OWASP TOP 10(2013)
java 开发安全实践

杭州安恒信息技术有限公司
刘志乐

OWASP 中国
The Open Web Application Security Project

About Me

OWASP 中国
The Open Web Application Security Project

- **刘志乐（Tony）**
  - OWASP中国区委员
  - OWASP中国杭州分会区域负责人
  - 安恒安全服务部总监
  - 浙江省计算机信息系统安全协会安全技术专业委员会副主任
  - 2011年中国计算机网络安全年会演讲嘉宾
  - 2011年OWASP亚洲峰会演讲嘉宾
  - ISF2011上海演讲嘉宾
  - 2012年OWASP AppSec Asia悉尼峰会演讲嘉宾
  - 2012年第四届中国云计算大会演讲嘉宾
  - 2012年计算机网络安全年会演讲嘉宾
  - 2013年第二届等级保护技术大会演讲嘉宾

目录

OWASP 中国
The Open Web Application Security Project

- OWASP TOP 10（2013）
- 安全开发编程概述
- 利用开源框架提升安全
- OWASP TOP 10 安全实践

OWASP TOP 10

OWASP 中国
The Open Web Application Security Project

- OWASP TOP 10
  - Injection(注入)
  - Broken Authentication and session Manager（失效的认证和会话管理）
  - XSS（跨站脚本）
  - Insecure Direct Object References（不安全的直接对象引用）
  - Security Misconfiguration（安全配置错误）
  - Sensitive Data Exposure（敏感信息泄露）
  - Missing Function Level Access Control（功能级访问控制缺失）
  - CSRF（跨站请求伪造）
  - Using Known Vulnerable Components（使用含有已知漏洞的组件）
  - Unvalated Redirects and Forwards（未验证的重定向和转发）

目录

OWASP 中国
The Open Web Application Security Project

- OWASP TOP 10（2013）
- 安全开发编程概述
- 利用开源框架提升安全
- OWASP TOP 10 安全实践

OWASP 中国
The Open Web Application Security Project

- **安全设计**
  - 在安全设计阶段，特别加入以下两方面的考虑
    - 减少攻击界面。例如，对一个网络软件的设计，它需要监听那些网络端口，是否可以减少监听端口的数目？那些用户可以与这些端口建立连接，是否要加强身份验证？
    - 深层防御。底层模块的设计中，假设上层模块有可能出现安全漏洞。对传递的数据考虑进一步校验

OWASP 中国
The Open Web Application Security Project

- **安全编程**
  - **独立、完整且集中的输入验证**

    创建并使用了独立的用户输入验证模块以完成对所有用户的输入校验，以此可带来：

    - 统一的输入检测策略
    - 统一的验证逻辑
    - 统一的错误验证处理
    - 降低升级和维护成本

OWASP 中国
The Open Web Application Security Project

- **校验全部的程序输入**

  保证所有变量在使用之前都经过严格的校验，防止被污染的数据进入程序。

- **校验全部的输入长度**

  通过限制输入长度，可以有效的控制一些攻击使其不给系统带来过大的威胁：

  - SQL Inject
  - XSS
  - File Include

OWASP 中国
The Open Web Application Security Project

- **校验全部的输入类型**

  不同的程序所接收到的参数类型应严格区分并校验，对于非法的类型应有相关异常进行处理以防止其进入程序。

- **不使用任何方式验证失败的数据**

  当程序对某个数据校验失败时（如：校验数据类型），相关的异常处理程序应抛弃该数据并中断操作，而不应对数据进行任何的修复尝试。

- **对HTTP所有内容进行校验**

  除需对传统的HTTP GET、POST等数据进行严格校验外，还应对HTTP内所有可能使用到的字段进行校验，防止字段中包含恶意字符而污染程序，如：

  - Referer
  - Host
  - Cookie
  - ……

OWASP 中国
The Open Web Application Security Project

- **校验向用户输出的数据**

  当程序通过查询后台数据库或其他方式从后台获取数据后，在将数据输出给用户前应对该数据进行校验，校验其中是否包含有非法字符、可执行客户端脚本等恶意信息。

- **使用安全的SQL查询方式**

  在进行SQL查询时，必须使用安全的查询方式，如：Prepared Statement，以避免查询语句中由用户恶意插入SQL语句所带来的风险。

- **禁止使用JavaScript进行任何校验**

  由于JavaScript为客户端脚本，因此任何试图使用JavaScript对用户数据进行校验的行为都可能被用户构造的本地脚本所绕过，因此，所有校验工作应由服务端程序完成而不是客户端。

