

HTTP DATA INTEGRITY VALIDATOR

*Protect your  
web applications  
against attacks*



# Contents



- **Introduction**
- HDIV
- OWASP Top Ten & HDIV
- Performance
- Conclusions
- References

# Introduction



- Most of the applications are vulnerable
  - Gartner
    - 75% of the attacks are executed in the application layer
  - NIST
    - 92% of the vulnerabilities belong to the application layer

# Introduction



- Origin of attacks
  - All the attacks have something in common:

**SEND MALICIOUS INFORMATION  
TO THE WEB SERVER**

# Introduction



- What does information refer to?
  - Url address
  - Parameters
  - Cookies
  - Headers

# Introduction

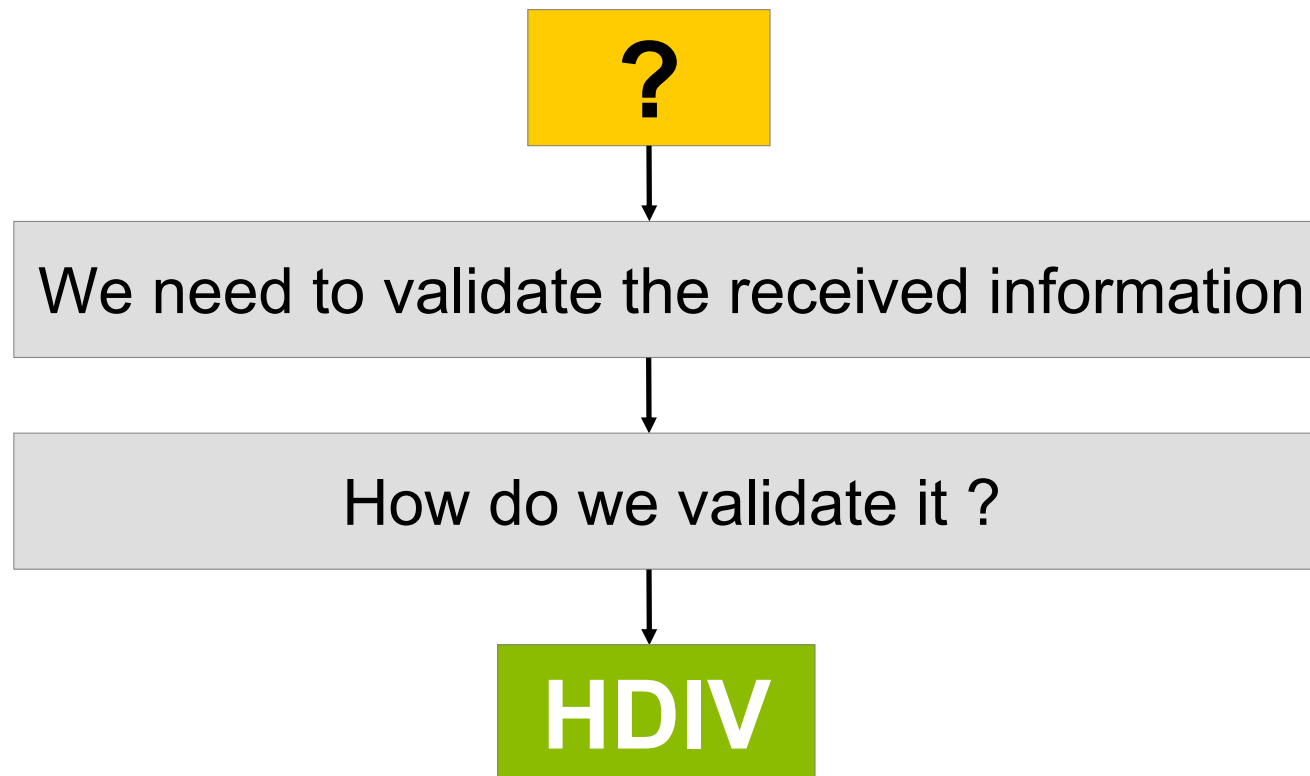


- Types of information
  - Editable data:
    - Textbox
    - Textarea
  
  - Non editable data:
    - The rest of the information

# Introduction



- How can we protect ourselves?



# Introduction



- How HDIV works
  - HDIV divides a request into two types of information:
    - Editable (textbox, textarea)
    - Non editable (the rest of the data)
  - Non editable data
    - Information generated in the server
    - Client should not (but it is possible) modify this information
    - HDIV guarantees that information has not been modified
    - HDIV hides from clients the real values of the non editable data, guaranteeing data confidentiality
    - So, it helps to eliminate vulnerabilities (known and unknown) of the non editable information

# Introduction



- How HDIV works
  - Editable data (textbox, textArea)
    - In this case, there is no base information to validate
    - Validation depends on the application and on the field that is being validated (name, telephone number, etc.)
    - It is the application who defines manually the validation for the editable data (a lot of information)
    - This causes a high risk to be vulnerable to many vulnerabilities
    - HDIV allows generic validations for the editable data, eliminating this risk to a great extent, for example:
      - Allow only alphanumeric characters
      - Not allow potentially dangerous characters, for example: > < = ‘

