

10 | 信息泄漏：为什么黑客会知道你的代码逻辑？

2019-12-30 何为舟

安全攻防技能30讲

[进入课程 >](#)



讲述：何为舟

时长 12:13 大小 9.79M




你好，我是何为舟。

你平时在 Debug 的时候，一定首先会去查看错误信息。根据错误信息，你能够了解究竟是什么情况引发了什么样的错误。同样地，黑客也能够通过错误信息，推断出你的后台代码逻辑。那么，黑客究竟是怎么做的呢？接下来，我们就一起看一下这个过程。

为什么错误信息会泄漏代码逻辑？

当黑客在登录某个页面时，在用户名位置输入一个单引号，在密码位置输入一个“g”之后，就会出现如下的错误信息。

```
1 An Error Has Occurred.
2     Error Message:
3     System.Data.OleDb.OleDbException: Syntax error (missing operator) in query
4     System.Data.OleDb.OleDbCommand.ExecuteNonQueryErrorHandling ( Int32 hr) a
5     System.Data.OleDb.OleDbCommand.ExecuteNonQueryForSingleResult ( tagDBPARAI
```

 复制代码

从这个错误信息中，我们可以看到，网页最终执行了一个 SQL 语句，这个 SQL 语句的部分内容为 `username = '' and password = 'g'`。因此，后台的大致逻辑应该是下面这样的。

第一，错误信息反馈的是 `Syntax error`，即语法错误。在密码位置输入单个字母“g”肯定不会引起错误，所以，这个 SQL 语句是因为多了一个单引号导致的报错。而如果使用了 `PreparedStatement` 等方法，是不会产生这个错误的。因此，后台的 SQL 查询应该是直接采用的字符串拼接，且没有过滤单引号。

第二，错误信息中显示了部分的 WHERE 条件是 `username = '' and password = ''`。这又是一个登录的逻辑，所以，只要用户名和密码正确，这个 SQL 语句会返回黑客需要的用户信息。因此，后台的 SQL 语句应该是形如 `select from where` 的格式。

根据这些信息，黑客很容易就可以发起 SQL 注入攻击了。

那错误信息中包含的敏感信息这么多，怎么避免被直接展示到前端呢？我们可以通过正确地配置文件，来进行合适的错误处理。比如，在 PHP 中，我们可以进行如下配置：

```
1 error_reporting = E_ALL ; 向 PHP 报告发生的每个错误
2 display_errors = Off ; 不显示满足上条指令所定义规则的所有错误
3 log_errors = On ; 决定日志语句记录的位置
4 log_errors_max_len = 1024 ; 设置每个日志项的最大长度
5 error_log = /var/log/php_error.log ; 指定产生的错误报告写入的日志文件位置
```

 复制代码

在 Java Spring 中，我们也可以通过配置 [ExceptionHandler](#) 等来进行处理。

避免错误信息泄漏代码逻辑，一方面是要通过正确的配置文件，避免错误信息被展示到前端；另一方面是要对错误信息进行检测，这里就需要用到“黑盒”检测了。

所谓“黑盒（Black Box Testing，功能测试）”，就是在不获取代码的情况下，直接运行应用，然后对应用的请求和响应进行扫描。比如，在错误信息泄漏的场景中，“黑盒”检测可以向应用发起一些必然会导致错误的请求（比如上述例子中的单引号），然后观察应用是返回完整的错误日志，还是返回某些经过处理的页面。

好了，现在你应该明白了，为啥错误信息会泄漏代码逻辑。实际上，错误信息泄漏属于一种间接的信息泄漏方式。间接的信息泄漏方式主要是通过拼凑各种零散信息，还原出代码整体的面貌，然后有针对性地发起攻击。所以我们常说，黑客的攻击本身就是一个“聚沙成塔”的过程。

