实时调度器中的一种新型侧信道

摘要

（文中提出了 SchedulLeak 算法，演示如何利用侧信道捕获一些定时信息，从而实现更高级的攻击）我们证明了在抢占式固定优先级实时系统（RTS）中存在一种新的调度器侧信道，这种系统可以在汽车系统、航空电子系统、发电厂和工业控制系统中找到。此侧信道可以泄漏重要的定时信息，例如实时任务的未来到达时间。然后可以使用这些信息来发起毁灭性攻击，这里演示了其中两种（在真实的硬件平台上）。请注意，由于调度中的运行时变化、系统中存在多个其他任务以及 RTS 设计中的典型约束（例如，截止时间），因此不容易捕获这些定时信息。我们的 SchedulLeak 算法演示了如何有效地利用这种侧信道。一个完整的实现是在真实操作系统（实时 Linux 和 FreeRTOS）上实现的。SchedulLeak 泄漏的定时信息可以显著地帮助其他更高级的攻击更好地实现目标。

1、引言

（敌方想要攻击嵌入式系统，需要获得大量的系统的特定信息）考虑这样一个场景：敌方想要攻击嵌入式实时系统（RTS）——自动汽车的部件、工业机器人、现代汽车的防抱死制动系统、无人机（UAV）、电网组件、美国宇航局漫游者、植入的医疗设备，这些系统通常具有有限的内存和处理能力，具有非常严格的设计（例如严格的定时约束），任何意外的行为都可以很快被阻止。因此，窃取关键信息的机会或发起控制系统的攻击的能力非常有限。因此，对此类系统的攻击需要大量特定于系统的信息。这些“信息”可以有多种形式——从对系统设计的理解到对关键组件（软件或硬件）的了解。攻击的准确程度取决于目标的类型和攻击的类型。例如，（a）窃取车载摄像机用于侦察的重要信息，或（b）从遥控车辆的地面操作员手中夺走控制权。

（周期性任务是攻击者的主要目标）实时系统中普遍存在的一个共同的基本主题是 timing 的重要性。“时序 timing”包括（i）某些事件发生的时间，（ii）它们发生的频率，以及，对于本文来说最重要的，（iii）它们将来何时（以及是否）再次发生。事实上，实时系统中的许多关键软件组件本质上是周期性的。正如我们将看到的，这些周期性任务本身就是攻击者的主要目标。

那么，如何攻击这样的系统，尤其是周期性的（和关键的）组件呢?

我们发现在实时系统中存在一个基于调度器的侧信道，它会泄露定时信息，尤其是那些具有固定优先级任务的系统。

（基于调度器的侧信道可以使攻击者得到周期任务的定时行为的信息）基于调度器的侧信道允许**非特权、低优先级任务**通过使用系统计时器观察其**自身的执行间隔**来学习**关键、周期性（被攻击者）任务**的精确定时行为。这使攻击者能够推断被攻击者任务的初始偏移量，并精确预测其在运行时的未来到达时间。我们将利用这种侧信道攻击的算法命名为**“SchedulLeak”**。

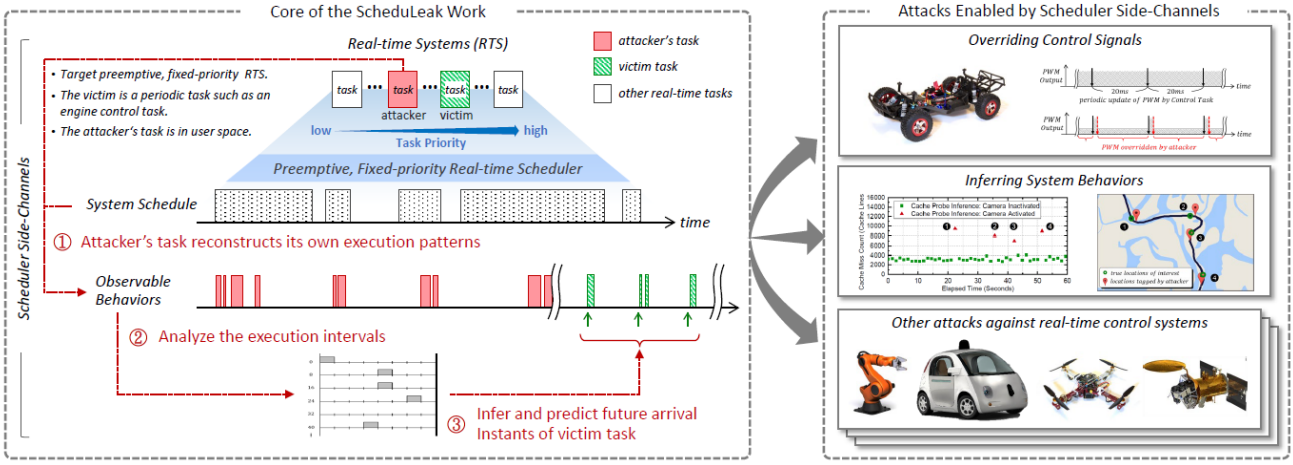


图 1：论文概述：我们展示了一个无特权的低优先级任务（在用户空间中）如何使用 ScheduLeak 算法来推断关键的、高优先级的周期性任务的执行行为。提取的信息有助于帮助其他攻击实现其主要目标（本文中实现了两个此类攻击实例作为可能的用例）。

（攻击者如何从调度器端基于信道的信息中获益）图 1 显示了侧信道的概述，以及攻击者如何从调度器端基于信道的信息中获益。图的左侧显示了一个由固定优先级任务组成的实时系统（顶部的框-被攻击者是周期性任务，而所有其他任务可以是周期性的或零星的），从而产生一个调度表（中间的虚线框，当每个任务在运行时无法与其他任务区分时），可以对其进行分析，以提取出被攻击者任务的精确未来到达时间点（绿色的向上箭头）。图的右侧显示了如何利用关键任务的这个定时信息来发动其他攻击，这些攻击要么泄漏更重要的信息，要么破坏实时控制系统的稳定性。请注意，如果没有这些精确的时间信息，攻击者要么被迫猜测被攻击者任务何时执行，要么在随机时间点发起攻击——这两种情况都会削弱攻击的效力，要么导致系统提前终止。

（运行时计时信息的提取是非常重要的，仅知道静态系统参数不足以重建被攻击者未来的到达时间，但是让攻击者跟踪自己的调度信息，可以高精度地重建目标定时信息。）这种运行时计时信息的提取是非常重要的；主要原因包括：（a）运行时调度严重依赖于系统启动时的状态、初始化变量和环境条件；（b）实时系统通常也包括多个非实时任务。即使对所有静态已知系统参数的精确了解也不足以重建被攻击者未来的到达时间，虽然有特权的攻击者可以以系统的调度程序为目标并提取所需的信息，但这种访问通常需要大量的努力和资源。另一方面，我们可以使用非特权用户空间应用程序以相同的精度重建信息，这是通过让攻击者的应用程序跟踪自己的调度信息来实现的，再加上一些容易获得的关于系统的信息（例如，被攻击者任务的周期），**攻击者可以高精度地重新创建目标定时信息。**

（举个具体的例子）更具体地说，我们想覆盖月球车的（远程）控制。在许多这样的系统中，周期性脉冲宽度调制（PWM）任务驱动转向和油门。在不知道 PWM任务何时可能更新电机控制值的情况下，攻击者被迫使用暴力或随机策略来覆盖 PWM值，这些可能最终无效，或者导致整个系统在攻击成功之前被重置（关于这个和另一个场景的更多细节，请参见第六节）。有了 ScheduLeak 的获得的信息，我们的智能对手现在可以在相应任务写入 PWM值后立即覆盖它们-有效地覆盖驱动命令。

