



# Input Validation Vulnerabilities, Encoded Attack Vectors and Mitigations

Marco Morana &  
Scott Nusbaum

**OWASP**

Cincinnati Chapter September 08 Meeting

Copyright 2008 © The OWASP Foundation  
Permission is granted to copy, distribute and/or modify this document  
under the terms of the OWASP License.

**The OWASP Foundation**

<http://www.owasp.org>

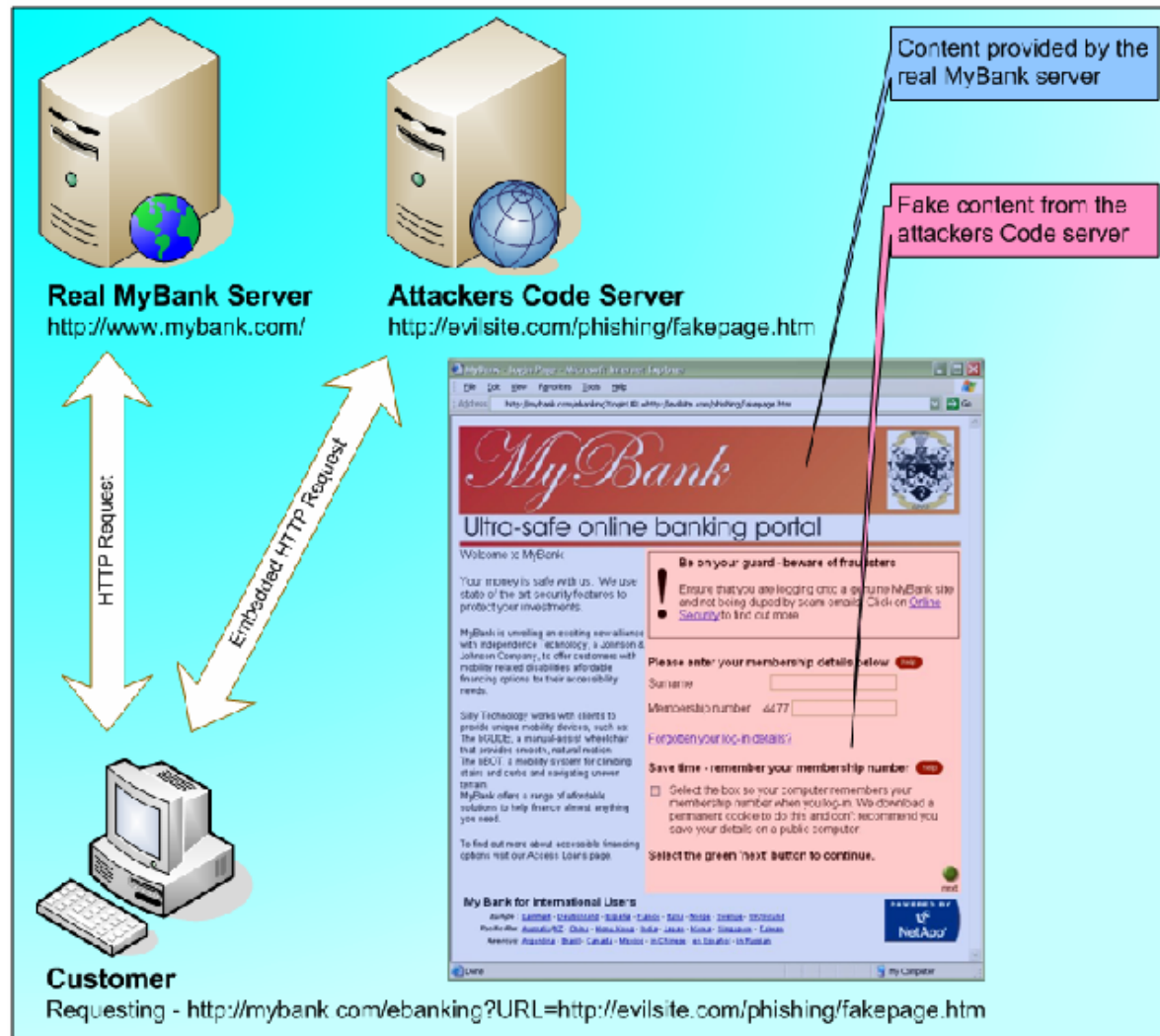
# Agenda

1. Input validation attacks: Cause, Exploits, Impacts
2. What is an attack vector: Definitions, Elements, Types (traditional old and new Web 2.0)
3. How attackers craft attack vectors: Encoding, Double Encoding and Filter Evasions
4. Attack vectors libraries (Cheat Sheets): XSS, SQL Injection
- 5. Live exploit examples**
6. How to find IV vulnerabilities: Web application security Assessments
7. How to protect from IV attack vectors
- 8. IV attack defenses live examples:** Structs Validators, Encoding Rules
9. Countermeasures and mitigation strategies
10. Q&A

# Input Validation Attacks: Cause, Exploits, Impacts

- **Cause:** Failure to properly validate data at the entry and exit points of the application
- **Exploits:** Injection of malicious input such as ***code, scripting, commands***, that can be interpreted/executed by different targets to exploit vulnerabilities:
  - ▶ Browser: XSS, XFS, HTML-Splitting
  - ▶ Data repositories: SQL Injection, LDAP injection
  - ▶ Server side file processing: XML, XPATH
  - ▶ Application/Server/O.S. :File uploads, Buffer Overflow
- **Impacts:** Phishing, Information Disclosure (e.g. PII), Data Alteration/Destruction, Denial/Degradation Of service, Financial Loss/Fraud, Reputation Loss

# IV Attack Example 1: Code Injection



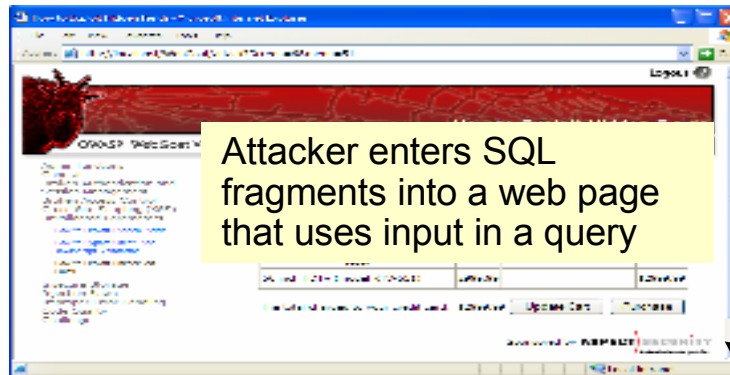
From: [www.technicalinfo.net/papers/Phishing.html](http://www.technicalinfo.net/papers/Phishing.html)