# Contents



- Introduction
- **HDIV**
- OWASP Top Ten & HDIV
- Performance
- Conclusions
- References

# HDIV



- Introduction

- Java Web Application Security Framework for Struts 1.x, Struts 2.x, Spring MVC and JSTL
  - New versions are being developed for JSF
- It eliminates most of the web vulnerabilities:
  - for non editable data
  - Most for the editable data (depends on the applied restrictions)
- Transparent for the programmers
  - Does not modify the source code of applications.
  - Can be applied to previously developed applications.
- Open Source project - Licensed under Apache License, Version 2.0

- Architecture

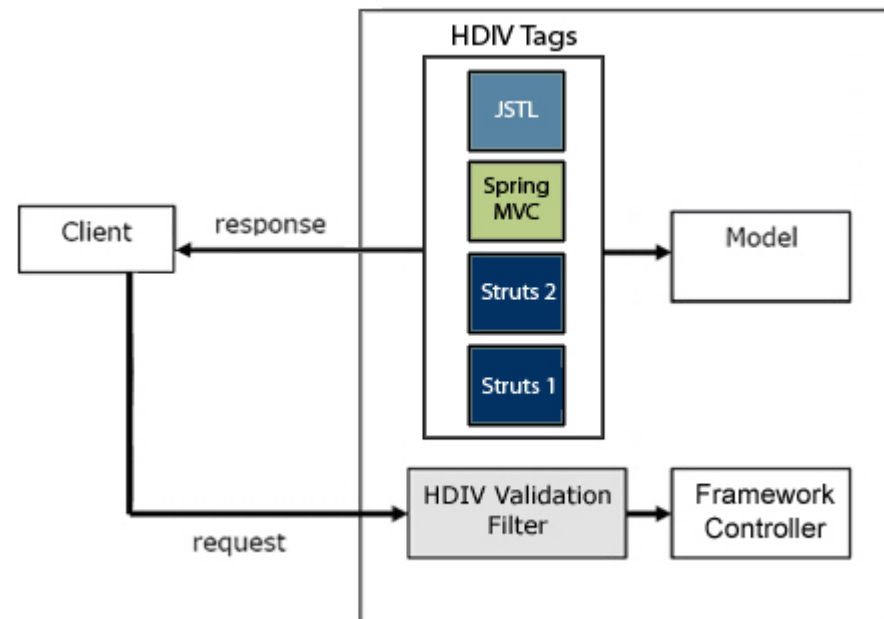
- All the client requests, except the first one (to home) must have an extra parameter called state (`_HDIV_STATE_`)

[http://host/action1.do?data=0&\\_HDIV\\_STATE\\_=6347dfhdfd84r73e94](http://host/action1.do?data=0&_HDIV_STATE_=6347dfhdfd84r73e94)

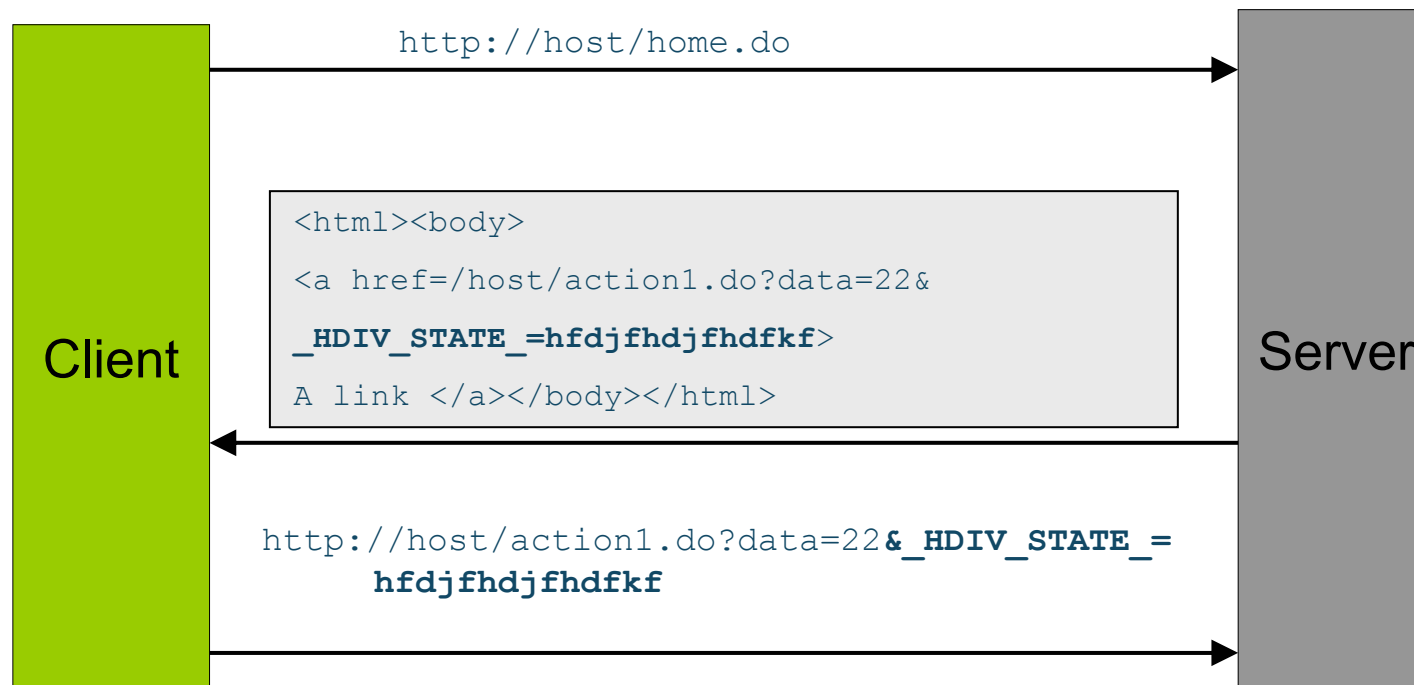
- State = recopilation of all non editable data that is part of a possible request to our server
- State makes possible to validate non editable data

