由于JABA跟踪对象，而不是变量但是对象定义上下文。也就是说，每个系统调用，一个对象被定义，只有在该对象上的操作和授权决定上下文。对这个对象和调用这个对象的调用的路径进行分析。这重复进行，直到所有相关对象（即，目标数据类型的对象）在每个系统调用进行了分析和他们的日志收集。­

第五步骤在所述上下文描述，并确定是值得分析的CFG节点。如果删除节点会导致节点的操作和授权受损那么这就是有价值的节点。因此，调用具有受控的相对于当前上下文对象的操作或授权的任何函数的功能是值得记录的。

第六步创建的节点值得分析的所有相关过程内路径。这是在分析的关键步骤，并且其详细描述在第5.3节中讨论。

第七步结合过程内的路径成单独的日志。从逻辑上讲，它列举ofintraprocedural在执行顺序记录所有组合。在实际中，日志生成被优化，以从提供新过程内路径的下一个组合，而不是从头开始构建每个路径中的功能建立新的日志。日志由收集的过程内路径序列为那些会在这些路径中调用的函数建。在接下来的通过，即具有多个过程内路径的日志中的最后一个函数被选择为定义下一个组合。日志到这一点是一样的，所以日志生成由新路径的开始只是做了。­

在第4.2节中描述的日志分析如之前在步骤（7）中生成的日志进行。在日志条目的唯一区别是：（1）JABA对象标识符用于对象，而不是特定于对象的标识符作为在运行时和（2）原语参数不被JABA捕获定义的，所以系统调用标记必须表示由我们在目前的前四个条件句的系统调用的参数的位置存储条件分支标识符。

* 1. 路径分析

过程内分析定义每个节点相关的路径的数量。由于节点的路径数是指数级的所以我们要进可能的减少。我们同时执行节点级和条件语句的分析，以尽量消除冗余路径。只有linux文件系统需要3条路径，其他只需要一条路径。

我们正在寻找的主要事情是操作和授权之间的关系。因此，节点中具有相同关系所有路径可以被组合成一个。因为授权的数量比控制操作的数目小得多，所以我们以授权为主：如果在一个节点的所有路径具有相同的授权，那么的控制操作可以组合成一个路径。由于授权控制的操作之前，通常会出现，这种优化是非常有效的。

在功能不同的路径是由它的条件语句的条件规定，每一个条件语句的分支都会生成一个路径。如果条件语句生成的路径并不不影响控制操作与授权之间的关系，那么这些路径不必单独考虑。我们称这个过程*合并。* 在大多数函数中，这导致在路径数一个显著减少。

1. 节点以拓扑顺序进行排序与CFG保持第一。
2. 确定节点调用有任何授权的节点。如果没有，压缩控制操作为一个路径。
3. 收集节点的基本块到一个路径。如果找到一个条件语句，分析条件语句生成的路径是否可以合并。如果不是，选择第一路径，将其他路径压入栈中继续进行分析。
4. 在进行路径分析的过程中保存函数调用的位置。
5. 当一个节点的路径分析完毕时，退栈分析其他路径，直到全部路径都被分析。

在Linux虚拟文件系统的唯一有困难的函数是path\_walk，其中有28条相关的路径。首先，函数中有很多很复杂的条件语句。然而，我们认为，这个函数调用，实际上是包含的很多语义不同的子程序，所以JABA数据流分析不能捕获正确的语义。许多子程序没有分析的必要。path\_walk函数路径分析需要3分钟，很多其他函数仅需几秒或者几十秒，因为大多数其他节点只具有1或2个路径。­­

Linux的虚拟文件系统的全部处理时间大约需要1小时。我们乐观地认为，静态分析性能可以显著改善。目前，我们的JABA系统的架构方式，我们必须重新计算path\_walk函数的路径分析，因为很多系统调用中使用它。除去该多余的计算减少大概一半的处理时间。另外，我们可以通过启动第一次看到节点的分析以进一步提高性能。

* 1. 静态一致性结果

我们的目标是使用JABA一致性分析，发现无论是4.3节的漏洞还是TOCTTOU漏洞，我们使用静态分析工具CQUAL发现。这两者都可以被归类为一致性错误，但后者是很难找到一个运行时分析，因为它需要每次成功的攻击造成结果不一致。但如果是我们已经知道的漏洞，我们就不需要工具。