## IV Attack Example 2: SQL Injection

1

Attacker Enters Malicious Inputs such as:

<http://www.bank.com/index.php?id=1> ***UNION ALL SELECT creditCardNumber,1,1, FROM CreditCardTable***

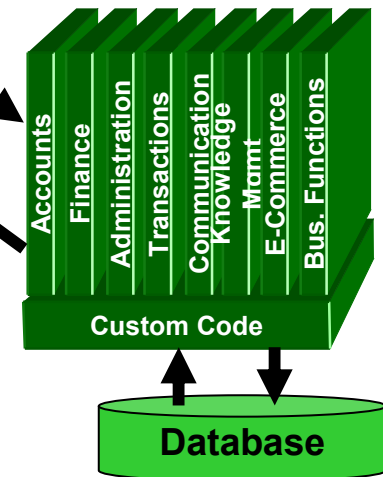


3

Attacker obtain other customers credit card numbers

2

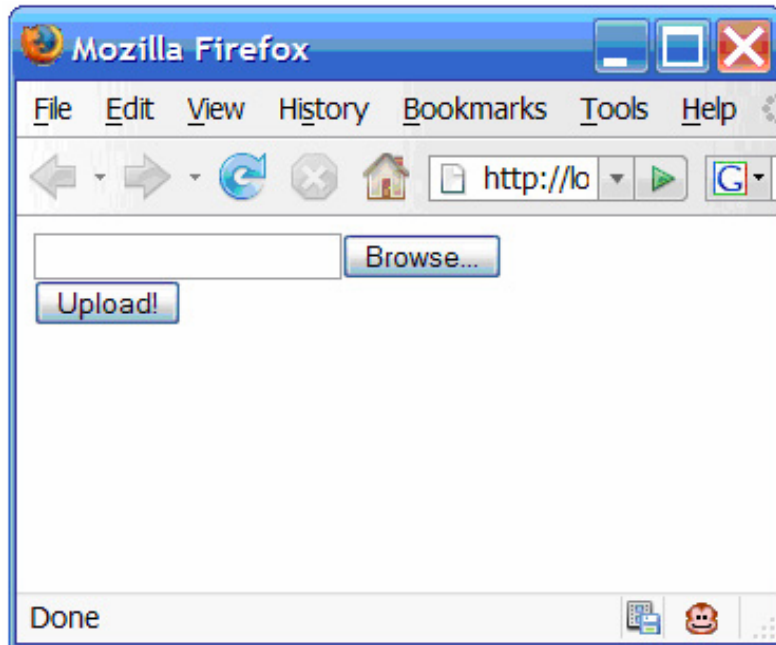
Application sends modified query to database such as ***SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber 1,1 FROM CreditCardTable***, which executes it



## IV Attack Example 3: Malicious File Upload

1) Malicious user passes the following information in the cmd parameter:

*cmd=%3B+mkdir+hackerDirectory*



2) The parameter from the request is used for command line process

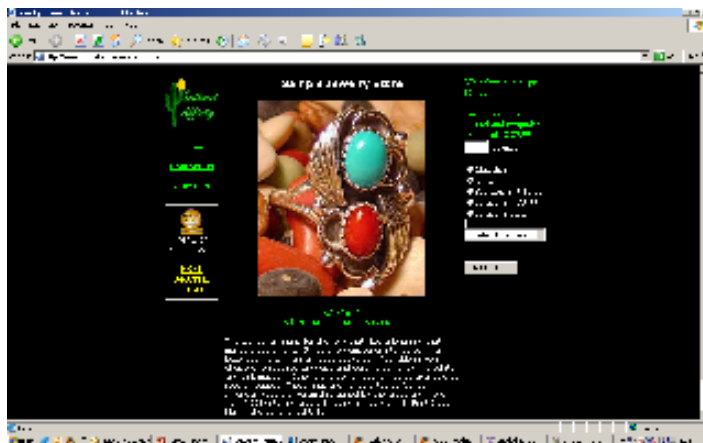
```
String fromRequest =  
request.getParameter("cmd");  
Process process =  
runtime.exec("cmd.exe /C" +  
fromRequest);
```

3) Final command executed is: *cmd.exe /C "dir; mkdir hackerDirectory"*

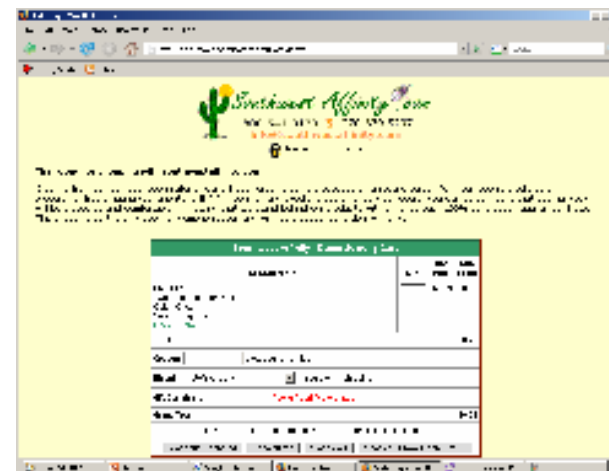
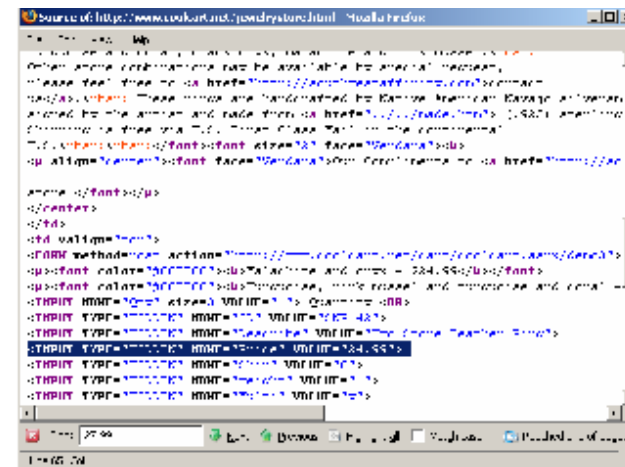
© 2014 Pearson Education, Inc. or its affiliate(s). All rights reserved.



