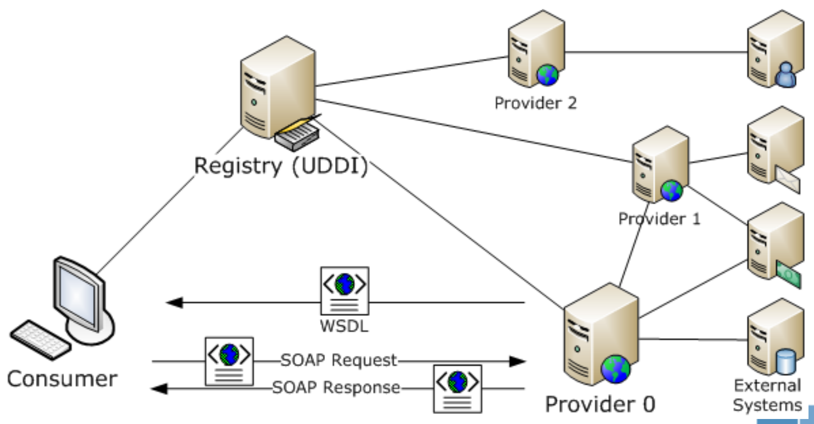
- What goes "on the wire"
  - Formats and protocols (e.g., html)
- What describes what goes on the wire
  - Description languages
- What allows us to find these descriptions
  - Discovery of services

Web Services Description Language

- Interface Description Language (IDL)
- Provides functional description of network services
- Platform independent description
- Extensible language

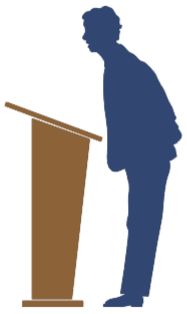
UDDI

- Universal Description, Discovery and Integration
- Defines the operation of a service registry



Outline

- Part #1: Computer Security
- Part #2: Security in Web Services
- Part #3: Security in Web Services Code
- Part #4: Open Discussion



Computer Security

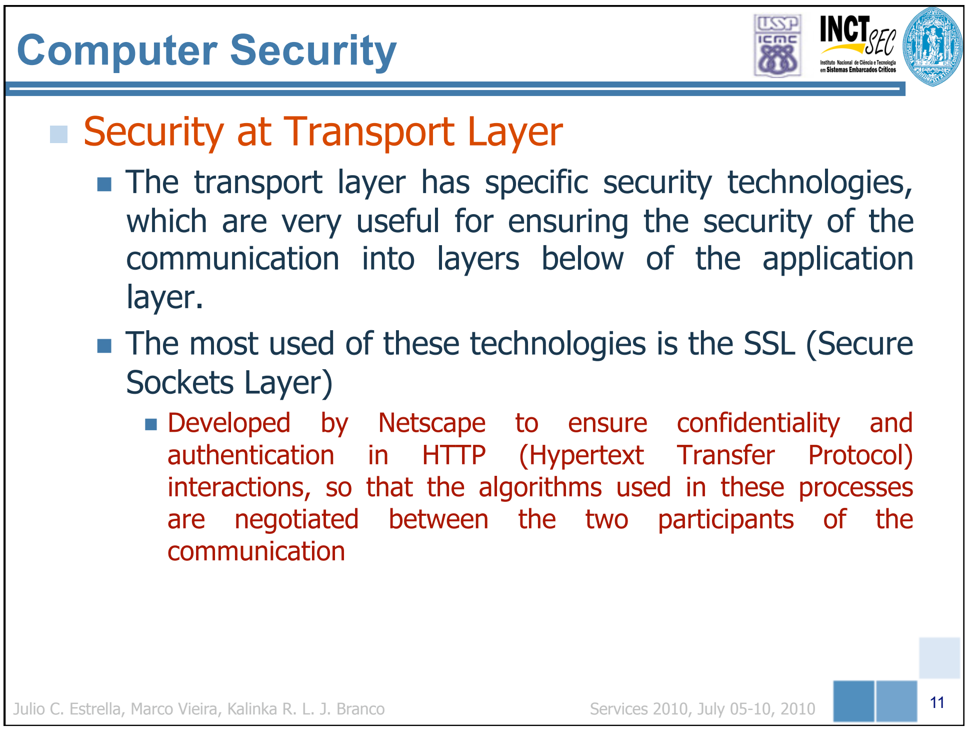
- Security became an indispensable requirement for computer systems, ensuring that access and information sharing occur without damaging the operating system and that its information will not be exposed to malicious users
- Security can be understood as a quality of service to ensure the provision of service and prevent violations in the systems

Computer Security

- Main security properties:
  - **Confidentiality**: ensures that information will be disclosed only to authorized users.
  - **Integrity**: ensures that information can not be changed, accidentally or intentionally, for users who do not have this right.
  - **Authenticity**: it ensures that the user is communicating is really who he claims to be.
  - **Non-repudiation**: ensures that the user can not deny his involvement in the occurrence of a transaction or event.
  - **Availability:** ensures that legitimate users have access to information and resources and this can not be denied access a maliciously.

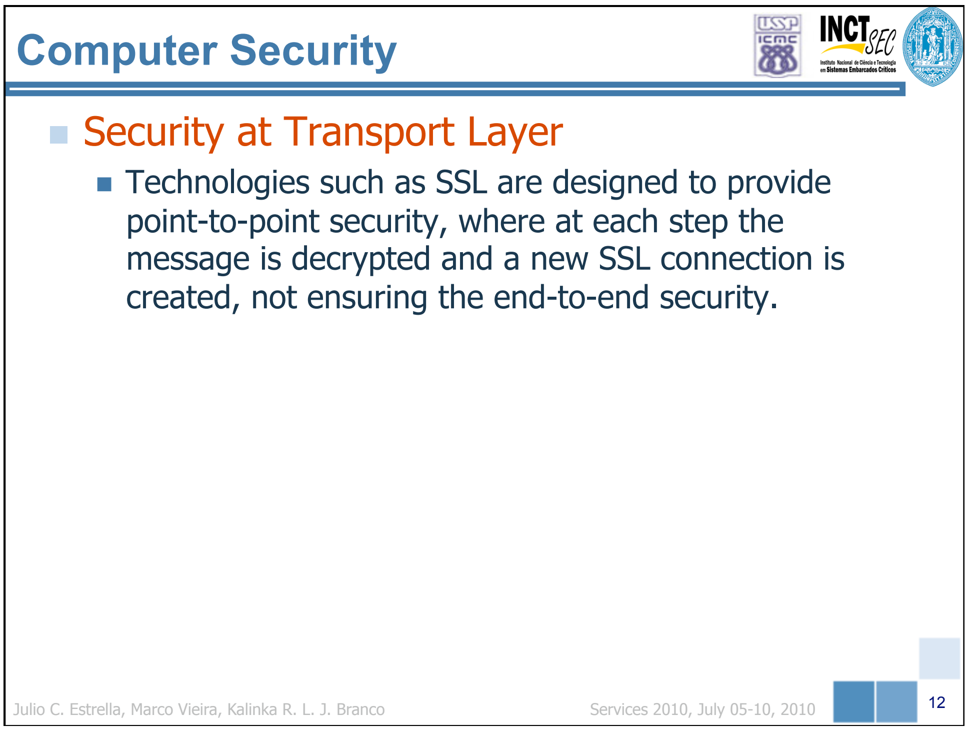
Computer Security

- Security at Network Layer
  - **IPSec Protocol:** The standard method for providing privacy, integrity and authenticity of information transferred across IP networks
  - IPSec is an extension of the IP protocol that aims to be the standard method for providing user privacy
  - IPSec combines several different technologies such as:
    - Mechanism for key exchange Diffie-Hellman
    - Public-key cryptography to sign the exchange of Diffie-Hellman key to ensure the identity of both parties and prevent attacks like man-in-the-middle
    - Encryption algorithms for large volumes of data, such as DES (Data Encryption Standard)



Computer Security

- Security at Transport Layer
  - The transport layer has specific security technologies, which are very useful for ensuring the security of the communication into layers below of the application layer.
  - The most used of these technologies is the SSL (Secure Sockets Layer)
    - Developed by Netscape to ensure confidentiality and authentication in HTTP (Hypertext Transfer Protocol) interactions, so that the algorithms used in these processes are negotiated between the two participants of the communication



Computer Security

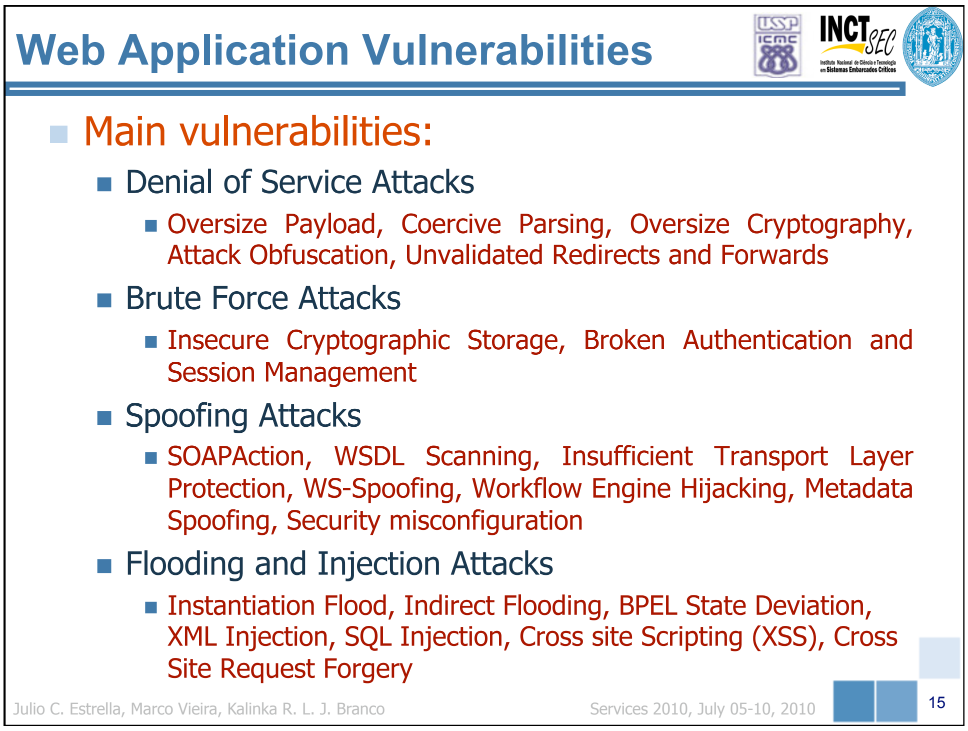
- Security at Transport Layer
  - Technologies such as SSL are designed to provide point-to-point security, where at each step the message is decrypted and a new SSL connection is created, not ensuring the end-to-end security.

Computer Security

- Security at Application Layer
  - Every layer of communication has its own unique security challenges. The application layer communication is a very weak link in terms of security because that the application layer supports many protocols which provide many vulnerabilities and access points for attackers
  - All this variability makes application-layer attacks very hard to defend against
  - Application-layer attacks are very attractive to a potential attacker because the information they seek ultimately resides within the application itself

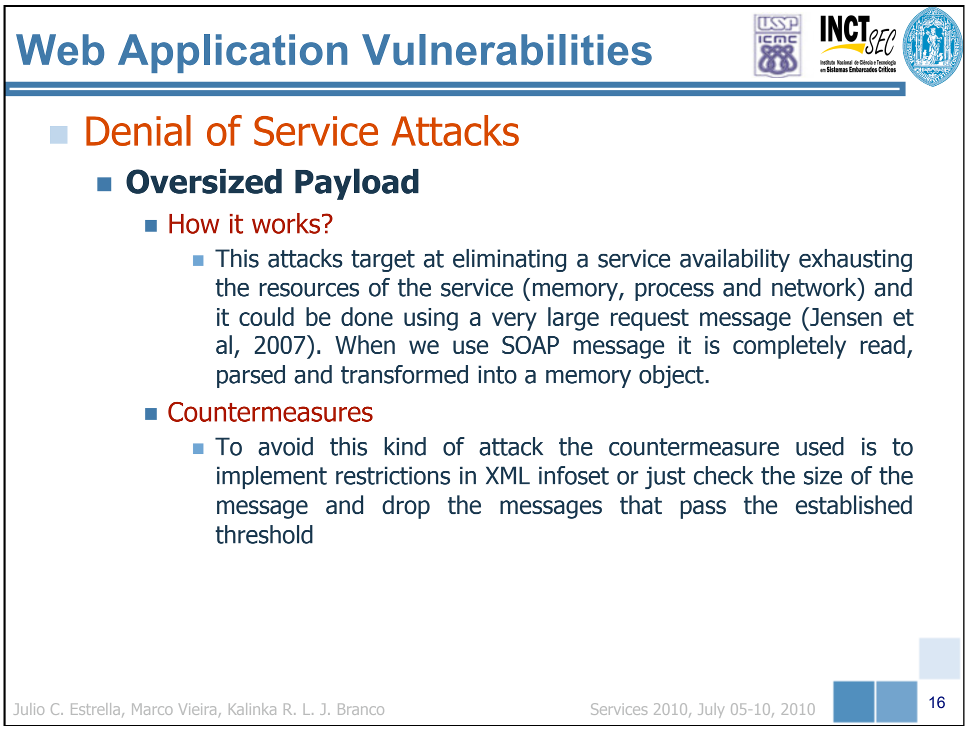
Computer Security

- The main categories of risks at the application level are as follows:
  - Web Security: Balance between security and accessibility
  - Email Security
  - Password Attack
  - Information Sniffing
  - DNS Attack
  - Instant Message Security
  - SNMP Attack
  - Operating Systems Risk



Web Application Vulnerabilities

- Main vulnerabilities:
  - Denial of Service Attacks
    - Oversize Payload, Coercive Parsing, Oversize Cryptography, Attack Obfuscation, Unvalidated Redirects and Forwards
  - Brute Force Attacks
    - Insecure Cryptographic Storage, Broken Authentication and Session Management
  - Spoofing Attacks
    - SOAPAction, WSDL Scanning, Insufficient Transport Layer Protection, WS-Spoofing, Workflow Engine Hijacking, Metadata Spoofing, Security misconfiguration
  - Flooding and Injection Attacks
    - Instantiation Flood, Indirect Flooding, BPEL State Deviation, XML Injection, SQL Injection, Cross site Scripting (XSS), Cross Site Request Forgery



Web Application Vulnerabilities

- Denial of Service Attacks
  - **Oversized Payload**
    - How it works?
      - This attacks target at eliminating a service availability exhausting the resources of the service (memory, process and network) and it could be done using a very large request message (Jensen et al, 2007). When we use SOAP message it is completely read, parsed and transformed into a memory object.
    - Countermeasures
      - To avoid this kind of attack the countermeasure used is to implement restrictions in XML infoset or just check the size of the message and drop the messages that pass the established threshold

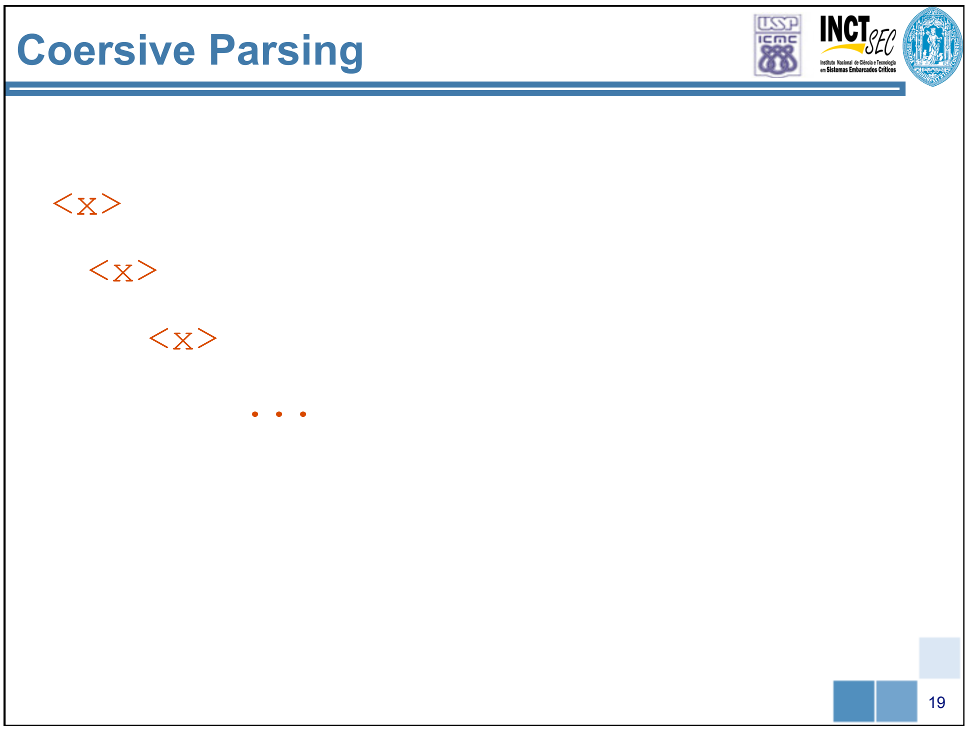
Oversize Payload

```
<Envelope>
  <Body>
     <getArrayLength>
          <item>x</item>
          <item>x</item>
          <item>x</item>
               ...
     </getArrayLength>
  </Body>
</Envelope>
```