回想一下，file\_ops-> set\_fowner错误，因为在该领域（读取在这种情况下操作）的受控操作由对于除lease\_modify所有功能的set\_fowner授权。静态日志包括fcntl\_setlease项，这表明，在某些日志set\_fowner授权并在某些日志事实并非如此。有趣的是，日志不会收集日志条目lease\_modify，因为它总是与相同的授权调用，只有的fcntl。因为关系是固定的，我们不需要考虑功能区内的路径。不过，我们也必须考虑到一个特定的控制操作可以使用可识别的一致性问题，比如这里的情况。在这种情况下，我们想知道的是，日志可以调用lease\_modify等功能，并执行其控制操作。由于控制运营和授权之间的关系是静态的，我们可以聚集所有的控制操作（即，将它们合并成一个总的路径）。这样做将使瓦利分析工具，以确定缺乏lease\_modify授权的。­­­­

该TOCTTOU漏洞Zhang等人发现，因为文件指针被授权sys\_fcntl，而是一个新的文件指针是基于fcntl\_getlk用户提供的文件描述符提取。由于用户可以控制文件描述符和它引用的文件之间的映射，竞争条件导致攻击者可以利用来进行未经授权的fcntl操作。JABA提供跟踪程序中的变量和字段的可能值的数据流图。在这种情况下，我们需要根据其来源来区分文件指针变量（即从文件描述符fget）。生成的静态记录表明，文件指针对象是sys\_fcntl被授权在所有情况下，而不同的文件指针对象在被从未授权fcntl\_getlk使用。­

因此，JABA分析工具可以覆盖我们以前使用的静态分析，以及更广泛的一致性分析。

1. 讨论

在本节中，我们简要分析一下在使用LSM一致性分析的三个问题：（1）静态分析的有效性; （2）LSM钩放置的验证; 和（3）用于回归测试。­

6.1静态分析的问题

在一般情况下，我们发现，我们可以使用通过两种动态或静态方法收集的日志数据所描述的一致性分析。目前证明概念的静态分析日志收集与构建和执行动态的日志收集大约需要同样多的时间。并且可以使用任何一种工具来执行相同的日志分析。

静态分析具有路径覆盖和准确性的优点。首先，它能够找到执行几乎所有相关的路径。基本上，为了执行正确的路径代码的静态分析是需要必要的基准。其次，以前的CQual静态分析表明，TOCTTOU错误可以发现，但是无法找到­­­

使用动态分析。查找使用动态分析这些错误的要求有一个主动的攻击者攻击TOCTTOU漏洞，所以我们已经知道攻击的存在。

有效利用静态分析依赖于处理所有必要的控制操作和使用数据流量来识别授权的对象。读操作扩展分析会延长日志的长度，并可能增加相关路径的数量。相关的路径数量取决于控制操作和授权之间的新关系，但由于每个系统调用的授权的数量较少，所以可以进一步细化上下文的概念，以限制日志生成的范围。

静态分析的一个重要用途是判断通过LSM钩子授权的对象。该LSM钩子没有明确描述他们授权的对象，因此必须确定。使用手动指定的运行时分析这样的特性有，但推断静态分析可以使得这样的性质。在一般情况下，传递到挂钩对象的授权。另外，也可以认为是被授权的对象通过固定字段引用的对象进行授权。这些场是否是不可改变的取决于是否有一个系统调用，可以得到一个别名相同的对象，并且可以修改字段。创建对象一般要求不符合，因为他们创造一个新的对象，所以他们不能修改现有对象。虽然文件的文件描述符关系可以通过系统调用进行修改，文件对象和其索引节点之间的关系一旦创建了文件对象很可能不会被修改。我们正在研究的数据流分析会使这种特征得到验证。­

1. 验证

大多数分析工作的目的是找bug。尽管一些分析方法是完整的，这些工具还没有被用来证明系统任何显著安全性能。有了足够的分析支持，我们希望能够真正验证系统的安全性。我们要考虑在完成这项任务时起到的作用。

