

IUSFUZZ: 智能无人系统控制器固件的覆盖率引导模糊测试

唐宁宁¹, 冯 博^{2†}, 万毕南¹, 罗 梦², 杨晓春³, 王丽娜¹

1. 空天信息安全与可信计算教育部重点实验室, 武汉大学 网络安全学院, 湖北 武汉 430072;
2. 区块链与数据安全全国重点实验室, 浙江大学, 浙江 杭州 310058;
3. 东北大学 软件学院, 辽宁 沈阳 110819

收稿日期: 2025-08-09 † 通信联系人 E-mail: bo.feng@zju.edu.cn

基金项目: 国家重点研发计划(2024YFB4709000); 国家自然科学基金(62472380)

第一作者: 唐宁宁, 男, 硕士生, 现从事信息安全方面的研究。E-mail: ningningtang@whu.edu.cn

摘 要: 现有覆盖率引导模糊测试工具在智能无人系统中存在效率衰减与状态不可控等问题。针对上述问题, 本文选取 PX4 开源无人系统作为研究对象, 首先分析了其在模糊测试中出现性能退化的三大核心成因: 异步线程执行、内外部信号交互与时空状态残留, 随后基于此三点核心成因设计了强制同步、消息过滤、时间变量替换、拓展改进后的选择性插桩等针对性缓解机制, 生成测试专用固件, 并构建了面向智能无人系统的模糊测试框架 IUSFUZZ。对比主流模糊测试工具 AFL++, IUSFUZZ 稳定性提升 16.5 个百分点, 挖掘相同漏洞, AFL++ 所需时长为 IUSFUZZ 的 7.5 倍。IUSFUZZ 成功挖掘出一个通过网络信道触发的零日漏洞并获得了 CVE 编号。

关 键 词: 智能无人系统; 漏洞检测; 模糊测试; 固件安全

中图分类号: TP368

文献标志码: A

文章编号: 1671-8836(2026)04-0579-13

IUSFUZZ: Coverage Guided Fuzzing for Intelligent Unmanned System Controller Firmware

TANG Ningning¹, FENG Bo^{2†}, WAN Binan¹, LUO Meng², YANG Xiaochun³, WANG Lina¹

1. Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, Hubei, China;
2. The State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou 310058, Zhejiang, China;
3. Software College, Northeastern University, Shenyang 110819, Liaoning, China

Abstract: Existing coverage-guided fuzzing tools in intelligent unmanned systems face issues such as efficiency degradation and uncontrollable states. To address this problem, this paper selects the PX4 open-source unmanned system as the research object. Firstly, it analyzes three core causes of performance degradation in fuzzing: asynchronous thread execution, internal and external signal interaction, and spatiotemporal state residue. Based on these three core causes, targeted mitigation mechanisms are designed, including forced synchronization, message filtering, time variable substitution and improved selective instrumentation, to generate test-specific firmware. Additionally, a fuzzing framework for intelligent unmanned systems, IUSFUZZ, is constructed. Compared with the mainstream fuzzing tool AFL++, IUSFUZZ has improved stability by 16.5 percentage points. To discover the same vulnerabilities, AFL++ requires 7.5 times as long as IUSFUZZ. IUSFUZZ successfully uncovered a zero-day vulnerability triggered through a network channel and obtained a CVE number.

Key words: IUS(Intelligent Unmanned System); vulnerability detection; fuzzing; firmware security

引用格式: 唐宁宁, 冯博, 万毕南, 等. IUSFUZZ: 智能无人系统控制器固件的覆盖率引导模糊测试[J]. 武汉大学学报(理学版), 2026, 72(4): 579-591. DOI: 10.14188/j.1671-8836.2025.0167.

TANG Ningning, FENG Bo, WAN Binan, *et al.* IUSFUZZ: Coverage Guided Fuzzing for Intelligent Unmanned System Controller Firmware [J]. *J. Wuhan Univ. (Nat Sci Ed)*, 2026, 72(4): 579-591. DOI: 10.14188/j.1671-8836.2025.0167(Ch).

0 引言

随着智能无人系统(如无人机、无人车及服务机器人等)在工业自动化、物流配送、环境监测及消费娱乐等领域的广泛应用^[1-3],其对外部环境的高度感知能力与自主决策机制使其能够执行日益复杂的任务,但这种高度互联与功能集成的特性也显著扩大了系统的攻击面,因为智能无人系统需要首先对各类数据(如传感器输入、用户指令和自身状态等)进行解析和判断才能确定后续执行的动作,错误或恶意的数据都可能影响智能无人系统的决策过程,引发安全问题。如通过伪造胎压监测系统传感器数据触发整数溢出漏洞,即可在无线范围内对无人车进行远程任意代码执行(CVE-2025-2082);又如通过向目标机器人发送精心构造的网络数据包,即可使机器人进入安全停止状态,从而导致生产中断(CVE-2024-6157)。无人系统控制器承担着状态感知、姿态解算、导航控制及执行机构驱动等关键功能,是保障系统安全稳定运行的核心模块,该模块需实时处理来自地面控制站(Ground Control Station, GCS)、遥控器、GPS模块及多种机载传感器的多源异构输入信号,形成了多维度、高并发的通信接口体系,从而引入了诸多潜在入侵途径。值得注意的是,当前主流开源无人系统广泛采用 MAVLink 协议(Micro Air Vehicle Link)进行跨组件通信。该协议设计之初以轻量化和低延迟为核心目标,缺乏身份认证、数据完整性校验和加密传输等基本安全机制^[4],导致其极易遭受中间人攻击(Man-in-the-Middle Attack)、数据注入、指令篡改及远程控制劫持等恶意行为。更为严峻的是,研究表明,PX4-Autopilot^[5]和 Ardupilot^[6]等开源无人系统占全球民用及行业级市场份额超过 30%^[7],这意味着一旦核心协议或软件存在可利用漏洞,将可能引发大规模的安全风险事件。

在此背景下,针对智能无人系统开展系统性的安全测试,尤其是通过网络接口实施模糊测试(Fuzzing),已成为发现潜在漏洞、提升系统鲁棒性的重要技术路径^[8]。模糊测试技术涵盖随机测试、变异驱动测试、规则驱动测试和覆盖率引导测试等技术路线,其中基于覆盖率引导的模糊测试方法因具备定向探索代码路径、提升测试深度的优势,已成为当前自动化漏洞挖掘领域的主流技术方案。然而,现有基于覆盖率引导的模糊测试方法在应用于智能无人系统时面临显著挑战。由于智能无人系统普遍具有高度耦合的软件架构、强状态依赖性、实时性约束以及多

线程并发执行等特性,测试过程常出现稳定性下降、状态不可控等问题,导致测试效率衰减、路径探索效率低下,表现出明显的性能退化现象^[9-10]。该现象直接削弱了覆盖率引导模糊测试对系统隐藏漏洞的挖掘能力,使得基于该技术的自动化漏洞检测难以发挥预期效能。最终造成智能无人系统的安全评估面临覆盖面不足、深度不够的困境,严重制约了覆盖率引导模糊测试在智能无人系统安全评估中的实际应用,亟需针对性的方法创新与框架优化。

为应对上述挑战,本文深入剖析智能无人系统在覆盖率引导模糊测试过程中所表现出的问题,进而提出一种面向智能无人系统的模糊测试框架 IUSFUZZ,旨在有效辅助研究人员发现以控制软件为代表的智能无人系统中的潜在安全漏洞,提升系统的整体安全性。本研究的主要贡献如下:

- 1) 系统性地分析了覆盖率引导模糊测试在应用于智能无人系统时出现性能退化现象的三大核心成因:异步线程执行、内外部信号交互与时空状态残留,为后续测试方法的优化提供了理论支撑与技术依据;
- 2) 针对上述不同类型的不稳定性根源,设计并实现了针对性的缓解机制,包括强制同步、消息过滤、时间变量替换与拓展改进后的选择性插桩策略,最终集成构建了 IUSFUZZ。

1 相关工作

模糊测试是一种通过注入大量异常或随机输入来发现安全漏洞的技术,在传统软件领域广泛应用^[11]。

在测试接口层面,现有模糊测试主要聚焦于 3 个维度:应用级、场景级和网络级^[8]。应用级模糊测试适用于近距离、可直接访问系统内部应用层接口的场景,核心针对无人系统的本地输入验证逻辑,通过注入异常参数(如无效命令参数、无效传感器值或超出范围的配置参数)直接触发漏洞。典型研究如 APFuzzer^[12]通过修改智能无人系统的配置参数实现漏洞挖掘;RVFuzzer^[13]对配置参数和一些环境因素(如风速、风向等)进行模糊测试,以查找输入验证漏洞。场景级模糊测试侧重在复杂环境条件下验证系统鲁棒性,适用于自动驾驶、户外巡检等需与物理环境深度交互的无人系统场景,核心围绕系统的感知、决策模块,通过扰动环境参数模拟极端工况,发现逻辑缺陷与响应异常。如 AV-FUZZER^[14]通过扰动交通路况参数对自动驾驶决策系统和感知模块进行测试,以发现其在复杂或异常场景下的逻辑缺陷、响应延迟或决策错误;DPFuzzer^[15]是针对无人系统路径

规划漏洞检测的工作,但尚未形成系统化的场景建模方法。网络级模糊测试聚焦于智能无人系统的各类特有通信协议(如 MAVLink 协议)适用于远程、非接触式测试场景,核心针对无人系统与地面站、其他设备间的通信链路,无需直接接触目标系统,尝试进行跨距离渗透测试。其典型研究包括:Domin 等^[16]通过变异 FTP 命令参数和字段值生成测试用例,4D-Fuzzer^[17]基于 MAVLink 数据包结构进行字节级随机组合生成,而 PGFUZZ^[18]将 MAVLink 接口中的配置参数、用户命令和环境因素确定为核心输入要素。由于无人系统的通信链路一旦被攻破,攻击者将直接掌控系统控制权,所以针对无人系统的网络攻击通常更频繁,危害性也更大。鉴于此,本文选择通过网络级测试接口对智能无人系统开展覆盖率引导模糊测试以提升其针对网络攻击的防御能力。

值得注意的是,大部分针对智能无人系统的模糊测试依托开源模拟器进行,其模拟方法主要分为软件在环(Software-in-the-loop, SITL)和硬件在环(Hardware-in-the-loop, HITL)两类^[19]。鉴于软件

在环方法能够完整模拟智能无人系统(包括控制算法、传感器及环境动力学模型)且具备成本低、安全性高等优势,当前绝大多数研究采用该方法。然而,HITL 在硬件真实交互上的优势使其在特定场景下仍不可替代。由于 SITL 环境已能覆盖本研究绝大多数测试场景,且为进一步提升测试效率、增强研究可拓展性,本研究最终选择在 SITL 环境中开展相关实验与验证工作。

2 稳定性下降成因分析

本文针对智能无人系统进行基于覆盖率引导的模糊测试研究,以 PX4 开源无人系统作为研究对象,以 SITL 仿真系统作为研究平台,以 MAVLink 协议栈作为测试接口,使用主流模糊测试工具 AFL++^[20]作为开发框架。本文利用覆盖率引导机制引导测试用例生成,针对 MAVLink 协议的固定字段进行变异(如表 1 和图 1 所示),将其发送给经过 AFL++ 插桩的 PX4 控制程序,以期触发潜在程序缺陷。

表 1 MAVLink V2 消息字段含义

Table 1 Meanings of MAVLink V2 message fields

数据段	取值范围	含义
STX	0XFD	数据包头特征符,标志着一个新数据包的开始
LEN	0X0-0XFF	声明 payload 段长度
INC FLAGS	0X0-0X1	不兼容标志,0x01 表示数据包已签名,数据包包含签名后缀
CMP FLAGS	0X0-0X1	兼容标志,如果不被理解就可以被忽视
SEQ	0X0-0XFF	数据包序列号,用于防止数据包丢失
SYS ID	0X1-0XFF	无人系统 ID,用于在同一网络中识别不同系统发送的消息
COMP ID	0X0-0XFF	组件 ID,用于区分系统组件(如惯性控制单元、相机等)
MSG ID	3 bytes	消息类型,用于正确解码 payload
PAYLOAD	0-255 bytes	载荷信息
CHECKSUM	2 bytes	校验和
SIGNATURE	13 bytes	签名,用于防篡改,是一个可选项

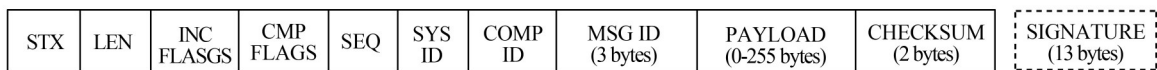


图 1 MAVLink 消息格式

Fig. 1 MAVLink message format

然而,AFL++ 的传统测试流程存在显著缺陷:控制程序每次执行均需经历启动-解析-关闭的完整周期,导致测试效率仅能达到 0.1 次/s,为突破这一瓶颈,本文又引入了 AFL++ 的持续交互(Persistent)模式^[21],该模式通过保持控制程序后台持续运行,在循环结构中实现测试用例的连续发送与接收,显著提升了测试吞吐量。值得注意的是,该模式下模糊测试器(Fuzzer)统计覆盖率的周期即为测试驱动执

行一次测试循环的起止时间,在每个周期内统计一次路径覆盖的情况,同时使用“稳定性(stability)”指标量化测试结果是否可靠,计算公式如(1)式所示。

$$\text{stability} = (1 - \Delta\text{Cov}) \times 100\% \quad (1)$$

其中, ΔCov 为覆盖率偏差,计算公式如(2)式所示:

$$\Delta\text{Cov} = \frac{\Delta E}{E_{\text{total}}} \quad (2)$$

其中, ΔE 表示重复执行同一测试用例时变化的程

序路径数量; E_{total} 为已探知的全部程序路径数量。

然而,无人控制系统的复杂特性导致测试稳定性极低(接近于零),这直接引发两方面问题:其一,程序崩溃(Crash)现象难以稳定复现,给漏洞验证带来显著挑战;其二,模糊测试器可能错误地将无效测试用例判定为有效输入,最终导致覆盖率引导机制失效,测试策略退化为黑盒模糊测试。为此,本文通过系统性观察与分析,揭示了导致稳定性下降的三大核心原因。

2.1 异步线程执行

PX4 控制程序为提升执行效率,采用多线程异步架构设计,根据功能不同可以大致划分为 3 类:

- 1) 消息接收线程:负责 MAVLink 协议等通信数据的实时接收。
- 2) 传感器数据处理线程:完成传感器原始数据采集与状态计算。
- 3) 控制执行线程:实现控制指令的实时解析与执行。