- **使用安全、统一的编码或转义方式**

  创建并使用独立、统一的编码或转移方式，而且编码或转移中，至少应包含对以下类别数据的编码或转移：

  - 可能造成SQL注入的数据，如：分号、单引号等
  - 可能造成XSS的数据，如：script、javascript等

- **设定有安全的权限边界**

  所有的程序都应清楚的了解到自己能做什么，而在其所能做的范围之外，均属于其权限边界之外，应严格禁止对其权限之外的任何操作。

- **校验被调用的后台命令**

  若程序需要调用后台可执行程序，则在调用时，应通过使用完整路径或对程序进行HASH校验等方式保证程序的调用正确。

OWASP 中国
The Open Web Application Security Project

- **校验被调用的文本或配置文件**

  若程序需要调用后台文本或配置文件，则在调用前，应相对文件或配置文件的完整性和有效性进行检查，以确保读入的文本或配置文件是正确可用的。

- **确保程序所记录的日志可控**

  若程序需要记录额外的操作日志等信息，应保证这些日志中的某些或全部内容不来自用户输入，否则用户可能通过外部恶意提交信息的方式填充日志。

目录

OWASP 中国
The Open Web Application Security Project

- OWASP TOP 10（2013）
- 安全开发编程概述
- 利用开源框架提升安全
- OWASP TOP 10 安全实践

OWASP 中国
The Open Web Application Security Project

- ESAPI（**Enterprise Security API**）

其实简单一点来说，ESAPI就是为编写出更加安全的代码设计出来的一些API，方便使用者 调用，从而方便的编写安全的代码。它本身是开源的，同时提供各种版本。

代码下载地址：http://code.google.com/p/owasp-esapi-java/

下图显示了提供的API与OWASP列出的10个安全问题的涵盖关系：

防护框架

OWASP 中国
The Open Web Application Security Project

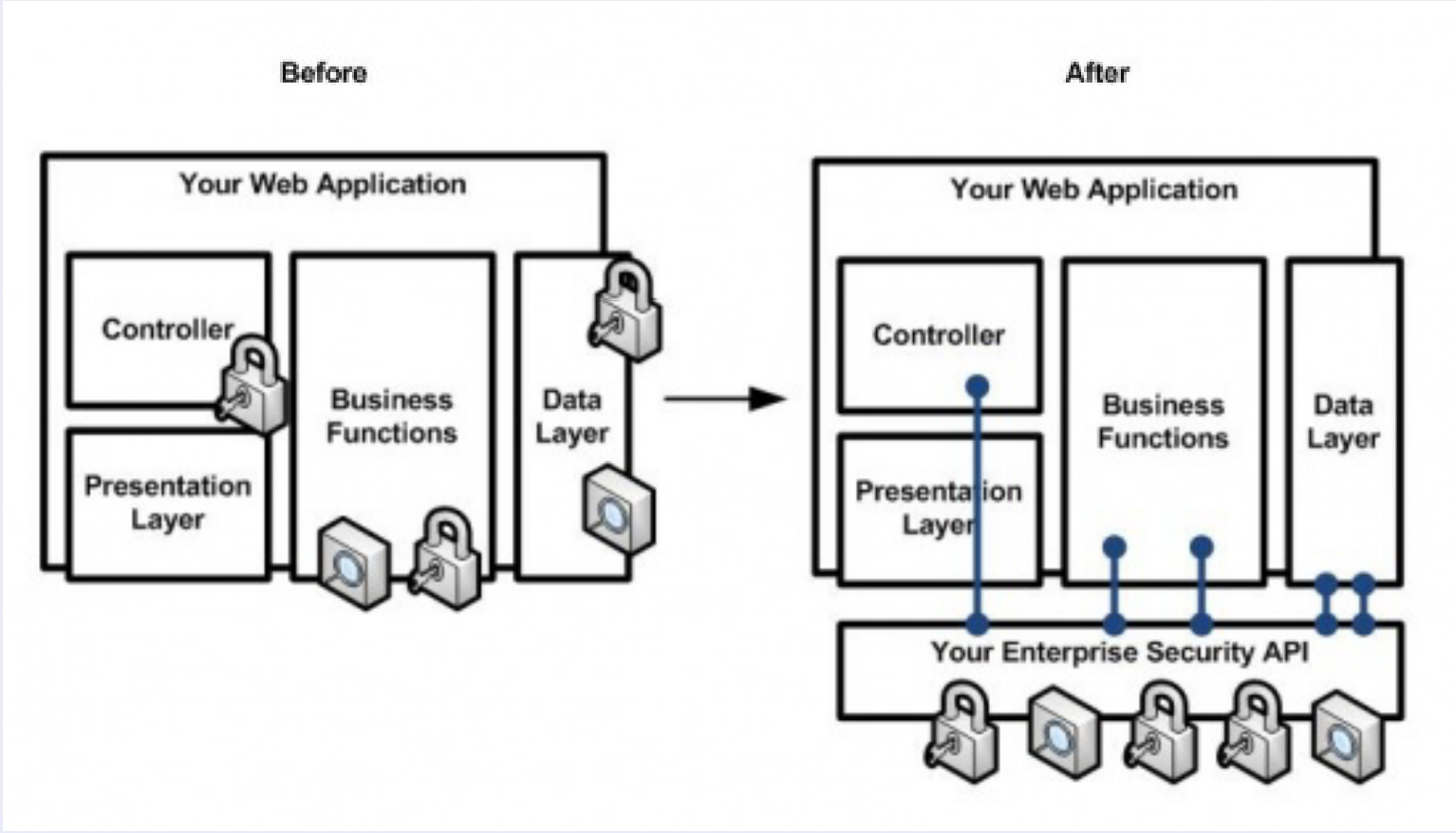
OWASP Top Ten	OWASP ESAPI
A1. Injection Flaws	Encoder
A2. Broken Authentication and Sessions	Authenticator, User, HTTPUtils
A3. Cross Site Scripting (XSS)	Validator, Encoder
A4. Insecure Direct Object Reference	AccessReferenceMap, AccessController
A5. Security Misconfiguration	SecurityConfiguration
A6. Sensitive Data Exposure	EnterpriseSecurityException, HTTPUtils
A7. Missing Function Level Access Control	Authenticator, User, HTTPUtils
A8. Cross Site Request Forgery (CSRF)	User (CSRF Token)
A9. Using Known Vulnerable Components	-
A10. Unvalidated Redirects and Forwards	AccessController