17

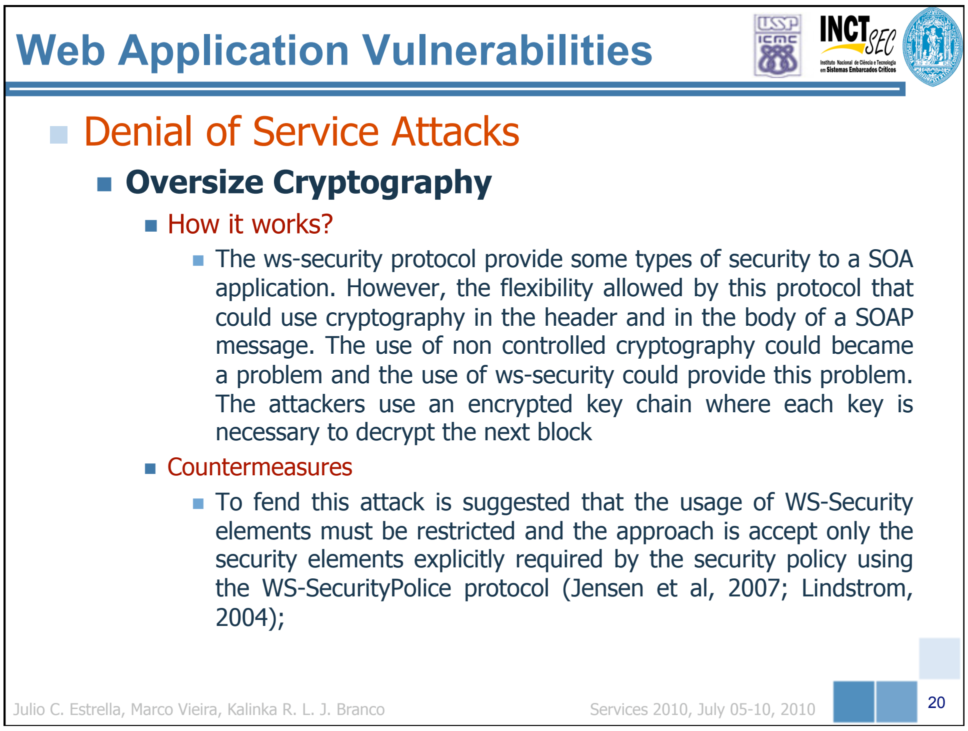
Web Application Vulnerabilities

- Denial of Service Attacks
  - **Coercive Parsing**
    - How it works?
      - Parsing the SOAP message is necessary in the Web Service request process. This becomes vulnerability when we can use a large number of namespace. The attackers can explore this vulnerability to provide a Denial of Service because the massively reduced of the service availability because the attack uses a continuous sequence of opening tags providing complex and nested XML documents
    - Countermeasures
      - This kind of attack is difficult to be preventing because the XML not limit the number of namespace declaration (Jensen et al, 2007);



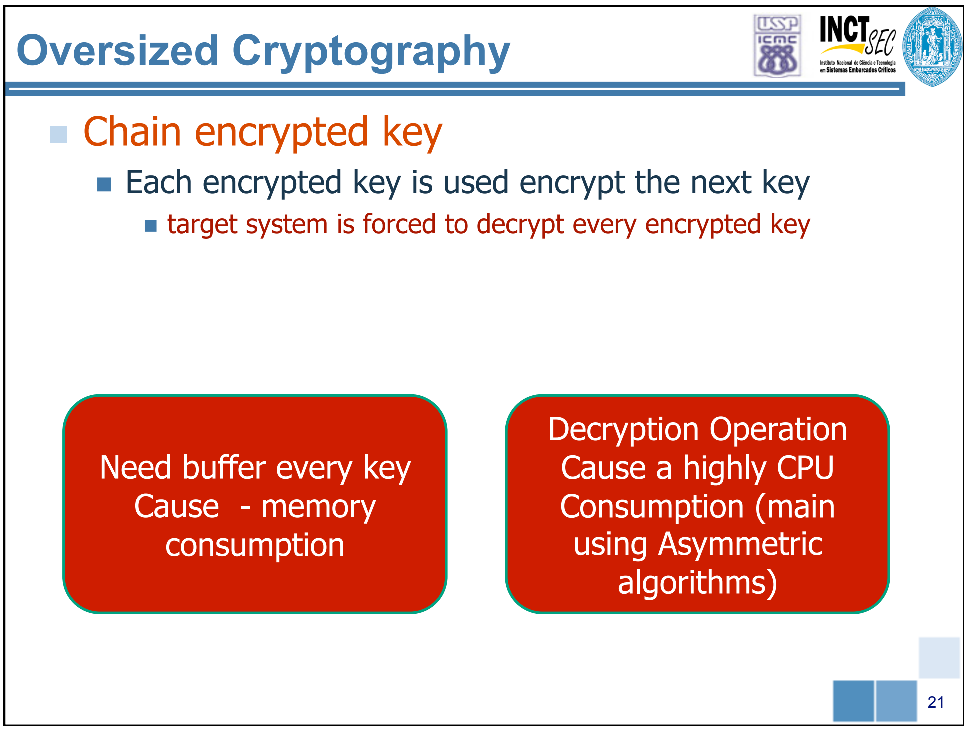
Coersive Parsing

```
<x>
  <x>
     <x>
          . . .
```



Web Application Vulnerabilities

- Denial of Service Attacks
  - **Oversize Cryptography**
    - How it works?
      - The ws-security protocol provide some types of security to a SOA application. However, the flexibility allowed by this protocol that could use cryptography in the header and in the body of a SOAP message. The use of non controlled cryptography could became a problem and the use of ws-security could provide this problem. The attackers use an encrypted key chain where each key is necessary to decrypt the next block
    - Countermeasures
      - To fend this attack is suggested that the usage of WS-Security elements must be restricted and the approach is accept only the security elements explicitly required by the security policy using the WS-SecurityPolice protocol (Jensen et al, 2007; Lindstrom, 2004);



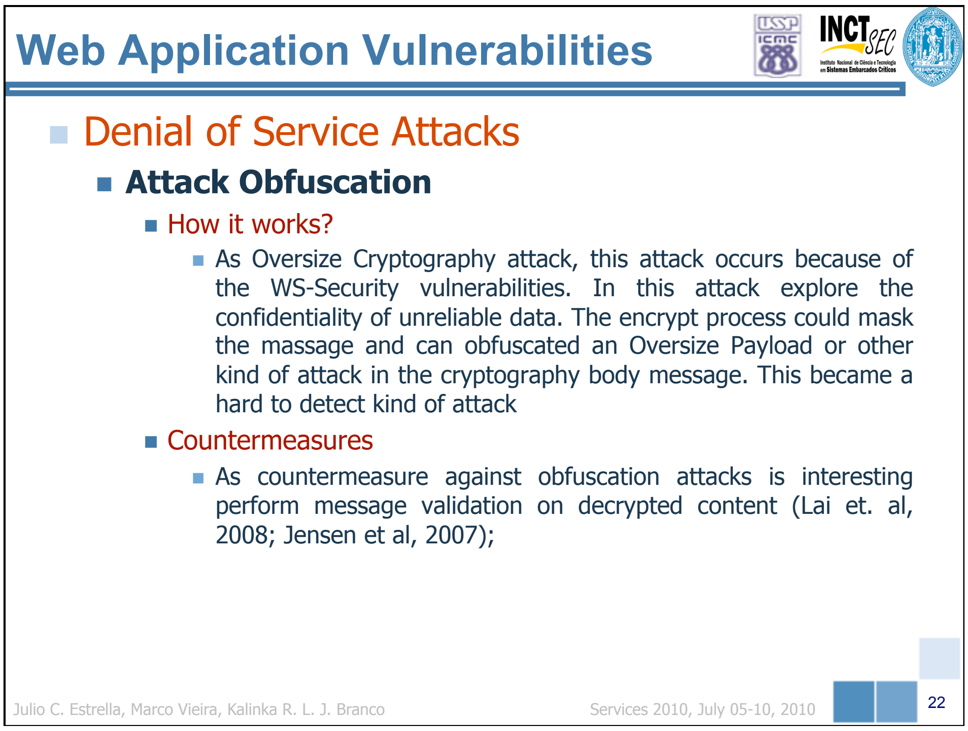
Oversized Cryptography

- Chain encrypted key
  - Each encrypted key is used encrypt the next key
    - target system is forced to decrypt every encrypted key

Need buffer every key
Cause - memory consumption

Decryption Operation
Cause a highly CPU Consumption (main using Asymmetric algorithms)

21



Web Application Vulnerabilities

- Denial of Service Attacks
  - **Attack Obfuscation**
    - How it works?
      - As Oversize Cryptography attack, this attack occurs because of the WS-Security vulnerabilities. In this attack explore the confidentiality of unreliable data. The encrypt process could mask the massage and can obfuscated an Oversize Payload or other kind of attack in the cryptography body message. This became a hard to detect kind of attack
    - Countermeasures
      - As countermeasure against obfuscation attacks is interesting perform message validation on decrypted content (Lai et. al, 2008; Jensen et al, 2007);

Julio C. Estrella, Marco Vieira, Kalinka R. L. J. Branco Services 2010, July 05-10, 2010 22

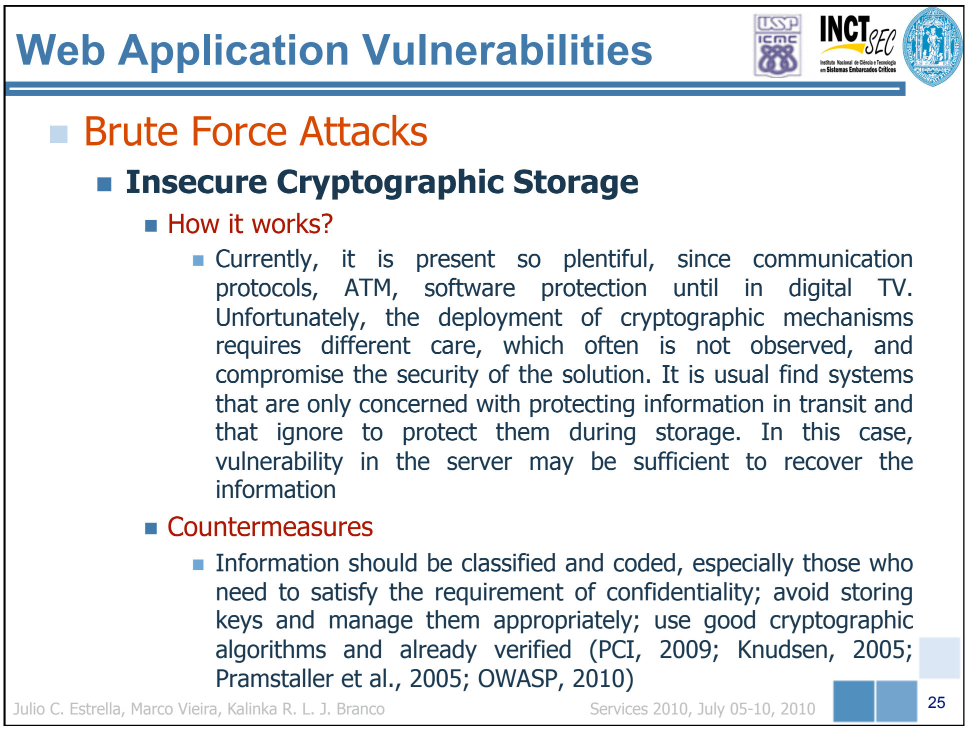
Attack Obfuscation

- A single SOAP message was sent to the server.
  - message containing an encrypted and signed body - 1 MB
    - Cause - full CPU load for 23 seconds and out-of-memory execution
  - unencrypted messages - 20 MB
    - Processed in one second.

Can also hide an Oversize Payload attack

Web Application Vulnerabilities

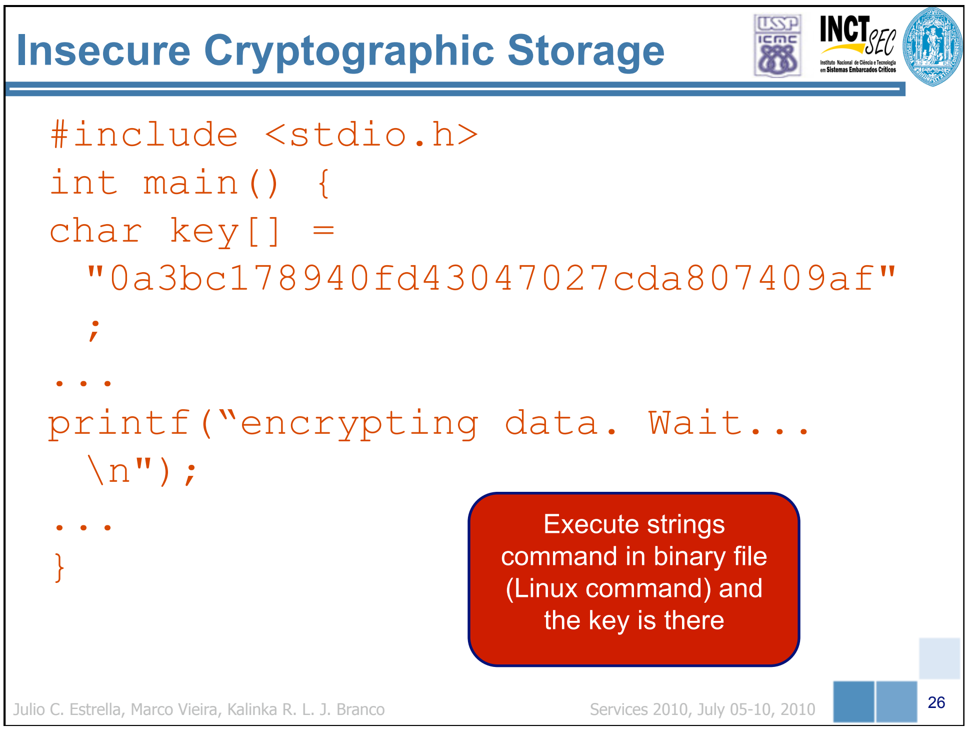
- Denial of Service Attacks
  - **Unvalidated Redirects and Forwards**
    - How it works?
      - This kind of vulnerability, unvalidated redirects, is related with the phishing or malware sites that could result in a Denial of Service attack or a unauthorized access to the client resources. The attackers use this vulnerability to redirect clients (victims) to unauthorized pages.
    - Countermeasures
      - Disabling the redirects and forwards (OWASP,2010).



Web Application Vulnerabilities

- Brute Force Attacks
  - **Insecure Cryptographic Storage**
    - How it works?
      - Currently, it is present so plentiful, since communication protocols, ATM, software protection until in digital TV. Unfortunately, the deployment of cryptographic mechanisms requires different care, which often is not observed, and compromise the security of the solution. It is usual find systems that are only concerned with protecting information in transit and that ignore to protect them during storage. In this case, vulnerability in the server may be sufficient to recover the information
    - Countermeasures
      - Information should be classified and coded, especially those who need to satisfy the requirement of confidentiality; avoid storing keys and manage them appropriately; use good cryptographic algorithms and already verified (PCI, 2009; Knudsen, 2005; Pramstaller et al., 2005; OWASP, 2010)

Julio C. Estrella, Marco Vieira, Kalinka R. L. J. Branco Services 2010, July 05-10, 2010 25

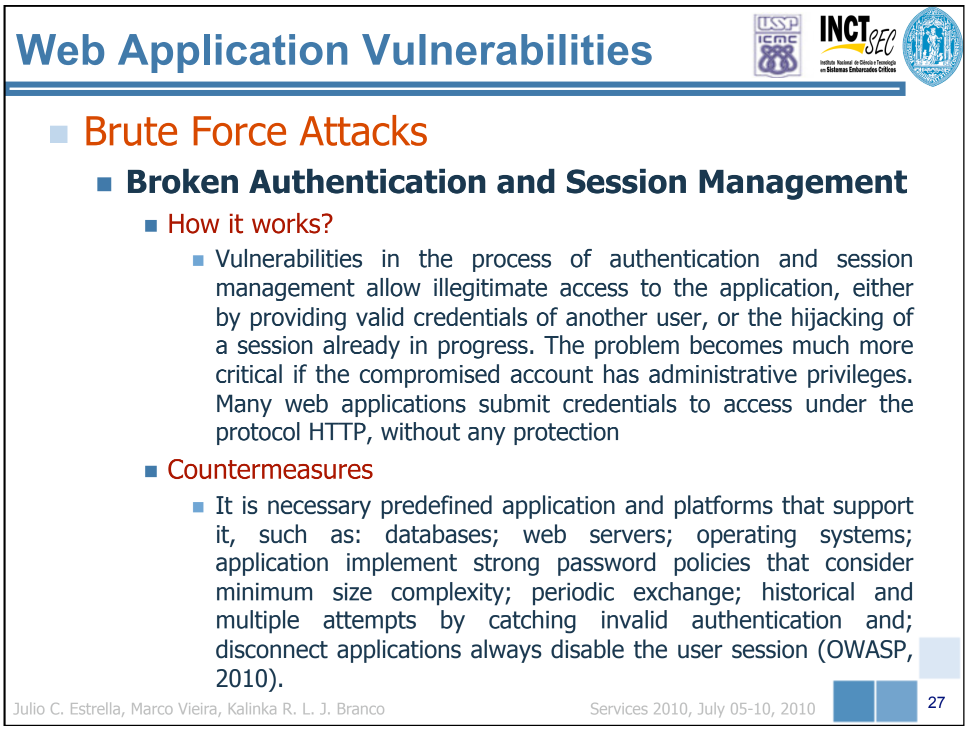


Insecure Cryptographic Storage

```
#include <stdio.h>
int main() {
char key[] =
  "0a3bc178940fd43047027cda807409af"
  ;
...
printf(“encrypting data. Wait...
  \n");
...
}
```