## ■ Architecture

- HDIV modifies the HTML returned to clients, including the state to each possible request to the server.
- Then it validates client request, through a web filter, using the received state.



- Architecture



- Architecture

- Non editable data

- Validation is automatic (links, hidden fields, combo values, radio buttons, destiny pages, cookies, etc.)
    - It's done using state (`_HDIV_STATE_`) parameter
    - All detected attacks are logged including application user identity

## ■ Architecture

### – Editable data

- Validation for editable data (text and textarea) is done by generic rules defined in XML format
- There are base validations in the HDIV distribution that make it possible to solve most of the vulnerabilities
- It's possible to apply them only to some urls, some parameters or component type (text or textarea)
- All detected attacks are logged including application user identity

# HDIV



- HDIV has 3 working strategies:
  - Cipher
    - State is stored in client as a parameter
    - State is ciphered to avoid being modified by the client
  - Memory
    - State is stored in the server, in the user session (HttpSession)
  - Hash
    - State is stored in the client as a parameter
    - The value is not ciphered
    - In order to guarantee its integrity, the same hash is stored in the server to be able to compare with the original one

# HDIV



- Working strategies – Example:
  - Original HTML page (without HDIV):

```
<html>
<body>
<a href=/struts-examples/action1.do?data=22>LinkRequest</a>

<form method="post" action="/struts-examples/processSimple.do">
<input type="text" name="name" value="" />

<input type="password" name="secret" value="" />
<select name="color">
    <option value="10">Red</option>
    <option value="11">Green</option>
    <option value="21">Blue</option>
</select>

<input type="radio" name="rating" value="10" />Actually, I hate it.<br />
<input type="radio" name="rating" value="20" />Not so much.<br />
<input type="radio" name="rating" value="22" />I'm indifferent<br />
<textarea name="message" cols="40" rows="6"></textarea>
<input type="hidden" name="hidden" value="15" />
<input type="submit" value="Submit" />
</form>
</body></html>
```

# HDIV



- Working strategies - Cipher

```
<html>
<body>
<a href=/struts-examples/action1.do?data=0&_HDIV_STATE=6347dfhdfd84r73e9483494734837487>
  LinkRequest</a>

<form method="post" action="/struts-examples/processSimple.do">
<input type="text" name="name" value="" />
<input type="password" name="secret" value="" />
<select name="color">
  <option value="0">Red</option>
  <option value="1">Green</option>
  <option value="2">Blue</option>
</select>

<input type="radio" name="rating" value="0" />Actually, I hate it.<br />
<input type="radio" name="rating" value="1" />Not so much.<br />
<input type="radio" name="rating" value="2" />I'm indifferent<br />
<textarea name="message" cols="40" rows="6"></textarea>
<input type="hidden" name="hidden" value="0" />
<input type="hidden" name="_HDIV_STATE_" value="jkhdfhgd948dkfhdfdkhffjfdf" />
<input type="submit" value="Submit" />
</form>
<body></html>
```

# HDIV



- Working strategies - Hash

```
<html>
<body>
<a href="/struts-examples/action1.do?data=0&_HDIV_STATE=wJTawJTawbVALQzFKJUMzJTQwJTE>
LinkRequest</a>

<form method="post" action="/struts-examples/processSimple.do">
<input type="text" name="name" value="" />
<input type="password" name="secret" value="" />
<select name="color">
  <option value="0">Red</option>
  <option value="1">Green</option>
  <option value="2">Blue</option>
</select>

<input type="radio" name="rating" value="0" />Actually, I hate it.<br />
<input type="radio" name="rating" value="1" />Not so much.<br />
<input type="radio" name="rating" value="2" />I'm indifferent<br />

<textarea name="message" cols="40" rows="3"></textarea>
<input type="hidden" name="hidden" value="0" />
<input type="hidden" name="_HDIV_STATE_" value="lRUMlQCUwM0YlMUy1RMUCwMRQlQzFj" />
<input type="submit" value="Submit" />
</form></body> </html>
```