OWASP 中国
The Open Web Application Security Project

OWASP ESAPI for Java

- ESAPI.encoder()
- ESAPI.encryptor()
- ESAPI.validator()
- ESAPI.accessController()
- ESAPI.logger()
- ESAPI.authenticator()
- ESAPI.randomizer()
- ESAPI.httpUtilities()

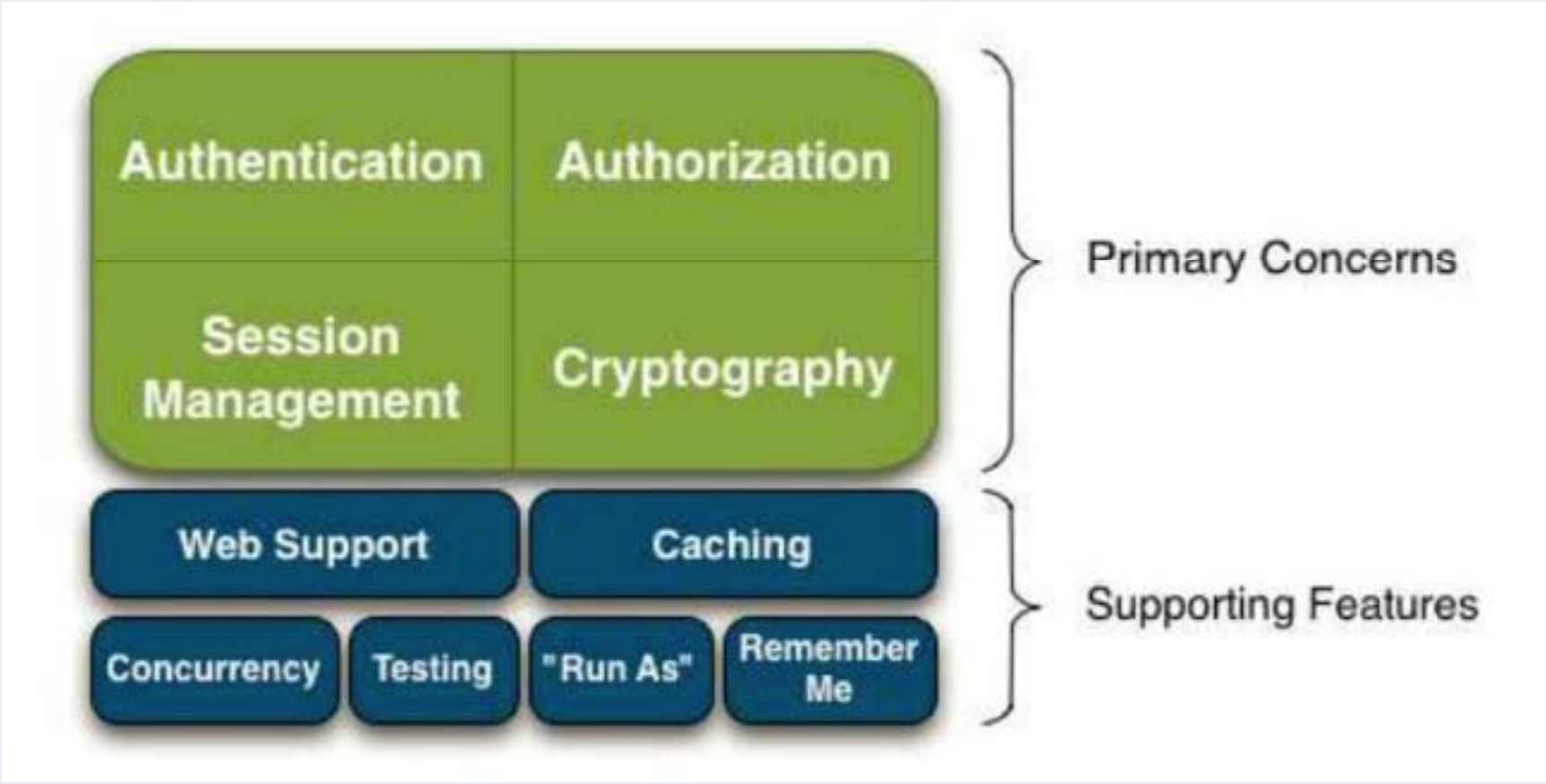
- 下图显示结合ESAPI设计你的程序



防护框架

OWASP 中国
The Open Web Application Security Project

- Shiro框架
  - Apache Shiro 是一个强大而灵活的开源安全框架
  - 主要能力包括了身份证人，授权，会话管理和加密。主要功能
    - 验证用户
    - 对用户的访问控制
    - 使用多个数据源


Authentication
Authorization
Session Management
Cryptography
Primary Concerns
Web Support
Caching
Concurrency
Testing
"Run As"
Remember Me
Supporting Features

OWASP 中国
The Open Web Application Security Project

- Sanitizer
  - 通过 Sanitizer 可快速配置html标签的过滤测试略，对于存在大量标签的请求数据包可以有效防止跨站攻击，简单的列子：

```
PolicyFactory policy = new HtmlPolicyBuilder()
    .allowElements("p")
    .allowElements(
        new ElementPolicy() {
          public String apply(String elementName, List<String> attrs) {
            attrs.add("class");
            attrs.add("header-" + elementName);
            return "div";
          }
        }, "h1", "h2", "h3", "h4", "h5", "h6"))
    .build();
String safeHTML = policy.sanitize(untrustedHTML);
```

