

图5曲线图授权为呼叫的fcntl关于^ SETLEASE（控制操作中的LEA ^修改和fput）和^ SETOWN（在d ^的fcntl控制操作和放）。当命令是^ SETOWN既FCNTL和SET\_0WNER被授权，但只能FCNTL被授权F\_SETLEASE。

（2）(2) fcnt1 (fd, F-SETLEASE, F-UNLCK)。受控操作节点包括位置(函数名、文件名、行号)和操作(数据类型、成员偏移量、操作类型)信息。授权节点包括authorization、命令和包含授权的函数。总是用实线表示边，有时用虚线表示边。如果受控操作和授权之间不存在边界，则永远不会为该操作执行该授权。

通过形象化的分析，我们可以分辨出当前图中是否包含在3.3节中描述的常量。这样，这样的话，有时候，fput和它的授权之间的关系可能表示存在问题。针对相同域的不同授权集合也可能表示出现了问题。这就需要人工检查来分辨到底是系统出了错误，还是一个合法的一致。

1. 一致性等级列表 一致性等级列表展示了一种通过一致性等级及其关联的权限要求的划分。这种划分是通过3.3节介绍的算法实现的。与授权图相比，授权等级列表为授权结果提供了一种新的视角。授权图显示了每个受控制操作与授权之间的关系，一致性等级列表使得数据中存在的不同授权情况变得更加明显。而且，由于它是文本性的，所以更容易进行回归测试[Jaeger等人的一致性类列表是更容易回归测试使用。2002]。

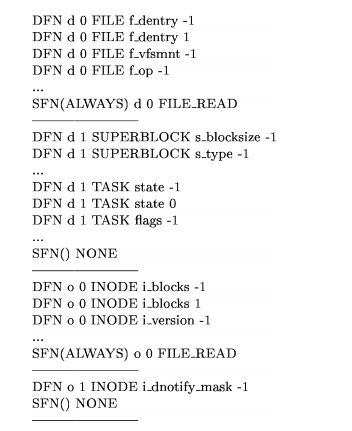
图6显示了如何把系统读调用的受控操作进行一致性等级划分，这个划分在3.3节中已经举过例子了。像那里描述的那样，一致性等级列表中数据类型的两个等级：一个是没有授权的任务和超级块，另一个是对文件的读权限。在对象一致性方面，有两个等级：一个是对inode的读操作，另一个没有读目录的权限。基本上，我们希望如果给与当前任务，文件的目录或者文件的超级块授予读权限，那么将会导致所有的受控操作拥有相同的权限（也就是系统调用一致性）。

图6.一致性类列表用于读系统调用具有以下字段：（1）入口类型（DFN或SFN）; （2）稠度（d为数据类型以及o为对象）; （3）类号码; （4）数据类型; （5）部件; （6）访问标识符。

我们大多数经验都是与文件系统相关的，同时我们也测试了任务相关的授权。由于大多数对象只有一个或没有权限，所以一致性等级列表不会太过复杂。取消链接系统调用是为数不多的一个对象含有多个权限的个例之一。使用一致性等级列表很容易看出目录inode拥有三个授权（执行、写和取消链接目录），而正在被删除的inode有一个（取消链接文件），因为它们是对象一致的且被放在不同的等级中。因此，对我们已经测试过的文件系统和任务操作来说，授权图和一致性等级列表已经足够去验证授权了。

* fcntl函数的F-SETOWN只有场f\_owner

1. =（+，ID\_TYPE，CONTEXT）（+，di\_cfm\_eax，的fcntl）（+，co\_ecx，F\_SETOWN）
2. （d，1）=（+，ID\_TYPE，SEC-CHK）
3. （d，1）=（+，ID\_TYPE，CNTL\_OP）（+，di\_dfm\_member，f.owner）

* 的fcntl适用于F ^ SETLEASE只有场f.owner

1. =（+，ID\_TYPE，CONTEXT）（+，二cfnueax，的fcntl）

（+，co\_ecx，F\_SETLEASE）（+，co\_edx，F\_UNLCK）

1. （d，4）=（+，ID\_TYPE，SEC\_CHK）
2. （d，4）= C +，ID\_TYPE＃CNTL\_OP）（+，di\_dfm\_member，F-所有者）

图7.规则异常发现歼所有者。

1. 样本分析。我们简要演示了对我们发现异常的样本分析。虽然发现异常的方法是与实际发现异常同时开发的，但是我们使用的方法与描述的方法大致相同，尽管有些方法不是自动的。这种异常发生在fcntl系统调用中。fcntl的一致性等级列表表明，fcntl的一致性是系统调用输入一致的。cmd和arg参数对fcntl的值可以更改所需的授权。我们使用授权图来查看不同输入下的授权，因为更容易看到粗粒度的问题（很多边缘问题会发生）。