除了错误信息，还有什么地方会泄漏代码逻辑？

除了错误信息之外，间接的信息泄漏方式还有两种：返回信息泄漏和注释信息泄漏。

注释信息你应该很熟悉。因为所有的前端代码基本都不需要编译就可以展示在浏览器中，所以黑客很容易就可以看到前端代码中的注释信息。但是，如果这些注释信息中出现服务器 IP、数据库地址和认证密码这样的关键信息。一旦这些关键信息被泄漏，将会造成十分严重的后果。

那该如何避免关键的注释信息出现在线上的代码中呢？我们经常会使用一种叫作“白盒”的代码检测方法。

所谓“白盒（White Box Testing，结构测试）”，即直接获取到线上的源代码，然后对它进行扫描。“白盒”扫描注释信息的原理比较简单，因为每一种语言的注释都会带有特殊的标记（比如 Java 和 PHP 中的 `/*` 等），可以比较准确地被识别出来。除此之外，“白盒”检测通常还会被用来做一些检测代码漏洞或者逻辑漏洞的工作，这一块比较复杂，现在你只需要有一个大概印象即可，我们会在后续的课程中专门来讲。

简单了解了注释信息泄漏，我们下面重点来看返回信息泄漏。

你可以回忆一下，在前面讲 [SSRF](#) 攻击的时候，我们模拟过这样一个场景：服务端在请求一个图片地址的时候，会根据地址的“存活”情况和返回数据的类型，分别返回三种结果：“图片不存在”“格式错误”以及图片正常显示。而黑客正是通过服务端返回信息的逻辑，利用一个请求图片的 SSRF，摸清整个后端的服务“存活情况”。

类似的多种返回状态的场景还有很多，你可以想想自己平时工作中有没有遇到过。这里我再再说一个常见的。当你在登录应用的时候，应用的返回逻辑可能是这样的：如果输入的用户名和密码正确，则登录成功；如果应用没有这个用户，则返回“用户名不存在”；如果输入的用户名和密码不匹配，则返回“密码错误”。

尽管这样清晰的登录提示对于用户体验来说，确实是一个较优解，但这个逻辑同样也暴露了过多的信息给黑客。黑客只需要不断地发起登录请求，就能够知道应用中存在的用户名，然后通过遍历常见的弱密码进行尝试，很容易就能够猜对密码。这样一来，猜对密码的成功率就比尝试同时猜测用户名和密码要高很多。


实际上，返回信息过于明确不算是代码层面的漏洞，更多的是产品层面的漏洞。因此，理论上没有任何技术手段能够对这种漏洞进行检测，只能依靠人为的分析审计来避免。解决方案也比较简单，直接将返回信息模糊化、统一化即可。比如，在上述登录的场景中，我们可以将两种登录失败的返回信息，统一修改为“用户名不存在或密码错误”。这样一来，既避免了用户体验受到太大影响，又消除了关键信息被黑客获取的隐患。

有哪些常见的直接泄漏方式？

在间接的泄漏方式中，黑客可以通过“蛛丝马迹”，推断出服务代码的逻辑。但是信息泄漏最普遍的方式还是直接泄漏。这里我会讲两种常见的直接泄漏方式。


第一种泄漏方式与版本管理工具中的隐藏文件有关。

在开发应用的过程中，你一定使用过版本管理工具（比如 SVN 和 Git），通过这些工具，你能够很方便地进行代码回滚、备份等操作。那你有没有想过，版本管理工具为什么这么方便呢？它的工作原理又是怎么样的呢？我们以 SVN 为例来说一说。

SVN 会在项目目录中创建一个.svn 文件夹，里面保存了应用每一个版本的源文件信息，这也是 SVN 实现代码回滚的数据基础。如果 SVN 可以通过.svn 中的数据提取应用任意版本的代码，那黑客也可以。只要你没有在上线代码的时候删除其中的.svn 目录，那就代表黑客可以通过.svn 中的 URL 访问里面的所有文件。接下来，只需要通过执行简单的  脚本，黑客就可以回溯出一个完整版本的代码了。