目录

OWASP 中国
The Open Web Application Security Project

- OWASP TOP 10（2013）
- 安全开发编程概述
- 利用开源框架提升安全
- OWASP TOP 10 安全实践

OWASP TOP 10安全实践

OWASP 中国
The Open Web Application Security Project

OWASP Top Ten	解决方法
A1. Injection Flaws	Query Parameterization Cheatsheet，Encoder
A2. Broken Authentication and Sessions	Apache SHIRO
A3. Cross Site Scripting (XSS)	OWASP Java Encoder, HTML and JSON Sanitizer
A4. Insecure Direct Object Reference	Apache SHIRO
A5. Security Misconfiguration	SecurityConfiguration
A6. Sensitive Data Exposure	EnterpriseSecurityException, HTTPUtils
A7. Missing Function Level Access Control	Apache SHIRO
A8. Cross Site Request Forgery (CSRF)	User (CSRF Token)
A9. Using Known Vulnerable Components	Patch & Update
A10. Unvalidated Redirects and Forwards	AccessController

注入漏洞（Injection Flaws）

OWASP 中国
The Open Web Application Security Project

- 定义

  简单来说，注入往往是应用程序缺少对输入进行安全性检查所引起的，攻击者把一些包含指令的数据发送给解释器，解释器会把收到的数据转换成指令执行，注入漏洞十分普遍，通常能在SQL查询、LDAP查询、Xpath查询、OS命令、程序参数等中出现。

- 危害

  注入能导致数据丢失或数据破坏、缺乏可审计性或是拒绝服务。注入漏洞有时甚至能导致完全接管主机。

- 种类

  SQL注入、XPATH注入、LDAP注入、OS命令注入等。

注入漏洞（Injection Flaws）

OWASP 中国
The Open Web Application Security Project

- 解决之道

➢ SQL注入实例

```
String sqlString = "SELECT * FROM users WHERE fullname = '" +
    form.getFullName() + "' AND password = '" +
    form.getPassword() + "'";
```

正常：username=tony，password=123456

```
SELECT * FROM users WHERE username = tony' AND password =
    '123456'
```

攻击： username=tony，password= **' OR '1' = '1**

```
SELECT * FROM users WHERE username = tony' AND password =
    ' ' OR '1' = '1'
```

➢ 参数化查询预处理

注入漏洞（Injection Flaws）

OWASP 中国
The Open Web Application Security Project

- 使用PreparedStatement()绑定变量

  下面的代码示例使用一个PreparedStatement，Java的一个参数化查询的执行情况，执行相同的数据 库查询。

  String custname = request.getParameter("customerName"); // This should REALLY be validated too // perform input validation to detect attacks

  String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

  PreparedStatement pstmt = connection.prepareStatement(query); pstmt.setString(1, custname);

  ResultSet results = pstmt.executeQuery( );

注入漏洞（Injection Flaws）

OWASP 中国
The Open Web Application Security Project

- 使用ESAPI

```
 //ESAPI version of query
Codec ORACLE_CODEC = new OracleCodec();
//we're using oracle
String query = "SELECT name FROM users WHERE id = " +
 ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
 validatedUserId) + " AND date_created >= '" +
 ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
 validatedStartDate) +"'";
myStmt = conn.createStatement(query);
...
//execute statement and get results
```

失效的认证和会话管理&不安全的直接对象引用

OWASP 中国
The Open Web Application Security Project

- 失效的认证和会话管理
  - 定义

    与认证和会话管理相关的应用程序功能往往得不到正确实施，这就导致攻击者破坏密码、密匙、会话令牌或利用实施漏洞冒充其他用户身份。
  - 危害

    这些漏洞可能导致部分甚至全部帐户遭受攻击。一旦攻击成功，攻击者能执行合法用户的任何操作。因此特权帐户会造成更大的破坏。
  - 解决之道
  - 使用内置的会话管理功能。
  - 通过认证的问候：
  - 使用单一的入口点。
  - 确保在一开始登录SSL保护的网页。

失效的认证和会话管理&不安全的直接对象引用

OWASP 中国
The Open Web Application Security Project

- 不安全的直接对象引用
  - 定义

    所谓“不安全的对象直接引用”，即Insecure direct object references，意指一个已经授权的用户，通过更改 访问时的一个参数，从而访问到了原本其并没有得到授权的对象。Web应 用往往在生成Web页面时会用它的真实名字，且并不会对所有的目标对象访问时来检查用户权限，所以这就造成了不安全的 对象直接引用的漏洞。

    我们看如下的一个示例，也许这样就更容易理解什么是不安 全的对象直接引用。

    - 攻击者发现他自己的参数是6065， 即?acct=6065；
    - 他可以直接更改参数为6066， 即?acct=6066；
    - 这样他就可以直接看到6066用 户的账户信息了。

失效的认证和会话管理&不安全的直接对象引用

OWASP 中国
The Open Web Application Security Project

- 获取注销的权利；
- 添加超时；
- 确保你使用的是安全相关的功能；
- 使用强大的认证；
- 不进行默认身份验证

```
//BAD - DON'T USE
public boolean login(String username, String password) {
boolean isAuthenticated = true;
    try {
  //make calls to backend to actually perform login against datastore
      if (! authenticationSuccess) {
           isAuthenticated = false;
           }
       } catch (Exception e) {
             //handle exc }
    return isAuthenticated;
}
```

失效的认证和会话管理&不安全的直接对象引用

OWASP 中国
The Open Web Application Security Project

- 通过 shiro 进行防护
  - 通过 shiro 框架可以快速的对会话进行管理，默认的 filter 已经有相当强的功能

```
[main]
...
sha256Matcher = org.apache.shiro.authc.credential.Sha256CredentialsMatcher
...
iniRealm.credentialsMatcher = $sha256Matcher
...

[users]
# user1 = sha256-hashed-hex-encoded password, role1, role2, ...
user1 = 2bb80d537b1da3e38bd30361aa855686bde0eacd7162fef6a25fe97bf527a25b, role1, role2,
```