图7包含两套规则：(1)一个收集所有授权和可控操作的文件结构字段f\_owner fcntl (fd, FSETOWN pid\_owner)的系统调用,和(2)一个收集所有授权信息与可控操作字段f\_ownerin fcntl (fd, F-SETLEASE F UNLCK)的系统调用。注意，这与用于生成图5中的图形使用相同的规则（fput 控制操作除外）。­

在图5中，我们看到，一些控制操作的被授权为 的fcntl 和 set\_fowner 授权和一些只授权 的fcntl。尽管这是一个事实，即控制操作访问相同的领域，f\_owner （偏移480）。鉴于这种异常，我们检查了内核源代码，以确定是否可以利用这种异常。我们将在下一节讨论这种分析的结果。

* 1. 结果

我们将2001年12月10日的LSM补丁应用到Linux 2.4.16源代码中，并使用我们修改过的GCC-3.0.3版本编译内核，以创建要分析的执行日志，我们在LMBench的三个并行实例中执行。SAINT漏洞工具(www.wwdsi.com/saint/)，这是一个内核编译器，以及我们在猜测并检查异常时编写的一些测试程序。由于运行时分析的有效性依赖于运行足够多的代码，因此必须开发涵盖足够多有趣路径的基准测试。例如，LMBench只运行大约20%的内核代码。­

我们进行内核测试，以收集对主要内核数据结构(文件、inode、超块、任务、套接字和skbuffs)的受控操作。到目前为止，我们只对文件系统授权做了详细的分析。以及对任务授权的初步分析。由于对文件系统的理解相当透彻，所以我们没有预期会出现大量异常，但是我们还是发现了一些异常。

-成员一致（多系统调用）：etgroups16是几个向后兼容ABI的16位任务操作之一，例如setuid16和setchown16。这些操作通常将其16位值转换为32位值，并调用包含授权的当前版本。但是，由于setgroups16设置了一个数组，因此不进行数组转换就更容易，所以不调用当前版本(包含一个钩子)。我们发现setgroups16函数中没有授权钩子，所以我们可以在没有授权的情况下重置任务组。注意，在libc的当前版本中没有setgroups16调用，因此我们不得不编写一个汇编程序来执行这个漏洞。­­

-成员一致（单系统调用）： 成员一致(单个系统调用):f\_owner的pid成员告诉内核关于这个文件上的IO向哪个进程发送信号。如果用户试图通过fcntl(fd, F-SETOWN, pid\_owner)函数直接设置该字段，则该字段由file\_ops->set\_fowner授权。但是，如果用户通过fcnt1(fd、F- setlease、F UNLCK)函数删除文件的租约，则所有者在未执行授权的情况下被设置为0。此外，进程可以在没有执行授权的情况下将通用TUN设备(驱动程序/网络/tun.c)的所有者设置为自己。为了实现这一点，流程在一个开放的、附加的TUN设备一致性(单个系统调用)上调用ioct1(fd、F-SETFL、FASYNC)函数。

-成员一致（单系统调用）： 在我们进行上述filp.f\_owner的一致性检测时，我们发现，获得的filp.f\_owner.signum（应该IO完成时发送的信号）可以在没有授权的情况下进行设置。

-系统调用一致（多系统调用）： 系统调用一致(多个系统调用):在每次读系统调用开始时执行读授权。此授权是必需的，因为由于进程更改其安全级别、文件更改其安全级别、新进程使用的文件或安全策略中的更改，在最初打开文件时执行的授权可能不再有效。但是，在内存映射文件的页面错误期间，不会执行此授权。因此，一旦进程拥有内存映射文件，它就可以继续读取该文件，而不必考虑对安全级别或安全策略的更改。

我们与LSM社区进行了讨论，除了读取内存映射文件的异常外，我们对所有异常都进行了修补。社区决定，需要读授权的文件不能被内存映射。我们很受鼓舞，因为我们能够帮助发现和修复钩子的放置问题。

1. 研究静态分析

我们现在已经使用JaBA statid分析工具实现了一个静态分析原型，该工具收集Linux ker nel中相关执行路径的日志。对于静态分析，可以比运行时分析更直接地评估完整的内核执行路径覆盖率。不幸的是，ker nel执行路径的数量通常是指数级的，因此模型必须设计成尽可能多地删除不相关路径，并进行优化，以便尽可能多地评估路径。静态生成的日志可以作为运行时生成的日志输入到4.2节中描述的分析工具中。­­­

* 1. JaBA分析工具

JaBA是基于指针的数据流分析的流敏感静态分析工具[Koved等，2002]。JaBA具有处理Java类字节码的基本函数，用于为变量和对象字段构建过程内基本块图、过程间控制流图和数据流图。我们将此基本分析信息作为输入，为日志收集构建进一步的静态分析。这些分析是过程性的，而不是声明性分析语言，如MC [Engler et al. 20001]。然后我们按照章节4.2的描述处理这些日志。­