<http://www.coolcart.com/jewelrystore.html>



The price charged for the  
"Two Stone Feather  
Ring" is now 99 cents



# Attack Vectors Definitions

- *"An attack vector is a **path** or **means** by which a hacker can **gain access** to a computer or network server in order to deliver a payload or malicious outcome"*
- *"Attack vectors are **routes** or **methods** used to get into computer systems, usually for nefarious purposes. **They take advantage of known weak spots to gain entry. Many attack vectors take advantage of the human element in the system, because that's often the weakest link.** "*

From SecuritySearch.com Definitions :<http://searchsecurity.techtarget.com/dictionary/definition/1005812/attack-vector.html>



# Understanding Attack Vectors

- Don't confuse attack vectors with the payload that is carried out

- ▶ **Attack vectors:** malicious email, attachments, worms, web pages, downloads, deception (aka social engineering), hackers
- ▶ **Payloads:** viruses, spyware, trojans, malicious scripting/executables.

## ■ XSS Example:

- ▶ The attack vector with a payload consisting in a script (**also encoded**) to capture sensitive information (e.g. cookie stored on the browser) such as in an alert dialog:
- ▶ *[http://server/cgibin/testcgi.exe?<SCRIPT>alert\("Cookie"+document.cookie\)</SCRIPT>](http://server/cgibin/testcgi.exe?<SCRIPT>alert("Cookie"+document.cookie)</SCRIPT>)*

# Traditional Vector Based Attack Types

- **Buffer overflows attacks (aka string injection)**
- **Code injection attacks:** also known as "code poisoning attacks" examples:
  - ▶ Cookie poisoning attacks
  - ▶ HTML injection attacks
  - ▶ File injection attacks
  - ▶ Server pages injection attacks (e.g. ASP, PHP)
  - ▶ Script injection (e.g. cross-site scripting) attacks
  - ▶ Shell injection attacks
  - ▶ SQL injection attacks
  - ▶ XML poisoning attacks



# New Web 2.0 Attack Vectors

- Cross-site scripting in AJAX
- XML Poisoning
- Malicious AJAX code execution
- RSS Atom Injection
- WSDL scanning and enumeration
- Client validation in AJAX routines
- Web service routing issues
- Parameter manipulation with SOAP
- XPATH injection in SOAP message
- RIA thick client binary vector

# Attacker Perspective: Crafting Attack Vectors

1. **Discover Entry Points:** Identify *first order injection* and *second-order injection* (e.g. to attack resources directly) Fingerprint application server and technology
2. **Scan and exploit** known vulnerabilities
3. **If not exploitable, try attack libraries**, bypass filtering, exploit IV vulnerability patterns:
  1. Input=>Output==XSS
  2. Input=>Query (SQL, LDAP) ==(SQL, LDAP) injection
  3. Input=>Code== Code injection
  4. Input=>XML doc == XML injection
  5. Input=>OS command==OS command injection
  6. Input=> Fixed buffer or format string== overflow



## Defense Perspective: Canonical Representation and Encoding

- **Fact:** filtering out bad input is not easy as it sounds and you can have many representations (i.e. more than just ASCII characters)
- **Canonicalization** (c14n): the process of translating every string character to its single allowed (standard) encoding for each character
- **Encoding:** Attack Examples for URL:
  - ▶ `< > %3c and %3e` (used in XSS)
  - ▶ `: %3a` (used in XSS with javascript: )
  - ▶ `' %27, -- %2D%2D, ; %3B` (used in SQL injections)
  - ▶ `../ %2E%2E%2F` (used in directory transversal, file upload)
  - ▶ `` %60` (used in command injections)
  - ▶ `/0 (null) %00` (used in NULL strings)
- **URL Encoding Tool:**
  - ▶ Napkin: <http://www.0x90.org/releases/napkin/>

# Browser Encoding Exploits: XSS

- Browsers encoding can be carried out automatically
  - ▶ Via browser settings (View Menu Encoding you can set UTF-8, UNICODE UTF-7, User defined)
  - ▶ Via HTML web pages meta tags you can declare the encoding to be used: `<head>`  
`<meta http-equiv="Content-Type" content="text/html; charset=utf-8">`  
`...</head>`
- By enforcing encoding on web pages you make sure the browser interprets any special characters as data and markup and non script to be executed for XSS for example:
  - ▶ `<` becomes **&lt;**
  - ▶ `>` becomes **&gt;**
  - ▶ **&** becomes **&amp;**
  - ▶ `"` becomes **&quot;**

## Server Encoding Exploits : Double Encoding And Filter Evasion

- **Problem:** Attacker can try three potential encodings for back-slash character "\"
  - ▶ 0x5C( ASCII) %5c (UTF-8), %c0%af(UNICODE UTF-7)
  - ▶ Attack vector: `http://www.example.com/app`  
`..%c0%af..%c0af../winnt/system32/cmd.exe?/c+dir` to perform a dir command
- **Microsoft solution:** release patch to filter all encodings (e.g. MS IIS4 and IIS5)
- **Attacker try filter evasion:** double encoding
  - ▶ (1) hex encode the "\" => %5C
  - ▶ (2) encode the "%" portion = %25
  - ▶ **(3) Yields double encoded \ = %255c**

# Web Application Filter Evasions: XSS

## ■ The application server side validation filters:

- ▶ `http://[server]/[path]/[file].asp?id=70-305zzz`  
`<script>alert();</script>`

## ■ Attacker Encodes Javascript with addition of a new STYLE attribute on the element which can contain a **Dynamic Property**

## ■ Attacker deliver attack vector that Internet Explorer will execute:

- ▶ `http://[server]/[path]/[file].asp?id=70-305zzz+"&style="background-position-x:expression\0028\0065\0076\0061\006C\0028\0061\006C\0065\0072\0074\0028\0027pwn3d\0027\0029\0029\0029"`