（本文工作的重点是侧信道可以将关键的、高优先级的实时任务的执行定时行为泄露给没有特权的低优先级任务）调度器隐蔽信道，其中两个进程使用调度器秘密通信，早已为人所知（例如，[1]，[2]，[3]）。相比之下，我们的重点是一个侧信道，它将关键的、高优先级的实时任务的执行定时行为（不是故意的，与调度器隐藏的信道相反）泄露给没有特权的低优先级任务。**我们专注于单处理器（即单核）系统，具有抢占式、固定优先级的实时任务调度器[4]，[5]，因为它们是目前实际部署的最常见的一类实时系统[6]**。对于攻击者来说，保持在分配给非特权任务的严格执行时间预算内是很重要的，尤其是在试图观察和重建被攻击者任务的定时行为的阶段，如果不满足此要求，则可能会导致其他关键实时功能失败，或触发重置系统的看门狗超时，从而导致攻击者过早被弹出。在所谓的高级持续威胁（APT）攻击的“侦察”阶段，该属性至关重要[7]，[8]。例如，有报道称，在震网[9]事件中，攻击者在发动实际攻击之前，已经侵入系统并在系统中呆了几个月而未被发现，一旦他们掌握了足够的系统内部信息，他们就能够针对特定系统设计有效的攻击。（攻击者在获取足够信息之前，要按照系统可调度性来执行，避免被发现）

SchedLeak 算法是在两种情况下实现的：（a）运行实时 Linux 和 FreeRTOS 的真实硬件平台（针对这两种攻击案例研究）和（b）模拟器。我们在第七节中评估了 SchedLeak 的性能和可伸缩性，并进行了设计空间探索（在模拟器上）。结果表明，我们的方法能够有效地重构调度信息，为以后的攻击提供有价值的信息，以更好地实现攻击目标。总而言之，主要贡献如下：

1） 新的调度器侧信道攻击算法，能够准确地重建实时系统中关键实时任务的初始偏移量和未来到达时间（无需特权访问）[第三节]。

2） 分析和量度，以衡量预测被攻击者任务的执行和时间特性的准确性[第四和第七节]。

3） 在运行实时 Linux 和 FreeRTOS 的真实硬件平台（即自治系统）上的实现和案例研究[第六节]。

2、系统和对手模型

A、时间模型

我们假设攻击者可以访问目标系统上的系统计时器，因此攻击者测量的时间具有与此系统计时器相等的分辨率。计时器可以是软件计时器，也可以是硬件计时器（例如，FreeRTOS 中的 64 位全局计时器或 Linux 中基于时钟单调的计时器）。我们考虑一个离散时间模型[10]。我们假设一个时间单位等于一个计时器的滴答声（攻击者可以访问的计时器的滴答声），滴答声计数是一个整数。所有系统和任务参数都是时间刻度的倍数。我们用[a，b）或[a，b-1]表示从时间点 a 开始到时间点 b 结束的间隔。

Table I: A summary of the system and adversary model.

Real-Time System Assumptions	
A1	A preemptive, fixed-priority real-time scheduler is used.
A2	The victim task is a periodic task.
Attacker's Capabilities (Requirements)	
R1	The attacker has the control of one user-space task (observer task) that has a lower priority than the victim task.
R2	The attacker has knowledge of the victim task's period.
R3	The attacker has access to a system timer on the system.
Attacker's Goals	
G1	Infer the victim task's initial offset and predict future arrivals.

B、系统模型

我们考虑一个由 n个实时任务组成的单处理器（即单核）、固定优先级、抢占式实时系统。任务可以是周期性任务，也可以是零星任务。每个任务 τi用（pi, di, ei, ai, prii）标识，其中 pi是周期（或最小到达时间），di是相对截止时间，ei是最坏执行时间（WCET），ai是初始任务偏移量（即到达时间），prii是优先级。我们假设每个任务都有一个不同的周期，任务的截止时间等于它的周期[5]（di= pi）。为了便于表示，我们使用相同的符号 τi来表示任务的作业（或实例）。我们假设任务释放抖动可以忽略不计。因此，周期性任务 τi的任何两个相邻到达都有一个恒定的距离 pi。我们进一步假设每个任务被分配一个不同的优先级，并且任务集可以由一个固定优先级的抢占式实时调度器调度。设 hp（τi）表示优先级高于 τi的任务集，lp（τi）表示优先级低于 τi的任务集。我们将任务的“执行间隔”定义为任务连续运行的时间间隔[a，b）。如果 τi被抢占，那么执行将被划分为多个执行间隔，每个间隔的长度都小于 ei。

C、对手模型

（本文开发的方法可以针对具有周期性、偶发性和非实时性任务的系统。）我们假设攻击者对系统中的一个关键任务感兴趣，我们将其称为“被攻击者任务”，用 τv∈ Γ 表示。我们还假设 τv是一个实时的周期性任务。实时控制系统中的许多关键功能本质上是周期性的，例如震网事件示例[9]中控制从变频驱动器频率的代码。在所有这些情况下，任务的周期与物理系统的特性严格相关，因此可以从物理特性中推断出任务的周期；因此，**我们可以假设攻击者能够事先获得被攻击者任务周期的信息**。通常情况下，在攻击复杂系统（如 CPS）之前，对手会研究此类系统的设计和细节。但是，攻击者不知道系统启动时的初始条件（例如，任务的初始偏移量），并且可能没有系统中所有任务的信息。系统中的所有其他任务可以是周期性的、零星的或非实时性的，具体取决于系统的设计。因此，本文开发的方法可以针对具有周期性、偶发性和非实时性任务的系统。

（本文中介绍了攻击算法，精准的推断被攻击者任务在将来的运行时间）最终目标因敌方和被攻击系统的不同而不同。例如，在高级持续威胁（APT）攻击[7]、[8]中，可以计划干扰关键任务的操作，通过共享资源窃听某些信息，甚至在被攻击者系统最脆弱的关键时刻实施削弱性攻击。通常，此类攻击需要攻击者精确地测量被攻击者任务的时序属性。在本文中，我们介绍了一些攻击算法，帮助攻击者在侦察阶段获得这些有价值的信息。在这种情况下，**攻击者的主要目标是精确地推断被攻击者任务在不久的将来（即未来到达时间）调度何时运行。**

（本文的重点是如何在不违反实时约束的情况下，利用调度器侧信道重构高优先级周期性被攻击者任务的定时行为）注意，我们在本文中的重点是如何在不违反实时约束的情况下，利用调度器侧信道重构高优先级周期性被攻击者任务的定时行为。我们从一个折衷的、低优先级的（“观察者”）任务的角度来做这件事，我们不关注攻击者如何访问观察者任务，许多业内人士通常都会采用复杂的供应链开发方法，比如从汽车到飞机等，软件和网络中的漏洞。最近的研究表明，像商用无人机这样的实时系统存在设计缺陷，因此容易受到损害[12]，[13]。访问观察者任务的细节超出了本文的讨论范围。然而，需要注意的是，我们并不要求观察者任务是系统中的特权任务。表 1 总结了假设、攻击者的能力和目标。（文中不讨论访问观察者任务的细节）

D、观察者任务

（文中考虑周期性观察者任务，其优先级低于被攻击者任务，且文中说明，既然周期性观察者任务可以成功，那么零星的观察者任务必然可以成功，因此文中只考虑周期性任务）如前所述，我们将攻击者控制的低优先级任务称为“观察者任务”，用τ o∈Γ 表示，它可以是用户空间任务。我们对τ o的唯一限制是它的优先级低于被攻击者任务 priv> prio。观察者任务可以是周期性任务，也可以是零星任务，其周期（或其最短到达时间）可以短于或长于被攻击者任务。特别是，作为一个周期性任务是一个更严格的限制条件，因为它降低了攻击者可用的灵活性（这将使我们在介绍算法时更加清晰）。**也就是说，在具有周期 Po和优先级 Prio的周期性观察者任务能够成功的情况下，零星的观察者任务（通过选择与最小到达时间相同的 Po和相同的优先级 Prio）也可以成功。**因此，在分析第四节的攻击能力时，我们将考虑一个周期性的观察者任务（或者一个以固定的到达时间运行的零星观察者任务）。