失效的认证和会话管理&
不安全的直接对象引用


```
//let's login the current user so we can check against roles and permissions:
if (!currentUser.isAuthenticated()) {
        UsernamePasswordToken token = new UsernamePasswordToken("lonestarr", "vespa");
        token.setRememberMe(true);
        try {
                currentUser.login(token);
        } catch (UnknownAccountException uae) {
                log.info("There is no user with username of " + token.getPrincipal());
        } catch(IncorrectCredentialsException ice) {
                log.info("Password for account " + token.getPrincipal() + " was incorrect!");
        } catch(LockedAccountException lae) {
                log.info("The account for username " + token.getPrincipal() + " is locked. " +
                        "Please contact your administrator to unlock it.");
        }
        // ... catch more exceptions here (maybe custom ones specific to your application?
        catch (AuthenticationException ae) {
                //unexpected condition? error?
        }
}
```

跨站脚本（XSS）

OWASP 中国
The Open Web Application Security Project

•定义

跨站脚本是最普遍的web应用安全漏洞。当应用程序在发送给浏览器的页面中包含用户提供的数据，但没有经过适当验证或转译那些内容，这就导致跨站脚本漏洞。

•危害

攻击者能在受害者浏览器中执行脚本以劫持用户会话、迫害网站、插入恶意内容、重定向用户、使用恶意软件劫持用户浏览器等等。

•种类

已知有三种著名跨站漏洞：1）存储式；2）反射式；3）基于DOM。

反射式跨站脚本通过测试或代码分析很容易找到。

跨站脚本（XSS）

OWASP 中国
The Open Web Application Security Project

- 解决之道

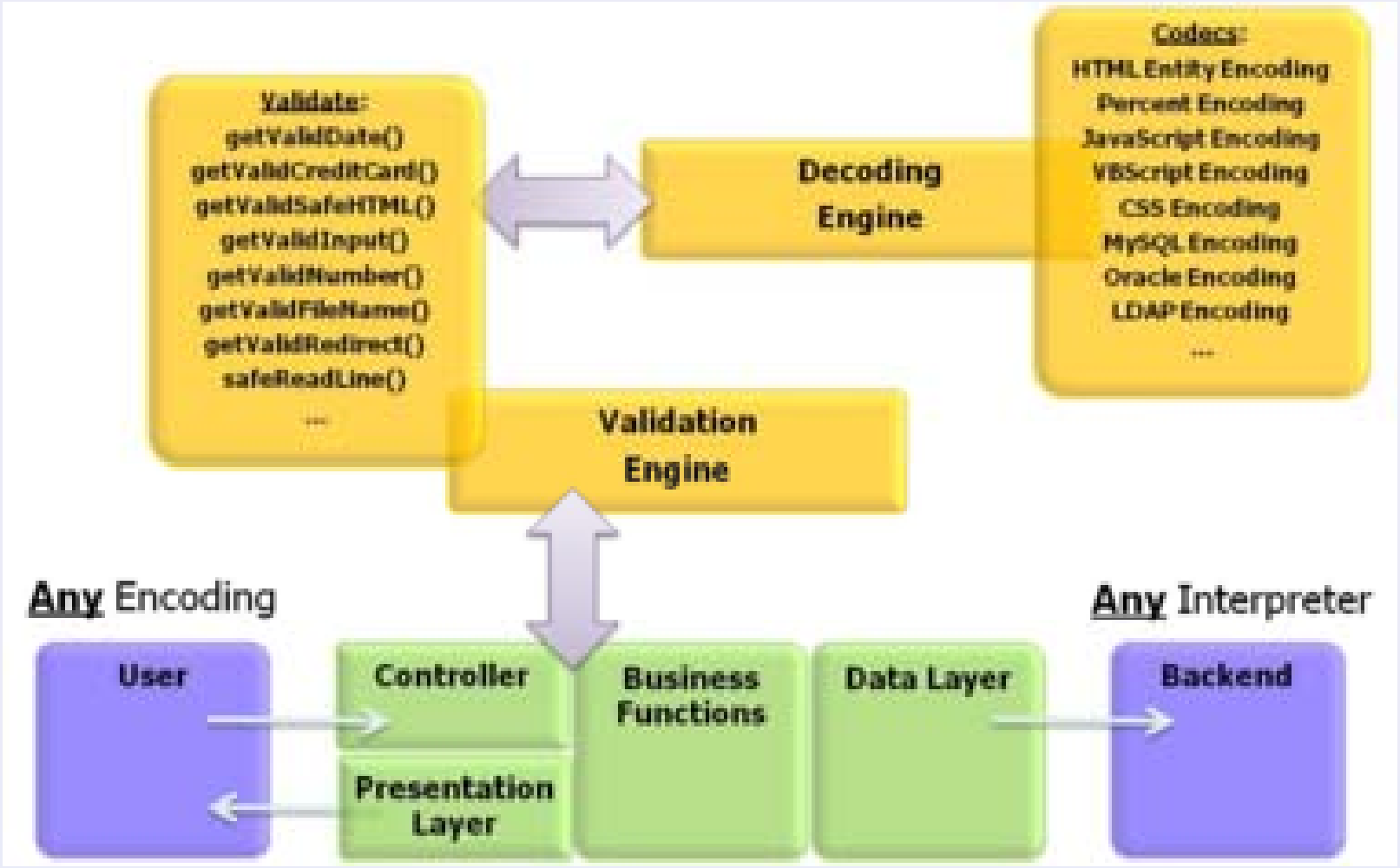
➢ 验证输入

验证输入很简单 - 检查每个输入的有效性。 这可能意味着很多东西，但在典型的和简单的情况下，这意味着检查输入类型和数据的长度。 例如，如果你是从一个文本框接受一个标准的邮政编码，你会知道，唯一有效的类型是一个数字（0-9），而长度应该是6，不能多也不能少。 并非所有 的案件都如此简单，但很多是相似的。