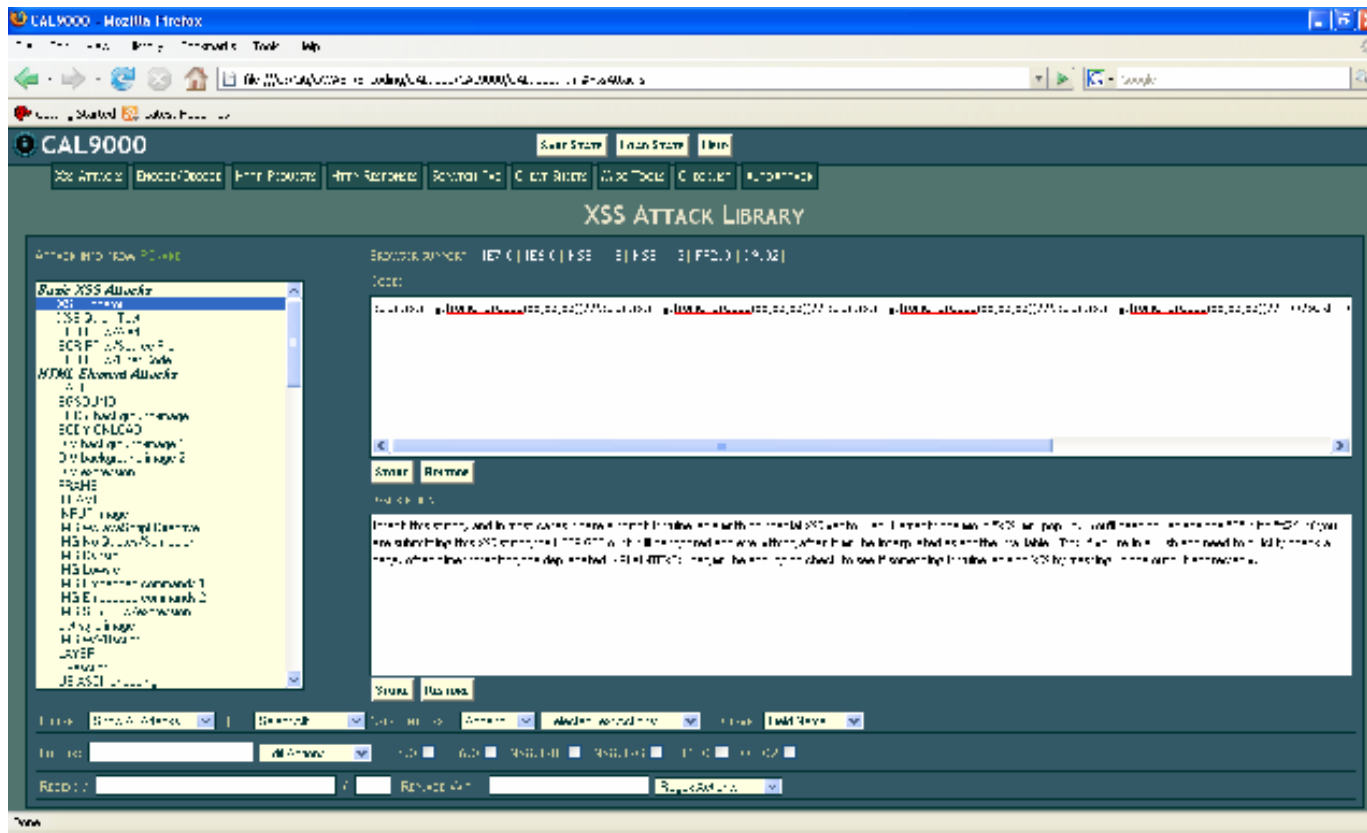


© 2014 Pearson Education, Inc. or its affiliate(s). All rights reserved.

Based on Robert Hansen (Rsnake) research: <http://ha.ckers.org/xss.html>

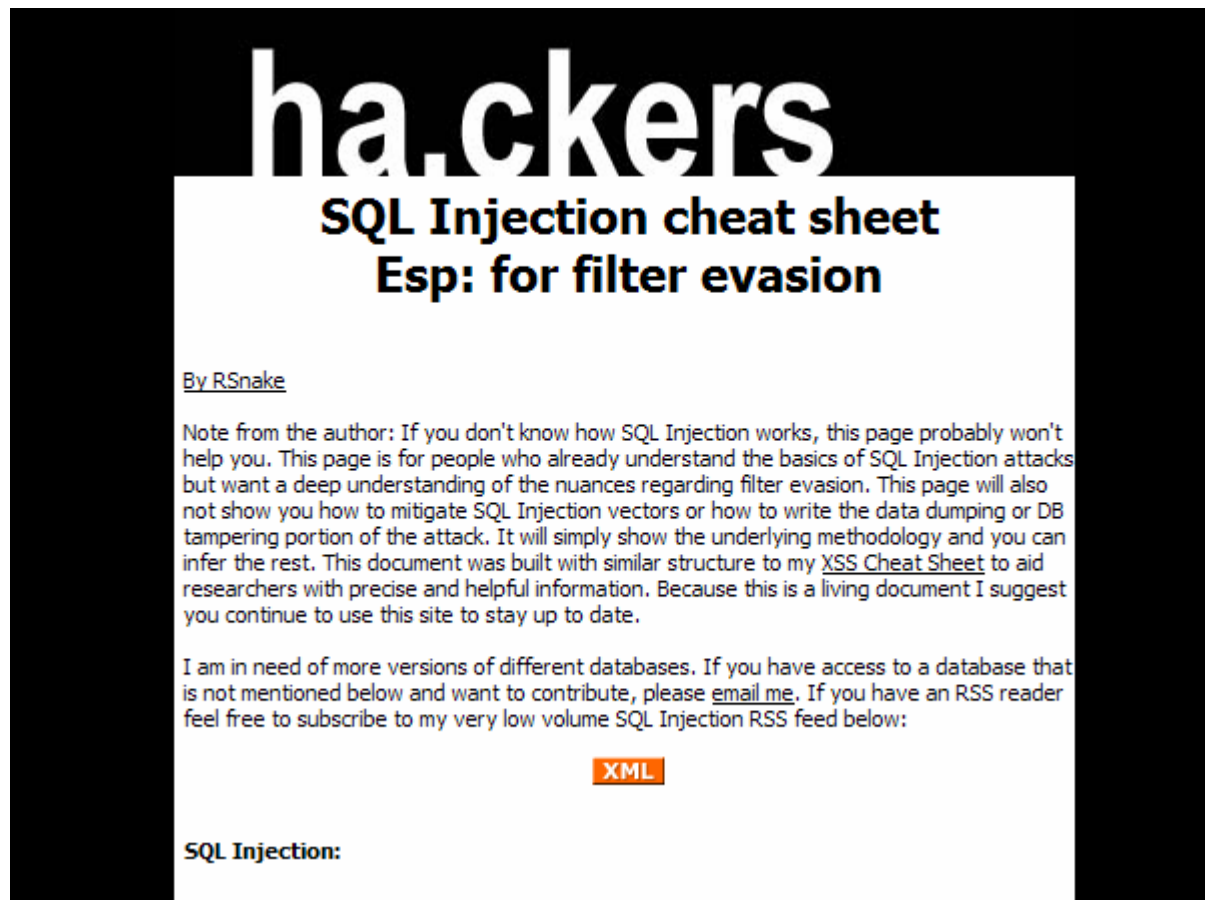
OWASP Project: [http://www.owasp.org/index.php/Category:OWASP\\_CAL9000\\_Project](http://www.owasp.org/index.php/Category:OWASP_CAL9000_Project)

