鉴释

为新手准备的OWASP十大WEB应用程序安全风险及其示例

鉴释

目录

引言

了解最常见的已知Web应用程序攻击的含义，对于保护软件安全至关重要。在这里，我们将用示例阐释漏洞和一些方法，因此可以避免这些风险。

1. 注入

注入攻击是黑客在试图访问数据时最常用的攻击方法。你首先需要做的是假设所有将要输入的数据在本质上都是恶意的。从这里开始，你可以构建所有必要的检查和过滤器来防止此类攻击。例如，对于SQL注入，你可以使用输入验证函数来防止传递不正确的字符，并只允许与输入类型相关的字符通过，例如电子邮件。其它可以做的包括在将控制权移交给输入变量时不允许使用动态SQL。许多数据库和操作系统不断需要安全补丁，因此具有一个补丁管理过程通常很关键。最后别忘了始终把你的数据库登录凭据分开并加密。

攻击示例

以下第一个示例使用SQL查询准备步骤来展示一个十分常见的漏洞，即SQL注入。该示例表明，该短程序在将用户输入发送至数据库前没有采取任何防御措施，因此用户可能插入某些恶意的SQL字符串，当输入提交到数据库时，这些恶意SQL字符串可能导致远程代码执行。

```
Java-SQL:

username = request.getArguments().getField("userName");

sql = "SELECT * FROM user WHERE username = '" + username + "'";


Java:

String userInput = request.getArguments().getField("userValue");

String basicResponse = "<div data-value='" + userInput +"'> Click to upload </div>";
```

第二个Java示例只是将用户输入结合到响应的HTML模板中。在这种情况下，危险的操作在第二行，用户输入没有经过任何形式的净化就被直接放置进结果中。为了防止这种情况的发生，可以在将输入与模板组合之前使用输入验证库执行检查。

2. 失效的身份认证

你可以通过仔细确定默认的登录凭据模式，并确保在初始设置后未将默认密码用于进一步的身份验证，来防止身份验证遭到破坏。

攻击示例

以下示例直接用MD5来执行密码检查，而不是正确地对其加盐。这意味着如果系统的任一用户碰巧使用了别人使用的密码，那么拥有系统管理控制权的人能检查该结果，然后提取用户密码的可能性就很大。

```
Java:

SysUser user = system.getUserByName(name);

if (MD5hash(user.getPassword()).equals(userInputPasswordHash)) {

  return true;

} else {

  return false;

}
```

3. 敏感信息泄露

当客户的私人数据或公司保密信息由于没有受到足够的保护而无意中遭到泄露时，就会发生敏感数据泄露。要防止这一点，作为开发人员，必须确保安全加密算法的正确使用、密钥的安全存储和传输安全。至关重要的是要识别任何缺少加密保护的发生。

攻击示例

以下示例显示开发人员不安全地把任何登录进系统的用户的登录凭据记录进日志。这意味着有权访问日志文件的人，特别是在日志文件对公众开放时，能得到包括用户名和密码在内的所有用户的凭据。

```
Java:

System.out.println("User password : " + user.getPassword());

Logger.info("User ID { } Name = {}, Password = {} ", user.getId(),

user.getName(), user.getPass());
```

当开发人员试图将服务器的私钥或访问密钥保存进日志文件或暗地里这么做时，会产生其他危险。例如，在记录环境变量时，这可能导致服务器的身份被伪造，这会完全破坏用户和服务器之间通讯的机密性。

4. XML 外部实体 (XXE)

防止XXE最安全的方式是始终完全禁用外部实体。禁用这些实体还让解析器变得安全而免受拒绝服务（DOS）攻击，例如专门针对XML文档解析器的Billion Laughs CVE-2019-5442 (https://nvd.nist.gov/vuln/detail/CVE-2019-5442)。在2019年10月，StackRox的Kubernetes API服务器GitHub库被发现在其YAML的部署上有安全问题，这使得它易受拒绝服务的billion laughs攻击。StackRox自己声称：“该问题再次提醒我们，与所有软件一样，Kubernetes易受零DAY漏洞的攻击”。

攻击示例

本Java示例用了XMLReader库，直接将这个XML文件与底层XML库一起解析。在此操作期间，将解析并获得外部实体，即common.xml，这可能是由系统多个用户共享的xml文件并可能被系统的某些外部用户所控制。如果有恶意的用户在此文件里精心地放入一些内容，这会危及到下游系统的配置。

```
XML:

Base.xml:

    <DOCTYPE ...>

    <ENTITY1>

    <...> <EXTERNAL foo SYSTEM “file:///usr/local/shared/common.xml”></EXTERNAL> </...>

    </ENTIY1>


Java:

xmlObject = XMLReader.parseEntityFromFile(“base.xml”);

xmlObject.getChild(“ENTITY1”);
```

5. 失效的访问控制

要避免中断访问控制的发生，至关重要的是在整个开发过程中为你的应用程序选择并坚持使用一种访问控制模式，并持续不断地测试它以确保少有故障点。四种标准的访问控制模式包括强制访问控制（MAC）、基于角色的访问控制（RBAC）、自主访问控制（DAC）以及基于规则的访问控制（RBAC或RB-RBAC）。每种模式都有优点和弱点，必须根据你的系统设计和用途仔细地进行选择。

攻击示例

部分检查访问控制（黑名单模式）是有风险的。如果开发人员添加了其它一些页面但忘记更新访问控制逻辑，则可能会有可行的漏洞，即没有适当访问权限的用户仍然可以访问这些页面并执行恶意操作。