（观察者任务通过使用系统计时器检视自己的执行间隔，从而推断被攻击者任务的初始偏移量，难点是观察者任务可能会被抢占，如何过滤掉不必要的信息是一个挑战）在本文中，我们使用观察者任务来推断可用于预测被攻击者任务未来到达时间的初始偏移量 av。我们让观察者任务通过使用系统计时器“监视”自己的执行间隔。**注意，在大多数操作系统中读取系统时间并不需要特权（例如，在 Linux 中调用 clock_gettime（））。**这里的**关键思想**是**观察者任务处于活动状态时的时间间隔不能包含被攻击者任务的执行或到达时间点**，因为**被攻击者任务会抢占观察者任务**。然而，还有其他更高优先级的任务会影响观察者任务的执行行为（这部分是影响算法准确度的因素之一）。对攻击者来说，挑战是过滤掉不必要的信息并提取出关于被攻击者任务的正确信息。这将在下一节中解释。

3、SchedLeak

A、概述

我们现在介绍核心算法。主要思想是，当观察者任务运行时，被攻击者任务不能运行，因为观察者任务的优先级较低，通过重构观察者任务自身的执行间隔，并根据被攻击者任务的周期分析这些间隔，我们可以推断出被攻击者任务的初始偏移量和未来到达时间。我们提出的 SchedLeak 算法的各个分析阶段的高层概述包括：

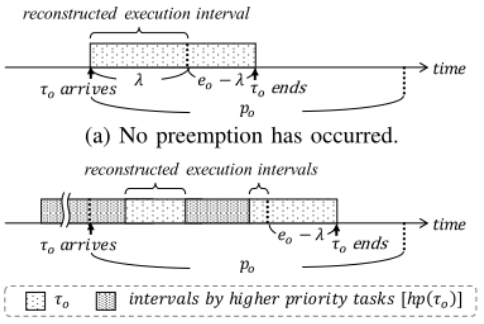
1） 重构观察者任务的执行间隔：首先，观察者任务使用一个系统计时器来测量和重构它自己的执行间隔（即，它本身是活动的时间）。[第 III-B 节]

2） 分析执行间隔：重构的执行间隔被组织在一个“调度梯形图”中，这个时间轴被划分为多个窗口，用以与被攻击者任务的周期匹配。[第 III-C 节]

3） 推断被攻击者任务的初始偏移量和未来到达时间：在最后一步中，推断被攻击者任务的初始偏移量。然后利用这些信息来预测被攻击者的未来到达时间。由于被攻击者任务本质上是周期性的，从它自己的窗口开始的偏移量转化为被攻击者任务的第一个实例执行时从启动开始的偏移量。[第 III-D 节]

B、重构执行间隔

（我们定义了一个参数 λ，作为最大重建时间，用于尽量节省执行预算，从而保证实时约束）第一步是重构观察者任务的执行间隔。**我们在观察者任务中实现了一个函数，它跟踪从系统计时器读取的时间，通过检查轮询的时间戳，可以识别抢占（如果有的话），并重构观察者任务的执行间隔。**虽然这个功能看起来很简单，但确保它遵守实时约束（即所有实时任务必须满足其截止日期）是至关重要的。也就是说，观察者任务的执行时间不应超过其 eo，WCET。此外，即使攻击者没有超出为自己分配的执行预算，它也可能希望为其他目的节省一些预算，例如执行分析以重建被攻击者的定时信息。因此，我们定义了一个参数 λ，该参数的值由攻击者设置，以限制观察者任务在每个周期内的运行时间。**“最大重建时间” λ** 是 0< = λ < = eo 范围内的整数。在每个周期中，重构的执行间隔的总长度为 λ，这使得观察者任务可以执行其他计算的时间跨度为 eo-λ。因此，原始（干净）系统保证的服务级别仍然保持不变，从而降低了触发系统错误的风险。另一方面，攻击者可能无法捕获所有可能的执行间隔，这可能会降低最终结果的保真度/精确度。第 IV-B 节讨论了如何计算 λ 的良好值。图 2 显示了重建的执行间隔的示例。在考虑 λ 的情况下，重构观察者任务的执行间隔的函数在附录 A 中作为算法 1 进行了详细说明。



(a) No preemption has occurred.

(b) Some tasks $\tau_i \in hp(\tau_o)$ preempt τ_o.

Figure 2: Examples of reconstructed execution intervals of the observer task. The total length of the reconstructed execution interval(s) is λ that leaves $e_o - \lambda$ for τ_o to perform original task functions.

图 2:观察者任务的重构执行间隔的例子。重构的执行间隔的总长度为λ，剩下的 eo-λ 用于执行 τo 的原始功能。

C、执行间隔分析

（将观察者的执行间隔组织成为一个时间轴，并匹配划分为被攻击者任务的周期，使攻击者能看到观察者任务的执行间隔是如何受到被攻击者任务以及其他高优先级任务的影响的）一旦观察者任务的执行间隔被重建，我们分析数据以提取关于被攻击者任务的信息。我们将观察者任务的执行间隔组织成一个时间轴，并被划分到长度为被攻击者任务的周期 Pv 的时间轴中（回想一下 Pv 是攻击者已知的量之一）。此步骤的目的是将观察者任务的执行间隔置于被攻击者任务的周期性窗口内。时间轴被分割成与被攻击者任务的周期相匹配的窗口，使攻击者能够看到观察者任务的执行间隔是如何受到被攻击者任务以及其他高优先级任务的影响的。

（图 3 是调度梯形图，用来说明时间轴的思想和提出的算法，这是基于 Pv 绘制的）为了更好地说明时间轴的思想和所提出的算法，我们将使用“调度梯形图”（定义如下）来表示本文中时间轴的构造。调度梯形图中的行可以合并为单行时间轴（这只是一个分析性的“技巧”）。调度梯形图是一个框架，由一组长度相等的相邻时间轴组成，这些时间轴与被攻击者任务的周期 Pv 相匹配。顶部部分的开始时间可以是攻击者指定的任意时间点（例如，首次调用算法的时间瞬间）。调度梯形图中的列是“单位时间列”。所以，一共有 Pv 个时间列。也就是说，调度梯形图与重构的执行间隔具有相同的时间分辨率。调度梯形图的框架如图 3 所示。从基于 Pv 绘制的图表中，我们进行了以下观察：

观察 1：任何 τv 的调度梯形图必须在每一行中正好包含一个 τv 到达实例，τv 的所有到达时间都位于同一时间列。（因为 τv 是周期性的任务）

这种观察是正确的，因为 τv 是一个周期性任务，它每隔 Pv 个时间单位到达，并且绘制的调度梯形图的间隔等于 Pv。我们将被攻击者的任务到达列定义为“真正到达列”，用 δv 表示。因此，初始偏移量 av 和真正到达列 δv 之间的相关性可以通过（t+ δv-av）mod Pv = 0 来推导（av 即为 δv），其中 t 表示攻击者分配调度梯形图的（任意）开始时间。图 3 也描述了这一点。基于这一观察，我们定义了以下关于观察者任务在调度梯形图上的执行的定理：

定理 1：观察者任务的执行间隔不出现在时间列[δv，δv+bcetv），其中 bcetv 是 τv 的最佳执行时间。（为了避免观察者任务在该时间间隔内被被攻击者任务抢占）

证明：从观察 1 可以看出，被攻击者任务 τv 有规律地到达时间列 δv，如果在 δv 列执行中存在低优先级任务 lp（τv），则被攻击者任务抢占这些任务，直到至少在 bcetv 时间之后完成任务结束作业。当存在高优先级任务 hp（τv）在[δv，δv+bcetv]期间执行或到达时，被攻击者任务 τv 被抢占。在这种情况下，如果观察者任务 τo 在[δv，δv+bcetv)期间到达，作为低优先级任务，它也会被抢占。因此，时间列[δv，δv+bcetv）不能包含观察者任务的执行间隔。