由于线程间存在解耦特性,Fuzzer 无需等待当前测试用例执行结束即可发送下一测试用例,输入测试用例的时序与程序执行时序存在显著偏差,覆盖率统计周期也与程序执行周期产生时间解耦,这种非确定性导致覆盖率统计结果呈随机变化趋势,同时在系统出现 Crash 时因 Fuzzer 早已将真正的漏洞数据包丢弃而导致无法复现。

2.2 内外部信号交互

PX4 控制程序为保证自身状态实时更新,需要持续处理多源信号输入,其中影响测试稳定性的关键信号类型包括:

- 1) 传感器数据:如姿态、速度、加速度等实时物理量。
- 2) 异步线程广播信号:异步线程通过频繁的信号交互实现数据更新,其中只有部分信号受 Fuzzer 输入的 MAVLink 数据包控制和影响,其余的线程间通信信号都会成为影响模糊测试稳定性的随机变量。
- 3) 无关的 MAVLink 数据包:除了接收 Fuzzer 输入的 MAVLink 数据包外,控制程序同时还会接收来自地面站的心跳数据包、来自外部无人系统的通信请求等无关 MAVLink 数据包。

这些内、外部信号未被 Fuzzer 输入的 MAV-Link 数据包完全约束,导致在重复执行相同测试用例时,控制程序因为无关信号的随机波动进入不同的路径分支,最终引发覆盖率的不可预测性变化。

2.3 时空状态残留

模糊测试采用 Persistent 模式,控制程序启动后便不再关闭,因操作过程中的时序相关特性,需维

护前一时刻的各类状态信息,这些残留信息会干扰后续测试用例执行结果的可预测性,导致程序覆盖率发生变化:

- 1) 时序残留:控制程序使用实时操作系统,程序执行路径与时间残留状态密切相关。如 GPS 数据包超时校验、温度传感器“冷浸泡”等待期以及陀螺仪固定积分周期等。
- 2) 空间残留:由于控制程序在 MCU 上运行,会尽可能减少资源消耗,程序执行路径受到空间残留状态影响,如对于已分配的传感器数据堆栈,程序倾向于使用已有内存而不是重新分配。
- 3) 状态残留:控制程序维护大量全局变量,初始值在改变后不可复位,而局部变量却在循环过程中不断刷新,数据生命周期的不同导致控制程序出现状态残留累计问题,使得程序首次运行与后续执行路径产生差异。如记录当前无人系统是否初次运行、状态是否更新等。

2.4 量化分析建模

为进一步量化各成因最终对测试效率衰减的影响,本文将构建简易量化关联模型,为后续研究拓展提供支撑。这些干扰因素通过引发覆盖率偏差(即重复执行相同测试用例时,程序变化路径占已探知全部路径的比例),间接导致测试效率下降,若其产生的覆盖率偏差越大,则测试效率越低,测试效率衰减程度呈指数级上升。基于这一关系,本文用(3)式量化测试效率衰减程度 λ_E :

$$\lambda_E = e^{-k \times \Delta \text{Cov}} - e^{-k} \quad (3)$$

其中, k 为比例系数。

该问题可转化为评估各干扰因素分别导致的覆盖率偏差。针对 3 类核心干扰因素,本文分别设计量化公式,计算其各自引发的覆盖率偏差:

- 1) 异步线程执行干扰因素 ΔCov_1 : 计算公式如(4)式所示:

$$\Delta \text{Cov}_1 = \frac{\alpha \times \sum_{i=1}^T E_i}{E_{\text{total}}}, \alpha \in [0, 1] \quad (4)$$

其中, T 为线程数; α 为解耦系数($\alpha = 0$ 表示完全同步, $\alpha = 1$ 表示完全解耦); E_i 为单个线程中的路径数量。该公式表明:待测系统中线程数量越多,线程间解耦程度越高,其引发的覆盖率偏差就越大,测试效率衰减越显著。

- 2) 内外部信号交互干扰因素 ΔCov_2 , 计算公式如(5)式所示:

$$\Delta \text{Cov}_2 = \frac{\beta \times \sum_{i=1}^T P_i}{E_{\text{total}}}, \beta \in [0, 1] \quad (5)$$

其中, β 为通信信号条件分支占比 ($\beta=0$ 表示不存在内外部通信, $\beta=1$ 表示所有条件分支均受内外部通信信号影响); P_i 为单个线程包含的分支路径数量。该公式表明: 待测系统中内外部通信信号在系统分支判断中的占比越高, 覆盖率偏差越大, 测试效率衰减越明显。

3) 时空状态残留干扰因素 ΔCov_3 , 计算公式如(6)式所示:

$$\Delta\text{Cov}_3 = \frac{E_\gamma + E_\delta + \sum_{i=1}^S B_i}{E_{\text{total}}} \quad (6)$$

其中, E_γ 为受时序残留影响的分支路径数量; E_δ 为受空间残留影响的分支路径数量; S 为全局状态数量; B_i 为每个全局状态影响的分支路径数量。该公式表明: 待测系统实时性要求越高, 内存空间复用率越高, 全局状态越多时, 其引发的覆盖率偏差就越大, 测试效率衰减越严重。

综合上述分析, 最终得到各干扰因素对测试效率衰减的影响量化公式如(7)式所示:

$$\begin{cases} \lambda_{E1} = e^{-k \times \frac{\alpha \times \sum_{i=1}^T E_i}{E_{\text{total}}}} - e^{-k}, & \alpha \in [0, 1] \\ \lambda_{E2} = e^{-k \times \frac{\beta \times \sum_{i=1}^T P_i}{E_{\text{total}}}} - e^{-k}, & \beta \in [0, 1] \\ \lambda_{E3} = e^{-k \times \frac{E_\gamma + E_\delta + \sum_{i=1}^S B_i}{E_{\text{total}}}} - e^{-k} \end{cases} \quad (7)$$

其中, $\lambda_{E1}, \lambda_{E2}, \lambda_{E3}$ 分别表示异步线程执行、内外部信号交互和时空状态残留导致的测试效率衰减影响量化结果。

3 本文方案

基于上述发现, 本文分别针对导致稳定性下降的不同原因, 提出了针对性的改进方案, 同时基于 AFL++ 模糊测试框架的核心组件, 实现了测试框架 IUSFUZZ。本文主要工作包括构建源码修改模块中的线程强制同步、消息过滤、时间变量替换功能以及改进编译模块中的选择性插桩功能, 整体架构及执行流程如图2所示。

3.1 基于阻塞更新的线程同步

PX4 控制程序内部通过 uORB 机制 (Micro Object Request Broker)^[22] 实现异步线程间的消息传递功能, 该机制采用发布-订阅的设计范式构建异步通信模型: 当数据生产者需要共享特定信息时, 将通过主题 (Topic) 进行广播式数据发布。这些 Topic 作为语义化的信息通道, 通常承载着如姿态估计、控制指令等具有明确