下图显示验证输入的架构。 这里的关键是，一切都进行验证，所有的输入，这并不来自于应用程序（包括用户输入，请求头，Cookie，数据库数据...）。

跨站脚本（XSS）

OWASP 中国
The Open Web Application Security Project



跨站脚本（XSS）

OWASP 中国
The Open Web Application Security Project

◆ 实例

- getValidInput(java.lang.String context, java.lang.String input, java.lang.String type, int maxLength, boolean allowNull, ValidationErrorList errors)

- isValidInput(java.lang.String context, java.lang.String input, java.lang.String type, int maxLength, boolean allowNull)

- String validatedFirstName = ESAPI.validator().getValidInput("FirstName", myForm.getFirstName(), "FirstNameRegex", 255, false, errorList);

- boolean isValidFirstName = ESAPI.validator().isValidInput("FirstName", myForm.getFirstName(), "FirstNameRegex", 255, false);

跨站脚本（XSS）

OWASP 中国
The Open Web Application Security Project

- 编码输出

对验证输入的另一面就是编码输出。 编码输出，是用来确保字符 被视为数据，而不是作为HTML元字符被浏览器解析。 这些技术定义一些特殊的“转义”字符。 没有正确转义的数据它仍然会在浏览器中正确解析。 编码输出只是让浏览器知道数据是不是要被解析，达到攻击无法实现的目的。