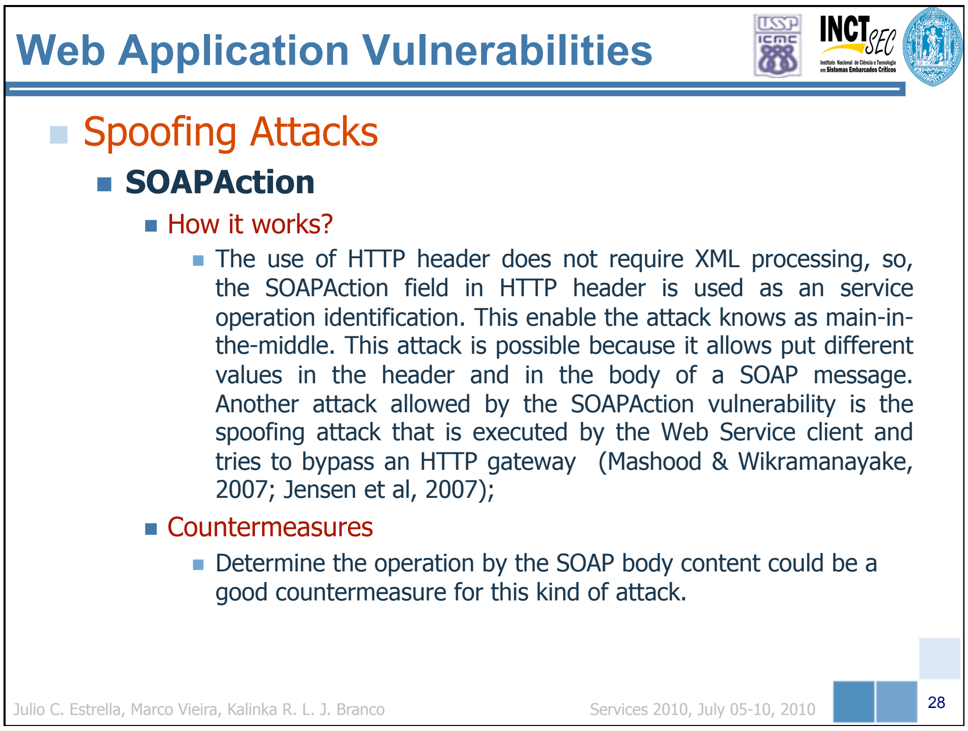
Execute strings command in binary file (Linux command) and the key is there

Julio C. Estrella, Marco Vieira, Kalinka R. L. J. Branco Services 2010, July 05-10, 2010 26



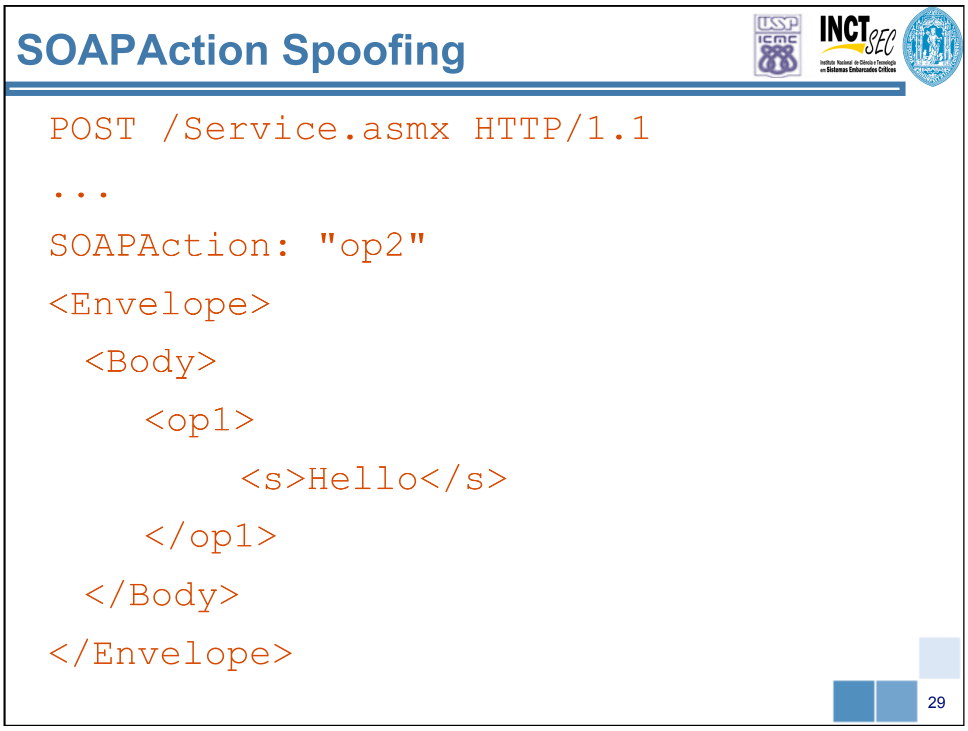
Web Application Vulnerabilities

- Brute Force Attacks
  - **Broken Authentication and Session Management**
    - How it works?
      - Vulnerabilities in the process of authentication and session management allow illegitimate access to the application, either by providing valid credentials of another user, or the hijacking of a session already in progress. The problem becomes much more critical if the compromised account has administrative privileges. Many web applications submit credentials to access under the protocol HTTP, without any protection
    - Countermeasures
      - It is necessary predefined application and platforms that support it, such as: databases; web servers; operating systems; application implement strong password policies that consider minimum size complexity; periodic exchange; historical and multiple attempts by catching invalid authentication and; disconnect applications always disable the user session (OWASP, 2010).



Web Application Vulnerabilities

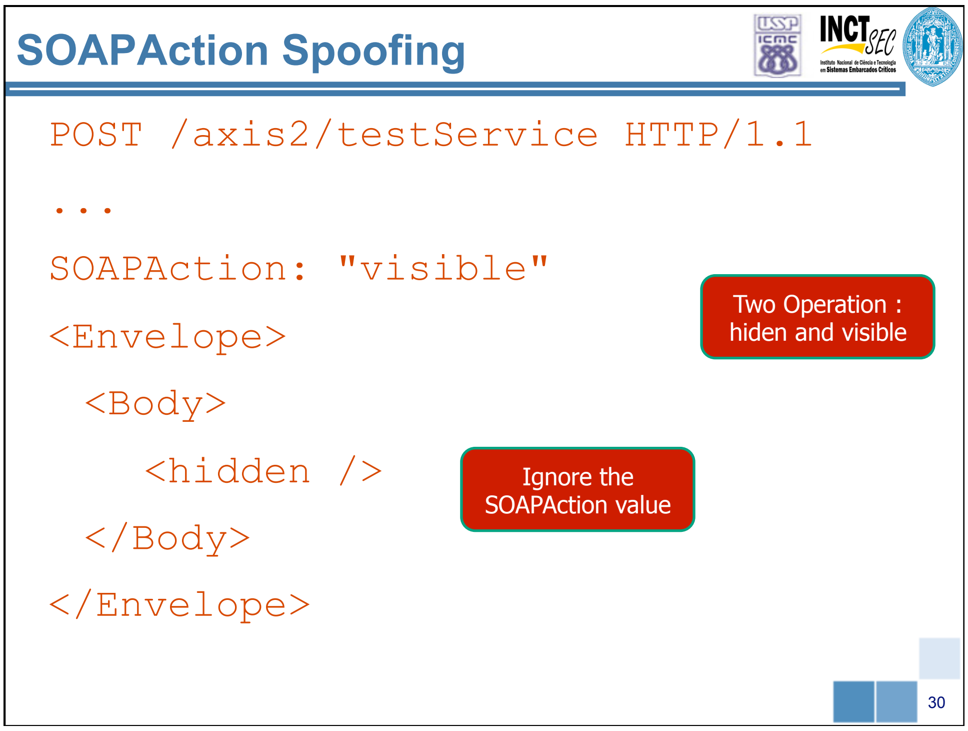
- Spoofing Attacks
  - **SOAPAction**
    - How it works?
      - The use of HTTP header does not require XML processing, so, the SOAPAction field in HTTP header is used as an service operation identification. This enable the attack knows as main-in-the-middle. This attack is possible because it allows put different values in the header and in the body of a SOAP message. Another attack allowed by the SOAPAction vulnerability is the spoofing attack that is executed by the Web Service client and tries to bypass an HTTP gateway (Mashood & Wikramanayake, 2007; Jensen et al, 2007);
    - Countermeasures
      - Determine the operation by the SOAP body content could be a good countermeasure for this kind of attack.



SOAPAction Spoofing

```
POST /Service.asmx HTTP/1.1
...
SOAPAction: "op2"
<Envelope>
  <Body>
     <op1>
          <s>Hello</s>
     </op1>
  </Body>
</Envelope>
```

29



SOAPAction Spoofing

```
POST /axis2/testService HTTP/1.1
...
SOAPAction: "visible"
<Envelope>
  <Body>
     <hidden />
  </Body>
</Envelope>
```

Two Operation : hiden and visible

Ignore the SOAPAction value

30

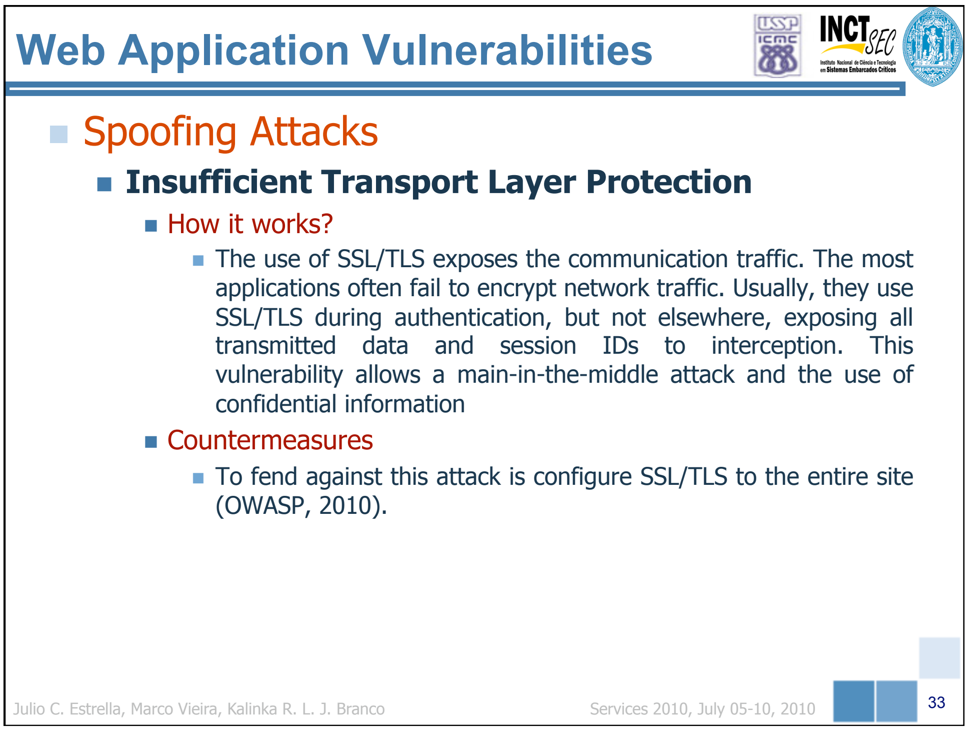
Web Application Vulnerabilities

- Spoofing Attacks
  - **WSDL Scanning**
    - How it works?
      - This class of attack provides a scanning in all the operations generated in WSDL. Once this document is open an external client could know all the internal operations and can invoke them
    - Countermeasures
      - To avoid this kind of attack is to provide a separate access WSDL to external clients that contain the external operations only, so the malicious user could try to guess the omitted operations and call them (Jensen et al, 2007);

WSDL Scanning

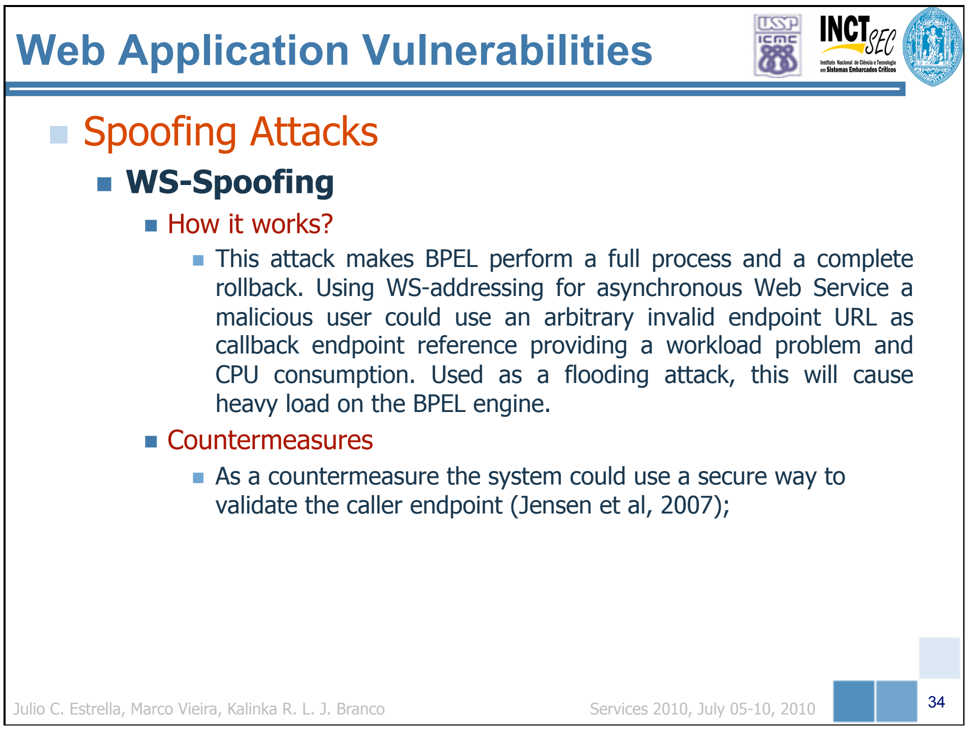
- Operation offered by Web Service
  - `(SendOrder) and (adminOrders)`

Attackers could easyly find all the methods



Web Application Vulnerabilities

- Spoofing Attacks
  - **Insufficient Transport Layer Protection**
    - How it works?
      - The use of SSL/TLS exposes the communication traffic. The most applications often fail to encrypt network traffic. Usually, they use SSL/TLS during authentication, but not elsewhere, exposing all transmitted data and session IDs to interception. This vulnerability allows a main-in-the-middle attack and the use of confidential information
    - Countermeasures
      - To fend against this attack is configure SSL/TLS to the entire site (OWASP, 2010).



Web Application Vulnerabilities

- Spoofing Attacks
  - **WS-Spoofing**
    - How it works?
      - This attack makes BPEL perform a full process and a complete rollback. Using WS-addressing for asynchronous Web Service a malicious user could use an arbitrary invalid endpoint URL as callback endpoint reference providing a workload problem and CPU consumption. Used as a flooding attack, this will cause heavy load on the BPEL engine.
    - Countermeasures
      - As a countermeasure the system could use a secure way to validate the caller endpoint (Jensen et al, 2007);

WS-Addressing Spoofing

- The use of WS-Addressing for asynchronous Web Service calls raises a lot of attack possibilities, which all have in common that they use modified callback endpoint references.

35

Web Application Vulnerabilities

- Spoofing Attacks
  - **Workflow Engine Hijacking**
    - How it works?
      - This kind of attack uses the WS-Addressing but with the attackers point to the target system URL. So the target system (the Web Service) receives a heavy amount of requests, providing a Denial of Service.
    - Countermeasures
      - As a countermeasure the system could use a secure way to validate the caller endpoint (Jensen et al, 2007);

Workflow Engine Hijacking

- This attack uses WS-Addressing spoofing again, but it points the attacker's endpoint URL to an existing target system, running a real service at the URL specified

Web Application Vulnerabilities

- Spoofing Attacks
  - **Metadata Spoofing**
    - How it works?
      - All the information in a Web Service application is exchanged in metadata. This metadata is exchanged using HTTP protocol allowing spoofing these metadata. The WSDL spoofing and Security Policy Spoofing is the most representative attacks of this class. In the first one the problem is the modification of the network endpoints (allowing the main-in-the-middle attack) and in the second the security policy could be changed
    - Countermeasures
      - As a countermeasure all the metadata contents could be carefully checked for authenticity (despite the mechanisms for securing metadata are not standardized) (Lai et. al, 2008; Jensen et al, 2007; Lindstrom, 2004);

Metadata Spoofing

- Supposably most promising for WSDL Spoofing is the modification of the network endpoints and the references to security policies. A modified endpoint enables the attacker to easily establish a man-in-the-middle attack for eavesdropping or data modification.

39

Web Application Vulnerabilities

- Spoofing Attacks
  - **Security Misconfiguration**
    - How it works?
      - Security depends on having a secure configuration defined for the application and the entire web service platform. All the stack application and web service platform must have a secure configuration. This vulnerability allows an attacker, using an automated scanner, detect missing patches and use default account and hijacking the target system**.**
    - Countermeasures
      - To fend the system against this kind of attack strong network architecture must be provide. This architecture must provide good separation and security between components. Automatic and periodic scanning must be performed to help detect future misconfigurations or missing patches (OWASP, 2010);

Web Application Vulnerabilities

- Flooding and Injection Attacks
  - **Instantiation Flood**
    - How it works?
      - BPEL start process when a new message arrives and creates an instance. The process immediately stars its execution based on the description instruction. This allows a kind of attack that creates new process and starts its execution, what improve the load of the engine.
    - Countermeasures
      - Fending such flooding attacks can only be achieved by identification and rejection of semantically invalid requests of semantically invalid requests. This kind of attack is consider a non trivial avoid class because is really difficult to identify an invalid message (Jensen et al, 2007; Lindstrom, 2004);

Web Application Vulnerabilities

- Flooding and Injection Attacks
  - **Indirect Flooding**
    - How it works?
      - This attack uses BPEL like an intermediate to provide a flooding attack in the target system making a lot of requests (calls). In this case, the BPEL engine will undergo a heavy load itself, but it merely will cause an equally heavy load on the target system
    - Countermeasures
      - This kind of attack could not be fended using WS-Security once the connection between the target system and BPEL is a trustful connection (Lai et. al, 2008; Jensen et al, 2007; Lindstrom, 2004)

Indirect Flooding

- The idea of this attack is to use the BPEL engine as intermediate for an attack on a target system behind the BPEL engine.