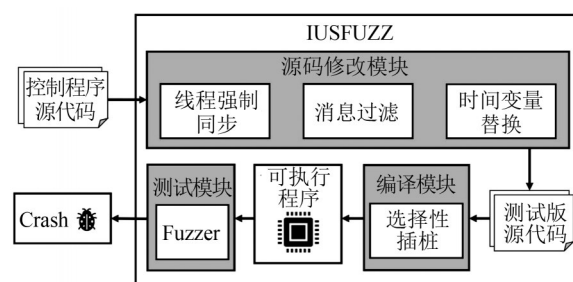


图2 IUSFUZZ 工作流程

Fig. 2 IUSFUZZ workflow

物理意义的数据结构。在订阅端, 数据消费者可以通过两种方式获取最新数据, 其一为轮询机制, 即订阅者以自主设定的频率主动查询 Topic 数据, 这些订阅者通常拥有自己的独立堆栈, 承担着较重的计算任务; 其二为事件触发机制, 当新数据写入 Topic 时, 系统将立即唤醒处于休眠状态的订阅线程, 这些线程受到工作队列的管理, 共享并复用同一个堆栈空间和线程优先级, 无法进行 sleep 操作或阻塞式输入输出。如图3所示, uORB 架构允许单个线程同时承担发布者与订阅者的双重角色, 支持单个线程对多个 Topic 进行并发订阅与发布, 但不会进行消息接收确认回复。

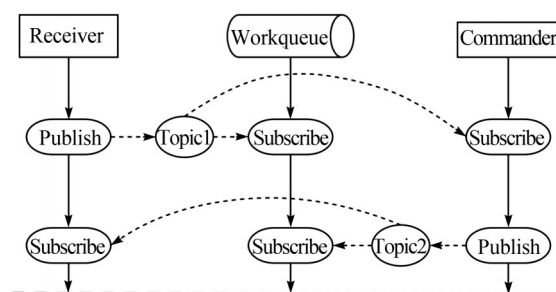


图3 uORB 通信机制

Fig. 3 uORB communication mechanism

为解决异步线程执行问题, 本文实现了基于阻塞更新的同步化处理, 本文重点针对 PX4 控制程序中具有独立堆栈的 5 个核心线程展开改造: 消息接收 (Receiver)、主控指挥 (Commander)、日志记录 (Logger)、导航控制 (Navigator) 和数据管理 (Data-man)。为实现模糊测试输入与系统执行流程的精确同步, 本文采取了以下分层同步策略。

3.1.1 Fuzzer 输入与控制程序消息接收周期同步

如前文所述, Fuzzer 在 Persistent 模式下从输入测试用例后开始统计覆盖率, 并在一轮测试循环结束后进行覆盖率回收和稳定性计算, 所以首先需要将 Fuzzer 的输入线程与控制程序消息接收线程进行同步, 使得 Fuzzer 可以统计控制程序接收消息并执行相应命令后的路径覆盖情况。

由于 Fuzzer 的输入线程与 PX4 控制程序的消

息接收线程分别处于独立进程空间,为确保两个线程的严格同步,本文在两个线程循环代码的首尾部建立双向管道连接,利用管道的阻塞式读写强制要求输入线程必须等待接收线程执行结束才能输入下一个测试用例,接收线程必须等待输入线程提供测试用例才能执行消息解析与广播。

3.1.2 控制程序消息接收与命令执行周期同步

由于消息接收线程会将 MAVLink 数据包解析成 uORB Topic 并进行广播,本文在此基础上设计了基于 uORB 阻塞式订阅更新的强制同步机制:新增两个专用同步 Topic: PubSend 与 PubReturn,分别用于标记线程执行周期的起始与终止,并在每个目标线程(Commander、Logger、Navigator、Dataman)的主循环入口处插入 PubSend Topic 的阻塞式订阅指令,强制要求线程在进入新执行周期前等待消息接收线程的同步信号,在各线程主循环出口处插入 PubReturn Topic 的发布指令,并在消息接收线程中相应位置添加对 PubReturn Topic 的阻塞式订阅,形成完整的执行周期同步闭环。需要注意的是,在执行单个测试用例时,本文仅对相应的目标线程进行强制同步以保证模糊测试吞吐量(每秒可执行测试用例数量)。

通过上述分层同步架构,本文实现了“Fuzzer 输入测试用例-控制程序接收消息-控制程序执行命令-Fuzzer 完成覆盖率统计”的强制同步,确保了覆盖率统计周期与系统执行状态的严格对应关系。

3.2 消息过滤

针对内外部信号交互问题,本研究对 PX4 控制程序进行了系统性改造:针对外界 MAVLink 数据包干扰问题,本文关闭 PX4 控制程序中除接收 Fuzzer 输入外的所有 MAVLink 接收线程,构建单向数据注入通道,有效隔离外部干扰源;针对来自异步线程的广播

Topic,本文设计了一种消息追踪-过滤机制以隔绝其对测试稳定性的影响,该机制主要包含两个部分:

1) 标记传播系统:在所有 uORB 消息结构中增加布尔型成员变量 fromreceiver。当 Fuzzer 输入的消息被接收线程处理时,系统自动将该标记设为 True。订阅该 Topic 的线程在后续广播时,其消息对象的 fromreceiver 标记将继承该状态,形成完整的传播路径标记链。

2) 广播过滤机制:通过修改 uORB 底层广播函数,在消息分发前强制执行双重验证:

- a. 检查 fromreceiver 标记是否为 True。
- b. 当无法获取消息具体内容时,通过程序调用回溯技术获取发布者的线程标识符进行二次验证。

系统仅允许通过上述验证的消息进行跨线程分发,其余非 Fuzzer 关联的消息将被阻断。PX4 官方定义 Topic 数量可达 200 个,为确保覆盖尽可能多的 Topic,本文开发了针对 PX4 源码的自动化修改工具,流程如算法 1 所示。

该机制最终实现效果如图 4 所示,Fuzzer 通过线程 Main 向消息接收线程 Receiver 发送测试用例,Receiver 接收测试用例并解析成对应的消息后,会将消息的 fromreceiver 标记设为 True 并执行广播操作;此时 uORB 底层的广播过滤机制将启动双重验证流程,首先检查 fromreceiver 标记是否为 True,若无法获取消息具体内容,则通过程序调用回溯机制获取发布者的线程标识符,以此完成二次验证。仅当双重验证通过时,Commander 线程才能接收到 Receiver 线程广播的消息。后续 Commander 线程尝试进行消息广播时,同样需要经过 uORB 底层的双重验证,通过后该消息才能被其他线程(如 thread3 和 thread4)接收。反观 thread5 线程,其广播请求因未满足验证