Local Web Page: <file:///C:/Citi/OWASP/Encoding/CAL9000/CAL9000/CAL9000.html#top>



# SQL Injection Cheat Sheet

■ <http://ha.ckers.org/sqlinjection/>

A screenshot of a webpage titled "ha.ckers" in large white letters on a black background. Below the title, the text "SQL Injection cheat sheet" and "Esp: for filter evasion" is displayed in bold black font on a white background. The author is listed as "By RSnake". A note from the author explains the page's purpose: it is for those who already understand the basics of SQL Injection and want a deep understanding of filter evasion, showing methodology rather than specific attack vectors. It mentions a similar "XSS Cheat Sheet" and suggests subscribing to an RSS feed. A small orange button labeled "XML" is visible. The section "SQL Injection:" is partially visible at the bottom.

**ha.ckers**

**SQL Injection cheat sheet**  
**Esp: for filter evasion**

By [RSnake](#)

Note from the author: If you don't know how SQL Injection works, this page probably won't help you. This page is for people who already understand the basics of SQL Injection attacks but want a deep understanding of the nuances regarding filter evasion. This page will also not show you how to mitigate SQL Injection vectors or how to write the data dumping or DB tampering portion of the attack. It will simply show the underlying methodology and you can infer the rest. This document was built with similar structure to my [XSS Cheat Sheet](#) to aid researchers with precise and helpful information. Because this is a living document I suggest you continue to use this site to stay up to date.

I am in need of more versions of different databases. If you have access to a database that is not mentioned below and want to contribute, please [email me](#). If you have an RSS reader feel free to subscribe to my very low volume SQL Injection RSS feed below:

[XML](#)