Web Application Vulnerabilities

- Flooding and Injection Attacks
  - **BPEL State Deviation**
    - How it works?
      - BPEL provides, with its vulnerability, this kind of attack. BPEL start endpoints communication. This attack provide a redundant work because service endpoints using messages that are correct regarding their message structure but not properly correlated to any existing process will be discarded within the BPEL. This cause a heavy amount of load provided by the many ports that could be request.
    - Countermeasures
      - Use specific firewall that could fend both correlation-invalid and state-invalid messages

BPEL State Deviation

- Suppose a BPEL engine running one BPEL process.
  - sequence of two receive activities: `first` and `second`, with only `first` initiating a new process instance.
- The attack used SOAP messages invoking operation `second` and

containing correlation properties that did not match to any running process instance.

- sequence of 1000 messages (0.5 MB) - the messages are discarded

Cause – Memory consumption and full CPU use

45

Web Application Vulnerabilities

- Flooding and Injection Attacks
  - **XML Injection**
    - How it works?
      - A XML Injection attack class intends modify the XML structure of a SOAP message by inserting new XML fragment containing XML tags, This situation may lead to unauthorized access to the protected resources.
    - Countermeasures
      - A countermeasure is try to provide a strict schema validation on the SOAP message aiming discover a invalid data type or message and reject them (Jensen et al, 2007; Knap & Mlýnková, 2009);

XML Injection

```
<Envelope>
  <Body>
      <HelloWorld>
             <a> <b>1</b> </a>
             <b> 2 </b>
      </HelloWorld>
  </Body>
</Envelope>
```

Web Application Vulnerabilities

- Flooding and Injection Attacks
  - **SQL Injection**
    - How it works?
      - SQL injection is the most common vulnerability in web applications nowadays (OWASP, 2010). This attack consists in the injection of a SQL code and parameters in the fields and application with the aim of which is performed on the data layer. In the simplest attacks, you can perform any operation on the database, limited to the privileges of the account that performs the access
    - Countermeasures
      - To prevent web applications vulnerable to SQL injection attacks has been to consider all information provided by users as malicious. Then, before processing it, check whether they agree with known values as valid

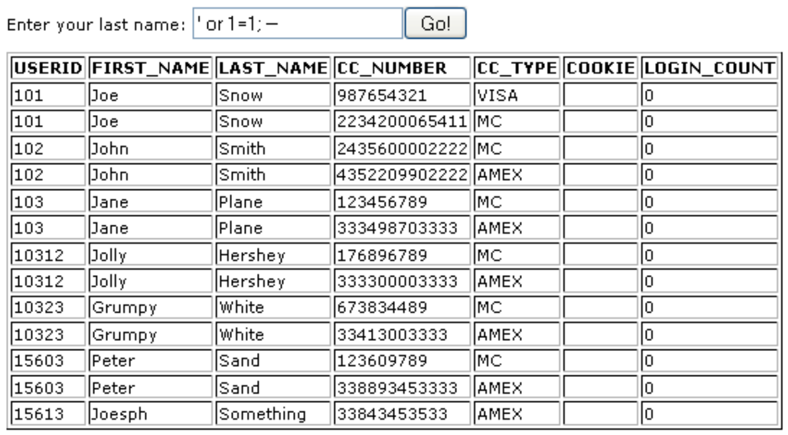
SQL Injection

sComando = "select * from user_data
  where last_name = '" +
  inputLastName + "'"

So...

select * from user_data
where last_name = '' or 1=1; --'

SQL Injection



Enter your last name: ' or 1=1; – Go!

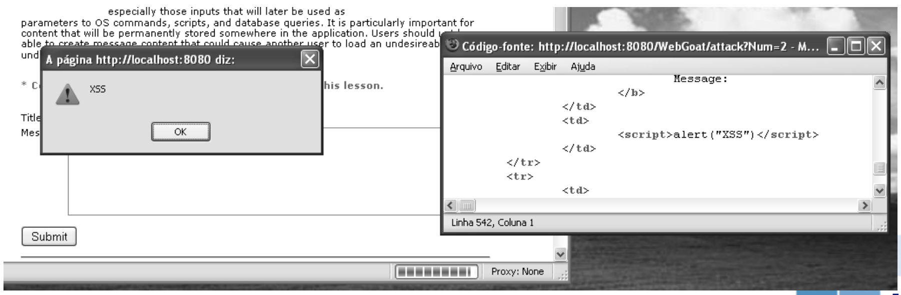
USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	White	673834489	MC		0
10323	Grumpy	White	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

Web Application Vulnerabilities

- Flooding and Injection Attacks
  - **Cross Site Scripting (XSS)**
    - How it works?
      - Also known as XSS, is the flaw most commonly found in web applications. This attack allows using a vulnerable application to carry malicious code, usually written in Javascript by the browser of another user. Given the relationship of trust established between the browser and server, that means that the code received is legitimate and therefore allows sensitive information such as the session identifier of the user. With this, a malicious user can hijack the session of the person attacked (OWASP, 2010; Uto & Melo, 2009; Stuttard and Pinto, 2007, Howard et al., 2005; Fogie et al., 2007)
    - Countermeasures
      - A countermeasure is satisfied when all information provided by users is considered malicious, and thus before processing it, make sure it complies with valid values for the field or parameter

Cross Site Scripting (XSS)

http://localhost:8080/WebGoat/attack?Screen=33&menu=900&search_name=X<script>alert(%22XSS%20Refletido%22)</script>&action=FindProfile


especially those inputs that will later be used as
parameters to OS commands, scripts, and database queries. It is particularly important for
content that will be permanently stored somewhere in the application. Users should
able to create message content that could cause another user to load an undesireab
und

* C his lesson.

Title
Mes

A página http://localhost:8080 diz:

XSS

Código-fonte: http://localhost:8080/WebGoat/attack?Num=2 - M...

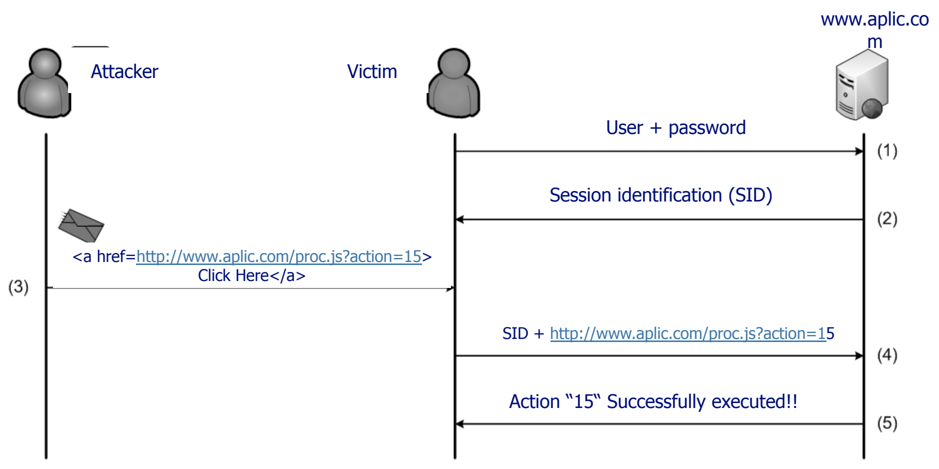
```
                              Message:
                    </b>
               </td>
               <td>
                    <script>alert("XSS")</script>
               </td>
          </tr>
          <tr>
               <td>
```

Linha 542, Coluna 1

Web Application Vulnerabilities

- Flooding and Injection Attacks
  - **Cross Site Request Forgery**
    - How it works?
      - CSRF is an attack that takes advantage of a user session already established in the vulnerable application to perform automated actions without the knowledge and consent of the victim. Examples of operations that can be performed range from simple closure of the session until the transfer of funds in a banking application. This attack is also known as XSRF, Session Riding and One-Click attack.
    - Countermeasures
      - The main countermeasure against CSRF attacks is to use information related to each session in the URLs and check them when a request is made. This prevents one knows in advance the format of the URL, it is necessary to mount the attack. (Uto & Melo, 2009; Stuttard and Pinto, 2007, Howard et al., 2005; Fogie et al., 2007).

Cross Site Request Forgery


www.aplic.com
Attacker
Victim
User + password
(1)
Session identification (SID)
(2)
<a href=http://www.aplic.com/proc.js?action=15>
Click Here</a>
(3)
SID + http://www.aplic.com/proc.js?action=15
(4)
Action "15" Successfully executed!!
(5)

Questions?

Security in Web Services

Protocols and Mechanisms

Security in Web Services

- For Web services to be widely adopted it is essential that their use is safe, since no company wants to risk exposing their applications and business flows with no damage
- For this reason, organizations as W3C, OASIS and WS-I are proposing specifications in order to make these services more secure

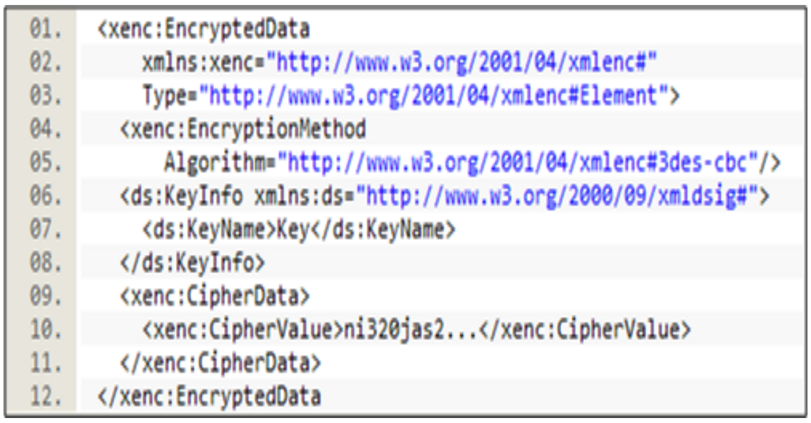
Security in Web Services

- Some specifications to deal with security in Web Services:
  - XML Encryption
  - XML Signature
  - WS-Security
  - WS-Policy
  - WS-Security Policy

XML Encryption

- Defines a way to encrypt data and represents the result in a structured way as an XML document, ensuring the confidentiality requirement
- Provides security end-to-end for applications that need to exchange data in XML format in a secure way, without concern that they can have their contents revealed and misused by third parties

XML Encryption


```
<xenc:EncryptedData
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Type="http://www.w3.org/2001/04/xmlenc#Element">
  <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:KeyName>Key</ds:KeyName>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>ni320jas2...</xenc:CipherValue>
  </xenc:CipherData>
</xenc:EncryptedData
```

XML Signature

- Specifies a process for generating and validating digital signatures expressed in XML, ensuring the integrity and authenticity in XML documents

```
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="...">
      <ds:Transforms .../>
      <ds:DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>pkKoUrEWaYhQhJKpfx9...</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>nnzElamjClN aMA0...</ds:SignatureValue>
  <ds:KeyInfo>...</ds:KeyInfo>
</ds:Signature>
```

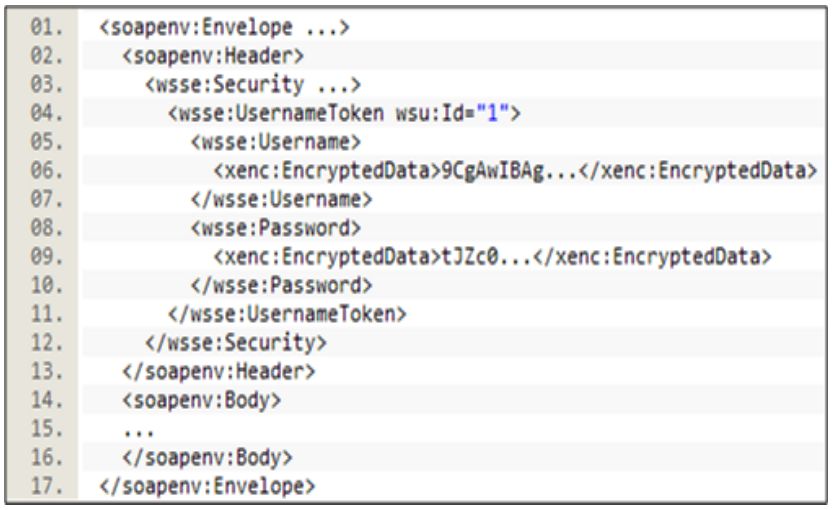
WS-Security

- Proposes a set of extensions to SOAP messages in order to implement secure Web services
- Provides ways of encryption and digital signature in a SOAP messages using XML Encryption and XML Signature, besides to include security credentials

WS-Security

- The purpose is to ensure security end-to-end in message-level with regard to three main points (Chou and Yurov, 2005):
  - Confidentiality of the message
  - Message integrity
  - Credentials Security
    - Security credentials are defined in WS-Security and access mechanisms and methods used for authentication and authorization (passwords, X.509 Certificate)
    - The main security credentials are (Mogollon, 2008):
      - UsernameToken
      - BinarySecureToken
      - XML Tokens

WS-Security



```
<soapenv:Envelope ...>
  <soapenv:Header>
    <wsse:Security ...>
      <wsse:UsernameToken wsu:Id="1">
        <wsse:Username>
          <xenc:EncryptedData>9CgAwIBAg...</xenc:EncryptedData>
        </wsse:Username>
        <wsse:Password>
          <xenc:EncryptedData>tJZc0...</xenc:EncryptedData>
        </wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
  ...
  </soapenv:Body>
</soapenv:Envelope>
```

WS-Policy

- Provides a model for the description of general purpose policies
- It has a flexible and extensible grammar that allows you to specify requirements and abilities to the environment of Web services
- A service provider sets its policy to determine the conditions that must be met for your service to be provided to a client.
- Thus, after reviewing the policy, a potential client is able to decide whether or not to access the service

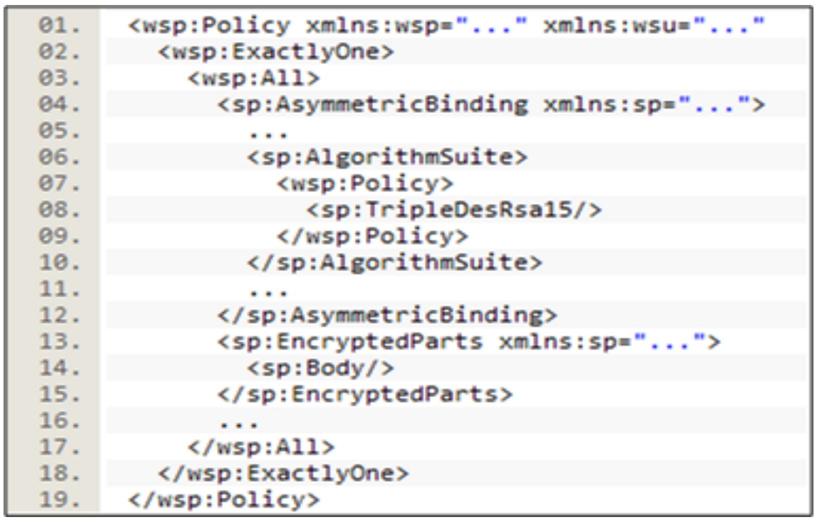
WS-Policy

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:SecurityBindingAssertion/>
      <sp:ProtectionAssertion/>
      <cs:CustomAssertion/>
    </wsp:All>
    <wsp:All>
      <sp:SomeOtherSecurityBindingAssertion/>
      <sp:SomeOtherProtectionAssertion/>
      <cs:SomeOtherCustomAssertion/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

WS-Security Policy

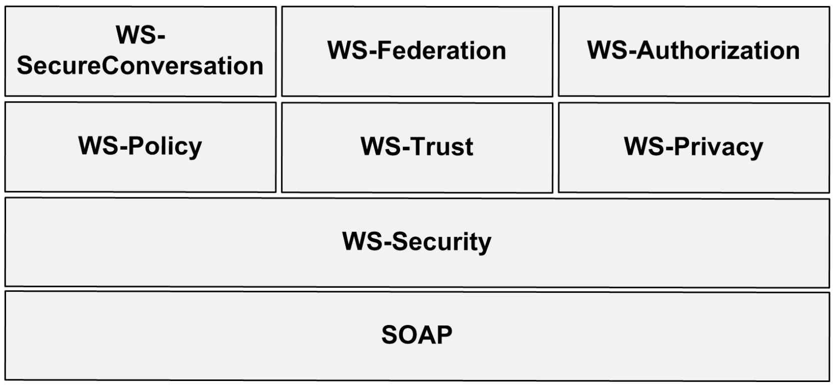
- Extension of WS-Policy
- Provides policy assertions related to security that can be used with WS-Policy
- Through these assertions can be specified the requirements, capabilities and limitations of an implementation of Secure Web services