JaBA中的上下文敏感控制流意味着可以区分进入同一功能的不同路径，从而可以区分进入同一功能的不同输入对象，并准确跟踪它们与授权的关系。

JaBA中的上下文敏感数据流跟踪程序中分配站点的数据流。对于Linux内核，大多数对象(例如文件和inode)使用相同的分配站点，因此在上下文不敏感的数据流中，该类型的所有操作都将使用相同的对象。JaBA通过调用站点将对象区分为创建对象(或任意级别)的函数，从而支持上下文敏感性的定义。但是，创建的对象越多，分析的成本就越高。。

由于Linux内核显然不是Java的，所以我们编写了从C到Java字节码的转换[Zhang et al.]。转换只保留与JaBA分析相关的Linux内核语义。例如，指向C结构的指针被转换为大小为1的数组，数组中包含相应类型的Java对象。另外，非类型安全指针置换被转换为wildPtr对象。。­­

* 1. 日志生成过程

JaBA日志分析过程使用两个输入:(1)根控制流图节点，通常是系统调用(例如sys\_open)的函数;(2)正在分析其控制操作的结构的数据类型。所有JaBA分析仅限于函数(即…可从根节点访问的CFG节点)。日志生成返回由运行时数据收集工具收集的表单的日志。

使用JaBA生成日志的过程如下：

1. 转储ID命名映射功能，授权，并通过日志分析工具的使用来控制操作。
2. 收集指定数据类型对象的所有受控操作。这些数据类型被指定为分析过程的输入，并且集合的范围受根函数的限制。
3. 收集指定的数据类型的对象的所有授权。
4. 为每个路径定义一个一致性上下文，当前定义为系统调用中使用的每个单独对象实例中的每个路径。由于系统调用标志值的关系，此上下文还可能包含条件分支限制，来模拟转换工作。­
5. 使用路径标识节点，这些路径将导致受控操作或授权。所有其他节点都与分析无关。
6. 生成所有相关的过程内路径(见下文)。
7. 从生成过程内的路径序列过程间的路径。
8. 重复步骤4，直到没有新的上下文需要检查。

第一步通过提供从数值表示(如函数CFG节点编号)到名称(如函数名称)的映射，使日志分析工具能够正确显示来自JaBA生成的日志的结果。­

第二步收集感兴趣的数据类型的所有受控操作，正如在运行时情况中所做的那样。在这个概念验证中，我们到目前为止只收集写访问:需要收集读访问来查找一些错误(例如从f-owner字段读取以更新其成员)，因此需要扩展到读访问。虽然增加更多的访问只会增加分析的复杂性，但是我们认为额外的复杂性并不像第6节中讨论的那样重要。。­

第三步收集所感兴趣的数据类型的授权。在运行时系统中，我们必须手动列出所有被认为是由每个LSM钩子授权的对象。在这里，我们希望使用数据流更准确地描述第6节中讨论的授权对象。但目前只收集直接授权。我们定义了一种从授权中推断授权对象的机制，但是这个步骤目前没有运行时分析中广泛。。

第四步定义了一组分析上下文。在运行时工具中，系统调用的执行定义了上下文。在静态分析中，系统调用定义了许多路径，因此我们需要标识值得收集的路径上下文。我们已经确定了一些方法来减少值得考虑的程序性路径的数量(见5.3节)，但是我们仍然发现需要进一步减少以便及时分析。我们遵循xgcc项目的领导，其中分析自动机处理是依赖于变量的[Hallem et al. 2002]。

因为JaBA跟踪对象而不是变量，所以对象定义上下文。也就是说，对于每个系统调用，将标识一个对象，并且只有对该对象的操作和授权才定义上下文。该对象的路径分析完成后，系统调用中的另一个对象被选中，所有相关上下文(即对象的路径)被记录。这样重复直到所有相关的对象(即分析每个系统调用中的目标数据类型的对象，并收集它们的日志。­

第五步进行上下文描述，确定值得分析的CFG节点。如果一个节点被移除，那么它就值得分析。删除它的任何后代将导致丢失受控操作或授权日志条目。因此，调用任何具有与当前上下文相关的受控操作或授权的函数的函数都值得日志记录。。

第六步为值得分析的节点创建所有相关的过程内路径。这是分析中的关键步骤，将在5.3节中详细讨论。。

第七步将过程内路径合并到单个日志中。从逻辑上讲，它按执行顺序列举了过程内日志的所有组合。实际上，日志生成是为了从为下一个组合提供新的过程内路径的函数构建新日志而优化的，而不是从头构建每个路径。日志是通过为那些在下一次传递中将在这些路径中调用的函数收集一系列过程内路径来构建的，选择日志中具有多个过程内路径的最后一个函数来定义下一个组合。到那一点的日志保持不变，因此日志生成只在新路径开始时进行。­

4.2节中描述的日志分析之前执行步骤(7)中生成的日志。唯一不同的日志条目有:(1)JaBA种标识符用于对象而不是特定对象标识符定义运行时和(2)不被JaBA原始参数,那么系统调用标记标识符必须由一个条件分支,我们存储在系统调用的参数位置第一目前四个条件。