**SQL Injection:**

---

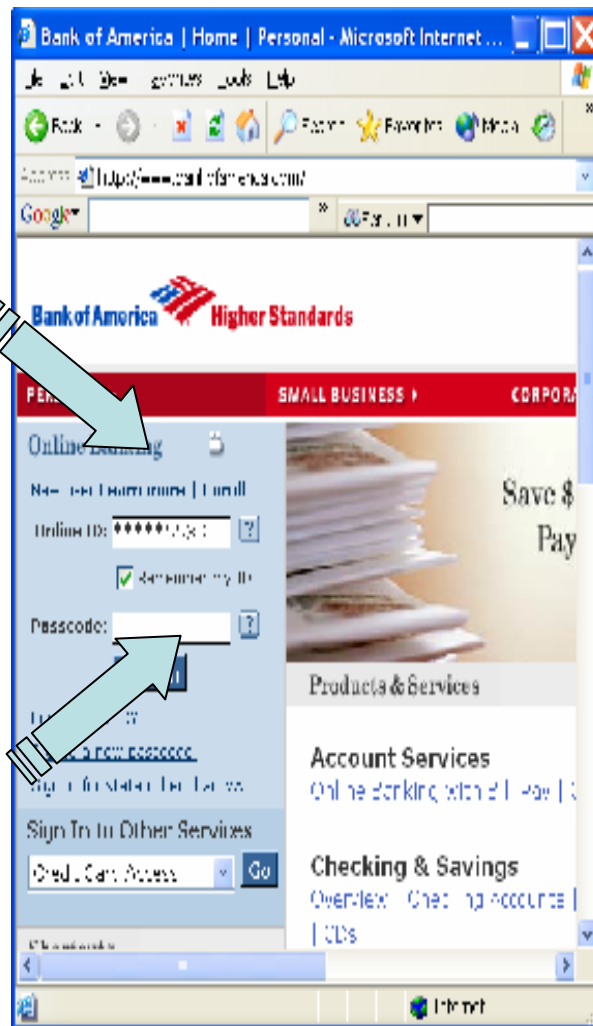
# **LIVE EXAMPLES PART I**

## **Input Validation Vulnerabilities**

### **Attack Vector Exploit Examples**

# How to Find IV Vulnerabilities: Web Application Security Assessments

**Manual Penetration Testing**



**Automated Vulnerability Scanning**

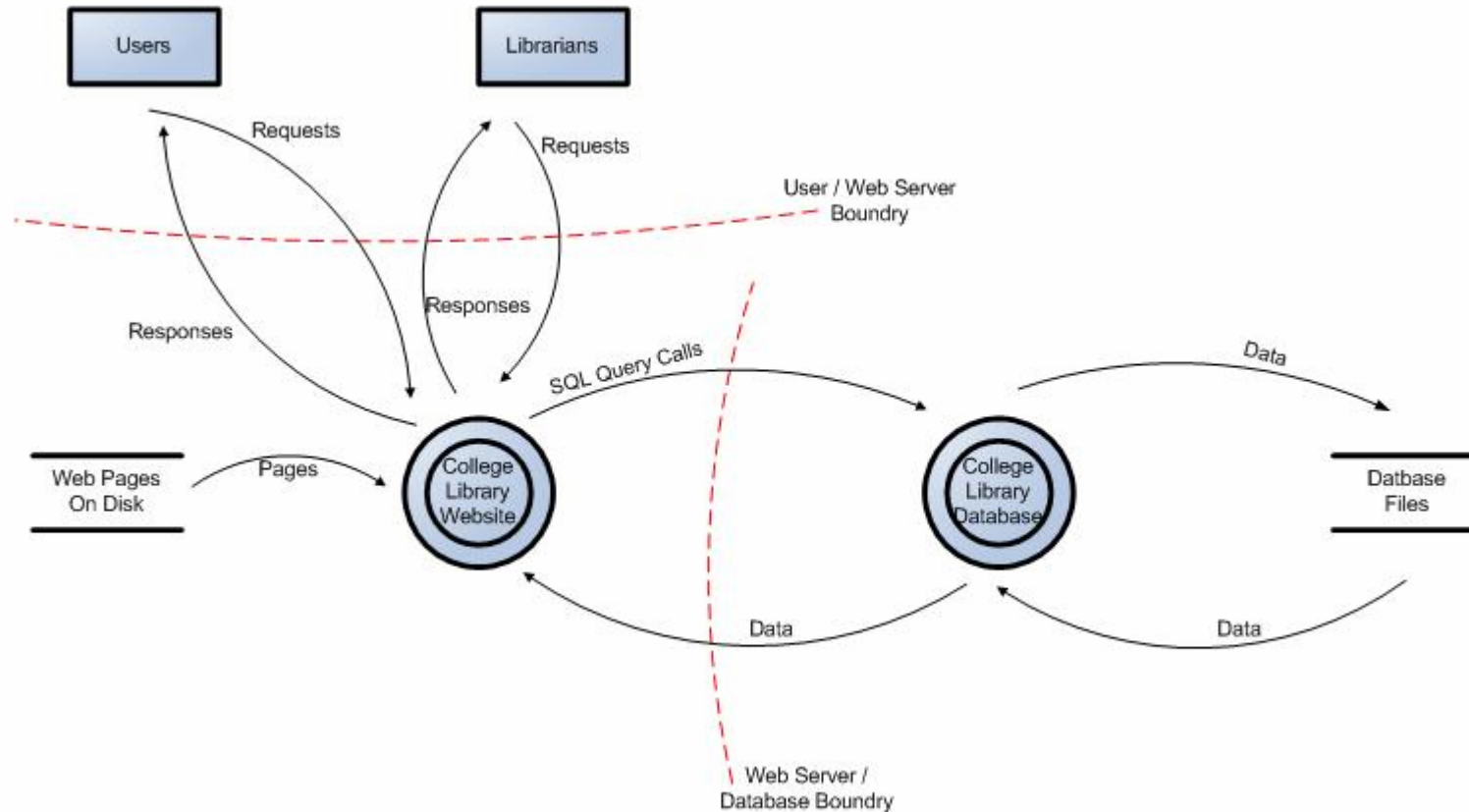
**Manual Code Review**

```
166 // check if the user wants userName set in
167 String rememberUserName = hreq.getParameter
168 if (rememberUserName != null) {
169     // set a cookie with the username in it
170     Cookie userNameCookie = new Cookie(COOK1
171     // set cookie to last for one month
172     userNameCookie.setMaxAge(2678400);
173     hres.addCookie(userNameCookie);
174 } else {
175     // see if the cookie exists and remove
176     Cookie[] cookies = hreq.getCookies();
177     if (cookies != null) {
178         for (int loop=0; loop < cookies.le
179             if (cookies[loop].getName().eq
180                 cookies[loop].setMaxAge(0)
181                 hres.addCookie(cookies[loc
182             }
183     }
184 }
185 }
186 }
187 }
188 //validate against the registered users
189 SignOnLocal signOn = getSignOnEjb();
190 boolean authenticated = signOn.authenticat
191 if (authenticated) {
192     // place a true boolean in the session
193     if (hreq.getSession().getAttribute(USE
194         hreq.getSession().removeAttribute(
195     }
196     hreq.getSession().setAttribute(USER_NP
197     // remove the sign on user key before
```

**Automated Static Code Analysis**



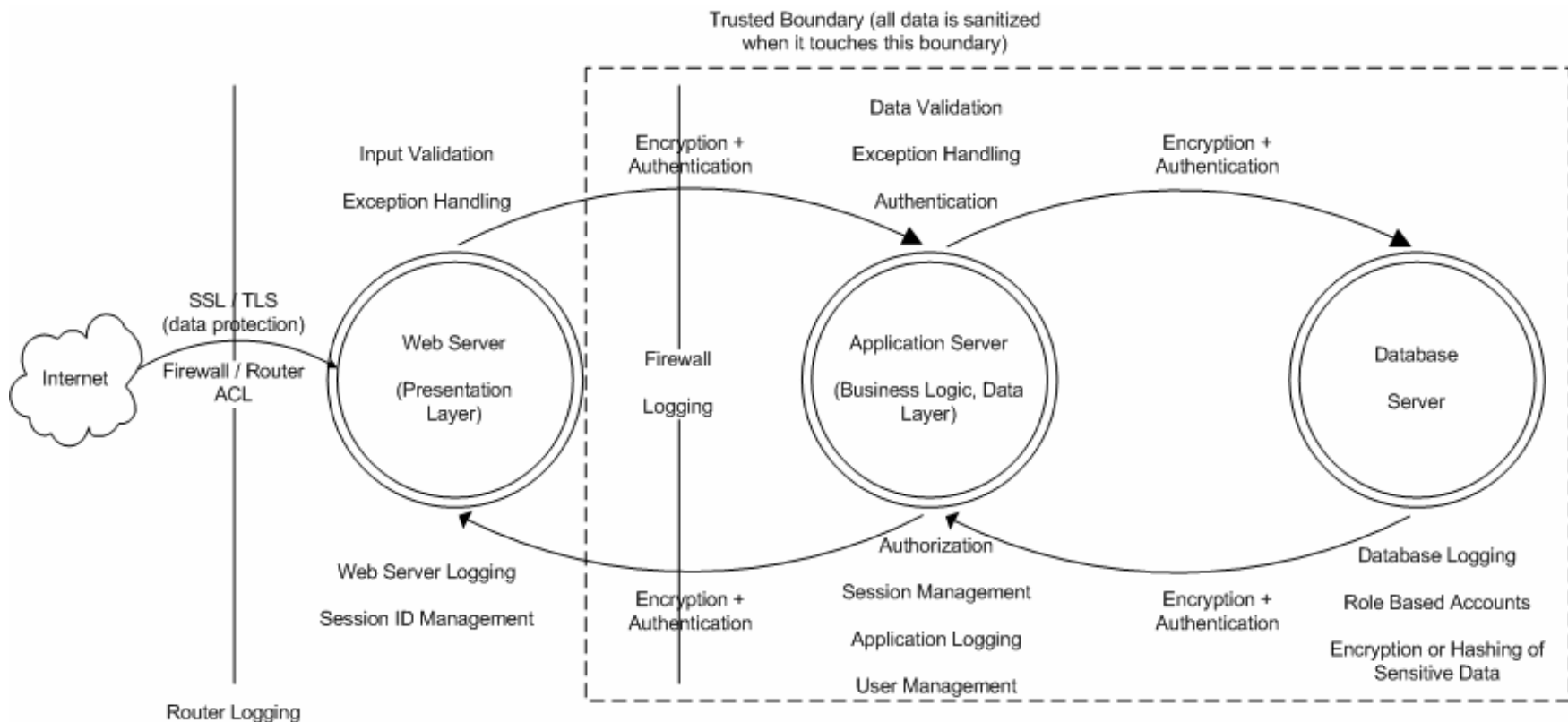
# How to Find Input Validation Flaws: Application Threat Modeling