WS-Security Policy



```
<wsp:Policy xmlns:wsp="..." xmlns:wsu="..."
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:AsymmetricBinding xmlns:sp="...">
        ...
        <sp:AlgorithmSuite>
          <wsp:Policy>
            <sp:TripleDesRsa15/>
          </wsp:Policy>
        </sp:AlgorithmSuite>
        ...
      </sp:AsymmetricBinding>
      <sp:EncryptedParts xmlns:sp="...">
        <sp:Body/>
      </sp:EncryptedParts>
      ...
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Other Security Specifications for WS



Other Security Specifications for WS

- WS-Trust
  - Standard of OASIS (OASIS, 2007a) that defines how trust relationships are established, allowing Web services interoperate safely.
- WS-Secure Conversation
  - OASIS standard (OASIS, 2007b) which provides mechanisms for establishing and identifying a security context, i.e., it allows the creation of sessions where several SOAP messages can be exchanged without the need to assess the authentication and authorization for each (Holgersson and Soderstrom, 2005).

Other Security Specifications for WS

- WS-Privacy
  - Proposal not yet standardized that aims to communicate the privacy policies set by organizations that are deploying Web services (Nordbotten, 2009).
- WS-Federation
  - Aims to describe how to manage and broker the trust relationship in a heterogeneous federated environment including support for federated identities. Proposal in progress by OASIS (OASIS, 2008)
- WS-Authorization:
  - proposal not yet standard that aims to describe how access policies for a Web service are specified and managed (Nordbotten, 2009).

Questions?

Security in Web Services Code

Principles

Outline revisited

- What are code vulnerabilities?
- Vulnerabilities detection approaches and tools
- Vulnerabilities in public Web Services
- How effective are vulnerability detection tools?
- Can we do better?
- Research challenges and opportunities

What is a vulnerability?

- A vulnerability is a weakness that may allow attackers to gain access to the system or info
  - [Stock07]
- There are many causes:
  - Complexity
  - Password and privileges management flaws
  - Operating system design flaws
  - Software bugs
  - Unchecked user input
  - …

Vulnerabilities in Web Applications (1)

- SQL Injection
  - It is possible to alter the construction of backend SQL statements
  - An attacker can read or modify database data and
  - In some cases, execute database administration operations or commands in the system
- XPath Injection
  - It is possible to modify an XPath query to be parsed in a way differing from the programmer's intention
  - Attackers may gain access to information in XML documents

Vulnerabilities in Web Applications (2)

- Code Execution
  - It is possible to manipulate the application inputs to trigger server-side code execution
  - An attacker can exploit this vulnerability to execute malicious code in the server machine
- Buffer Overflow
  - It is possible to manipulate inputs in such a way that causes buffer allocation problems
    - Including overwriting of parts of the memory
  - An attacker can exploit this causing Denial of Service
    - Or, in worst cases, alter application flow and force unintended actions

Vulnerabilities in Web Applications (3)

- Username/Password Disclosure
  - A response contains information related to usernames and/or passwords
  - An attacker can use this information to get access to private data
- Server Path Disclosure
  - A response contains a fully qualified path name to the root of the server storage system
  - An attacker can use this info to discover the server file system structure and devise other security attacks
- …

Examples of SQL Injection vulnerability

' OR 1=1 --

```
public String auth(String login, String pass)
               throw SQLException {
    String sql = "SELECT * FROM users WHERE "+
        "username='" + login + "' AND "+
        "password='" + pass + "'";
```

```
"SELECT * FROM users WHERE username='' OR 1=1 -- ' AND
                  password=''";
```

```
}
```

' OR ''='

```
public void delete(String str) throw SQLException{
    String sql = "DELETE FROM table
        "WHERE id='" + str + "'";
```

```
"DELETE FROM table WHERE id='' OR '' = ''";
```

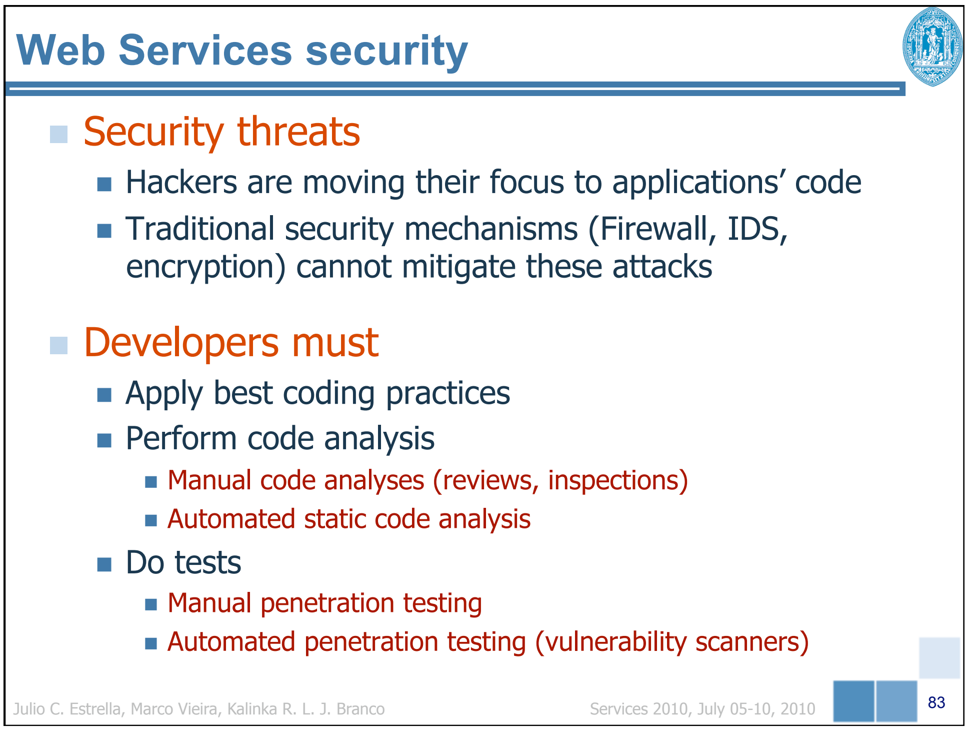
Questions?

Security in Web Services

Vulnerabilities detection approaches and tools

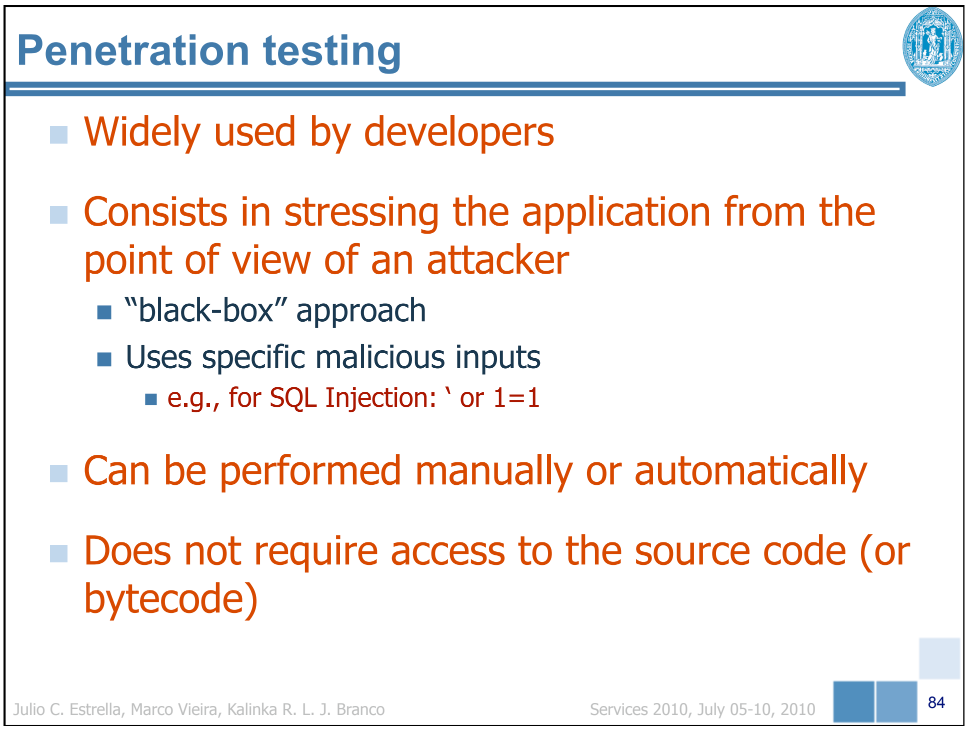
Are Web Services vulnerable?

- Web Services are widely exposed
- Any existing vulnerability will most probably be uncovered/exploited
- Both providers and consumers need to assess Web Services' security



Web Services security

- Security threats
  - Hackers are moving their focus to applications' code
  - Traditional security mechanisms (Firewall, IDS, encryption) cannot mitigate these attacks
- Developers must
  - Apply best coding practices
  - Perform code analysis
    - Manual code analyses (reviews, inspections)
    - Automated static code analysis
  - Do tests
    - Manual penetration testing
    - Automated penetration testing (vulnerability scanners)



Penetration testing

- Widely used by developers
- Consists in stressing the application from the point of view of an attacker
  - "black-box" approach
  - Uses specific malicious inputs
    - e.g., for SQL Injection: ' or 1=1
- Can be performed manually or automatically
- Does not require access to the source code (or bytecode)

Penetration testing tools

- Provide an automatic way to search for vulnerabilities
- Avoid the repetitive and tedious task of doing hundreds or even thousands of tests by hand
- Many tools available
  - Including commercial and open-source
- Different tools target different types of vulnerabilities
- The effectiveness of penetration testing tools is doubtful

Examples of penetration testing limitations

```
public void operation(String str) {
 try {
  String sql = "DELETE FROM table" +
          "WHERE id='" + str + "'";
  statement.executeUpdate(sql);
 } catch (SQLException se) {}
}
```

No return value and exceptions related with SQL mal-formation do not leak out to the invocator

```
public String dumpDepositInfo(String str) {
 try {
  String path = "//DepositInfo/Deposit"+
                "[@accNum='" + str + "']";
  return csvFromPath(path);
 } catch (XPathException e) {}
 return null;
}
```

This lack of output information

Examples of penetration testing tools

HP WebInspect™

acunetix Web Vulnerability Scanner 5

WSFuzzer

Foundstone A division of McAfee

WSDigger

IBM watchfire Rational. AppScan.

HP WebInspect (1)

- *"Web application security testing and assessment for complex web applications*
- *Built on emerging Web 2.0 technologies*
- *Fast scanning capabilities, broad security assessment coverage*
- *Accurate web application security scanning results"*

HP WebInspect (2)

- Includes pioneering assessment technology
  - Including simultaneous crawl and audit (SCA) and concurrent application scanning
- Broad scanning capabilities
  - Targets many different types of vulnerabilities
- Can be applied for penetration testing in web-based applications
  - Including Web Services

IBM Rational AppScan

- *"Is a leading suite of automated Web application security and compliance assessment tools*
- *Scan for common application vulnerabilities"*
- Suitable for users ranging from non-security experts to advanced users
- Supports extensions for customized scanning environments
- Can be used for security testing in web applications, including Web Services

Acunetix Web Vulnerability Scanner

- *"Is an automated web application security testing tool*
- *Audits your web applications by checking for exploitable hacking vulnerabilities"*
- Broad scanning capabilities
  - Targets many different types of vulnerabilities
- Can be applied for security testing in web applications in general
  - Including Web Services

Static code analysis

- "white-box" approach
- Consists in analyzing the source code of the application
  - Without execution it
- Looks for potential vulnerabilities
  - Among other types of software defects
- Can be performed manually or automatically
- Does require access to the source code (or bytecode)

Static code analysis tools

- Analyze the code without actually executing it
- The analysis varies depending on the tool sophistication
  - Ranging from tools that consider only individual statements and declarations
  - To others that consider the complete code
- Have other usages
  - e.g., model checking and data flow analysis
- These tools provide an automatic way for highlighting possible coding errors

Examples of static analysis limitations

```
public void operation(String str) {
 int i = Integer.parseInt(str);
 try {
  String sql = "DELETE FROM table" +
           "WHERE id='" + str + "'";
  statement.executeUpdate(sql);
 } catch (SQLException se) {}
}
```

Analyzers identify the vulnerability because the SQL query is a non-constant string