# HDIV



- Working strategies - Memory

```
<html>
<body>
<a href=/struts-examples/action1.do?data=0&_HDIV_STATE=0-1-526189>
  LinkRequest</a>
<form method="post" action="/struts-examples/processSimple.do">
<input type="text" name="name" value="" />
<input type="password" name="secret" value="" />
<select name="color">
  <option value="0">Red</option>
  <option value="1">Green</option>
  <option value="2">Blue</option>
</select>
<input type="radio" name="rating" value="0" />Actually, I hate it.<br />
<input type="radio" name="rating" value="1" />Not so much.<br />
<input type="radio" name="rating" value="2" />I'm indifferent<br />
<textarea name="message" cols="40" rows="3"></textarea>
<input type="hidden" name="hidden" value="0" />
<input type="hidden" name="_HDIV_STATE_" value="0-2-526189" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

# Contents



- Introduction
- HDIV
- **OWASP Top Ten & HDIV**
- Performance
- Conclusions
- References

# OWASP TOP TEN & HDIV



- OWASP (Open Web Application Security Project)
  - OWASP publishes a list of the most important vulnerabilities
  - OWASP TOP 10 – 2007 UPDATE:
    - A1 - Cross site scripting
    - A2 - Injection Flaws
    - A3 - Insecure Remote File Include
    - A4 - Insecure Direct Object Reference
    - A5 - Cross – site Request Forgery
    - A6 - Information Leakage and Improper Error Handling
    - A7 - Broken Authentication and Session Management
    - A8 - Insecure Cryptographic Storage
    - A9 - Insecure communications
    - A10 - Failure to restrict URL access

# OWASP TOP TEN & HDIV



- Cross site scripting or XSS (A1)
  - It is the most extended vulnerability
  - It allows the execution of scripts (Javascript,HTML,..) in the client's browser
  - There are different types ofXSS:
    - Reflected: when the page shows the data (script) sent by the attacker directly
    - Stored: when the script is stored in the server (DB, file). This is specially dangerous because it can affect many users: forums, etc.

# OWASP TOP TEN & HDIV



- Cross site scripting or XSS (A1) - Example

<http://myserver/myapp/XSSDemo?userInput=bar>

### XSS Vulnerability Sample

Enter string here:

---

Output from last command: bar

# OWASP TOP TEN & HDIV



- Cross site scripting or XSS (A1) - Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head> <title>XSS Vulnerability Sample</title> </head>
<body>
  <h1>XSS Vulnerability Sample</h1>
  <form method="GET" action="XSS.jsp">
    Enter string here:
    <input type="text" name="userInput" size=50>
    <input type="submit" value="Submit">
  </form>
  <br> <hr> <br>
  Output from last command: <%=
  request.getParameter("userInput") %>
</body>
</html>
```

# OWASP TOP TEN & HDIV



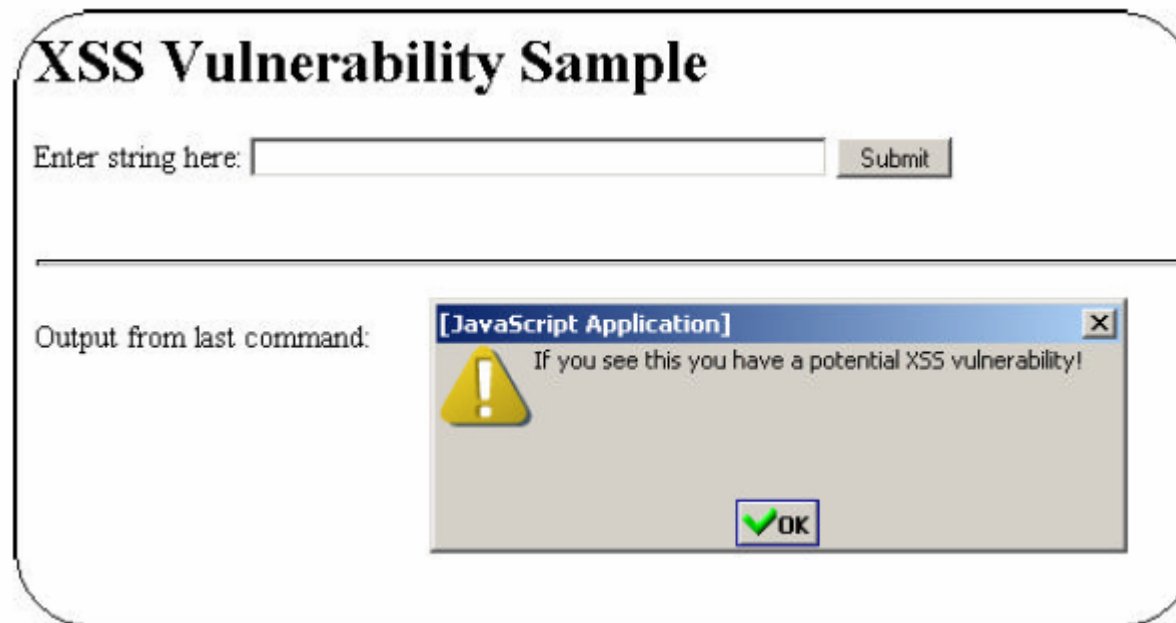
- Cross site scripting or XSS (A1) - Example
  - If we type the following text in the textbox:

```
<script>  
  alert("If you see that you have a potential XSS Vulnerability !");  
</script>
```