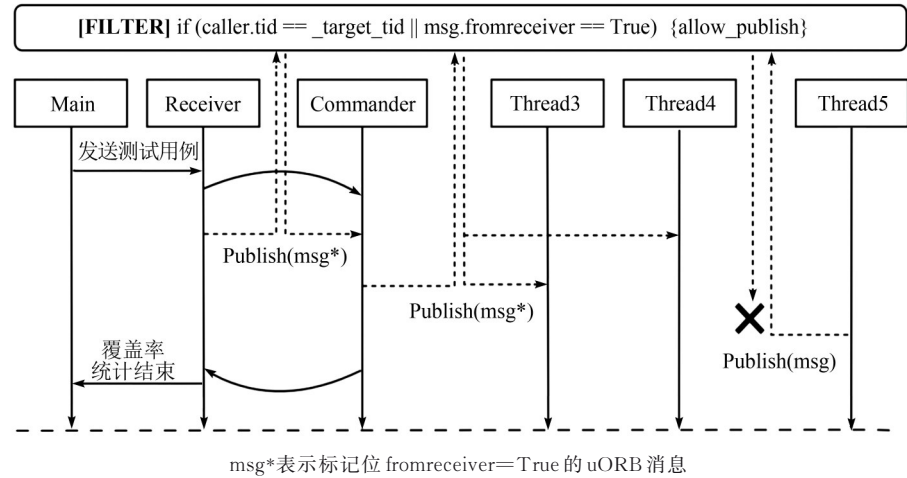


图 4 消息追踪-过滤机制

Fig. 4 Message tracking - filtering mechanism

算法1 消息追踪-过滤代码注入算法

输入: 待修改的源代码文件列表、函数调用图

输出: 修改后的源代码

```
1 for File in Source_File_List: #对每个源代码文件
2     for Function in File: #对文件中的每个函数
3         if HavePublishBehavior(): #检查是否存在发布
           行为
4             if HaveUpdateBehavior(): #检查是否存在
           检查 Topic 更新行为
5                 LocateUpdateVariable() #定位更新 Topic
           变量
6                 InjectCode() #注入消息追踪-过滤代码
7             else if CallerUpdate(Function, CG): #检查
           当前函数调用者是否存在 Topic 更新行为
8                 InjectCodeUpdate() #调用者中注入消息
           过滤代码(检查更新 Topic 标记位)
9                 InjectCodePublish() #当前函数注入消息
           追踪代码(标记发布消息)
10            else:
11                Vars = SelectAllVariable() #筛选所有
           可访问变量
12                Ps = CalVarsPosibility(Vars) #计算各
           变量为 Topic 变量的置信度
13                if HaveHighOnes(Ps): #如果存在高概率
           可访问变量
14                    InjectCode() #注入消息追踪-过滤代码
```

条件而被拦截,无法被其他线程获取。同时由于强制同步机制的约束,Fuzzer必须等待Commander线程执行完毕,并逐步将执行结果反馈给Receiver线程和Main线程后才算完成单轮测试用例的执行流程,进而开始统计覆盖率等相关信息。

3.3 时间变量替换

由于空间残留和状态残留问题过于复杂,且为了平衡PX4控制程序的可测试性和可用性,本文仅提出了时间变量替换方法以解决时序残留问题。经过对PX4控制程序源代码的深入分析,本文发现其大量路径分支采用了基于时间的条件判断机制,相关统计结果如表2所示。

为解决此类时间相关变量对测试稳定性带来的负面影响,本文将表2中统计的全部时间判定条件统一替换为可以被Fuzzer直接控制的测试参数。此外,针对代码中其他与程序运行时长相关的变量,本研究也采取了相同的参数化改造措施。

此次重构通过消除时间依赖性变量的不可控性,有效提升了系统测试的可重复性和覆盖率。改

表2 时间相关分支条件

Table 2 Time related branch conditions

数量	代码类型
1	if (hrt_absolute_time() > CONST_TIME)
2	if (hrt_absolute_time() - last_time > CONST_TIME)
3	if (hrt_elapsed_time(last_time) > CONST_TIME)
4	if (now_time - last_time > CONST_TIME)
5	if (now_time > last_time + CONST_TIME)

注:hrt_absolute_time()函数用于获取系统自启动以来的累计运行时间;hrt_elapsed_time(last_time)函数则返回自指定时间戳last_time至当前时刻的时间间隔。

造过程中特别注意保持原有逻辑结构的完整性,确保时间参数化替换不会引入新的功能缺陷。所有变量替换均遵循“可控性-保真度”平衡原则,在保证测试场景可控性的同时,最大限度保留原始时间约束的工程语义。

3.4 拓展改进后的选择性插桩技术

AFL++原生工具提供多种插桩模式,可在程序编译的不同环节(预处理、编译、汇编、链接)执行插桩任务。其中,链接时插桩具有显著优势:针对PX4这类包含多个子模块的大型程序进行插桩时,其覆盖范围相较于其他模式更加全面,且原生工具支持选择性插桩能力^[23],为本文编译模块的功能开发提供了核心技术支撑。

然而,AFL++原生工具选择性插桩技术存在设计局限性:仅支持函数粒度的选择性插桩,无法对程序基本块进行筛选,进而无法完全屏蔽特定代码引发的执行路径干扰,这一问题反而将链接时插桩的全面覆盖优势转化为劣势,导致引入更多的覆盖率噪声,并进一步降低测试稳定性。为此,本文以AFL++选择性插桩技术为基础,对其核心插桩逻辑进行针对性拓展与改进,将插桩粒度从函数级调整至基本块级:首先,依托LLVM IR层的程序结构分析能力,遍历提取每个基本块的关键特征;其次,构建多维度基本块指纹,融合所属文件名、函数名、模块名、后继节点数量及前驱节点数量五大特征,形成唯一标识以精准定位目标基本块;最后,基于该指纹实现精细化筛选逻辑,可按需排除工作队列线程、冗余传感器信号处理等无关代码对应的基本块。通过上述改进,最终构建形成了IUSFUZZ框架中的编译模块。该改进既保留了AFL++链接时插桩的覆盖范围优势,又弥补了其插桩粒度粗糙的不足,可有效屏蔽工作队列线程异步执行带来的干扰、随机变化的传感器信号等各类覆盖率噪声,有效提升测试稳定性和测试效率。

4 实验设计与评估

4.1 实验设置

4.1.1 实验环境

本研究在SITL仿真环境中对PX4控制程序实施模糊测试,并将发现的潜在漏洞实例在真实无人系统中进行验证。实验环境采用的关键软硬件配置参数如表3所示,其中LLVM用于构建IUSFUZZ和AFL++编译器。

表 3 软硬件配置

Table 3 Software and hardware configuration	
关键设备与软件	参数
CPU中央处理器	AMD EPYC 7763 64-Core
内存	2 TB
操作系统	Ubuntu 22. 04
AFL++版本	4. 32a
LLVM版本	18. 1. 8
PX4控制程序版本	1. 16. 0-beta1
无人系统控制器版本	Pixhawk-6c

4.1.2 实验设计

为评估IUSFUZZ的运行效果,本文设计实验以系统回答以下关键问题:Q1 本文提出的改进方案是否能够有效提升PX4控制程序模糊测试的稳定性(4.2节);Q2 针对修改后PX4控制程序的模糊测试可行性及效率如何(4.3节);Q3 各干扰因素对测试效率衰减的影响权重如何分布(4.4节);Q4 强制同步是否会影响模糊测试吞吐量以及是否会影响整体漏洞挖掘效率(4.5节)。

4.2 消融实验

为了系统性阐述各个策略是否有效,对稳定性的提升效果如何,本文设计了多组实验,对比分析不同策略与不同策略组合对稳定性提升的综合影响。

首先比较仅使用单个改进策略对测试过程稳定性的提升效果。本文将未采用任何改进策略的AFL++的测试数据(Persistent)作为基准,其稳定性随执行次数的增加而降低并最终稳定在0.2%(图5),此时覆盖率引导机制已经完全失效,测试过程完全退化为黑盒模糊测试。然后分别使用拓展改进后的选择性插桩(Select),强制同步(Sync),消息过滤(Filter),时间变量替换(Replace)4种改进策略对其进行优化,实验结果如图5所示,Replace策略展现出最优的稳定性提升效果,因为其直接修改了分支判断条件,使其受Fuzzer输入控制;Sync策略的优化效果次之,证明其有效锚定了覆盖率统计周期与程序执行之间的时序关系;相比之下,Select