对于这种因为目录中额外内容（.svn/.git）导致的源码泄漏，我们一方面需要对线上代码进行人工的代码审查，确保无关的文件和文件夹被正确地清除；另一方面，我们也可以在

HTTP 服务中对部分敏感的路径进行限制。比如，在 Apache httpd 中配置下面的内容，来禁止黑客对.svn 和.git 目录的访问。


 复制代码

```
1 <DirectoryMatch \.(svn|git)>
2     Order allow,deny
3     Deny from all
4 </DirectoryMatch>
```

除此之外，还有一种最常见、也最不容易注意的泄漏方式，那就是上传代码到 GitHub 上。

我们知道，Git 除了是一个版本管理工具之外，还是一个很流行的代码管理工具。除了前面讲过的隐藏文件漏洞之外（Git 会生成.git，同样包含应用各种版本的文件信息），Git 还存在将代码上传到公开平台的问题。但是，使用 GitHub 上传代码通常属于个人行为，所以，我们很难从技术层面上进行预防。

那我们有没有一些有效的防护措施，可以尽可能地提高安全性呢？

我个人认为，公司应该从加强员工安全意识的培训、强化公司管理制度入手，避免员工私自上传代码。除此之外，公司还可以对 GitHub 发起巡检（比较知名的工具有  Hawkeye），通过定期检索公司代码的关键字（比如常用的包名、域名等）来进行检测。通过这些方式匹配到的结果，很可能就是员工私自公开的代码。确认之后，我们就可以联系上传的人员进行删除了。

总结

好了，今天的内容讲完了。我们来一起总结回顾一下，你需要掌握的重点内容。

信息泄漏这类漏洞很容易理解，但它能够造成的危害却不容小觑。**基本上，所有攻击的第一步都是从信息泄漏开始的。**而且黑客没有办法攻击一个未知的系统，所以黑客会通过这些泄漏的信息，去推断出应用的整体架构和逻辑。