# OWASP TOP TEN & HDIV



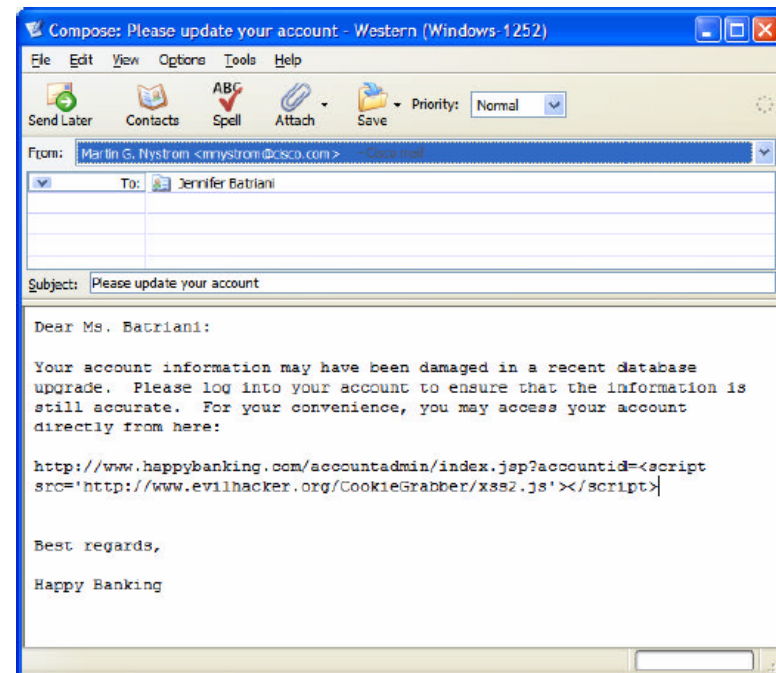
- Cross site scripting or XSS (A1) - Example



# OWASP TOP TEN & HDIV



- Cross site scripting or XSS (A1) - Consequences
  - Session hijack
  - Page defacement
  - It's a base for the phishing



# OWASP TOP TEN & HDIV



- Cross Site Scripting (A1) - **HDIV**
  - Non editable data:
    - HDIV eliminates this vulnerability
  - Editable data
    - HDIV provides generic validations to avoid this vulnerability

# OWASP TOP TEN & HDIV



- Cross Site Scripting (A1) - Recommendations
  - Use framework' tags:
    - They escape the data output avoiding XSS

# OWASP TOP TEN & HDIV



- Injection Flaws (A2)
  - The most important: SQL injection
  - It allows the modification of the SQL queries executed in the server

# OWASP TOP TEN & HDIV



- Injection Flaws (A2) - Example

```
http://host?data=12'
```

```
String data = request.getParameter("data");  
String sql= "select field1 from table1 where id='" + data  
+ '"
```

```
select field1 from table1 where id='12'
```

# OWASP TOP TEN & HDIV



- Injection Flaws (A2) - Example

```
http://host?data=12' or '1'='1
```

```
String data = request.getParameter("data");  
String sql= "select field1 from table1 where id='" + data  
+ '"
```

```
select field1 from table1 where id='12' or '1'='1'
```

# OWASP TOP TEN & HDIV



- Injection Flaws (A2) - Consequences
  - DB modification
  - Access to other's people data
  - etc.

# OWASP TOP TEN & HDIV



- SQL Injection (A2) - **HDIV**
  - Non editable data:
    - HDIV eliminates this vulnerability
  - Editable data
    - HDIV provides generic validations to avoid this vulnerability

# OWASP TOP TEN & HDIV



- SQL Injection (A2) - Recommendations
  - Use PreparedStatement in the queries
    - It solves this vulnerability totally
    - It requires code checking to guarantee that it's been used correctly

# OWASP TOP TEN & HDIV

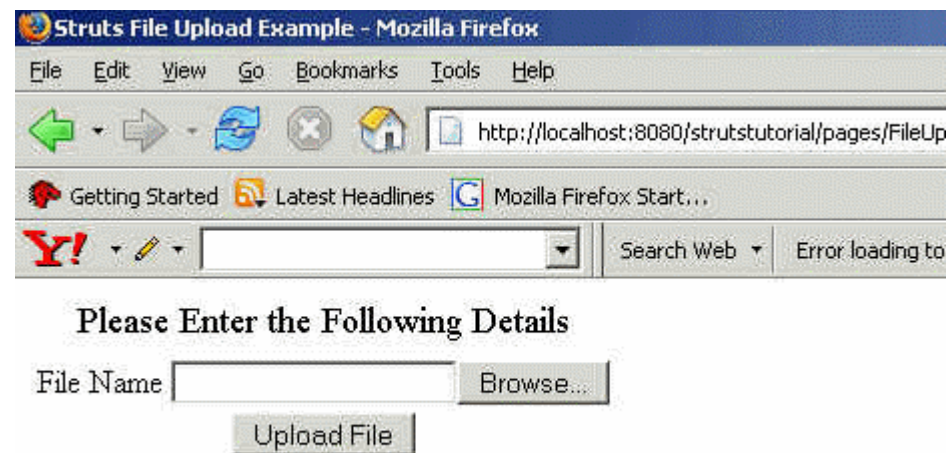


- Malicious File Execution (A3)
  - It means the execution of files which has been previously uploaded to the server or existing at the server
  - Usually related with non editable data
    - url parameters
    - forms parameters
  - Platforms based on PHP are specially vulnerable