策略和Filter策略改进效果较为有限,因为它们依然受到异步程序执行带来的严重干扰。

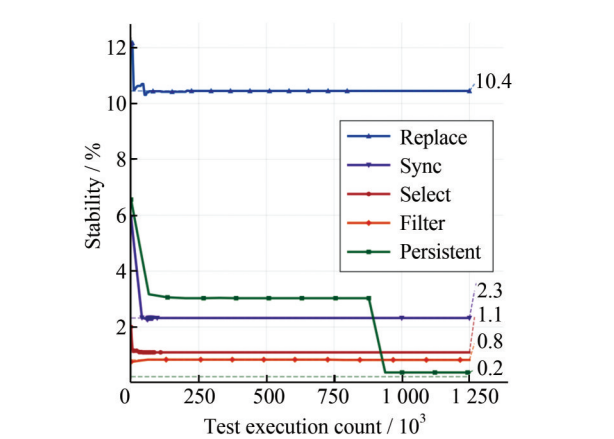


图 5 单个策略的测试稳定性随执行测试用例数量变化
Fig. 5 Variation of stability with test execution count based on single strategy

鉴于异步线程执行会对Fuzzer的覆盖率统计过程产生严重干扰,为进一步评判各优化策略的实际效果,本文在强制同步策略的基础上,与其他策略进行组合验证。图6中的实验数据表明,强制同步与拓展改进后的选择性插桩的协同应用(Sync+Select)可以让测试稳定性提升至60%及以上,但由于多线程环境的无关信号干扰及程序残留时空状态影响而持续下降,最终稳定在6.0%左右。进一步引入消息过滤策略(Sync+Select+Filter),测试稳定性显著提升至13.6%,证明其可以有效屏蔽非目标信号引发的路径覆盖率干扰,但控制程序内部的残留信息仍对测试稳定性构成制约。最终追加时间变量替换策略,使用所有优化策略后(Use All),测试的初始阶段可实现100%的稳定性,但受空间状态与全局变量状

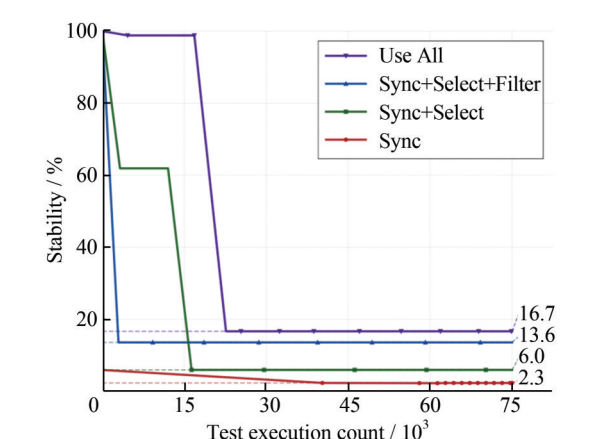


图 6 基于同步策略组合的测试稳定性随执行测试用例数量变化
Fig. 6 Variation of stability with test execution count based on synchronous strategy combinations

态残留的影响,测试稳定性逐步收敛至约 16.7% 的稳定区间,值得注意的是,此时 IUSFUZZ 已经具备正确捕获触发 Crash 的异常测试用例的能力。除此之外,相较于 AFL++ 测试(Persistent)的稳定性而言(0.2%),使用所有优化策略(Use All)的稳定性提升 16.5 个百分点,稳定性提升效果十分明显。

IUSFUZZ 使用的策略组合(Use ALL)的最终稳定性,不论是相较于单个策略(Filter、Select、Sync、Replace),还是同步策略组合,都有着较大幅度的提升。值得注意的是,Select 策略虽然也可以通过屏蔽所有路径干扰以提升稳定性,但 Fuzzer 将因为无法回收路径执行信息,导致覆盖率引导机制失效并失去测试意义。而通过本文多维度优化策略的合理协同,既充分发挥了单个策略的独特优势,又互补了彼此的短板,最终显著提升了 PX4 控制程序在模糊测试中的执行稳定性,实现了程序可测试性、测试速度与测试效率上的平衡,为基于覆盖率引导的模糊测试在智能无人系统的应用提供了重要改进依据。

4.3 漏洞挖掘效率分析

为验证 IUSFUZZ 的漏洞挖掘效率,本文将其与针对 PX4 控制程序 MAVLink 接口的黑盒模糊测试工具 4DFuzzer 和主流模糊测试工具 AFL++ 进行了对比实验。值得注意的是,为了实时捕获触发拒绝服务(Deny of Service, DoS)类漏洞的测试用例,本文在 PX4 控制程序的关机请求函数中插入了一段检测代码,即控制程序在解析 MAVLink 数据包后尝试进行关机时,检测代码就会通知内存检测工具(如 Address Sanitizer)触发警报,以便测试工具及时进行响应,防止其因控制程序的延时命令执行机制而导致关键数据丢失。测试漏洞的信息如下:

- 1) CVE-2025-57427: 由 battery_status 类(msgid=147)的 MAVLink 数据包触发,该漏洞位于 Commander 线程,具体漏洞成因与触发机制详见第 5 节;
- 2) CVE-2024-40427: 由 serial_control 类(msgid=126)的 MAVLink 数据包触发,位于 Receiver 线程,因未实施数组边界校验引发栈溢出漏洞。

实验设置要求所有工具都在指定 msgid 后执行漏洞挖掘,本文通过 10 次独立测试统计了挖掘漏洞

的平均时间与使用的测试用例数量并以此来展示各个工具漏洞挖掘的效率,挖掘漏洞使用的平均时间和测试用例数量越少,则说明该工具的漏洞挖掘效率越高。需要注意的是,由于各工具测试吞吐量不同,所以其在相同时间内挖掘漏洞所使用的测试用例数量可能也各不相同。

实验结果如表 4 所示,根据实验数据可以发现:

1) IUSFUZZ 整体漏洞挖掘效率优于 AFL++ 与 4DFuzzer: 在 CVE-2025-57427 漏洞挖掘结果中, IUSFUZZ 的漏洞挖掘能力表现显著优于 4DFuzzer 和 AFL++, 4DFuzzer 和 AFL++ 挖掘 DoS 漏洞所需时间分别是 IUSFUZZ 的 10 倍和 7.5 倍,使用的测试用例数量是 IUSFUZZ 的 20 倍和 227 倍,进一步验证了 IUSFUZZ 在跨线程漏洞触发场景中的优势。在 CVE-2024-40427 漏洞挖掘结果中, IUSFUZZ 测试用例数量略多于 4DFuzzer,但二者在漏洞触发时长上未呈现出显著差异,因为 Receiver 线程直接对 Fuzzer 输入的 MAVLink 数据包进行解析处理,执行路径尚不深入,覆盖率引导机制的优势发挥并不明显;相比之下, AFL++ 漏洞挖掘所用时间和使用测试用例数量呈随机变化趋势,在多次测试中漏洞挖掘所用时长超过 5 min 的最大时间,其根本原因在于 AFL++ 在测试过程中出现稳定性下降问题,覆盖率信息随机变化,覆盖率引导机制无法正常工作,测试策略彻底退化为黑盒模糊测试。