Java:

```
if (pageName.equals("/home/admin/landing")) {
 checkUserAccessControls ();
}

// The access control are left unchecked from some other admin pages.
```

6. 安全配置错误

你应该意识到的一点是配置错误通常和人为错误有关，而不是弱点或攻击。要避免这一点，你可以设计和使用微分段策略，以确保在破坏发生时受到保护，如果配置错误没有得到解决或者补丁管理有延迟时，则可以限制攻击面。

攻击示例

该示例显示了每次在登录请求里给出某些特定参数时，如何打开供内部使用的后门。这是十分危险的，因为用户可能在程序逻辑上发起服务路径遍历攻击。

Java:

```
userInput = request.getArguments().getField("extraParams");
if (userInput.equals("debug=on")) {
  return debugInfo;
}
```

7. 跨站脚本（XSS）

谨慎地显示和存储可能向其他用户显示的用户输入，可以预防跨站脚本攻击。用户输入还可以通过HTML消毒引擎运行，该引擎可以检查XSS代码。跨站脚本攻击是代码注入的一种形式。

攻击示例

该示例展示了跨站脚本漏洞如何处理用户输入，尤其是带有一小段准备显示给其他人的文本的输入。因为文本（在Java代码里）将直接添加到HTML，这让攻击者有机可乘，滥用这一点并注入会在其他用户的浏览器上运行的任意代码。此处的<script>是简化后的示范。

```
Java:

...

return "<html> ... " + userComment.getBody() + "</html>"

...


Where user input included:


<script> ... ... </script>
```

8. 不安全的反序列化

对不安全的反序列化的预防，能通过使用加密、监控反序列化过程和不接受来自未知来源的序列化对象来达到。RASP技术也常常用来处理Web应用程序不安全的反序列化问题。

攻击示例

该示例显示了在安卓中发送IPC消息的一种流行方式，这可能会反序列从用户环境中的恶意App发送来的对象（Intent），因此是有危险的。如果该Intent在正常函数（例如以下的Intent.getStringExtra）背后有某些隐藏代码，这会导致远程代码执行。

```
Java:

response.write(user.serialize()); // Sending the serialized object to the user
...
user = MyUserImplementation.deserialize(request.getArguments().getField("userInfo"));
  // Retrieve the serialized object from the user.
...

Android Java:

Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra("EXPECTED", "someString"); //An expected parameter
intent.putExtra("UNEXPECTED", someObject); //Unexpected object

....

Intent intent = getIntent();
String expected = intent.getStringExtra("EXPECTED");
```

9. 使用含有已知漏洞的组件

防止由此漏洞引起的攻击的最佳方法是确保应用严格的过程和策略，来不断地监控库和组件，以确保使用最新版本并定期更新补丁。

攻击示例

在物联网（IoT）领域里，由于产品开发的速度和多个设备相互交互的不安全策略，难以维持当前库的使用和补丁程序的应用。攻击者可以创建程序来检测未打补丁或配置错误的应用程序。即使在今天，于2014年发现的Heartbleed漏洞（https://nvd.nist.gov/vuln/detail/CVE-2014-0160）仍然活跃在不受保护的设备里。简而言之，作为OpenSSL里原始风险的Heartbleed漏洞都可以归结为这行代码：

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
```

在这个实例里，根据用户输入的值从有效负载+填充中分配内存。因为没有长度检查，攻击者可以强迫OpenSSL服务器读取任意内存位置。

10. 不足的日志记录和监控

正确的日志记录和错误条件检查，对防御诸如猜测密码或检测非法网络流量之类的暴力破解攻击至关重要。日志和监控非常重要，因为它们为安全团队提供了宝贵时间来进行取证分析和发展其防御机制。不足的日志记录和监控机制常常会导致严重后果。

没有适当的反暴力破解检测的登录失败已经导致了数亿个用户账户被黑客入侵。在网络渗透测试期间，攻击团队的一个重要因素是寻找进入目标环境而不被监视或日志记录的方法。如今，像攻击者一样思考的开发人员在软件开发里是至关重要的，他们在编写代码的同时会试图找出缺陷和漏洞。

攻击示例

该示例显示了如果在用户登录失败后未采取任何措施，其他人很有可能在系统上施加暴力破解攻击来猜测该用户的密码。正确的做法是正确记录失败的登录尝试，一旦登录尝试失败的次数过多，系统应该在“冻结”期间阻止该用户登录。

```
...
If (isPasswordMatch(user.getPassword(), recordPassword)) {
  return “login failed”;
}
...
```

结论

重要的是要注意，必须把OWASP知识作为增强安全性的活动的一部分包括进去。企业要制定安全计划，领导层在所有团队中建立意识，确保持续审查和测试，在开发环境里使用具有适当安全性的安全编码惯例，以及适当的管理和监督等。

鉴释的理念之一是真正的安全性来自对人员进行风险培训，然后为他们提供确保安全的方法和工具。软件开发生命周期（SDLC）里的质量保证和测试是很关键的，但我们认为，安全性必须从一开始就考虑在内，开发时必须牢记安全性。建立意识并确保学习了解这份十大风险清单是一个好的开始！

关于鉴释

鉴释的使命是通过创建简单操作的工具来协助开发人员构建并调配安全可靠的代码。我们通过使用高级静态分析技术帮助客户降低成本，提高生产力，并确保其软件开发人员具备相应的能力以开发更好、更可靠的软件。

了解更多，请访问www.xcalibyte.com