```
public String dumpDepositInfo(String str) {
 try {
  String path = "//DepositInfo/Deposit"+
               "[@accNum='" + str + "']";
  return csvFromPath(path);
 } catch (XPathException e) {}
 return null;
}
```

Depending on the complexity of csvFromPath method A static analysis tool may not be able to find the vulnerability

Examples of static analysis tools

- FindBugs
- Yasca (Yet Another Source Code Analyzer)
- IntelliJ IDEA
- …

Case Examples

FindBugs

- *"A program which uses static analysis to look for bugs in Java code"*
- It is able to scan the bytecode of Java applications
- Detects, among other problems, security issues
- It is one of the most used tools for static code analysis

Yasca (Yet Another Source Code Analyzer)

- *"A framework for conducting source code analyses"*
- Wide range of programming languages, including java.
- Yasca includes two components:
  - A framework for conducting source code analyses
  - An implementation of that framework that allows integration with other static code analyzers
    - e.g., FindBugs, PMD, Jlint

IntelliJ IDEA

- A commercial tool that provides a powerful IDE for Java development
- Includes "inspection gadgets" plug-ins with automated code inspection functionalities
- IntelliJ IDEA is able to detect security issues in java source code

Questions?

99

Security in Web Services

Vulnerabilities in public Web Services

100

Experimental study

- [Vieira09]
- Question: Are Web Services vulnerable?
- Apply leading commercial scanners in public Web Services
- 300 Web Services tested
  - Randomly selected
- 4 Scanners used
  - Including two different versions of a brand

Experimental procedure

- Preparation
  - Select services and scanners
- Execution
  - Test the services using the scanners
- Verification
  - Identify false positives
- Analysis
  - Analysis and systematization of results

Scanners

HP WebInspect™

IBM

watchfire

Rational. AppScan.

acunetix

Web Vulnerability Scanner

5

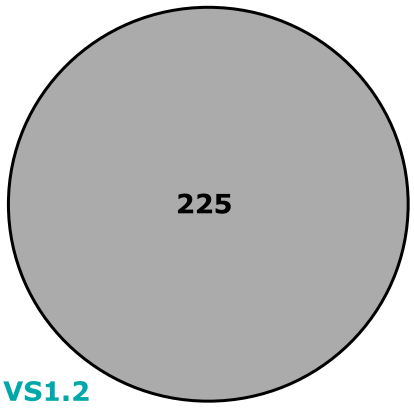
Vulnerabilities Found

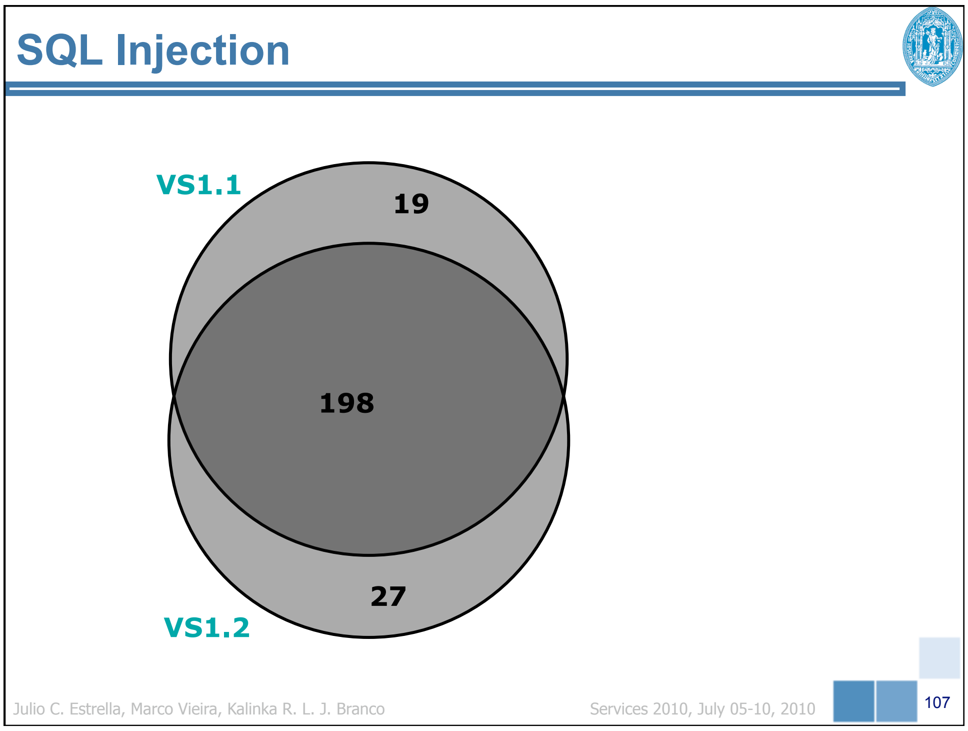
- SQL injection
- XPath Injection
- Code Execution
- Possible Parameter Based Buffer Overflow
- Possible Username or Password Disclosure
- Possible Server Path Disclosure

Overall results analysis

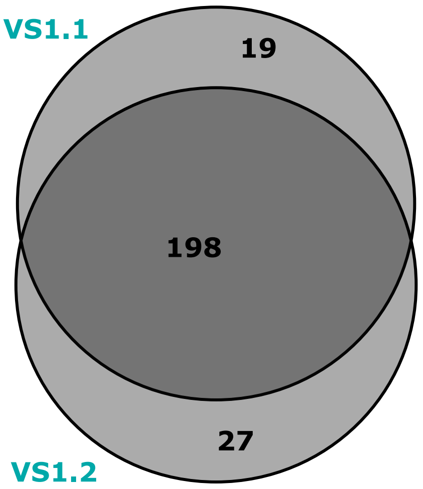
Vulnerability Types	VS1.1		VS1.2		VS2		VS3	
	# Vuln.	# WS	# Vuln.	# WS	# Vuln.	# WS	# Vuln.	# WS
SQL Injection	217	38	225	38	25	5	35	11
XPath Injection	10	1	10	1	0	0	0	0
Code Execution	1	1	1	1	0	0	0	0
Possible Parameter Based Buffer Overflow	0	0	0	0	0	0	4	3
Possible Username or Password Disclosure	0	0	0	0	0	0	47	3
Possible Server Path Disclosure	0	0	0	0	0	0	17	5
Total	228	40	236	40	25	5	103	22

SQL Injection

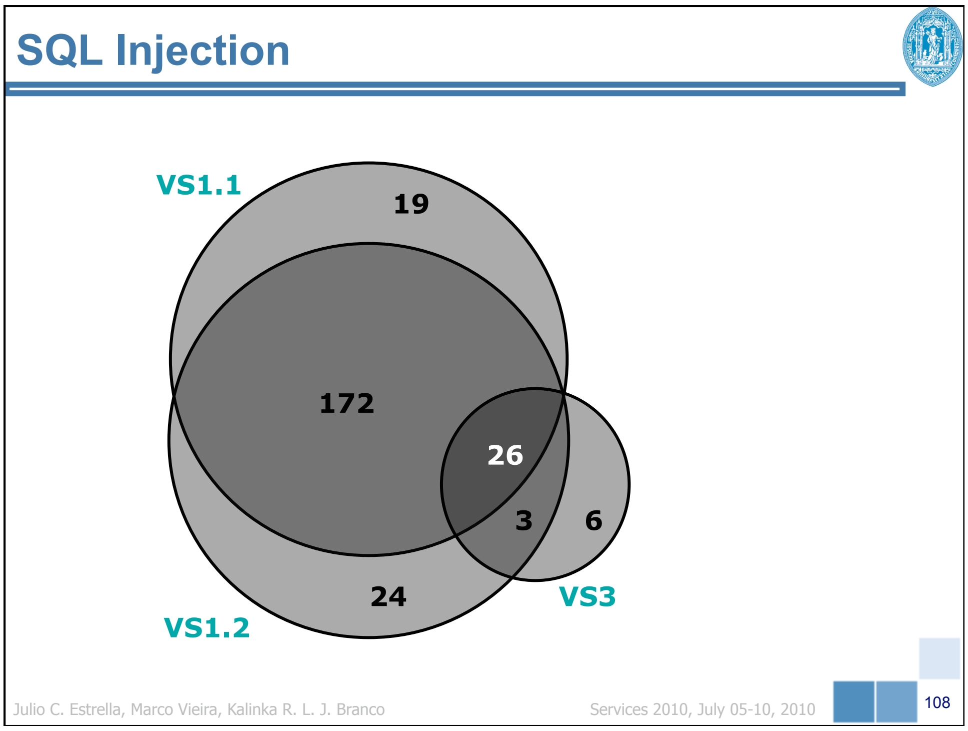




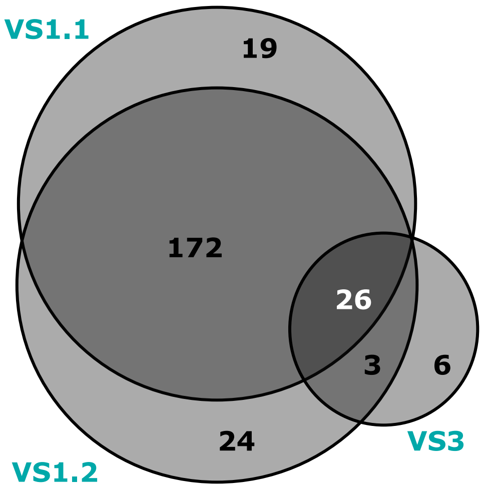
SQL Injection



Julio C. Estrella, Marco Vieira, Kalinka R. L. J. Branco Services 2010, July 05-10, 2010 107

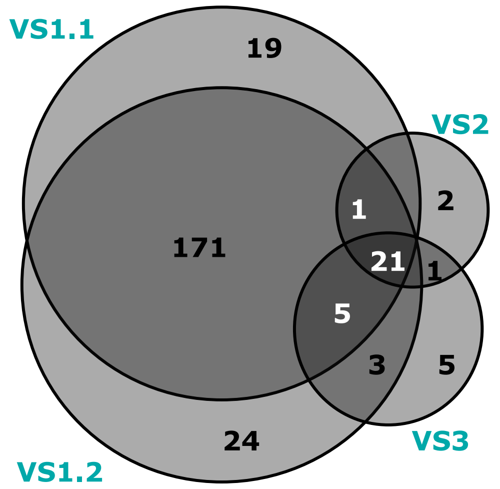


SQL Injection

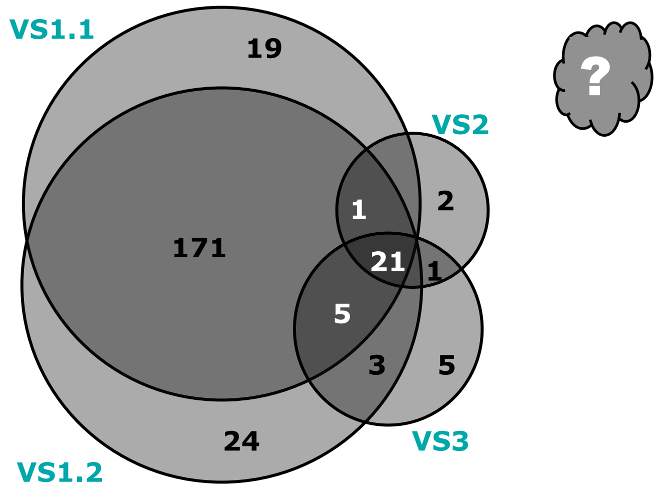


Julio C. Estrella, Marco Vieira, Kalinka R. L. J. Branco Services 2010, July 05-10, 2010 108

SQL Injection



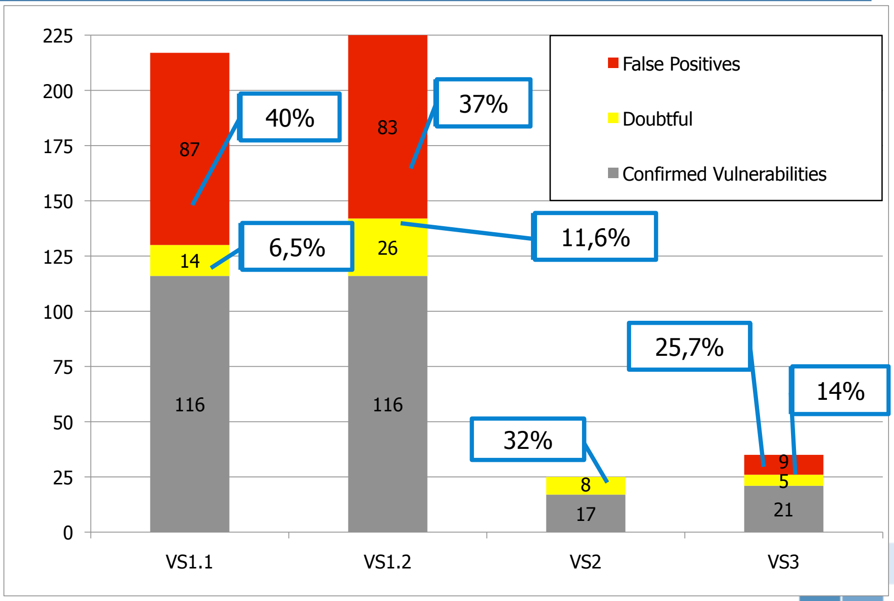
SQL Injection



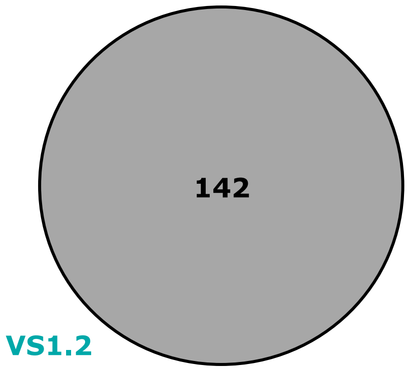
False positives examination

- False positive when
  - The error/answer obtained is related to an application robustness problem.
  - The same problem occurs when the service is executed with valid inputs
- Confirmed vulnerability when
  - Is possible to observe that a SQL command was invalidated by the "injected" values
  - The "injected" values lead to exceptions raised by the database server
  - Is possible to access unauthorized resources

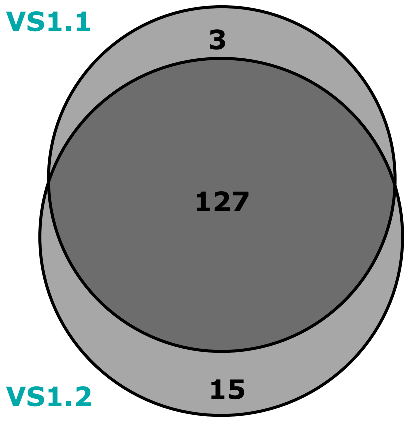
False positives results



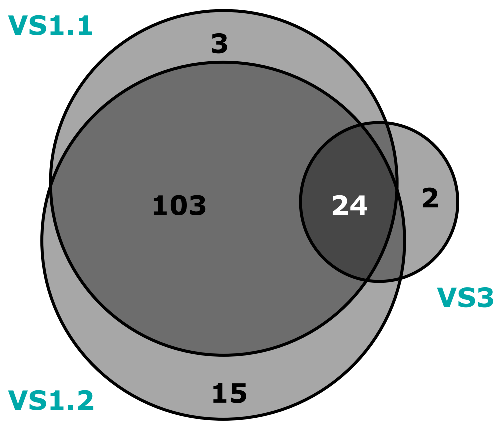
SQL Injection without False Positives



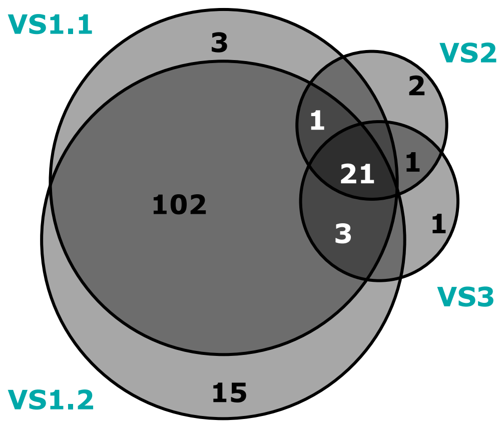
SQL Injection without False Positives



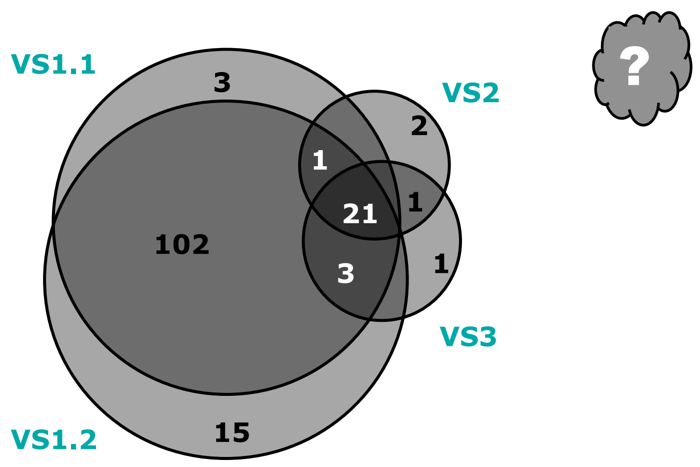
SQL Injection without False Positives



SQL Injection without False Positives



SQL Injection without False Positives



Julio C. Estrella, Marco Vieira, Kalinka R. L. J. Branco   Services 2010, July 05-10, 2010   117

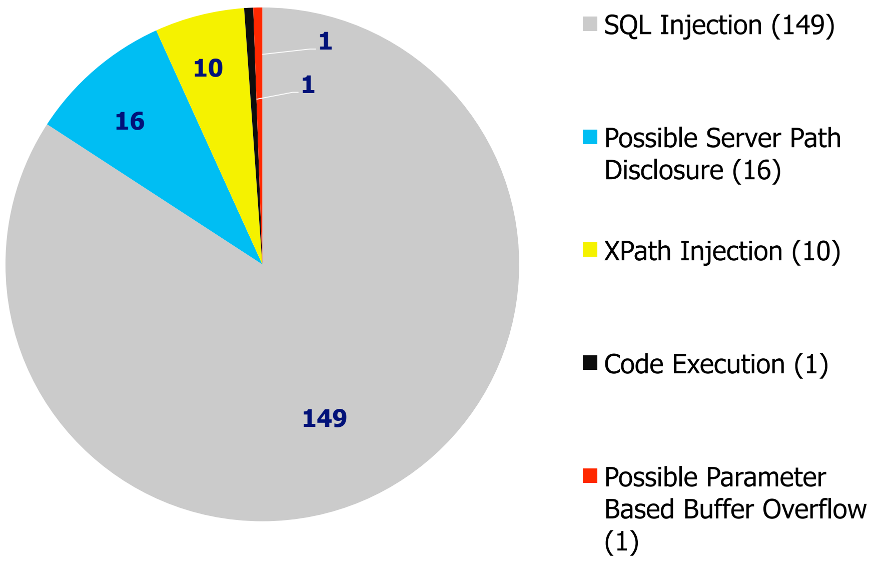
Coverage analysis

- Real number of vulnerabilities unavailable
  - It is possible to make a comparative analysis
- Overestimated Coverage values!!

Scanner	# SQL Injection Vulnerabilities	Coverage %
VS1.1	130	87.2%
VS1.2	142	95.3%
VS2	25	16.8%
VS3	26	17.4%
Total	**149**	**100%**

Julio C. Estrella, Marco Vieira, Kalinka R. L. J. Branco   Services 2010, July 05-10, 2010   118

Common vulnerabilities



Are Web Services secure?

- A large number of vulnerabilities was observed
- SQL Injection vulnerabilities are prevalent
- Selecting a scanner for Web Services seems to be a very difficult task
  - Different scanners detect different types of vulnerabilities
  - High false positives rates
  - Low coverage rates
- How effective are vulnerabilities detection tools?

Questions?

Security in Web Services

How effective are vulnerabilities detection tools?

Experimental study [Antunes09a]

- Evaluate several automatic penetration testing tools and static analysis tools
  - In a controlled environment
- Focus on two key measures of interest:
  - Coverage
    - Portrays the percentage of existing vulnerabilities that are detected by a given tool
  - False positives rate
    - Represents the number of reported vulnerabilities that in fact do not exist
- Target only SQL Injection vulnerabilities
  - Extremely relevant in Web Services

Steps

- Preparation
  - Select the penetration testers and static code analyzers
  - Select the Web Services to be considered
- Execution
  - Use the tools to identify potential vulnerabilities
- Verification
  - Perform manual verification to confirm that the vulnerabilities identified by the tools do exist
    - i.e., are not false positives
- Analysis
  - Analyze the results obtained and systematize the lessons learned

Web Services tested

- Eight Web Services
  - A total of 25 operations
- Four of the services are based on the TPC-App performance benchmark
- Four other services have been adapted from code publicly available on the Internet
- Implemented in Java and use a relational database

Web Services characterization

	Service	Short Description	#Op	LoC	LoC/Op	Avg. C.
TPC-App	ProductDetail	Get details about a product	1	105	105,0	6,0
	NewProducts	Add new product to the database	1	136	136,0	6,0
	NewCustomer	Add new customer to the database	1	184	184,0	9,0
	ChangePayment Method	Change customer's payment method	1	97	97,0	11,0
Public-Code	JamesSmith	Manages personal data about students	5	270	54,0	6,0
	PhoneDir	Phone book	5	132	26,4	2,8
	Bank	Manages bank operations	5	175	35,0	3,4
	Bank3	Manages bank operations (different from the Bank service)	6	377	62,8	9,0

Tools studied

- Penetration testing
  - HP WebInspect
  - IBM Rational AppScan
  - Acunetix Web Vulnerability Scanner
- Static code analysis
  - FindBugs
  - Yasca
  - IntelliJ IDEA
- Decided not to mention the brand of the tools to assure neutrality
  - VS1, VS2, VS3 (without any order in particular)
  - SA1, SA2, SA3 (without any order in particular)

Tools and environment configuration

- Penetration-testing
  - Underlying database restored before each test
    - This avoids the cumulative effect of previous tests
    - Guarantees that all the tools started the service testing in a consistent state
  - If allowed by the testing tool, information about the domain of each parameter was provided