2) IUSFUZZ 漏洞挖掘结果复现稳定性更高: 4DFuzzer 和 AFL++ 因缺乏线程同步机制,在触发漏洞后难以进行稳定复现。具体来说, 4DFuzzer 通过检测接收回复是否超时来判断当前测试用例是否触发程序崩溃, AFL++ 通过检测程序执行的异常状态判断当前测试用例是否触发程序崩溃,然而由于 Fuzzer 与控制程序的 Receiver 线程、命令执行线程相互异步执行, Fuzzer 无需等待 Receiver 线程完成消息接收即可发送下一条测试用例, Receiver 线程无需等待命令执行线程执行结束即可开始接收下一条数据包。所以当控制程序发生崩溃时,极有可能是由 Fuzzer 数秒前发送的测试用例触发的,然而 Fuzzer 已经将其丢弃并开始执行后续数轮测试,并错误地将

表 4 漏洞挖掘效率对比
Table 4 Comparison of vulnerability mining efficiency

工具名	CVE-2025-57427 挖掘时间/s	CVE-2025-57427 触发用例数/个	CVE-2024-40427 挖掘时间/s	CVE-2024-40427 触 发用例数/个	复现概率/%
IUSFUZZ	8	1 139	15	27 554	100
AFL++	61	277 413	>300	>1 000 000	0
4DFuzzer	80	22 267	10	1 028	0

当前测试用例当作引起控制程序崩溃的恶意数据包,这使得4DFuzzer和AFL++即使能够触发程序崩溃,也无法进行稳定复现,需要额外依赖目标侧或Fuzzer日志进行溯源。而IUSFUZZ通过强制同步机制,无需额外手段辅助,实现了漏洞的100%复现率,验证了其在复杂系统漏洞挖掘场景中的可靠性。

4.4 干扰因素影响权重分析

基于2.4节构建的量化关联模型,本文进一步分析各核心干扰因素对测试效率衰减的影响。

首先基于4.2节与4.3节的实验数据对(3)式中的系数 k 进行拟合。在实验4.2中,IUSFUZZ稳定性为16.7%,则IUSFUZZ在模糊测试过程中出现的覆盖率偏差 $\Delta\text{Cov}_{\text{IUS}}=0.833$;AFL++的稳定性为0.2%,则AFL++在模糊测试过程中出现的覆盖率偏差 $\Delta\text{Cov}_{\text{AFL}}=0.998$ 。在实验4.3中,IUSFUZZ的测试效率(100%-IUSFUZZ的测试效率衰减 $\lambda_{\text{E,IUS}}$)至少为AFL++测试效率(100%-AFL++的测试效率衰减 $\lambda_{\text{E,AFL}}$)的7倍,建立等式拟合后确定(2)式比例系数 k 取值为3。

其次,模糊测试过程中已探知全部程序路径数量 E_{total} 为9822个。在Persistent模式下,导致测试稳定性下降的变化路径数量 ΔE 为9802个;归因于异步线程执行的变化路径数量为9802个,由于各线程间呈完全解耦状态,取解耦系数 α 值为1,利用(4)式计算得到 $\Delta\text{Cov}_1=0.998$;内外部信号交互干扰对应的变化路径数量为4364个,利用(5)式计算得到 $\Delta\text{Cov}_2=0.445$;时空状态残留引发的变化路径数量为4614个,利用(6)式计算得到 $\Delta\text{Cov}_3=0.469$ 。需要说明的是,由于部分变化边同时受多种干扰因素共同作用,故存在部分变化路径被重复统计的情况。将 ΔCov_1 , ΔCov_2 , ΔCov_3 分别代入(7)式中计算可得:异步线程执行导致的测试效率衰减 λ_{E1} 达94%,时空状态残留造成的测试效率衰减 λ_{E2} 为26%,内外部信号交互引发的测试效率衰减 λ_{E3} 为24%。

量化结果表明,异步线程执行是导致测试效率衰减的首要因素,其影响远高于其他两类因素;时空状态残留的影响次之,内外部信号交互的影响相对最小。

4.5 强制同步性能影响分析

消息过滤、时间变量替换、拓展改进后的选择性插桩策略,因机制特性仅涉及局部代码的执行路径调整,对系统整体性能的影响极小,在实际测试场景中可忽略不计;而线程强制同步策略是唯一直接作用于多线程异步执行时序的机制,会显著干扰模糊测试吞吐量,因此,本节聚焦该策略展开具体的性能影响分析。

为了进行具体的量化分析,本文记录了不同强制

同步策略下模糊测试的执行速度与稳定性,具体包括:无同步策略(no_sync)、仅同步输入线程(input_sync)、同步输入线程与1个目标线程(input_sync_one)、同步输入线程与2个目标线程(input_sync_two)、同步输入线程与3个目标线程(input_sync_three)、同步输入线程与4个目标线程(input_sync_four)的模糊测试执行速度与稳定性,实验结果如图7所示。

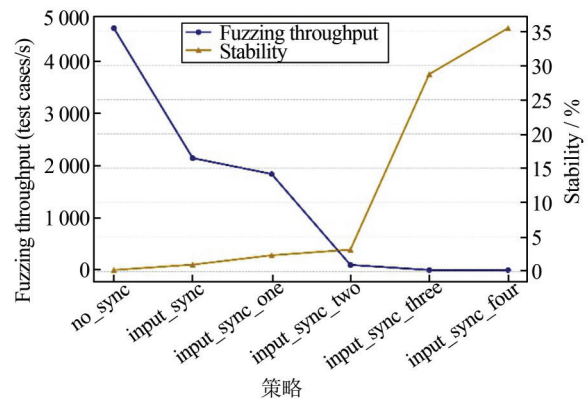


图7 不同强制同步策略下的模糊测试吞吐量与稳定性
Fig. 7 Fuzzing throughput and stability under different mandatory synchronization strategies

实验结果表明,随着同步线程数量的增加,模糊测试吞吐量呈持续下降趋势,而稳定性则逐步提升,其中,核心性能损失来自对输入线程的强制同步,由于本文采用的是阻塞式管道与输入线程进行通信,该通信方式的传输速率成为了模糊测试吞吐量提升的瓶颈。为进一步量化评估本文采用的同步策略(同步输入线程与1个目标线程/input_sync_one)对漏洞挖掘效率的提升效果,构建量化模型如(8)式所示。

$$E_e = E_s \times P_{\text{rep}} \times e^{-k \times \Delta\text{Cov}} \quad (8)$$

其中, E_e 表示每秒钟执行的有效测试用例(根据覆盖率引导机制进行有效变异并且可以复现的测试用例)数量; E_s 表示模糊测试吞吐量,即每秒钟执行的测试用例数量; P_{rep} 为挖掘漏洞后的复现概率。本文采用sigmoid函数表征其与覆盖率之间的关系,如(9)式所示。

$$P_{\text{rep}} = \frac{1}{1 + e^{-a \times (1 - \Delta\text{Cov} - b)}} \quad (9)$$

其中, a 、 b 为比例系数,通过4.3节中的复现概率数据对(9)式中的比例系数进行拟合后分别取值为: $a=200$, $b=0.05$ 。将图7中的稳定性数据代入到(1)、(2)式中可得到每种同步策略的 ΔCov 值,再与图7中的模糊测试吞吐量 E_s 数据先代入到(9)式中进行计算,再结合(9)式的计算结果代入(8)式,即可得到各同步策略每秒钟执行的有效测试用例数量 E_e ,如图8所示。