（我们将观察者任务可能出现的时间列的问题转化为消除时间列的过程）换言之，观察者任务 τo 可能出现的时间列不是真实到达列 δv。为此，更容易将问题看作是消除这些观察者任务的时间列的过程。如果我们把得到的 τo 的执行间隔放在调度梯形图上，去掉相应的时间列，那么，必须至少存在一个长度等于或大于 bcetv 的连续时间列的间隔，而这个间隔最终没有被删除。这些时间列是 τv 的真实到达时间的候选者，**也可能存在由于其他高优先级任务而没有删除的时间列。**然而，由于其他任务具有不同的到达周期（或零星任务的随机到达），这些时间列往往是分散的（与[δv，δv+bcetv)相比），并且随着观察者任务的执行间隔的增加，这些时

间列将被消除。在实践中，我们的结果表明，这个过程是有效的，并且在攻击持续时间为 5·LCM（po，pv）后基本稳定（见第 VII-B1 节）。

例 1：考虑一个 RTS，由四个任务组成的 Γ = {τ1，τo，τv，τ4}。为了简单起见，在这个例子中，我们假设所有任务都是周期性的（尽管我们的分析也可以处理周期性的、零星的和混合的系统）。任务参数见下表（左侧）。注意 prii> prij 意味着 τi 比 τj 高。因此，任务 τ1 的优先级最低，任务 τ4 的优先级最高，τv 的优先级高于 τo。让最大重建持续时间 λ 为 1，攻击开始时间为 0（因此，在本例中，av 等于 δv）。假设攻击者已经执行了第一步/算法一段时间，下表列出了观察者任务的重建执行间隔。（这里重构时间间隔的方法在附录 A 中的算法 1 中）

	p_i	e_i	a_i	pri_i
τ_1	15	1	3	1
τ_o	10	2	0	2
τ_v	8	2	1	3
τ_4	6	1	4	4

Reconstructed Execution Intervals
[0,1)
[12,13)
[20,21)
[30,31)
[43,44)

注意，由于 τ1 的优先级低于观察者任务 τo，因此它不影响 τo 的执行。然后，我们将重构的执行间隔放置在宽度等于被攻击者任务周期 Pv 的调度梯形图中。这个操作如图 4 所示。为了更好地理解调度梯形图在分析被攻击者任务行为方面的有效性，我们在附录图 12 中的梯形图上绘制原始的、完整的调度梯形图，以便读者更好地理解它。这使我们深入了解 τo 的执行间隔与被攻击者任务的执行间隔之间的关系。

（然后我们得到了候选时间列）从图 4 中的调度梯形图中，我们删除观察者任务的执行间隔所占用的时间列。结果显示在图 4 的底部，剩下的是候选时间列，其中包含我们要提取的被攻击者的真实到达时间。这些间隔被传递到最后一步，以推断被攻击者任务的初始偏移量/到达时间。

D、初始偏移量 av 和未来到达时间 δv 的推断

我们现在进入最后一步——推断被攻击者任务的未来到达时间——我们最初的目标。但是，首先，我们需要计算被攻击者任务的初始偏移量。我们从**上一步得到的是一组候选时间列的间隔，**其中包含被攻击者任务的真正到达列。**间隔的数量取决于收集到的执行间隔的数量以及其他优先级更高的任务引入的“噪音”（因此，无法保证最终可以消除所有错误的时间列）**。然而，根据我们的实验和定理 1，错误时间列往往是分散的。因此，我们以可能包含被攻击者任务的真正到达列的最大间隔作为推断。然后我们**选择这个区间的起点作为推断的真正到达列，用 δv^表示**。虽然这种策略并不总是能保证成功，但我们的评估（第六节中的案例研究和第七节中的性能评估）表明，我们能够实现较高的推断精度。所需的**初始偏移量（用 av^表示）可以导出为 av^ =（t+ δv^）mod Pv**，其中 t 代表调度梯形图的开始时间。（初始偏移量就是第一个被攻击者任务开始执行的时间距离攻击开始的时间的距离）

例 2：从例 1 获得的间隔与时间列[1，3），[5，6）和[7，8）相对应。根据该算法，选取最大区间[1, 3）。这样一个间隔的起始点被认为是被攻击者任务的真正到达列的推论，它变成 δ v^= 1。在本例中，真实到达列为 δ v= 1。因此，正确推断初始偏移量的真正到达列的算法就可以相应地导出。

现在，**被攻击者任务的未来到达时间可以很容易地由 av^ + Pv · T，T∈N来计算**，其中 av^ 是 τ v 的推断初始偏移量，Pv 是 τ v 的周期，T 是期望到达数。**计算的结果是第 T 个被攻击者任务到达的确切时间。**

4、算法分析

A、攻击能力分析

（如何确定观察者任务对于被攻击者任务的攻击能力？本节中的分析集中于观察者任务是一个周期性的任务）我们现在讨论如何确定观察者任务对于被攻击者任务的**攻击能力或有效性**。也就是说，在这种情况下，观察者任务是否可以删除所有错误的时间列，从而正确地推断出被攻击者任务的到达信息。请注意，本节中的分析集中于观察者任务是一个周期性任务，因为正如我们在第 II-D 节中所提到的，它对攻击者是一个更严格的限制条件。在相同的目标系统下，**零星观察者任务的性能可能更好**，因为零星任务自然具有更灵活的到达时间，而这些到达时间仅受其最小到达间隔时间的限制。

（我们首先分析观察者任务和被攻击者任务之间的执行间隔之间的关系）一个能保证从 τ v 的调度梯形图中删除所有错误时间列的保守条件是：**当观察者任务的执行间隔出现在所有可能的时间列中时**。因此，我们首先分析**观察者任务的执行**与**被攻击者任务的执行**之间的关系。当 τ v 和 τ o 都是周期性任务时，我们有以下观察和定理：

观察 2：在调度梯形图中，在每个观察者任务的到达时间列（即调度执行）和真正到达列之间的偏移量在它们的最小公倍数 LCM（po，pv）之后重复。（即观察者任务的到达时间和被攻击者任务的到达时间之间的偏移量）

定理 2：如果给定的观察者任务 τ o 和被攻击者任务 τ v 满足不等式 eo> = GCD（po，pv），则可调度的 τ o 的执行保证出现在 τ v 的调度梯形图的所有时间列中。

证明：根据观察 2，观察者任务执行的时间列偏移量在每个 LCM（po，pv）上重复。因此，上述条件（即 τ o 的调度执行出现在所有可能的时间列中）可以用不等式来描述：

$\frac{LCM(p_o,p_v)}{p_o} \cdot e_o \geq p_v$，然后，利用 $LCM(p_o,p_v) = \frac{p_o p_v}{GCD(p_o,p_v)}$（这是最小公倍数的计算方式），我们可以得到一个条件来保证观察者任务能够检测到被攻击者任务的到达，即 eo> = GCD（po，pv）。（重复就意味着 τ o 出现在所有可能的时间列）

由定理 2 可知，在 LCM（po，pv）中，当 eo > GCD（po，pv）时，观察者任务的调度执行可以多次出现在某些时间列中，冗余覆盖意味着相比于与较小的 eo 到 GCD（po，pv）的比值，τ o 将更频繁地访问错误时间列（这样可以尽量减少真实到达列的候选列数量）。相反，如果 eo < GCD（po，pv），则不是所有的假时间列都能被观察者任务覆盖和检查。为了更好地描述观察者

任务的覆盖，我们进一步定义了一个**覆盖率**，该覆盖率描述了观察者任务对被攻击者任务的能力，如下所示

定义 1：（覆盖率）覆盖率，用 C（τ o，τ v）表示，计算公式如下：

$$\mathbb{C}(\tau_o, \tau_v) = \frac{e_o}{GCD(p_o, p_v)} \qquad (1)$$

覆盖率可以粗略地解释为观察者任务可能出现在调度梯形图中的时间列的比例。如果观察者任务可以覆盖所有 Pv 时间列，则 C（τ o，τ v）> = 1。否则 0< = C（τ o，τ v）< 1。