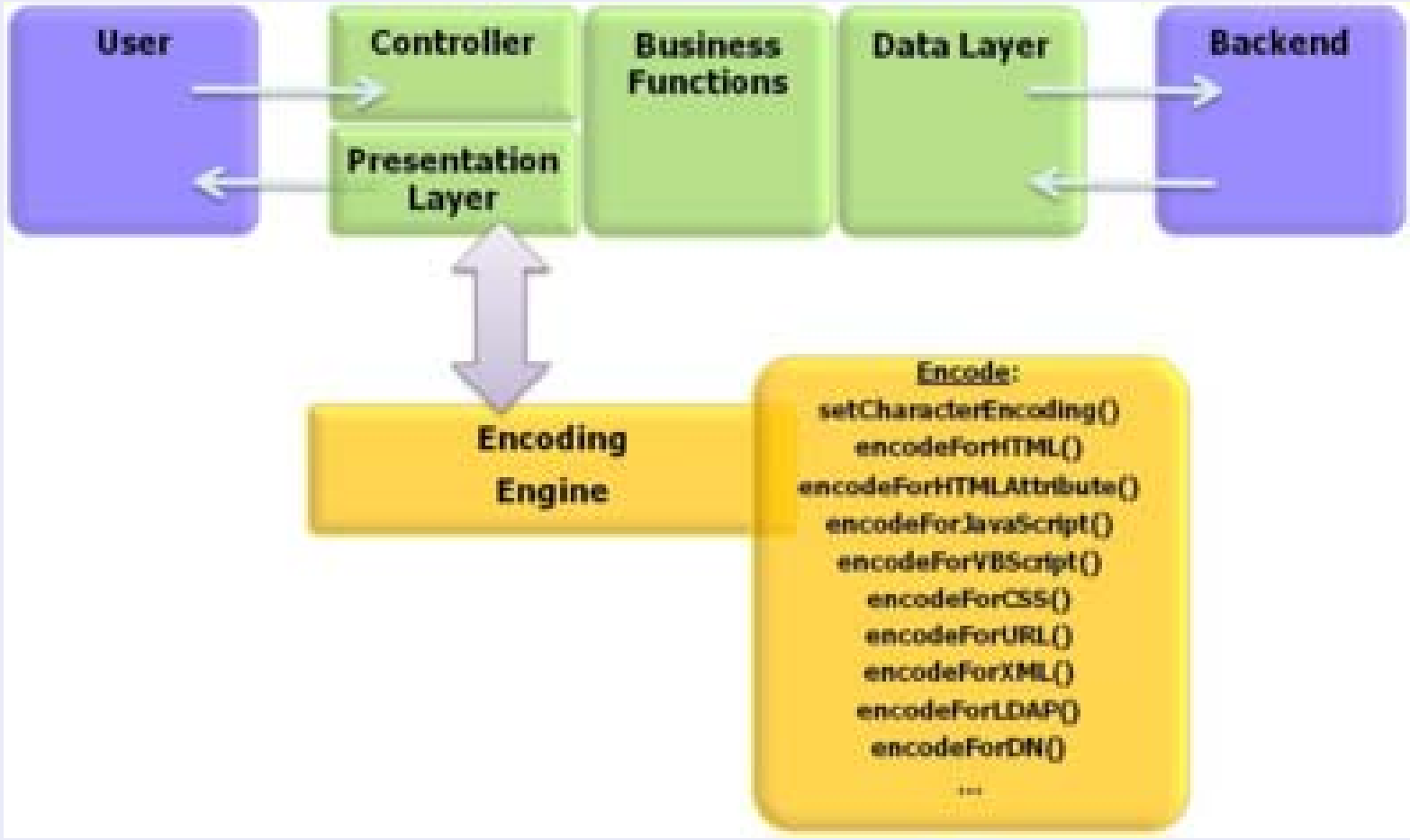
需要编码的部分：

1、HTML实体
2、HTML属性
3、Javascript
4、CSS
5、URL

下图像显示编码输出的架构。

跨站脚本（XSS）

OWASP 中国
The Open Web Application Security Project



跨站脚本（XSS）

OWASP 中国
The Open Web Application Security Project

◆ 实例1——HTML实体编码

```
//performing input validation String cleanComment =
    ESAPI.validator().getValidInput("comment",
    request.getParameter("comment"), "CommentRegex", 300,
    false, errorList); //check the errorList here ... ... //performing
    output encoding for the HTML context String safeOutput =
    ESAPI.encoder().encodeForHTML( cleanComment );
```

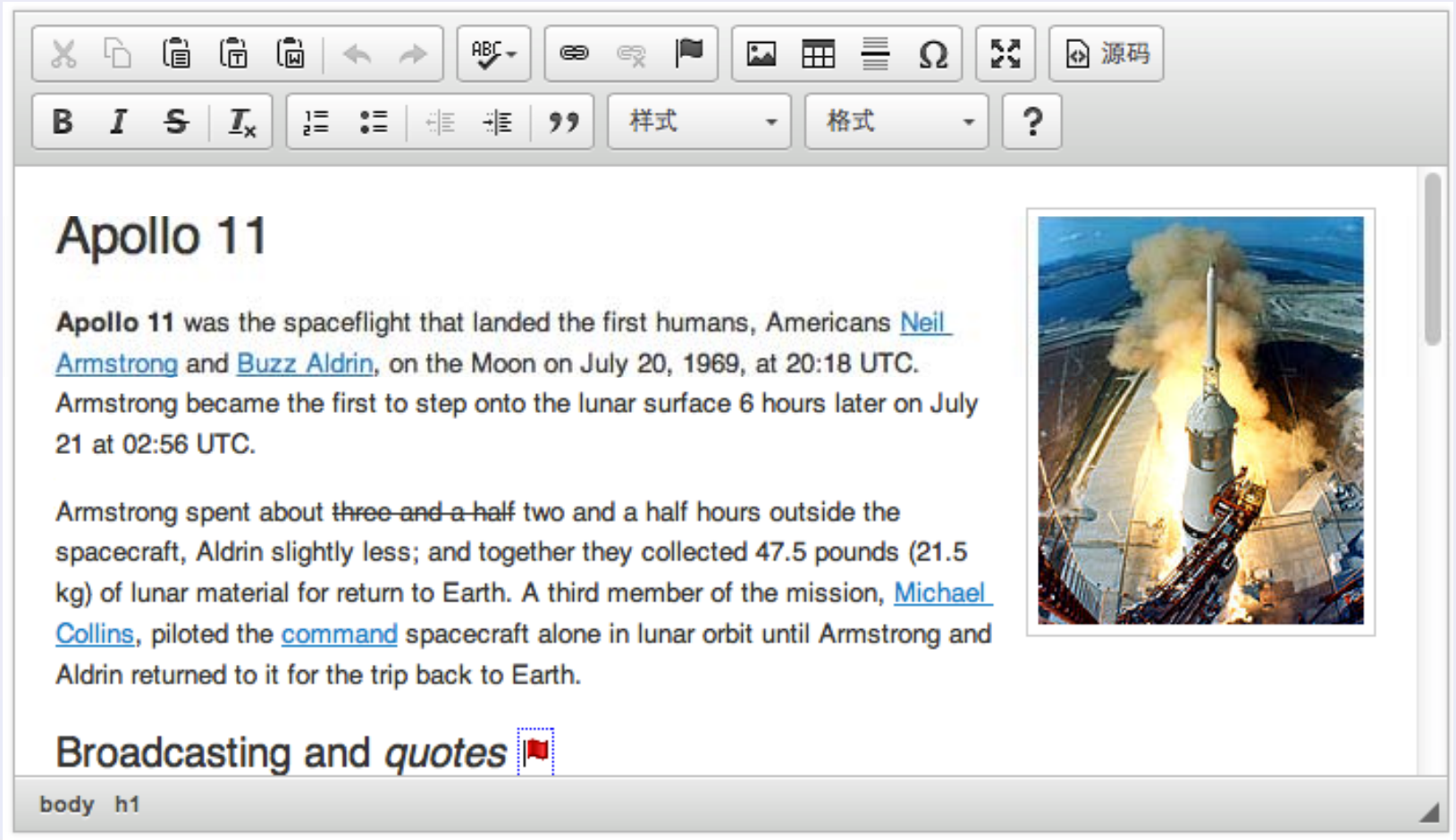
◆ 实例2——URL编码

```
//performing input validation String cleanUserName =
    ESAPI.validator().getValidInput("userName",
    request.getParameter("userName"), "userNameRegex", 50,
    false, errorList); //check the errorList here ... ... //performing
    output encoding for the url context String safeOutput =
    "/admin/findUser.do?name=" +
    ESAPI.encoder().encodeForURL(cleanUserName);
```