    - If the tool requires an exemplar invocation per operation, the exemplar respected the input domains of operation
    - All the tools in this situation used the same exemplar
- Static code analysis
  - Configured to fully analyze the services code
  - For the analyzers that use binary code, the deployment-ready version was used

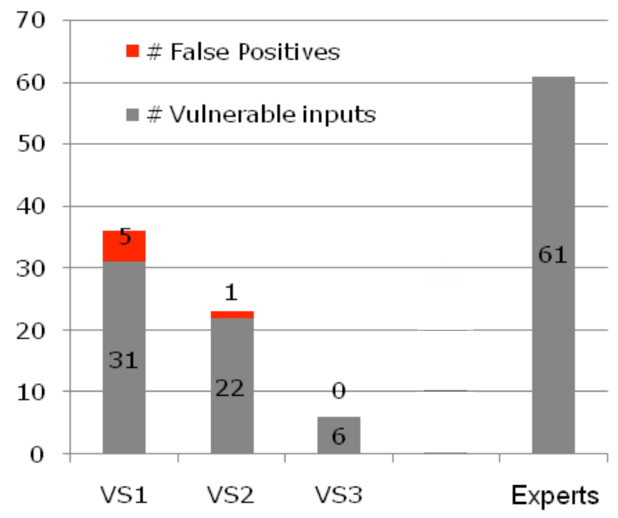
Web Services manual inspection

- It is essential to correctly identify the vulnerabilities that exist in the services code
- A team of experts was invited to review the source code looking for vulnerabilities
  - False positives were eliminated by cross-checking the vulnerabilities identified by different people
- A key difficulty is that different tools report (and count) vulnerabilities in different ways
  - Penetration testing: a vulnerability for each vulnerable parameter
  - Static analysis: a vulnerability for each vulnerable line in the service code

Vulnerabilities found

Service	#Vuln. Inputs	#Vuln. Lines
ProductDetail	0	0
NewProducts	1	1
NewCustomer	15	2
ChangePaymentMethod	2	1
JamesSmith	20	5
PhoneDir	6	4
Bank	4	3
Bank3	13	12
Total	**61**	**28**

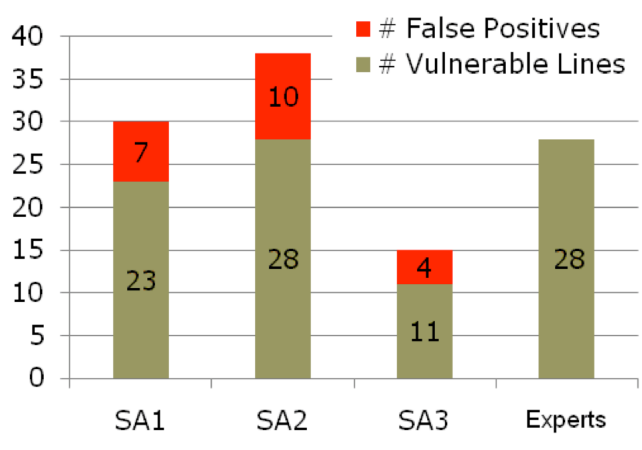
Penetration testing results



Tool	% F. P.
VS1	14.0%
VS2	4.0%
VS3	0.0%
VS4	0.0%

Tool	Coverage
VS1	50.8%
VS2	36.1%
VS3	9.8%
VS4	45.9%

Static code analysis results

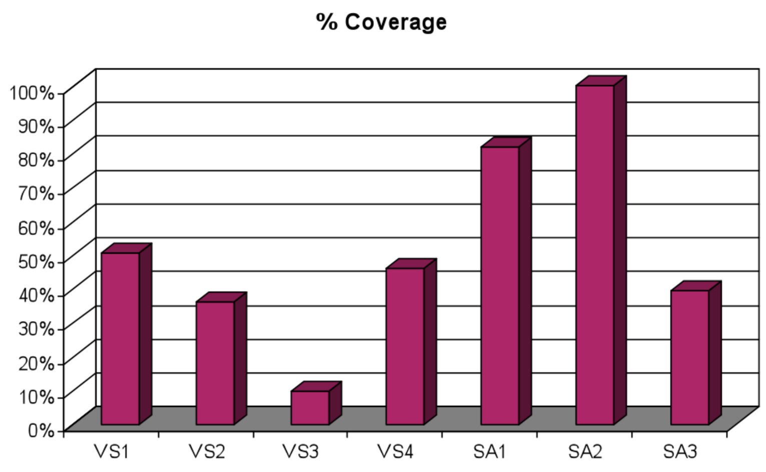


Tool	% F. P.
SA1	23.3%
SA2	26.3%
SA3	26.7%

Tool	Coverage
SA1	82.1%
SA2	100.0%
SA3	39.3%

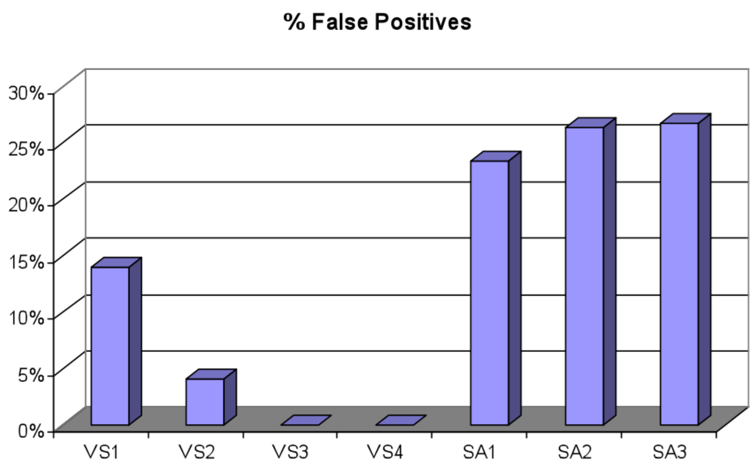
Penetration testing *vs* Static analysis (1)

- Coverage



Penetration testing *vs* Static analysis (2)

- False positives



Key observations

- The coverage of static code analysis is typically higher than of penetration testing
- False positives are a problem for both approaches
  - But have more impact in the case of static analysis;
- Different tools report different vulnerabilities in the same piece of code
  - Even tools implementing the same approach frequently
- Poor results!
  - Can we do better?

Questions?

Security in Web Services

Can we do better?

Yes, we can! ☺

- [Antunes09a]
  - Proposes a new penetration testing approach to detect SQL Injection vulnerabilities in Web Services code
- [Antunes09b]
  - Approach based on anomaly detection to find SQL/ XPath Injection vulnerabilities

[Antunes09a]

- New penetration testing approach to detect SQL Injection vulnerabilities
- Main improvements:
  - A representative workload to exercise the services and understand the expected behavior
  - A broader set of attacks
  - Well defined rules to analyze the service's responses
    - To improve coverage and remove false positives
  - Completely automatic

Execution steps

1. Prepare the tests
   1.1. Gather information about the Web Service' operations, call parameters, data types, and input domains
   1.2. Generate the workload
2. Execute the tests
   2.1. Execute the workload to understand the expected behavior of the service in the absence of attacks
   2.2. Perform the attacks to trigger faulty behaviors and disclose SQL Injection vulnerabilities
3. Analyze the responses to detect and confirm the vulnerabilities

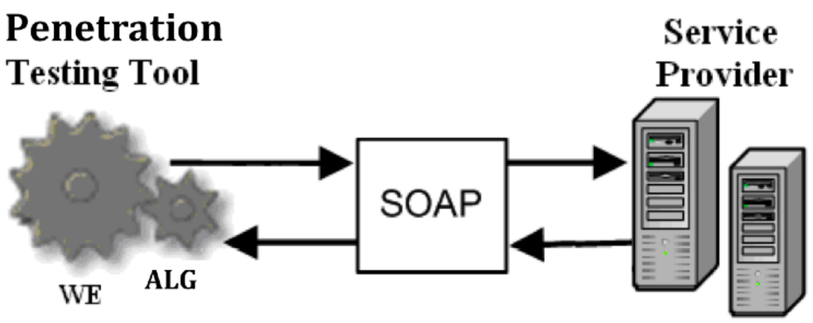
Prepare the tests: Gather information

- Web Service interfaces are described as a WSDL file
- This file is processed automatically to obtain:
  - Operations
  - Call parameters
  - Data types
- The valid values for each parameter (i.e., input domains) have to be provided by the user

Prepare the tests: generate the workload

- Two options:
  - User-defined workload
  - Random workload
- Random workload is generated automatically
  - Generate test values for each input parameter
  - Generate test calls for each operation
  - Select test calls for each operation
    - It may be unfeasible to use a workload based on all the test calls generated (e.g., due to time constraints)
    - It is up to the user to specify the size of this subset

Execute the tests: Configuration



Execute the tests: Type of attacks

- Examples:

SQL Injection Attack
" or 1=1 --
" or 1=1 or ""="
' or (EXISTS)
' or uname like '%
' or userid like '%
' or username like '%
' UNION ALL SELECT
' UNION SELECT
char%2839%29%2b%28SELECT
&quot; or 1=1 or &quot;&quot;=&quot;
&apos; or &apos;&apos;=&apos;

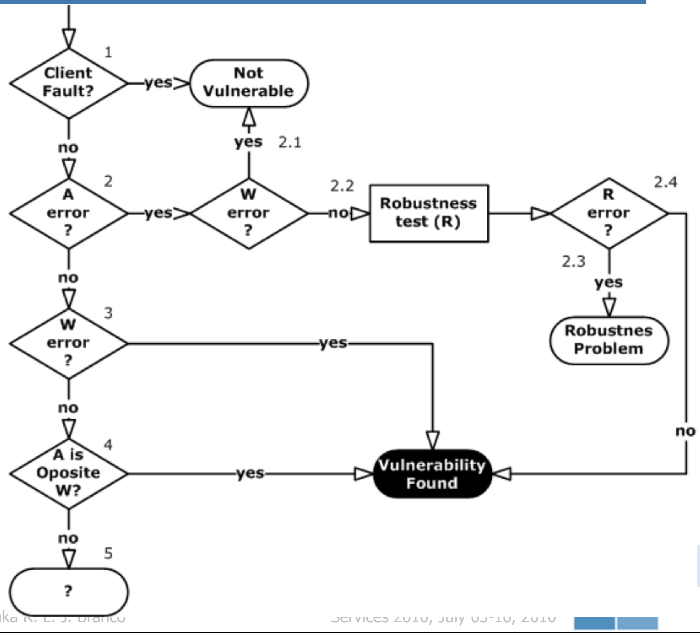
- A total of 137 types
  - The list can be continuously improved
  - Just add new attack patterns to a configuration file

Execute the tests: Attacks generation

- Mutation of the workload test calls
  - Valid values are replaced by malicious values
- Number of attacks can be extremely large, e.g.:
  - 3 operations with 5 parameters each
  - A workload with 25 test calls per operation
  - 137 attack types ➔ 52500 attacks
- The tool allows specifying the number of test calls to be used for the attack load generation
  - The original test calls are ranked based on their ability to help us detecting vulnerabilities
    - e.g. test calls that that lead to valid Web Service responses (i.e., no error) are in the top of the list

Analyze the responses

- W
  - Valid call
- A
  - Attack call



Experimental evaluation

- Web Services tested
  - 262 public Web Services
- Four steps:
  - Preparation: select a large set of Web Services.
  - Execution: use the vulnerability scanners to scan the services to identify potential vulnerabilities
  - Verification: perform manual testing to confirm that the vulnerabilities identified do exist
  - Analysis: analyze the results and compare the effectiveness of our tool to the commercial ones

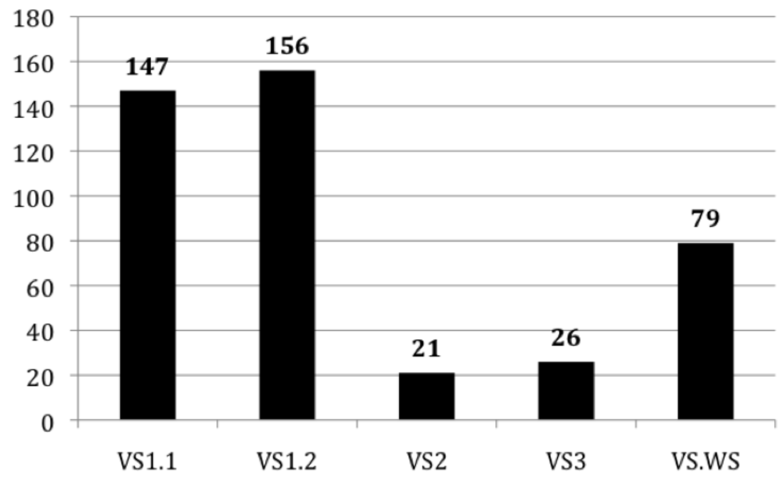
Scanners

HP WebInspect™

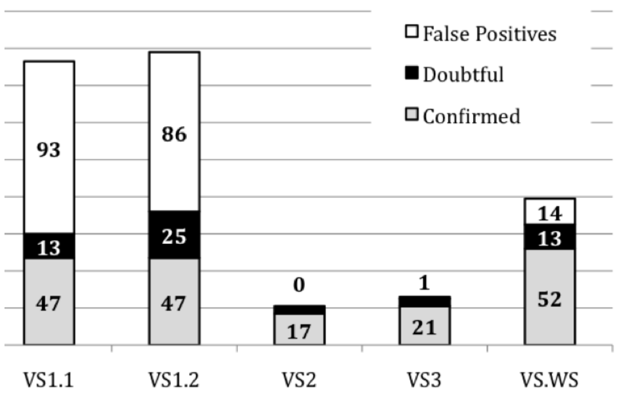
IBM watchfire Rational. AppScan.

acunetix Web Vulnerability Scanner 5

Raw results for public Web Services



After removing false positives…



Scanner	% FP
VS1.1	63
VS1.2	55
V2	0
VS3	4
VS.WS	18

Scanner	% Doubtful
VS1.1	9
VS1.2	16
V2	19
VS3	15
VS.WS	16

Detection coverage

- Based on limited knowledge
  - Probably we don't know all the existing vulnerabilities

Scanner	# Vul. Detected	Coverage %
VS1.1	47	84%
VS1.2	47	84%
VS2	17	30%
VS3	21	38%
VS.WS	52	93%
Total	**56**	**100.0%**

Can we do better?

- Results show that it is possible to achieve much better results than commercial tools
  - Concerning both coverage and false positives
- The tool was able to detect vulnerabilities that were not detected by the commercial scanners
  - And, at the same time, was able to eliminate most of the false positives
- This shows that it is possible to improve the state of the art in vulnerabilities detection

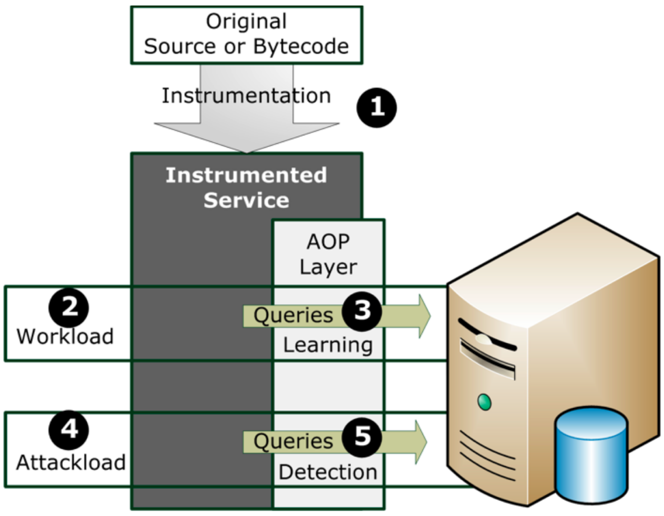
Questions?

[Antunes09b]

- Approach based on anomaly detection to find SQL/XPath Injection vulnerabilities
- Steps:
  1. Instrument the web service to intercept all SQL/XPath commands executed
  2. Generate a workload
  3. Execute the workload to learn SQL commands and XPath queries issued
  4. Generate an attackload based on a large set of SQL Injection and XPath Injection attacks
  5. Execute the attackload to detect vulnerabilities

Web Service instrumentation

- Based on Aspect-Oriented Programming (AOP)



Workload generation

1. Get relevant information from the WSDL file
   - Operations, parameters, datatypes
2. Generate test values for each input parameter
   - Random generation of a set of valid input values
   - The number of test values is defined by the user
3. Generate test calls for each operation
   - Generate a large set of calls for each operation
   - Sum of all combinations of the test values generated for all the parameters of each operation