B、选择最大重建持续时间 λ

回想一下，最大重建持续时间 λ 用于限制观察者任务运行攻击算法所占用的执行时间（在一个周期内）。由于攻击者希望保持秘密，并尽量减少对原始功能的破坏，因此最好使用尽可能小的 λ 值。剩余的执行时间 eo- λ 可被攻击者用来交付 τ o的原始功能，同时在捕获执行数据方面取得进展。基于这一思想， λ 可由以下公式确定：

$$\lambda = \begin{cases} GCD(p_o, p_v) & \text{if } \mathbb{C}(\tau_o, \tau_v) \geq 1 \\ e_o & \text{otherwise} \end{cases} \qquad (2)$$

在 C（ τ o； τ v）> = 1 的情况下，观察者任务具有冗余覆盖，由于一次性覆盖足以让观察者任务检查所有 Pv 时间列，因此附加覆盖可以用于其他目的，否则，如果（C（ τ o， τ v）< 1），攻击者可能需要利用其所有计算资源进行攻击。

5、评估指标

为了评估 ScheduLeak，我们定义了以下两个指标：（我们推断的对象是 τ v 的初始偏移量）

（1）**推断成功率**：如果攻击者能够准确地推断出被攻击者任务的初始偏移量，我们成功的定义一个推断（回想一下第 III-D 节，一旦我们知道了初始偏移量，我们就可以很容易地预测到未来到达实例），因此，推断的结果要么是真的，要么是假的，推断成功率是针对一组任务集的给定测试条件的真/假结果的平均值。

（2）**推断精度比**：在推断不精确的情况下，我们定义一个度量来评估推断精确的程度（即，我们与实际值的接近程度）。在本文中，推断目标是被攻击者任务的初始偏移量。我们首先通过 $\epsilon = |\hat{a_v} - a_v|$ 计算推断量与真实值之间的差距，其中 av 是被攻击者任务的初始偏移量，av^ 是推断的初始偏移量，然后我们定义推断精度比：

定义 2：（推断精度比）推断精度比用 $\mathbb{I}_v^o$ 表示，计算方式为：

$$\mathbb{I}_v^o = \begin{cases} 1 - \frac{p_v - \epsilon}{\frac{p_v}{2}} & \text{if } \epsilon > \frac{p_v}{2} \\ 1 - \frac{\epsilon}{\frac{p_v}{2}} & \text{otherwise} \end{cases} \quad (3)$$

（第一个条件是推断量和真实值之间的差距超过 pv/2）

该推断精度比是在 0< = $\mathbb{I}_v^o$ < = 1 范围内的一个真实值，这让我们可以知道推断值与真实偏移量之间的差距，$\mathbb{I}_v^o$ = 1 表示初始偏移量 av 的推断是完全正确的。

6、基于实际平台的案例研究评估

在评估文中介绍的算法的性能之前，本节中我们首先致力于评估真实平台上此类算法的可行性，ScheduLeak 算法实施在两个有着实时调度能力的操作系统上：（i）实时 Linux[14]和（ii）FreeRTOS[15]。在接下来的内容中，提出了两个攻击案例，它们从提出的算法获得的信息中受益，并利用这些信息来实现其主要攻击目标。这些攻击案例的演示视频可以在 https://scheduleak.github.io/中找到。

A、覆写控制信号

攻击场景和目标：大量的实时控制系统封装了控制执行器的子系统。例如，现代自动系统中，引擎控制单元（ECU）控制电子节气门体（ETB）中的阀，以实现电子节气门控制（ETC）。在大多数无人驾驶飞机中，飞行控制器通过电子速度控制器（ESC）来管理电动机的转速。在这些系统中，诸如 PWM信号之类的致动信号会定期更新，以确保对控制任务的快速一致响应。

（希望隐秘的覆盖控制信号，因此了解控制信号更新的确切时间极为重要）让我们考虑一个攻击者，它希望能够隐秘地覆盖此类系统中的控制权，目的是通过引起不良行为，甚至在很短的时间范围内接管系统控制权，从而实现不良控制。 为此，攻击者将其作为恶意任务进入系统，并尝试覆盖控制信号。在这种情况下，过度覆盖控制信号的暴力策略将不起作用，因为其高攻击开销可能导致其他实时任务错过其截止日期并导致系统崩溃。在这种情况下，了解控制信号更新的确切时间并在正确的时间对其进行覆盖可以使攻击者以较低的开销有效地进行控制。

实现方式：我们在自定义漫游车上实施此攻击，其控制系统由 Raspberry Pi 3 Model B 板构建，封装了各种惯性传感器的 Navio2 模块板连接到 Raspberry Pi 板上。 该系统运行带有 Ardupilot [16]自动驾驶仪软件套件（远程和自治控制社区中最流行的开源代码堆栈之一）的实时 Linux（即 Raspbian，内核 4.9.45，带有 PREEMPT RT 补丁）。 它由一组实时和非实时任务组成，以执行与控制相关的工作，例如刷新 GPS 坐标，解码远程控制命令，执行 PID 计算和更新输出信号。 任务之一是以 20ms 的周期定期更新 PWM值，以用于转向和节流，更新通过串行外围设备接口（SPI）

发送到 Navio2 模块，该模块将 PWM信号输出到伺服器和 ESC。图 5（a）显示了正常情况下工作的 PWM输出信道。

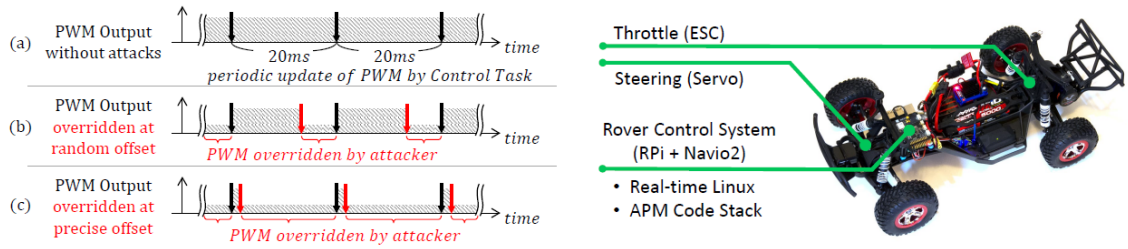


Figure 5: An illustration of PWM channels on a rover system. (a) The PWM outputs are updated periodically by a $50Hz$ task. (b) A naive attack issuing the PWM updates at random instants may not be effective. (c) By carefully issuing the PWM updates right after the original updates, the PWM outputs can be overridden.

图 5：漫游车系统上的 PWM信道示意图。（a）PWM输出由 50Hz 任务定期更新。（b）在随机时刻发出 PWM更新的简单攻击可能无效。（c）通过在原始更新之后立即发布 PWM更新，可以覆盖 PWM输出。

在这种攻击中，我们假设攻击者可以访问低优先级的周期性任务（作为观察者任务，Po = 50ms）和非实时 Linux 进程（用于发起 PWM覆盖攻击），攻击者的最终目标是覆盖被攻击者任务（即 50Hz 周期性任务）更新的控制信号。**在此处的实施工作中，观察者任务使用系统调用 Clock gettime（）从 CLOCK_MONOTONIC 获取时钟计数（以纳秒为单位）**。运行 ScheduLeak 算法时，时间测量值进一步舍入为微秒，因为 Ardupilot 中的所有任务参数均为 1us 的倍数。一旦确定了被攻击者任务的初始偏移量，攻击者就会参与非实时进程，以通过被攻击者任务使用的同一接口发布 PWM更新。请注意，由于设计使 Raspberry Pi 板和 Navio2 模块之间缺乏身份验证，因此的可能这样做的。该过程通过使用时钟 gettime（）来跟踪时间，并且每当它确定已超过被攻击者任务的到达时刻时即发出两次 PWM更新（一次用于转向，一次用于油门）（即 t-av^mod pv >= 0，其中 t 是当前时间，a v^ 是推断的被攻击者任务的初始偏移量。该过程在两个 PWM更新之间保持空闲，以减少攻击占用空间。