跨站脚本（XSS）

OWASP 中国
The Open Web Application Security Project

复杂的 HTML 代码提交，如何处理？



OWASP 中国
The Open Web Application Security Project

- 使用 Sanitizer 防护
  - 允许title全部可用
  - 允许href只能在<a>标签中使用
  - 指定lang、align属性的格式
  - 确定允许使用的标签列表

```
public static final Function<HtmlStreamEventReceiver, HtmlSanitizer.Policy>
     POLICY_DEFINITION = new HtmlPolicyBuilder()
         .allowStandardUrlProtocols()
         .allowAttributes("title").globally()
         .allowAttributes("href").onElements("a")
         .requireRelNofollowOnLinks()
         .allowAttributes("lang").matching(Pattern.compile("[a-zA-Z]{2,20}"))
             .globally()
         .allowAttributes("align")
             .matching(true, "center", "left", "right", "justify", "char")
             .onElements("p")
         .allowElements(
             "a", "p", "div", "i", "b", "em", "blockquote", "tt", "strong",
             "br", "ul", "ol", "li")
         .allowElements("quote", "ecode")
         .toFactory();
```

敏感信息泄露

OWASP 中国
The Open Web Application Security Project

- 定义

  应用程序常常产生错误信息并显示给使用者。 很多时候，这些错误信息是非常有用的攻击，因为它们揭示实施细则或有用的开发信息利用的漏洞。

- 危害

➢ 泄露太多的细节（如错误堆栈跟踪信息、SQL语句等等）；

➢ 登录失败后，通知用户是否用户ID或密码出错——登录失败可能是由于ID或密码错 误造成的。这为一个对关键资产发动蛮力攻击的攻击者提供重要信息。

- 解决之道

➢ 案例1

通过web.xml配置文件实现

```
<error-page>
    <exception-type>java.lang.Throwable</exception-type>
     <location>/error.jsp</location>
</error-page>
```

跨站点请求伪造（CSRF）

OWASP 中国
The Open Web Application Security Project

- 定义

  跨站请求伪造，也被称成为“one click attack” 或者session riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。尽管听起来像跨站脚本（XSS），但它与XSS非常不同，并且攻击方式几乎相左。XSS利用站点内的信任用户，而CSRF则通过伪装来自受信任用户的请求来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不大流行（因此对其进行防范的资源也相当稀少）和难以防 范，所以被认为比XSS更具危险性。

- 危害

  攻击者能让受害用户修改可以修改的任何数据，或者是执行允许使用的任何功能 。

跨站点请求伪造（CSRF）

OWASP 中国
The Open Web Application Security Project

- 解决之道

第一步，新建CSRF令牌添加进用户每次登陆以及存储在http session里，这种令牌至少对每个用户会话来说应该是唯一的，或者是对每个请求是唯一的。

```
//this code is in the DefaultUser implementation of ESAPI
/** This user's CSRF token. */
 private String csrfToken = resetCSRFToken();
 ...
 public String resetCSRFToken() {
        csrfToken = ESAPI.randomizer().getRandomString(8,
DefaultEncoder.CHAR_ALPHANUMERICS);
         return csrfToken;
  }
```

跨站点请求伪造（CSRF）

OWASP 中国
The Open Web Application Security Project

第二步，令牌同样可以包含在URL中或作为一个URL参数标记/隐藏字段。

```
//from HTTPUtilitiles interface
 final static String CSRF_TOKEN_NAME = "ctoken";
//this code is from the DefaultHTTPUtilities implementation in ESAPI public
  String addCSRFToken(String href) {
          User user = ESAPI.authenticator().getCurrentUser();
          if (user.isAnonymous()) { return href; }
// if there are already parameters append with &, otherwise append with ?
   String token = CSRF_TOKEN_NAME + "=" + user.getCSRFToken();
   return href.indexOf( '?') != -1 ? href + "&" + token : href + "?" + token;
   }
   ...
  public String getCSRFToken() {
        User user = ESAPI.authenticator().getCurrentUser();
        if (user == null) return null; return user.getCSRFToken(); }
```

跨站点请求伪造（CSRF）

OWASP 中国
The Open Web Application Security Project

第三步，在服务器端检查提交令牌与用户会话对象令牌是否匹配。

```
//this code is from the DefaultHTTPUtilities implementation in ESAPI
public void verifyCSRFToken(HttpServletRequest request) throws
   IntrusionException { User user = ESAPI.authenticator().getCurrentUser();
  // check if user authenticated with this request - no CSRF protection required
   if( request.getAttribute(user.getCSRFToken()) != null ) {
            return;
       }
   String token = request.getParameter(CSRF_TOKEN_NAME);
   if ( !user.getCSRFToken().equals( token ) ) {
        throw new IntrusionException("Authentication failed", "Possibly forged
  HTTP request without proper CSRF token detected");
        }
 }
```

跨站点请求伪造（CSRF）

OWASP 中国
The Open Web Application Security Project

第四步，在注销和会话超时，删除用户对象会话和会话销毁。

```
    //this code is in the DefaultUser implementation of ESAPI
 public void logout() {
    ESAPI.httpUtilities().killCookie( ESAPI.currentRequest(), ESAPI.currentResponse(),
     HTTPUtilities.REMEMBER_TOKEN_COOKIE_NAME );
   HttpSession session = ESAPI.currentRequest().getSession(false);
    if (session != null) {
            removeSession(session);
            session.invalidate();
            }
    ESAPI.httpUtilities().killCookie(ESAPI.currentRequest(), ESAPI.currentResponse(),
      “JSESSIONID”  );
    loggedIn = false;
    logger.info(Logger.SECURITY_SUCCESS,  “Logout successful”  );
     ESAPI.authenticator().setCurrentUser(User.ANONYMOUS);
}
```

OWASP 中国
The Open Web Application Security Project

Think You