# OWASP TOP TEN & HDIV



- Malicious File Execution (A3) - Example



# OWASP TOP TEN & HDIV



- Malicious File Execution (A3) - Consequences
  - Attacker can executed a program uploaded by him
  - Consequences depend on server configuration

# OWASP TOP TEN & HDIV



- Malicious File Execution (A3) - **HDIV**
  - Non editable data:
    - HDIV eliminates this vulnerability
    - In addition to that HDIV hides non editable data, making difficult to know server's file system paths
  - Editable data
    - It is not usual to use editable data for this type of attack
    - HDIV provides generic validations to avoid this vulnerability

# OWASP TOP TEN & HDIV



- Insecure Direct Object Reference (A4)
  - Usually a HTML page is full of references of server information (BD identifiers, file paths, etc.)
  - Clients can modify this data and access data that don't belong to them

# OWASP TOP TEN & HDIV



- Insecure Direct Object Reference (A4) - Example
  - In this case the client can choose an account number that doesn't belong to him

**User Accounts**

Select an account:

```
<select name="accounts">
  <option value="20771111422000086456">account 1</option>
  <option value="20771111422000086487">account 2</option>
</select>
```

# OWASP TOP TEN & HDIV



- Insecure Direct Object Reference (A4)
  - Consequences
    - An attacker can access data that don't belong to him
    - An attacker can modify or create data in the name of another person

# OWASP TOP TEN & HDIV



- Insecure Direct Object Reference (A4) - **HDIV**
  - Non editable data:
    - HDIV avoids this vulnerability
  - Editable data:
    - This vulnerability only affects to non editable data

# OWASP TOP TEN & HDIV



- Cross Site Request Forgery (A5) - **HDIV**
  - It is based on causing an authenticated user to make a request to help the attacker
  - This type of attack has similarities to XSS
  - XSS requires the attacker to inject unauthorized code into a website, while CSRF merely transmits unauthorized commands from a user the website trusts.

# OWASP TOP TEN & HDIV



- Cross Site Request Forgery (A5) -  
Example

– An attacker has posted this XSS vulnerable message on a forum:

```

```

# OWASP TOP TEN & HDIV



- Cross Site Request Forgery (A5) -  
Consequences
  - Execution of an action in the name of another person
  - For example, as we have seen previously, to make a transaction to the attacker's account

# OWASP TOP TEN & HDIV



- Cross Site Request Forgery (A5) - **HDIV**
  - HDIV adds a token in each form and link of the HTML response, making difficult a possible attack

# OWASP TOP TEN & HDIV



- Information Leakage and improper error Handling (A6)
  - It happens when we give the attacker too much information.

# OWASP TOP TEN & HDIV



- Information Leakage and improper error Handling (A6) - **Examples**
  - Visualization of the stack traces
  - DB identifiers are included in the HTML
  - HTTP error codes (500,...)

# OWASP TOP TEN & HDIV



- Information Leakage and improper error Handling (A6) - Consequences
  - The obtained information is used to execute other types of attacks:
    - Sql Injection (A2)
    - Insecure Direct Object Reference (A4)
    - etc.
  - This information its very useful for automatic auditing tools

# OWASP TOP TEN & HDIV



- Information Leakage and improper error Handling (A6) - **HDIV**
  - HDIV's catches uncontrolled exceptions
  - HDIV hides the non editable data values
    - It reduces the information amount available for the attacker
    - Consequently, it reduces the risk of this type of attack

# OWASP TOP TEN & HDIV



- Information Leakage and improper error Handling (A6) - Recommendations
  - Define a generic exception handler:
    - Web frameworks such as Struts and Spring MVC makes it possible to define exception handlers
    - Not include technical information in the error message sent to the client
    - Use server's logging mechanisms to obtain information about the error

# OWASP TOP TEN & HDIV



- Broken Authentication and session management (A7)
  - It means to break the authentication system or the session management system
  - Systems provided by the servers are secure

# OWASP TOP TEN & HDIV



- Broken Authentication and session management (A7) - Example
  - An attacker makes a request to the server with a random jsessionID (session identifier)

# OWASP TOP TEN & HDIV



- Broken Authentication and session management (A7) - Consequences
  - Supplant another person's identity

# OWASP TOP TEN & HDIV



- Broken Authentication and session management (A7) - **HDIV**
  - HDIV doesn't interfere in the authentication system
    - It is recommended to use the one offered by the application server
  - HDIV doesn't interfere in the session management mechanism
    - It is recommended to use the mechanisms offered by the application server