我们的分析使验证，即一个特定的一组 *中介业务­* （例如，LSM钩子授权）被一致地执行，当一组的 *使用操作*（例如，内核控制操作）被执行时。如果中介操作可以在事先确定（即，我们可以找出所需的授权）就可以关联到操作，那么这个分析就可以完成。

在LSM钩子分析，事先未指定用于控制操作的授权，因此一致性分析所述来确定所需操作。有三个问题，使这个假设是困难的：⑴ 有可能是LSM的钩子只是完全缺失，因此不存在不一致的问题;⑵上下文描述可能不包括所有相关路径; （3）的设定由LSM钩子标识的对象可以不包括某些已经被授权的对象。在第一种情况下，如果是一个钩子在任何情况下不丢失并不存在一致性问题。这个想法是，授权控制的操作关系的聚集使这个手动验证。然而，在人工验证错误的可能性。在第二种情况下，上下文规范必须保证一致性分析涵盖的相关日志路径。为了实现这一目标，每个控制操作必须属于一个范围内声明的授权关系是一致的，对失踪钩子进行验证。三种类型的上下文的简单性（即召回系统调用，系统调用的标志，和操作集）我们的目标是使用JABA数据流分析，以验证该属性。

因此，如果我们能保证所有的内核路径，这种影响的一致性，保证他们的授权的所有对象的全覆盖，并可靠地验证授权要求，那么我们就可以进行有效的验证。我们预计，我们可以验证，如对象标识（使用前标签），对象初始化，审计和对象重用性。例如，审计业务应与安全决策匹配。这些都是很容易识别，所以这个验证应该是简单的推断，虽然受影响的对象依然是一个问题。对于标签，我们可以识别标签的操作，并确定它们是否对前安全检查的所有对象运行。然而，对象可以被重新标记。确定是否发生重新贴在必要时需要重新标记定义的要求。指示重新标签和表示使用一个重新标记对象的受控操作必须被确定。因此，我们可以验证初始标记，但正确的重新贴标签的验证更加困难。­­­­­­

6.3回归

一旦安全性能得到验证，必须确保当系统更新后此属性保持不变。该过程通常被称为回归测试。由于很多人使用Linux系统因此回归测试比较简单。特别是，我们想最大限度保证一个简单的变化将不会导致回归的数据没有手动检查的可能性。

对于LSM钩子验证，使用稠度类提供针对改变一些弹性。只要一致性类列表不改变，那么输出基本上是相同的。由于一致性的变化是指示性的安置问题，在回归测试一致的一致，表明有效安置。为了让这个假设持有，我们还必须考虑的事实是必要的挂钩可以在任何情况下丢失。因此，只要没有新的控制操作，则此回归方法的工作原理。对于增加新的控制操作，需要做更多的工作。­­

但存在潜在的问题才会对受控操作进行测试。一般而言，较大的不同集合的授权给该受控操作在一致性列表相关联的数是从特定的授权其独立的一个量度。这种情况将被视为­需要一个新的LSM钩子。在另一方面，如果控制操作是一组定义的上下文控制操作的一部分，则有必要对一个LSM钩的可能性要高得多。我们可以汇总基于这些种类的标准来判断我们需要的控制操作。

1. 结论

在本文中，我们提出了一致性分析用于验证LSM框架的正确性。LSM框架由一组放置在内核内部授权挂钩子所组成，所以它更难以识别完整的调试点。我们利用的事实是大多数LSM钩子会被放到正确的地方，并通过这一事实判断错误的钩子。我们使用对主要内核数据结构作为中介界面结构件业务，并收集这些操作的授权。通过分析运行时测试的输出，判断它的授权是否是不一致的操作。我们分析了文件系统和一些任务操作，发现一些可能被利用的异常。与LSM社区沟通，这些问题已经被修复。

最终，我们发现一致性分析验证，认定与规范不一致的可能是错误的方式非常有用。然而，运行时间日志收集在路径覆盖的局限性和会错过一些重要的错误，比如TOCTTOU，这个漏洞依赖于异常的输入数据或上下文。我们已经发现，这种分析方法也可以应用于通过静态分析收集的执行日志。我们演示了如何使用JABA分析工具收集这些日志。此外，我们还讨论了改进，可以使用JABA的数据流分析进行了全面的分析。我们正在积极寻找完整的，易于使用的验证的LSM授权钩子和检查等安全性能的方法。­­­