攻击结果：图 5（b）和 5（c）显示可以使用与 PWM硬件不同的值来覆盖 PWM输出。 但是，如果没有确切的调度信息，攻击者只能以**随机选择**的初始偏移量周期性地发送更新（图 5（b）），随机初始偏移量可以是 20ms 周期内的任意点，根据我们的实验，只有初始偏移在 av 和 av+8.3ms 之间的范围内的攻击才能产生对转向和油门控件的有效覆盖，结果，攻击者有 41.5% 的机会选择有效的初始偏移并导致有效的攻击。

另一方面，攻击者在发起 ScheduLeak 攻击并确切了解被攻击者任务何时到达后，可以在原始更新之后立即发出 PWM更新以覆盖 PWM输出（图 5（c））。在这种情况下，攻击者首先在观察者任务中运行 ScheduLeak 算法，并在 1 秒钟的时间内得出 0.9985 的推断精度比（用于推断受害任务的初始偏移）。这样，攻击者就可以在非实时进程中准确推断被攻击者任务的初始偏移量，从而发起 PWM覆盖攻击。请注意，在被攻击者任务的作业完成之后（因此在原始 PWM更新之后），攻击者尝试在被攻击者任务的到达瞬间进行 PWM更新，因为非实时进程的优先级低于被攻击者任务，因此，攻击者可以接管转向和油门的控制，通过探测 PWM信号，我们观察到被覆盖的 PWM

信号在 85%的时间内处于活动状态。结果，我们看到漫游车不再响应原始控制，相反，漫游车是由攻击者的命令驱动的，由于攻击者的任务在两次 PWM更新之间保持闲置状态，因此占用的 CPU利用率低至 2. 6%。

B、推断系统行为

攻击场景和目标：让我们考虑一个执行监视任务的无人机系统，当飞越感兴趣的位置时，它会捕获高分辨率图像。在这种情况下，**攻击者的目标是提取无人机所针对的位置**。该策略是监视无人机上的监视摄像机何时切换到正在处理高分辨率图像的执行模式。**这可以通过利用缓存定时侧信道攻击来衡量处理图像的任务的粗粒度内存使用行为来完成，此任务对高速缓存的使用率很高，这表示正在处理高分辨率图像，否则，它将使用较少的缓存。**但是，**对缓存进行随机采样会导致数据嘈杂（例如图 6（a））**（并且常常是无用的），因为系统中还存在其他也使用缓存的任务。 相比之下，知道任务调度何时运行将使攻击者可以非常接近目标任务的执行程度进行攻击和探测攻击[17]，[18]。（我们需要知道摄像机什么时候切换到处理高分辨率图像的执行模式）（同样，当我们检测到有处理高分辨率图像任务的时候，可以采取我提出的防御手段。）

实现方式：这种攻击是通过运行 FreeRTOS的 Zedboard在硬件在环（ HIL）仿真中实现的，该 FreeRTOS可以模拟无人机上的控制系统。该系统由一个图像处理任务（被攻击者任务，pv = 33ms）以 30Hz的速度处理照片以及其他四个任务（攻击者不知道）-全部以周期性方式运行。 当无人机到达预载列表上的目标位置时，被攻击者任务会处理大量数据，其他任务占用不同的内存量。在这种情况下，我们假设**攻击者作为最低优先级的周期性任务**进入系统，Po = 40ms。攻击者使用此任务来运行 ScheduLeak 算法和执行缓存定时侧信道攻击。攻击者的最终目标是观察被攻击者任务的内存使用情况并了解系统行为。

攻击结果：首先，我们考虑不使用 ScheduLeak 攻击的攻击者。攻击者在每个周期中发起缓存定时侧信道攻击，以尝试估计被攻击者任务的缓存使用情况。如图 6（a）所示，这会产生许多高速缓存探测器，并且很难区分被攻击者任务与其他任务的高速缓存使用情况，由于无法识别被攻击者任务的使用模式，因此导致攻击未成功。

接下来，让我们考虑攻击者利用 ScheduLeak 攻击的情况。在这种情况下，算法在 3· LCM（pq pv）的窗口（即 4秒）内产生 0. 99的推断精度比。然后，攻击者可以在被攻击者任务被执行之前和之后立即发起缓存定时侧信道攻击，并跳过那些无关紧要的时刻。图 6（b）显示了针对被攻击者任务的精确缓存探测的结果。**我们发现，该攻击极大地降低了其他任务引起的噪音（省略了 96. 9%的缓存探针），并且能够准确地识别受害任务的内存使用行为。**结果，可以从图 6（b）所示的尖峰（红色三角形点）中识别出四个相机活动实例。如图 6（c）所示，当与攻击者通过其他手段获得的飞行路线信息结合在一起时，就可以推断出高度关注的位置。（利用 ScheduLeak 攻击的情况，极大的降低了其他任务引起的干扰，使攻击者很轻易地能够找到感兴趣的位置）

（总之：不使用 ScheduLeak 算法的情况下，观察者任务和被攻击者任务以外的许多任务都会引起干扰，它们的执行会导致攻击者无法分清什么时候被攻击者任务会执行。在使用了

SchedLeak 算法之后，我们可以排除其他任务的干扰，将目光准确放在观察者任务和被攻击者任务身上，我们知道所有观察者任务的运行时间，于是我们可以推断出被攻击者任务会在哪些时间段运行，这样，我们就知道了哪些位置是原系统高度感兴趣的位置。从而达到了攻击的目的。）

7、性能评价与设计空间探索

A、评估配置

我们使用随机生成的合成任务集来测试我们的算法，该任务集按 CPU 利用率[0.001+0.01*x，0.1+0.1*x]分组，其中 0 <= x <= 9，每个利用率组由 6 个子组组成，**这些子组有着固定数量的任务（5，7，9，11，13，15）**，每个子组包含一百个任务集，在每个任务集中，50%的任务都生成为周期性任务（每个子组分别为 3，4，5，6，7，8 个周期性任务），其余的任务为零星任务，任务周期从[100，1000]中随机抽取，我们假设攻击者可以访问解析度为 1 的系统时间，任务初始偏移量从[0，Pi]中随机选择，在零星任务的情况下，**我们将生成的任务周期作为最小的间隔时间，**利用 **RM单调速率**分配算法分配任务优先级，我们只选择那些可调度的任务集。

（观察者任务总是定为最低优先级的周期性任务，被攻击者任务考虑 priv=1 和 priv=|hp(τo)|两种情况，覆盖率 C 设置为 C>=1，保证算法能产生有效推断）观察者任务和被攻击者任务在生成任务集时被分配，在仿真中，我们考虑了一个周期性观察者任务，因为它代表了对手最坏的攻击情况，如第 IV-A 节所述，由于只有具有较高优先级的任务才会影响观察结果，因此我们跳过了低优先级任务 lp(τo)的生成。因此，观察者任务在这些生成的任务集中总是具有最低优先级(即 prio=1)。对于被攻击者任务，考虑两个条件: (i) priv=2 和(ii) priv=|hp(τo)|，这是为了测试两个边界条件。更进一步，当生成任务集的时候我们设置了覆盖率 C(τo，τv)≥1(除了评估覆盖率的影响)，这是为了评估算法能否真正产生自信的推断，因为攻击者有着理论上的攻击能力的保证，(例如，在全面覆盖所有 pv 时间列，根据定理 2)。每个章节 IV-B 的最大持续时间 λ 设置为 $\lambda = GCD(p_o, p_v)$.