[https://www.owasp.org/index.php/Application\\_Threat\\_Modeling](https://www.owasp.org/index.php/Application_Threat_Modeling)

# How to Find Input Validation Flaws: Secure Architecture Reviews

- Check input validation on every tier and when crossing trust boundaries



# How to protect web applications from IV attack vectors

- **Web Server Mitigations:** Apache Web Server Modules (e.g. mod rewrite, mod security), SunONE's NSAPI, Microsoft's ISAPI
- **Common Framework-based libraries validations:** use regular expressions for input validation/sanitization and output (HTML, URL) encoding
  - ▶ J2EE world the struts framework commons validators
    - <http://www.owasp.org/index.php/Struts>
    - [http://www.owasp.org/index.php/Data\\_Validation\\_\(Code\\_Review\)](http://www.owasp.org/index.php/Data_Validation_(Code_Review))
  - ▶ .NET framework validations implementations for XSS:
    - <http://msdn.microsoft.com/en-us/library/ms998274.aspx>
  - ▶ .NET framework validation strategies for SQL:
    - <http://msdn.microsoft.com/en-us/library/ms998271.aspx>
- **Secure APIs/Encoders**
  - ▶ .NET Anti XSS Libraries
    - <http://msdn.microsoft.com/en-us/security/aa973814.aspx>
  - ▶ OWASP ESAPI, AntiSamy Encoding Libraries
    - <http://www.owasp.org/index.php/ESAPI>
    - <http://www.owasp.org/index.php/AntiSamy>
    - [http://www.owasp.org/index.php/Category:OWASP\\_Encoding\\_Project](http://www.owasp.org/index.php/Category:OWASP_Encoding_Project)