信息泄漏的方式和原因有很多，这其中，除了黑客主动发起攻击导致的信息泄露之外，有很多非技术原因导致的信息泄漏。所以，相应的防护手段也比较零散。不过总体来说，我们可

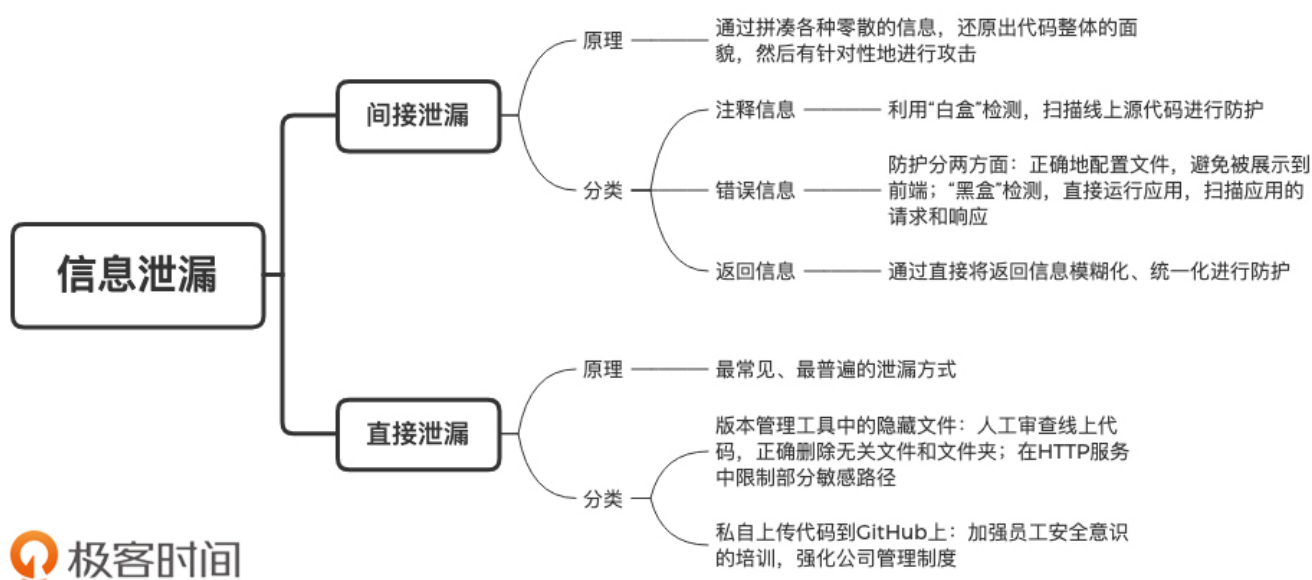
以从以下几个方面进行防护：

屏蔽信息：通过技术手段，将不该被访问的资源进行屏蔽，从而避免信息泄漏的产生；

代码检测：从“白盒”和“黑盒”两个方向，对代码、应用等进行检测，对可能的泄漏进行预警；

人工审计：对于非技术原因造成的泄漏，加强人工审计的工作。同时从公司制度上，去提高员工的安全意识。

今天的内容虽然比较简单，但是为了方便你记忆，我还是总结了一张知识脑图，你可以利用它来查缺补漏，加深记忆。



思考题

最后给你留一个思考题。

通过今天的讲解，你可以回忆一下，你的公司或者你负责的应用当中，是否发生过类似的信息泄漏事件呢？如果发生过，对你的公司或者应用都造成了什么影响呢？最后又是如何解决的呢？

欢迎留言和我分享你的思考和疑惑，也欢迎你把文章分享给你的朋友。我们下一讲再见！

点击查看 

来参加打卡，攻克 工作中 80% 的安全问题



PC端用户扫码参与



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | 反序列化漏洞：使用了编译型语言，为什么还是会被注入？

精选留言 (4)

 写留言



qinsi

2019-12-30

还有旁路信息泄漏。以上面的用户登录场景为例，对于无效用户和有效用户的登录请求，如果服务端处理耗时不一样也会泄漏信息。又比如padding oracle攻击，只要服务端返回的信息可以区分解密成功与否，就可以在没有密钥的情况下经过有限次尝试枚举出解密后的信息

展开 

作者回复：你好，感谢你的留言。旁路泄露确实也比较重要。其实间接的泄露方式还有很多，比如robots.txt里面的url路径等。这些在平常都很难意识到问题的存在，更多的是吃一堑长一智～



1



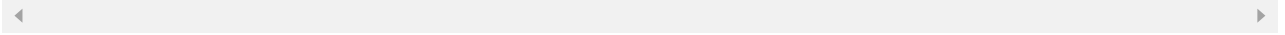
leslie

2019-12-30

直接泄漏没有，不过间接泄漏估计不好说，注释和抛错做过了居然会引发间接泄漏的风险。看来后面要好好检查调整相关的策略了。谢谢老师的分享。

展开 ▾

作者回复: 错误信息一般还是会暴露比较多的代码信息的，最好能做统一处理，返回一个静态的错误页面，或者错误码，都行。



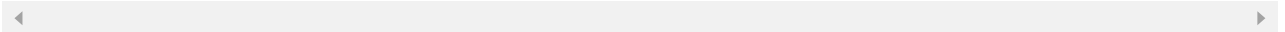
小晏子

2019-12-30

想到了著名的B站事件，源代码上传到github，感觉这个很大一部分原因是工程师对github理解的不到位，以为是个人仓库，别人看不到，这块应该加强培训，在入职的时候就应该培训，尤其对于年轻的工程师，安全意识太薄弱。

展开 ▾

作者回复: 其实类似的事件很多，大小公司都有。毕竟这么多工程师，很难保证哪个人不注意就传上去了。因此，除了加强培训管理，扫描也是很必要的。



Cy23

2019-12-30

每个目录没有默认页面的都创建空index.php，禁止返回错误信息给客户，模糊返回校验错误提示信息