为了改变任务的执行时间并在到达时间间隔中添加抖动(对于零星任务)，我们分别使用正态分布和泊松分布。注意，泊松分布用于到达时间间隔的变化，因为在这样的分布模型中，每次发生的概率(即，零星任务的每次到达)是独立的。首先，生成一个可调度任务集(使用前面提到的参数)。然后，对于任务数量 τi，平均执行时间由 woeti·80%计算。接下来，我们对任务 ii 拟合一个正态分布 $\mathcal{N}(\mu, \sigma^2)$，取平均值为 woeti·80%，得到累积概率 P(X≤woeti)为 99.99%的标准差标准。这样的正态分布会产生变化，95%的执行时间都在±10%·woeti 范围内。为了确保任务集保持可调度性，如果它超过 WCET，我们将最大修改执行时间调整为等于 WCET。对于零星任务，平均到达时间间隔按 pi·120%计算。在模拟过程中，我们使用 pi·120%作为平均值的泊松分布来产生不同的到达时间。类似地，为了不违反零星任务给定的最小到达时间，我们在修改后的到达时间低于 pi 时重新生成它。（这一段没怎么看懂，也就是数据结构的设计而已）

B、结果

（1）**攻击持续时间：**我们的第一个目标是了解攻击持续时间的影响。回想一下，调度梯形图的覆盖范围重复每个 LCM(po，pv)(观察 2)。因此，我们使用 LCM(po，pv)作为评估算法的时间单

位。以 Ardupilot 软件为例，任何实时任务(即 AP HAL 线程)对的最大 LCM为 20ms。当 LCM(po, pv)随着系统的不同而变化时，这让我们对 LCM(po, pv)的规模有了更深入的了解。在本实验中，我们按照 VII-A 部分的说明生成任务集，并对每个任务集运行 SchedLeak 算法，其持续时间固定为 10 · LCM(po, pv)。图 7 显示了本次实验的结果。在图 7 中，推断精度比的每个点为给定攻击持续时间下 12000 个任务集的单个推断精度比的平均值。结果表明，攻击持续时间越长，算法的成功率和准确率就越高。这是因为较长的攻击时间意味着观察者任务将重新构建更多的执行时间间隔。另一方面，在 5 · LCM(po, pv)后，成功率和正确率均出现平缓期，成功率和正确率分别高于 97%和 0.99。试验结果表明，所提出的算法可以在非常短的时间内产生精确的推断，而通过长时间运行而获得的额外收益则微不足道，因此，在下面的其余实验中，我们将评估算法的持续时间为 10 · LCM（Po, Pv）。

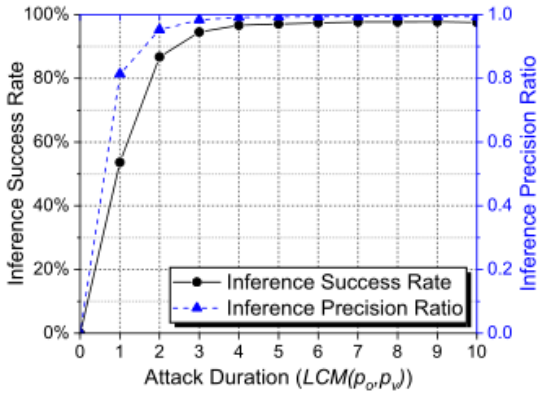


图 7:攻击持续时间变化的结果。结果表明，攻击持续时间越长，推断成功率越高，推断精度越高。这些点只是作为一个引线连接起来。

（2）**任务的数量和任务集的利用率：**图 10 显示了一个 3D 图，其中显示了任务数量和任务利用率子组的每个组合的平均推断精度比。结果表明:(i)推断精度比随着任务集中任务数量的增加而降低，(ii)推断精度比随着任务集利用率的增加而提高。当一个任务集中有 15 个任务，利用率组为[0.001, 0.1]时，推断准确率最差——这是本实验中数量任务和利用率的边界条件。任务数量的影响是直接的，因为在 hp(τo)中有更多的任务意味着更频繁地抢占，这使得观察者任务很难消除错误的时间列。对于任务集利用率的影响，低利用率值意味着任务的执行时间较小，并且在调度中存在很多差距。因此，观察者可能得到许多小而分散的间隔。由于我们让算法选择最大的间隔来推断真实到达列，多个小间隔是有问题的——算法很难选择包含真实到达时间的正确间隔。因此，错误是复杂的。

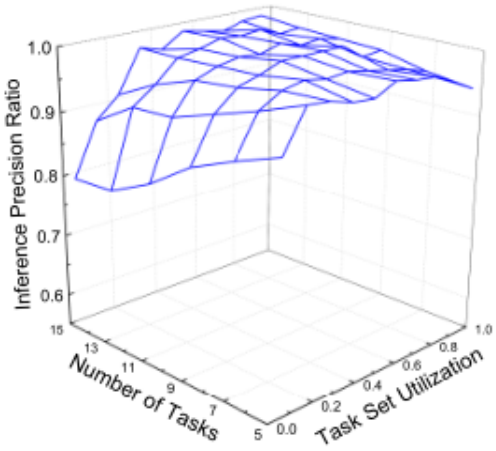
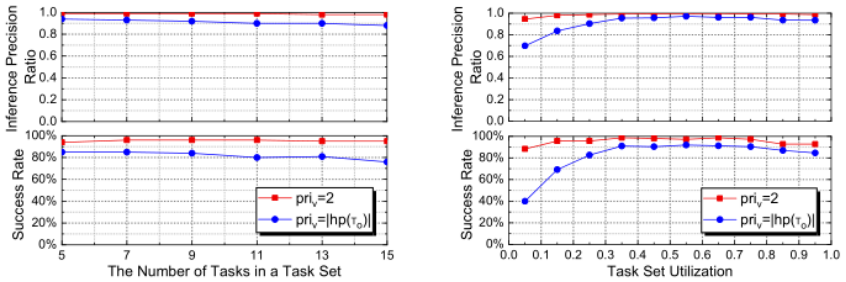


图 10:任务数量和任务集利用率的影响。实验结果表明，该算法在任务数量少、任务集利用率高的情况下具有较好的性能。

（3）**被攻击者任务的优先次序：**我们分析了任务集中被攻击者任务的优先级对任务的影响。在第Ⅶ- a 节中，我们考虑了被攻击者任务位置的两个边界条件:(i) priv= 2 和(ii) priv= |hp(τ o)|。图 11(a)和图 11(b)给出了两种情况下的实验结果。图 11(a)显示，图 10 中的巨大下降(随着任务数量的增加)主要是由条件 priv= |hp(τ o)|造成的。图 11(b)也显示了类似的迹象，即图 10 中低利用率组的下降是条件 priv= |hp(τ o)|造成的。值得注意的是，由于我们使用速率-单调算法来分配优先级，priv= 2 意味着 τ v 有一个大的周期，因此可能有更多的执行时间。它有利于算法，因为我们选择最大的区间，以作出推断的最后一步。



(a) Grouped by the number of tasks. (b) Grouped by task set utilization.