# OWASP TOP TEN & HDIV



- Insecure Cryptographic storage (A8)
  - This vulnerability is based on not having an appropriate (unsecure) ciphered storage system
  - HDIV doesn't cover this vulnerability

# OWASP TOP TEN & HDIV



- Insecure Cryptographic storage (A8) -  
Recomendations
  - Use hashes instead of encryption
  - Use standards encryption algorithms

# OWASP TOP TEN & HDIV



- Insecure Communications (A9)
  - This vulnerability is based on not having secure communication systems (SSL)
  - HDIV doesn't cover this vulnerability
  - This vulnerability is solved by configuring the server properly

# OWASP TOP TEN & HDIV



- Failure to restrict url Access (A10)
  - This vulnerability is based on accessing an unauthorized url

# OWASP TOP TEN & HDIV



- Failure to restrict url Access (A10) - Example

- Access to an administration url which hasn't been provided to users:

`http://www.host.com/admin`

- It is usual to find pages which security depends on the fact that the attacker doesn't know the url address to access to it

# OWASP TOP TEN & HDIV



- Failure to restrict url Access (A10) -  
Consequences
  - Access to a forbidden url address
  - Execution of administrative tasks
  - etc.

# OWASP TOP TEN & HDIV



- Failure to restrict url Access (A10) - **HDIV**
  - HDIV eliminates completely this vulnerability by not allowing to access to any url which hasn't been previously sent to the client

# Contents



- Introduction
- HDIV
- OWASP Top Ten & HDIV
- **Performance**
- Conclusions
- References

# Performance



- Introduction

- HDIV's performance depends on the chosen working strategy

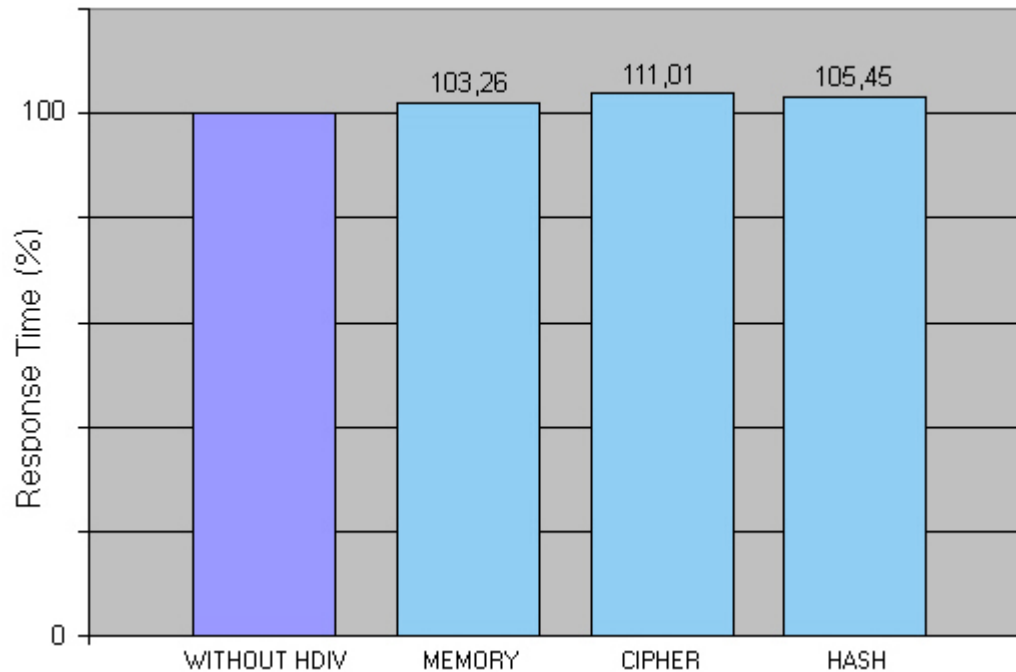
- Strategies:

- Cipher
    - Memory
    - Hash

# Performance



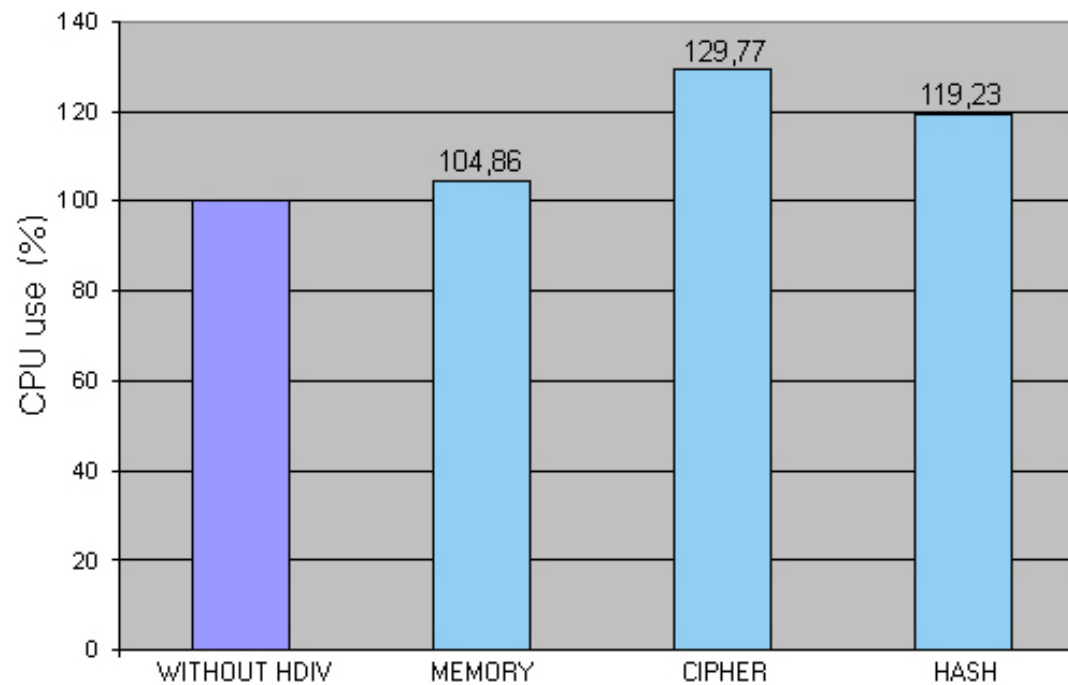
- Response time



# Performance



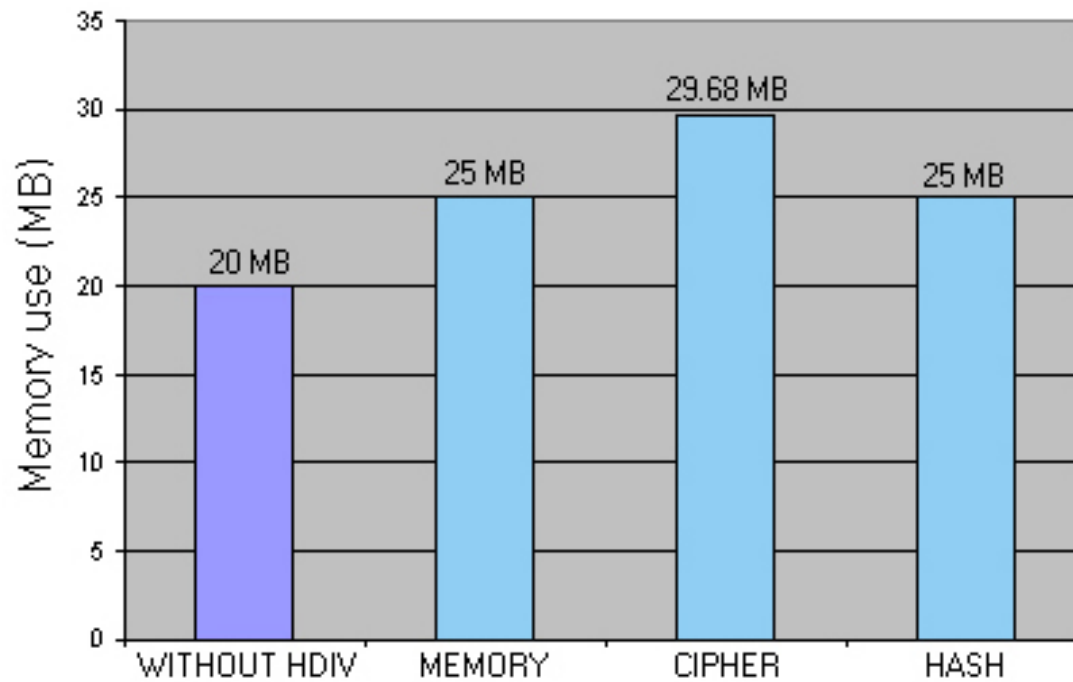
- CPU use



# Performance



- Memory use



# Contents



- Introduction
- HDIV
- OWASP Top Ten & HDIV
- Performance
- **Conclusions**
- References

# Conclusions



Vulnerability	HDIV
A1	✓
A2	✓
A3	✓
A4	✓
A5	✓
A6	✓
A7	Delegates on server
A8	Delegates on server
A9	HDIV doesn't cover
A10	✓

# Conclusions



- HDIV helps to eliminate most of the web vulnerabilities:
  - 7 of 10 defined by OWASP
  - A9 is not covered by HDIV
  - The rest (A7,A8) are covered by the application server
- It is transparent for the programmers
- It is possible to apply HDIV to applications that has been previously developed using Struts 1.x, Struts 2.x, Spring MVV or JSTL
- Allows to know the attacks and the identity of the attacker (on authenticated web)

# Contents



- Introduction
- HDIV
- OWASP Top Ten & HDIV
- Performance
- Conclusions
- **References**

# References



- HDIV
  - <http://www.hdiv.org>
- Struts
  - <http://struts.apache.org>
- Spring MVC
  - <http://www.springframework.org/docs/reference/mvc.html>
- OWASP
  - <http://www.owasp.org/>