4. Select test calls for each operation
   - Defined by the user
   - Determines the final size of the workload

SQL/XPath commands learning

1. Exercise the service by executing the workload
2. SQL and XPath commands are intercepted
3. Commands are parsed to remove the data variant part (if any)
4. A hash code is used to identify each command
5. Each hash signature is associated with a source code entry point
6. Workload coverage is analyzed
    - If not satisfactory, then more calls should be performed

Attackload

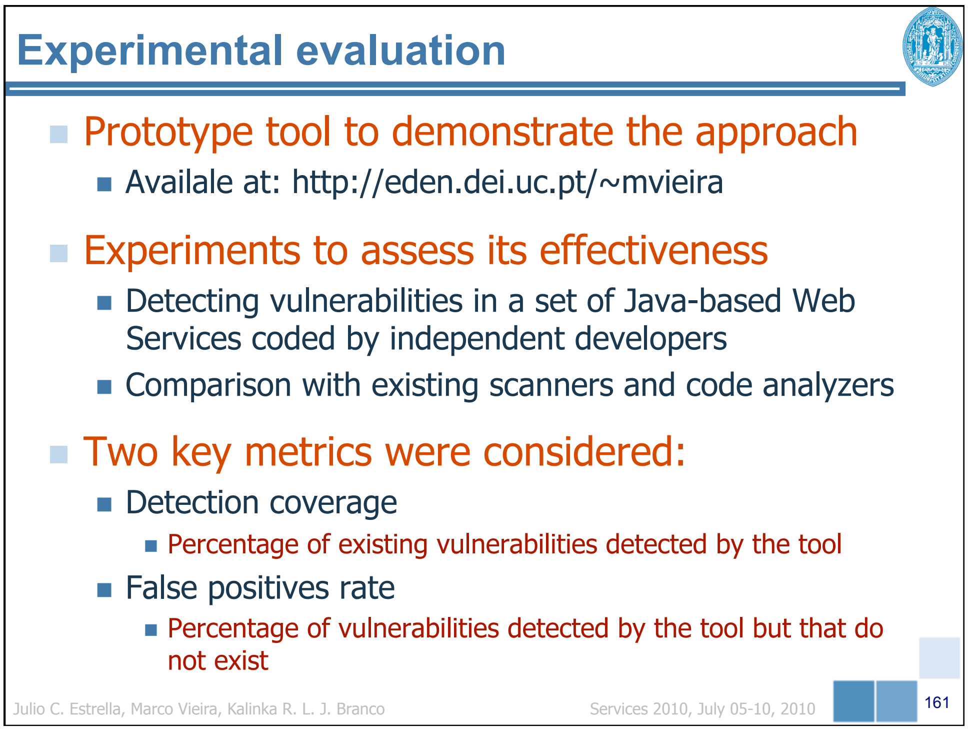
- Includes parameter values that attempt to perform SQL/XPath injection
- Attack types are based on the compilation of the types used by a large set of scanners
- Complemented based on practical experience and on information available in the literature
- Attackload generation:
    - Generate a new workload
    - Malicious values are selectively inserted by applying the attack rules

Examples of attack types

SQL Injection Attacks
" or 1=0 --
" or 1=1 or ""="
' or (EXISTS)
' or uname like '%
' or userid like '%
' or username like '%
char%2839%29%2b%28SELECT
&quot; or 1=1 or &quot;&quot;=&quot;
&apos; or &apos;&apos;=&apos;

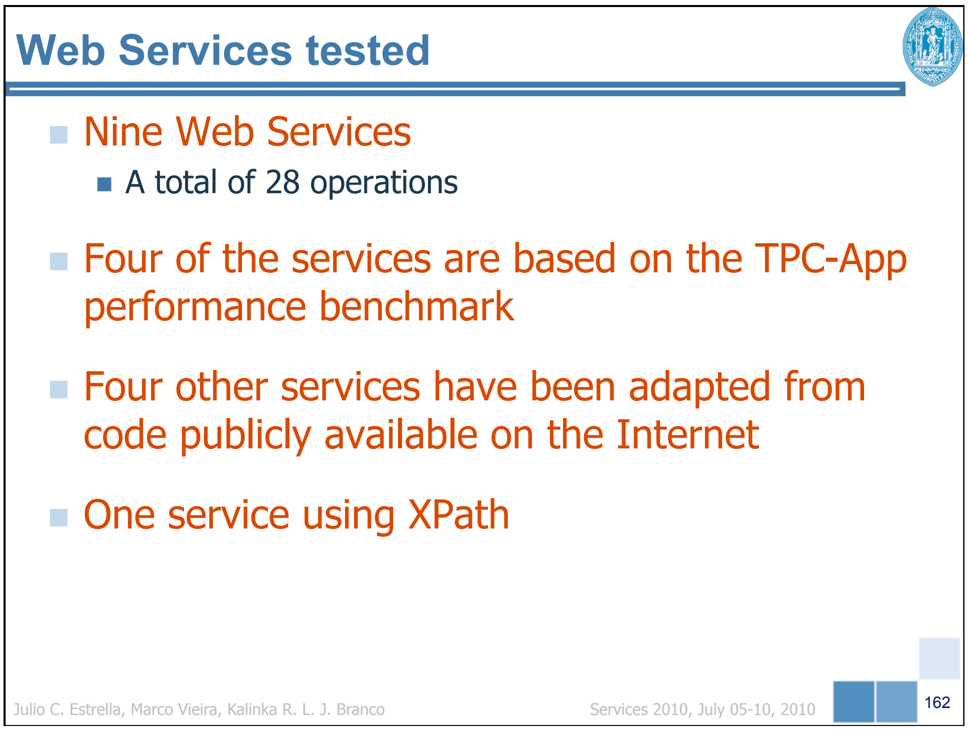
Vulnerabilities detection

1. Execute the attackload and perform security checks per each data access executed
2. SQL and XPath commands are intercepted and hashed
3. The calculated hash codes are compared to the values of the learned valid commands
   - For the code point at which the command was submitted
4. If hash code is NOT found then:
   - There is a vulnerability
   - The source code location was not learned correctly



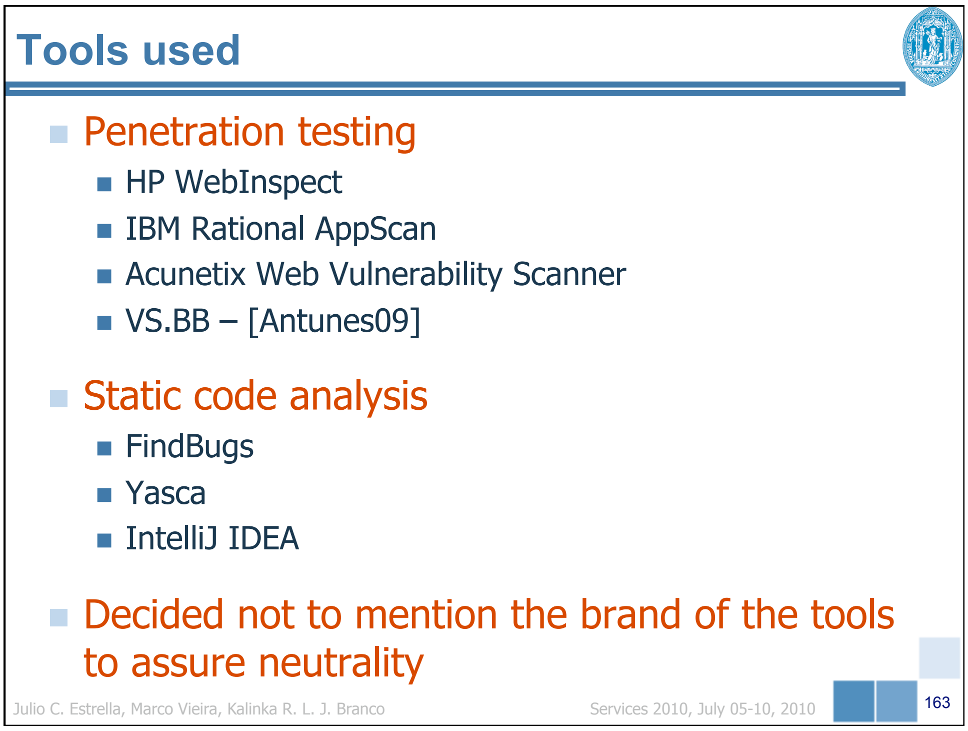
Experimental evaluation

- Prototype tool to demonstrate the approach
  - Availale at: http://eden.dei.uc.pt/~mvieira
- Experiments to assess its effectiveness
  - Detecting vulnerabilities in a set of Java-based Web Services coded by independent developers
  - Comparison with existing scanners and code analyzers
- Two key metrics were considered:
  - Detection coverage
    - Percentage of existing vulnerabilities detected by the tool
  - False positives rate
    - Percentage of vulnerabilities detected by the tool but that do not exist



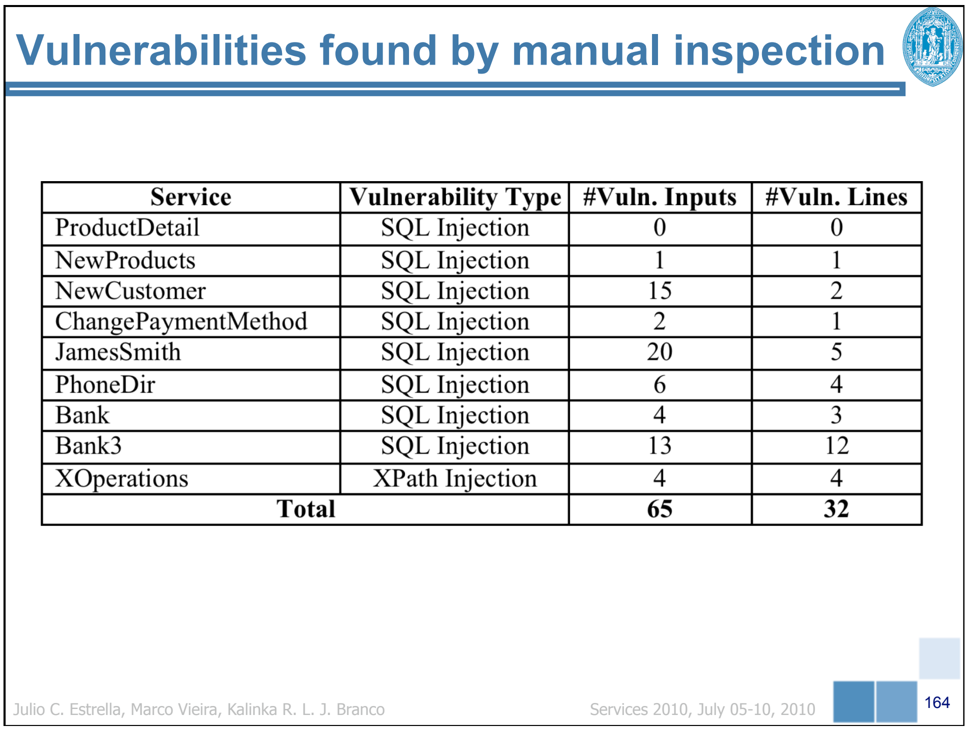
Web Services tested

- Nine Web Services
  - A total of 28 operations
- Four of the services are based on the TPC-App performance benchmark
- Four other services have been adapted from code publicly available on the Internet
- One service using XPath



Tools used

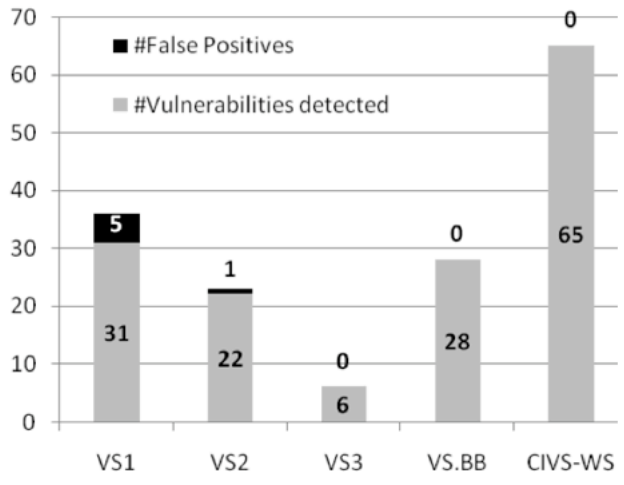
- Penetration testing
  - HP WebInspect
  - IBM Rational AppScan
  - Acunetix Web Vulnerability Scanner
  - VS.BB – [Antunes09]
- Static code analysis
  - FindBugs
  - Yasca
  - IntelliJ IDEA
- Decided not to mention the brand of the tools to assure neutrality



Vulnerabilities found by manual inspection

Service	Vulnerability Type	#Vuln. Inputs	#Vuln. Lines
ProductDetail	SQL Injection	0	0
NewProducts	SQL Injection	1	1
NewCustomer	SQL Injection	15	2
ChangePaymentMethod	SQL Injection	2	1
JamesSmith	SQL Injection	20	5
PhoneDir	SQL Injection	6	4
Bank	SQL Injection	4	3
Bank3	SQL Injection	13	12
XOperations	XPath Injection	4	4
Total		**65**	**32**

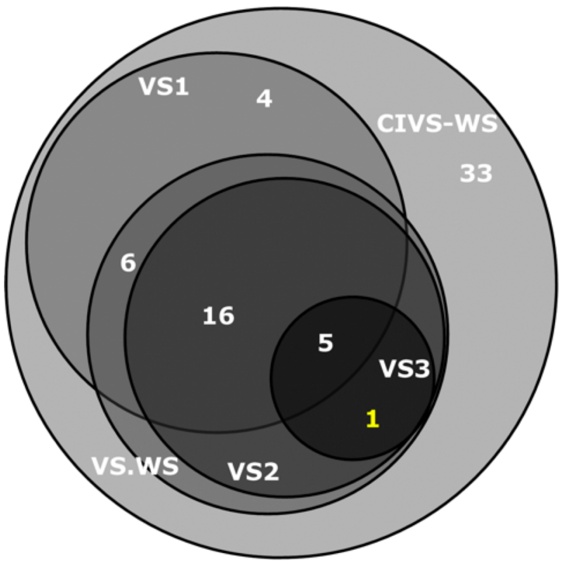
Comparing with penetration testing…



Tool	% F. P.
VS1	14%
VS2	4%
VS3	0%
VS.BB	0%
CIVS-WS	0%

Tool	Coverage
VS1	48%
VS2	34%
VS3	9%
VS.BB	43%
CIVS-WS	100%

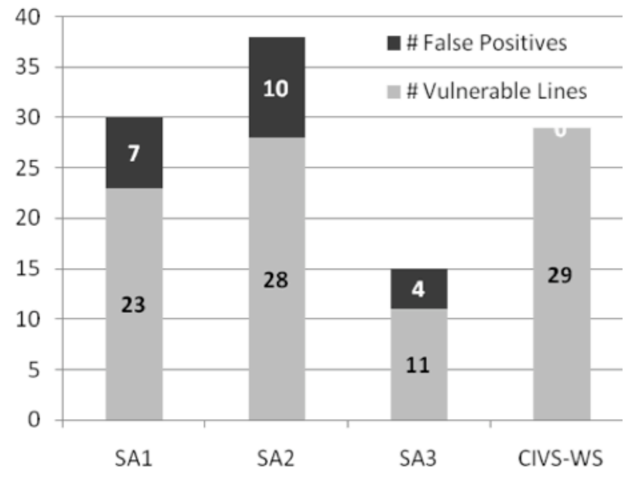
Intersections for penetration testing



Legend:

CIVS-WS
VS1
VS.WS
VS2
VS3

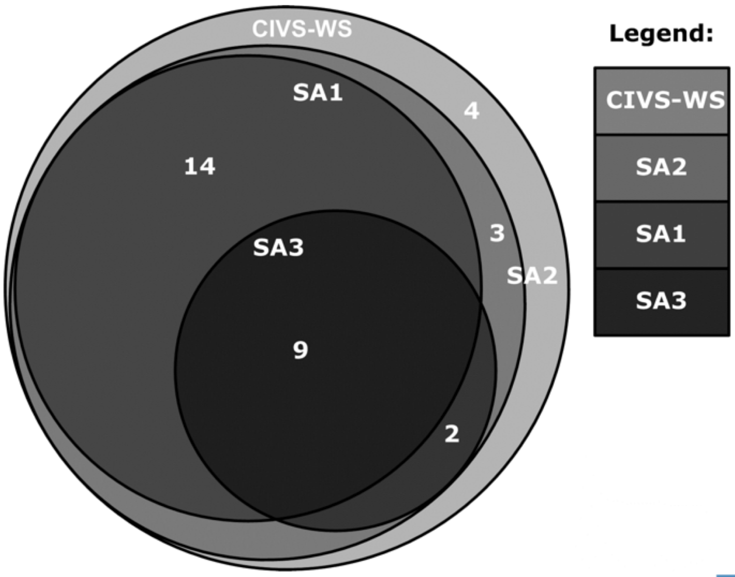
Comparing with static code analysis…



Tool	Coverage
SA1	71,9%
SA2	87,5%
SA3	34,4%
CIVS-WS	100,0%

Tool	% F. P.
SA1	23,3%
SA2	26,3%
SA3	26,7%
CIVS-WS	0,0%

Intersections for static code analysis



Questions?

Security in Web Services

Research challenges and opportunities

Challenges and opportunities (1)

- Fact: The effectiveness of vulnerability detection tools is very low
- How to improve penetration testing?
  - Increase representativeness of the workload
  - Guarantee high coverage
  - Improve the attacks performed
  - Improve the vulnerability detection algorithms
- How to improve static analysis?
  - Include new vulnerable code patterns
    - How to identify those patterns?

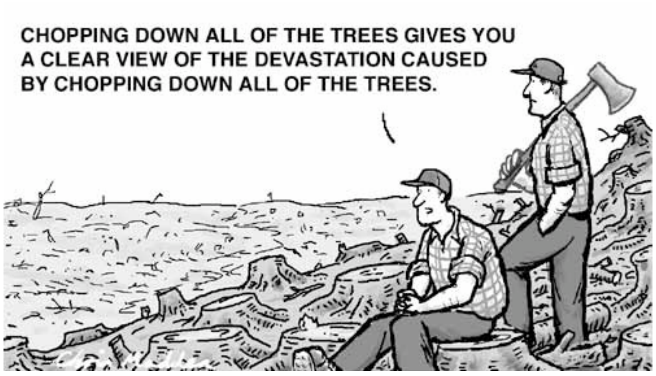
Challenges and opportunities (2)

- Merge penetration testing and static code analysis techniques?
- Fact: Web Services are typically deployed with vulnerabilities
- How to improve the current situation?
  - Better tools for vulnerability detection
  - Approaches to automatically remove vulnerabilities
- What about attack detection?
  - Can we include in the Web Services attack detection mechanisms?

Challenges and opportunities (3)

- Improve the software development process?
  - More effective testing?
  - More effective code reviews and inspections?
    - Use targeted checklists?
- How to take advantage of Web Services diversity to change the current situation?

Questions?


CHOPPING DOWN ALL OF THE TREES GIVES YOU A CLEAR VIEW OF THE DEVASTATION CAUSED BY CHOPPING DOWN ALL OF THE TREES.

Open Discussion

Your Time!

Key references (1)

- [Antunes09a] Antunes, N. and Vieira, M. , "Detecting SQL Injection Vulnerabilities in Web Services", Fourth Latin-American Symposium on Dependable Computing (LADC 2009), João Pessoa, Paraíba, Brazil, September 2009.
- [Antunes09b] Antunes, N. and Laranjeiro, N. and Vieira, M. and Madeira, H. , "Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services", IEEE International Conference on Services Computing (SCC 2009), Bangalore, India, September 2009.
- [Chou05]. D. C. Chou, K. Yurov. Security development in web services environment. Computer Standards & Interfaces, v. 27, n. 3, p. 233–240, 2005.
- [Holgersson05] J. Holgersson, E. Soderstrom. Web service security - vulnerabilities and threats within the context of ws-security. 2005, p. 138 – 146
- [Mogollon08] M. Mogollon. Cryptography and security services: Mechanisms and applications. IGI Global, 2008.
- [Nordbotten09] N. A. Nordbotten. Xml and web services security standards. IEEE Communications Surveys Tutorials, v. 11, n. 3, p. 4 –21, 2009
- [Oasis07a] Ws-trust 1.3. Available at: http://docs.oasis-open.org/ws-sx/ws-trust/200512. Último acesso: 18/02/2010, 2007a.
- [Oasis07b] Ws-secureconversation 1.3. Available at: http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.html. Último acesso: 18/02/2010, 2007b.

Key references (2)

- [Oasis08] Web services federation (wsfed) tc. Available at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsfed. Último acesso: 18/02/2010, 2008.
- [Siblini05] Siblini, R., Mansour, N., "Testing Web Services", The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.
- [Stock07] Stock, A., Williams, J., Wichers, D., "OWASP top 10", OWASP Foundation, July, 2007.
- [Vieira09] Vieira, M. and Antunes, N. and Madeira, H. , "Using Web Security Scanners to Detect Vulnerabilities in Web Services", 39th Annual IEEE/IFIP Intl. Conference on Dependable Systems and Networks (DSN 2009), Estoril, Lisbon, Portugal, June 2009.
- [Xu05] Xu, W. et al., "Testing Web Services by XML perturbation", 16th IEEE International Symposium on Software Reliability Engineering, 2005.

Questions?

Thanks for your participation!

Julio Cezar Estrella
Kalinka R. L. J. C. Branco
University of Sao Paulo - Brazil
jcezar@icmc.usp.br, kalinka@icmc.usp.br

Marco Vieira
Center for Informatics and Systems
University of Coimbra
mvieira@dei.uc.pt

179