图 11:被攻击者任务在任务集中位置的影响。说明优先级较高的被攻击者任务使得算法很难做出正确的推断。这个结果在任务集中存在不同数量的任务以及不同的任务集利用率。此外，具有低任务集利用率的高优先级被攻击者任务会降低推断性能。这解释了图 10 中巨大的下降。

（4）**我们研究了零星任务和周期性任务混合的影响**。我们在一个任务集中生成 0%、25%、50%、75%和 100%零星任务的任务集。任务集中的其余任务是周期性任务。比较图 8 所示的所有周期性任务的结果和所有零星任务的结果，我们发现算法在处理零星任务时表现得更好。随着零星任务比例的增加，呈现上升趋势。然而，性能变化不到 1%，这是微妙的。因此，我们的推断算法对于系统中零星/周期性任务的实际混合相当不可知。

（5）**覆盖率和最大重建持续时间：**以上实验表明，当 C(τo, τv) ≥ 1，λ = GCD(Po, Pv)，算法的精度和成功率均达到一定的水平。然而，攻击者可能面临一个被攻击者系统，其中 C(τo, τv) < 1。也就是说，观察者任务的执行不能保证出现在所有 pv 时间列中。为了评估算法在这种情况下的性能，我们生成了任务集，其中 0 < C < 1，（因此 λ = ∞），并且运行算法的持续时间为 10 · LCM(Po, Pv)，在本实验中，任务集按覆盖率从[0.001 + 0.1· x, 0.1 + 0.1· x]开始分组，其中 0≤ x≤ 9。图 9 显示了结果。这表明，当覆盖率较低时，攻击者可能无法完全推断出被攻击者任务的初始偏移量，然而，这些算法在某些情况下仍然可以成功，因为定理 1在低覆盖率情况下仍然有效。当观察者覆盖时间列([0.401, 0.5]组)约为一半时，其成功率为 59.9%，平均推断准确率为 0.819。当观察者任务观察到更多的时间列时，精度和成功率就会提高，这是因为较高的覆盖率使算法有更高的机会捕获真实到达列并删除其他列。因此，**推断成功率与覆盖率成正比。**

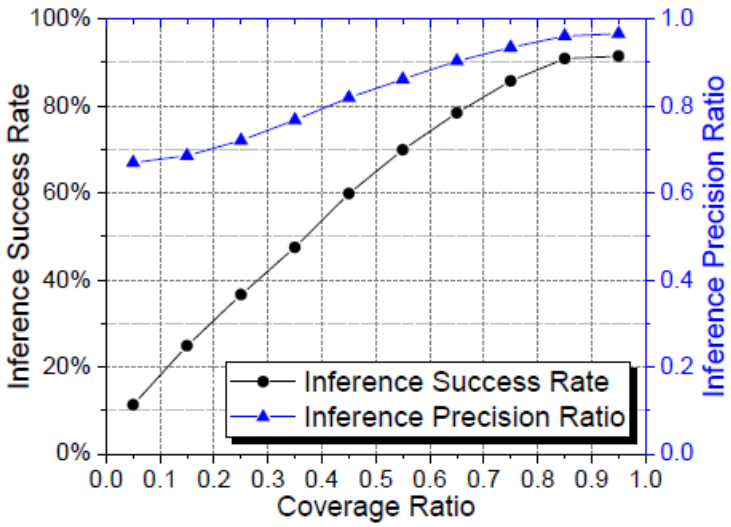


图 9：当 C（τo, τv）< 1时算法的性能。圆点和三角形分别代表推断成功率和推断精度比。

8、讨论-潜在的防御策略

为了防御提出的攻击算法，一种策略是通过调整任务参数，在任何低优先级的任务和关键实时任务之间施加低覆盖率，这降低了攻击者的可观察性/能力(根据 VII-B5 节的结果)。此外，精心设计和使用一个和谐任务集也可能降低 ScheduLeak 的推断精度（？？？），因为它会在算法的最后一步创建多个候选对象。但是，任务参数中的任何更改都必须同时满足实时需求和所需的性能。因此，更改任务参数可能并不总是适用于实时系统，特别是已经部署的遗留系统。

由于所提出的算法依赖于被攻击者任务的重复模式，一个潜在的对策是干扰系统调度的周期性。然而，由于实时任务的实时约束，该度量将不是微不足道的，一个粗心的解决方案很容易导致一些实时任务错过最后期限，从而导致系统故障。Yoon 等人提出的速率单调调度器的随机化协议[19]是移除 RTS 调度器侧信道的一个很好的尝试。然而，他们的工作并不适用于

我们的情况，因为他们只关注具有所有周期性任务的系统，而我们的工作对于具有周期性和零星任务的系统是可行的(这是大多数实时控制系统的情况)。因此，一个有效的解决方案需要考虑同时覆盖这两种任务类型。**（此处显然已经给出该作者未来的研究方向）**

9、相关工作

侧信道信息泄漏问题在文献中已经得到了很好的研究。例如，它已经表明，基于缓存的测信道对于信息泄漏方面来说是无价的[20]，[21]，[17]，[18]。随着多租户公共云的出现，基于缓存的侧信道和它们的防御再次受到关注(如[22]、[23]、[24]、[25])。其他类型的侧信道，如差动功率分析[26]，电磁和频率分析[27]，[28]也被研究。我们这里的重点是实时系统中的调度器侧信道。

也有一些通过调度器的信息流的工作，研究了两个任务利用秘密信道泄漏私有信息的问题。Volp等[1]、[29]研究了实时任务不同优先级之间的隐蔽渠道，并提出了避免这种隐蔽渠道的解决方案，量化调度程序中信息泄漏的方法也被研究，之前的研究主要集中在一些调度程序中的隐信道上，而我们的研究重点是实时调度程序中的新型侧信道，在这些侧信道中，一个没有特权的低优先级任务可以推断高优先级实时任务的执行时间行为。另外，与依赖主动抢占实时任务的隐蔽信道相比，我们工作中的侧信道没有违反任何实时约束，观察者任务只观察自己的行为。

将安全集成到实时调度器中是一个正在发展的研究领域。Mohan等人将实时系统安全需求作为一组调度约束，并引入了一种改进的固定优先级调度算法，该算法将安全级别集成到调度决策中。Pellizzoni等人[11]将上述方案扩展到更一般的任务模型中，并提出了确定任务可抢占性的最优优先级分配方法。一些研究人员还专注于实时系统的防御技术(如[33]、[34]、[35]、[36]、[37]、[38]、[39])。**但是，这些解决方案不能保护系统免受 ScheduLeak 攻击。**

最密切相关的解决方案是采用随机化技术来混淆调度表。Yoon等人[19]为抢占式固定优先级调度器引入了一种随机化协议，该调度器仅对(完全)周期性任务工作。Kruger等人[40]在此基础上提出了一种用于时间触发系统的在线作业随机化算法。然而，这些解决方案并不适用于大多数实时系统，在这些系统中，抢占式的固定优先级调度器同时支持周期性和零星的实时任务。这使得那些系统仍然很容易受到我们的调度表泄露攻击。

10、结论

在具有实时特性的控制系统（包括网络物理系统）中成功的安全漏洞可能会产生灾难性的后果。在许多这样的系统中，掌握关键任务的精确时间信息可能对对手有利。我们在这篇论文中的工作展示了如何以一种隐蔽的方式来捕捉这个调度时间信息，即，在不被检测到或对原始

系统造成任何干扰的情况下。这种系统的设计者现在需要认识到这种攻击媒介，并在设计系统时包括能够阻止潜在入侵者的对策。最终的结果是，实时系统可以对总体安全威胁更加稳健。

附录

算法 1 以观察者任务的最坏情况执行时间 e0、当前作业实例的剩余执行时间 e0′ 和最大重构持续时间 λ 作为输入。它输出检测到的执行间隔的开始时间 tbegin 和结束时间 tend 以及当前实例 e0 ′ 的更新后的剩余执行时间。第 1-4 行初始化算法使用的变量。具体地说，当算法达到当前实例的给定最大重建持续时间 λ 时，第 3 行计算时间点（停止条件）。第 5-9 行用于检测抢占并检查当前时间是否超过计算的停止时间点。这些行通过从全局计时器（即系统计时器）读取当前时间并将其与前一个循环的时间进行比较，来跟踪每个循环之间的时间差。如果时间差超过我们预期的时间（循环的执行时间），我们就知道发生了抢占（即一个或多个高优先级任务被执行）。当检测到抢占或当前时间超过计算的停止时间点时，循环将退出。第 10-12 行确定重建执行间隔的结束时间。如果循环因抢占而退出，则抢占之前的最后一个时间点将作为该执行间隔的结束时间（第 11 行）。否则，不会检测到抢占，所有的持续时间都会用完，最后一个时间点作为执行间隔的结束时间（第 13 行）。第 15 行为下一次调用更新当前作业的剩余执行时间。第 16 行返回重建的执行间隔（它的开始时间 tstart 和结束时间 tend）以及更新后的剩余执行时间。