---

# **LIVE EXAMPLES PART II**

## **Attack Vectors Filtering Examples:**

### **White-list, Black-list, Sanitization, Encoding Rules**

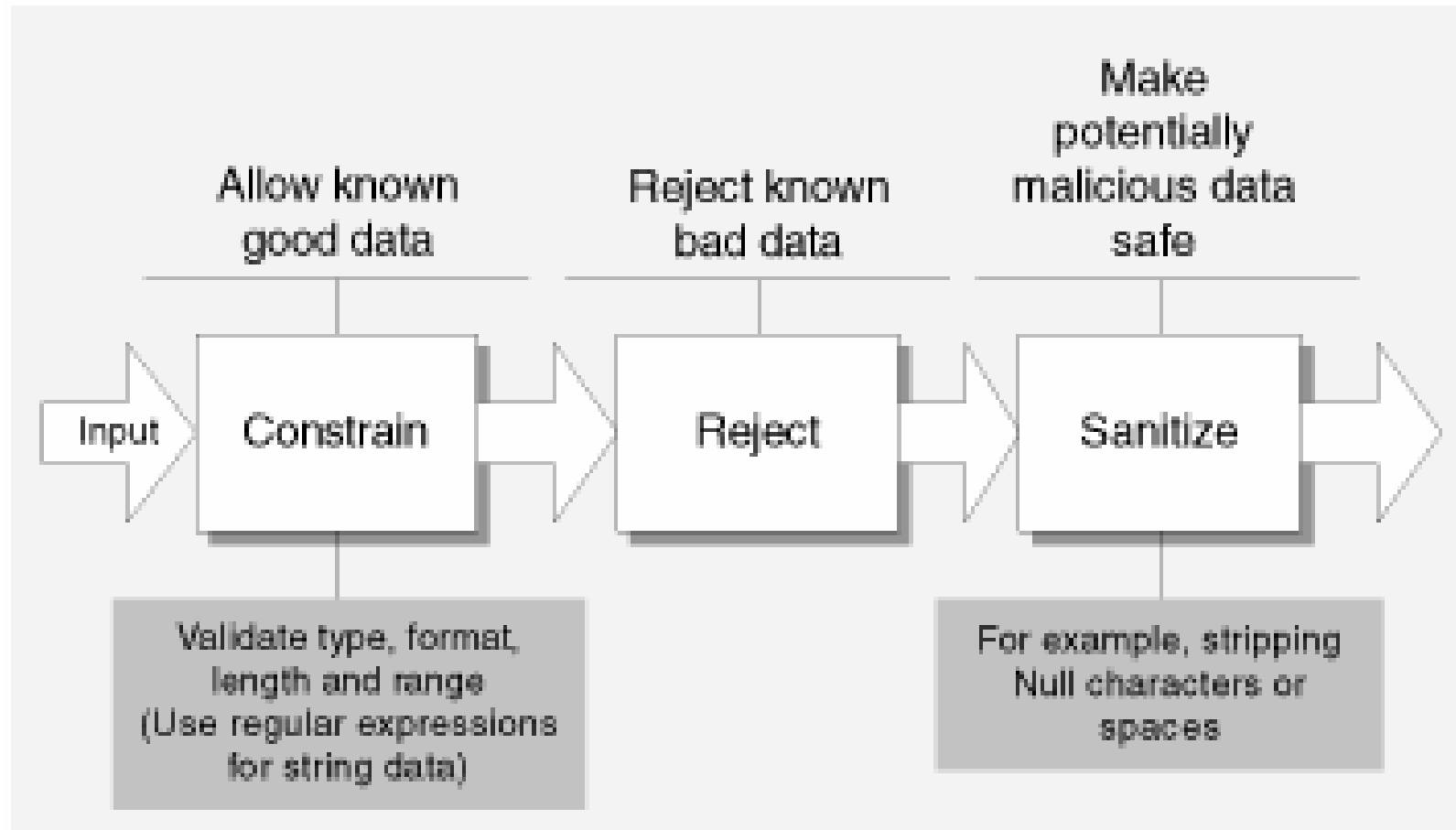


# Where to Validate? From Outside to Inside

Validation method or component / Evaluation criteria	Validation in front of the application and before the input data will arrive the web server	Inside the web server but before the "coded" part of the application	Inside the application with central modules	Within the code "form by form"
Examples	Web application firewalls (WAF)	<ul style="list-style-type: none"> <li>WAF web-server plugins</li> <li>Apache's mod_security</li> </ul>	Using the validation routines in Struts	Using validation controls in ASP.NET
Typical purpose	Protection of Common of the Shelf (COTS) web applications	Protection of COTS web application but one has control about the webserver	Self developed applications based on a framework or with validation services	Selfdeveloped applications based but not based on a framework
Individual advantages of such a solution	A web application firewall (WAF) can implemented independently of the underlying business application. There are no (or only few) changed necessary.	Similar to the WAF scenario but one can spare an additional proxy. This improves performance, maintaining and costs.	Central validation control point. It is easy to check that all input will be validated. Changes in the whitelist can implemented at a single point.	Easy to implement. Sufficient and rigid approach for small and middle size applications.
Individual disadvantages of such a solution	<ul style="list-style-type: none"> <li>Requires additional hardware and software</li> <li>Reduces performance and stability</li> <li>Needs extensive training of the WAF</li> <li>No programmatic interaction with the application.</li> </ul>	<ul style="list-style-type: none"> <li>Increases slightly the complexity of the workflow</li> <li>Reduces slightly performance and stability</li> <li>Needs extensive training of the filter.</li> <li>No programmatic interaction with the application.</li> </ul>	<ul style="list-style-type: none"> <li>Only applicable when such frameworks are used.</li> <li>There is no well accepted design pattern for input validation.</li> </ul>	<ul style="list-style-type: none"> <li>It is difficult to verify the completeness and rigidity of the validation.</li> <li>Changes of business data specification can result in complex refactoring.</li> </ul>
Occurs for the first time in which phase of software lifecycle	<ul style="list-style-type: none"> <li>Testing or</li> <li>Production</li> </ul>	<ul style="list-style-type: none"> <li>Testing or</li> <li>Production</li> </ul>	<ul style="list-style-type: none"> <li>Design</li> </ul>	<ul style="list-style-type: none"> <li>Implementation</li> </ul>
Performance issues	Moderate: "yet another proxy"	Good	Good	Good
False positive rate	Annoying	Annoying	Low	Low
Request specific error handling	None	None	Possible	Possible
Protected components	<ul style="list-style-type: none"> <li>Application</li> <li>Web server</li> </ul>	Application	Application	Application



# How to validate? Input Validation Strategies



Source: Design Guidelines for Secure Web Applications <http://msdn.microsoft.com/en-us/library/aa302420.aspx>

# White-list filtering: Accept known good

## ■ This strategy is also known as positive validation.

The idea is that you should check that the data is one of a set of tightly constrained known good values. Any data that doesn't match should be rejected. Data should be:

- ▶ Strongly typed at all times
- ▶ Length checked and fields length minimized
- ▶ Range checked if a numeric
- ▶ Unsigned unless required to be signed
- ▶ Syntax or grammar should be checked prior to first use or inspection
- ▶ If you expect a postcode, validate for a postcode (type, length and syntax):

## ■ Example: `Regex("^[A-Za-z0-9]{16}$")`

## Black-List Filtering: Reject Known Bad

- **This strategy, also known as "negative" or "blacklist" validation** that is if you don't expect to see characters such as %3f or JavaScript or similar, reject strings containing them.

- **Example:**

```
▶ public String removeJavascript(String  
  input) { Pattern p =  
    Pattern.compile("javascript",  
    CASE_INSENSITIVE); p.matcher(input);  
    return (!p.matches()) ? input : '' ; }
```

- **Problem**

- ▶ Maintenance ( up to 90 regular expressions, see the CSS Cheat Sheet in the Development Guide 2.0)
- ▶ Subjectible to Filter evasion

# Sanitize or Canonicalize

■ **Eliminate or translate characters** (such as to HTML entities or to remove quotes) in an effort to make the input "safe". Like blacklists, this approach requires maintenance and is usually incomplete.

■ **Example:**

▶ Remove special characters:

' " ` ; \* % \_ = & \ | \* ? ~ < > ^ ( ) [ ] { } \$ \ n \ r

▶ *public String quoteApostrophe(String input) { if (input != null) return input.replaceAll("[\\ ']", "&rsquo;"); else return null; }*

## Data Validation: Include Integrity Checks (Server Side Business Validations)

- **What:** Ensure that the data has not been tampered with (e.g. client-server) and is the same as before
- **Where:** Integrity checks must be included wherever data passes from a trusted to a less trusted boundary
- **What:** The type of integrity control (checksum, HMAC, encryption, digital signature) should be directly related to the risk of the data transiting the trust boundary.

### ■ Example:

- ▶ The account select option parameter ("payee\_id") is read by the code, and compared to an already-known list.

```
▪ if (account.hasPayee(  
    session.getParameter("payee_id") )) {  
    backend.performTransfer(  
        session.getParameter("payee_id") ); }  

```

---

# Q & A

**QUESTIONS**  
**ANSWERS**

# Book References

## ■ Further Reading:

- ▶ OWASP Guide 2.0: A guide to building secure web applications and web services
- ▶ OWASP Testing Guide v2
- ▶ OWASP Code Review vs1.0
- ▶ Mike Andrews, J. A Whittaker: How to break Web Software
- ▶ Mike Shema, Hack Notes; Web Security
- ▶ Tom Gallagher et al, Microsoft Press, Hunting Security Bugs
- ▶ David LeBlanc, Microsoft Press, Writing Secure Code 2<sup>nd</sup> ed)