图8的量化结果显示,input_sync_one策略的每秒执行有效测试用例数量是最多的,为1个/s,而其

他同步策略(包括无同步策略/no_sync)的每秒执行有效测试用例数量都接近于0;no_sync和input_sync策略虽然模糊测试吞吐量较高,但因为稳定性太低导致执行测试用例几乎无法复现,每秒执行测试用例数量逼近于0;input_sync_two、input_sync_three和input_sync_four策略虽然实现了较高的稳定性,但因为同步策略过于激进导致模糊测试吞吐量快速下降,使得每秒可执行的有效测试用例数量有限;input_sync_one策略则在模糊测试吞吐量和稳定性之间取得了平衡,成为IUSFUZZ框架最终采用的同步策略。说明本文设计的强制同步策略虽然会导致模糊测试吞吐量有所下降,但该策略仍对整体漏洞挖掘效率的提升发挥着不可或缺的关键作用。

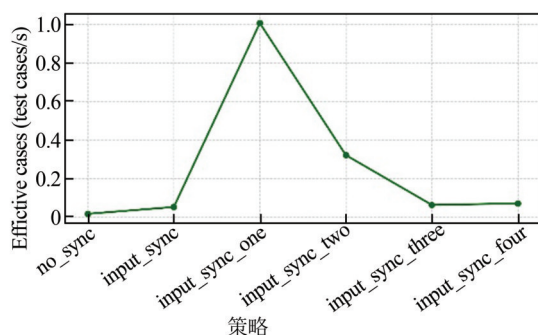


图8 不同强制同步策略下的每秒执行有效测试用例数量
Fig. 8 Effective executions per second under different mandatory synchronization strategies

5 案例分析

CVE-2025-57427是基于本文提出的方法完成挖掘与复现分析的。其他工具虽能触发类似现象,但因无法捕获测试用例导致其无法复现,更无从分析。因此,该漏洞是本研究发现的首个零日漏洞,并通过申请获得了CVE编号。该漏洞允许攻击者在不与搭载PX4控制程序的机器建立连接的前提下,仅通过广播恶意MAVLink数据包就可以迫使机器进入低电量状态并停止运行(若机器为无人机则会垂直降落着陆,且无法再起飞)。现详细阐述如下。

5.1 漏洞成因

该漏洞的产生源于Commander线程在数据冲突处理时的决策逻辑缺陷。PX4控制程序的核心控制组件Commander线程负责检查机器异常状态并依据状态机进行模式切换,Commander线程通过监听uORB消息总线实现实时的跨线程机器状态数据同步。在电池状态监控机制中,Commander线程的相关处理代码存在多数据源融合缺陷:一方面持

续轮询内部传感器采集的电池剩余电量数据;另一方面通过MAVLink网络接口接收地面站或其他机器传输的电池状态通知,当系统同时接收到多个来源的电池状态数据时,由于uORB消息机制未实现发布者身份认证与优先级标记功能,Commander线程无法对数据源进行可信度评估。PX4设计者为了尽可能保证机器的电池安全,没有采用加权平均或冗余校验等容错机制对电池状态数据进行处理,而是简单地将所有有效数据中最低电量的电池状态设置为全局电池状态并执行相应的预设命令(如垂直降落并着陆)。这种设计导致PX4控制程序容易遭受恶意数据注入攻击,攻击者可通过持续向机器发送伪造的MAVLink数据包通知机器当前电池剩余电量低,强制系统触发低电量保护机制。

5.2 漏洞复现条件

该拒绝服务(DoS)漏洞的触发需要满足以下攻击条件:

- 1) 攻击者建立与目标机器的MAVLink协议通信链路。
- 2) 恶意数据包构造:指定MAVLink数据包的消息类型为BATTERY_STATUS(msgid=147);剩余电量字段设置低于系统安全阈值(低于5%)。
- 3) 数据包发送策略:数据包发送频率需高于正常传感器上报速率(≥ 10 Hz);采用持续不间断的数据流注入策略。

攻击成功的关键在于维持足够的数据注入强度,使伪造数据在uORB消息队列中所占比例与真实传感器数据相当,甚至占据主导地位,从而实现对Commander线程决策的有效干扰。

5.3 漏洞危害验证

1) 在SITL仿真环境中,攻击者向目标机器控制程序的任意MAVLink端口持续发送恶意数据包,攻击目标在遭受攻击后会在日志系统中输出低电量警报“Dangerously low battery! Shutting system down in 60 seconds”,并会在1 min后直接关机,如果目标机器控制程序在遭受攻击前连接了QGC(QGroundControl)地面站程序^[24],在攻击者实施攻击后将会看到QGC状态栏出现多个低电量图标并直接与目标机器控制程序断开连接。

2) 在实机运行环境中,受害者同时使用QGC和遥控器连接目标机器,攻击者位于距离目标机器2 000 m范围内的任意位置(视电磁信号频率、强度与无线电模块性能而定),并通过无线电模块广播恶意数据包,若此时目标机器:

- a. 处于等待启动状态。受害者尝试使用QGC

向目标机器发送启动命令,目标机器将因为无法通过安全检查(低电量警报)而拒绝启动。

b. 处于运行状态。如图9所示,受害者将观察到QGC状态栏显示了一个低电量图标警报,状态栏变为红色,同时受害者无法再利用QGC控制目标机器。

若目标机器为无人机,那么使用遥控器可以勉强控制目标无人机的水平方向,但无法控制目标无人机的垂直升降,目标无人机的运动轨迹呈现随机变化趋势。当攻击者停止攻击后,无人机将立刻进入低电量紧急状态并采取安全措施,开始垂直降落并准备着陆。

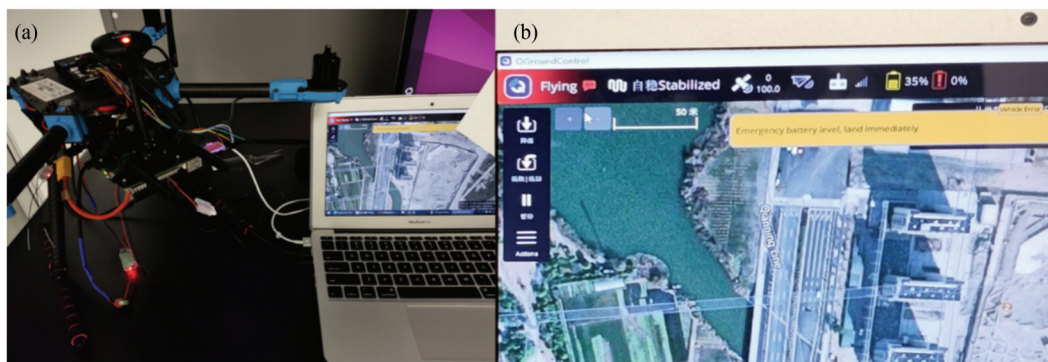


图9 (a) 搭载PX4控制程序的无人机;(b) QGC地面站控制程序遭受攻击后的界面

Fig. 9 (a) A drone equipped with the PX4 control program; (b) The interface of the QGC ground station control program after being attacked

5.4 漏洞危害评估与缓解措施

该漏洞具备远程可利用性、攻击门槛低和影响范围广等特点。攻击者无需与目标机器建立任何形式的可信连接,也无需突破认证机制,仅依赖广播恶意MAVLink数据包即可诱使PX4控制程序错误触发低电量保护逻辑导致机器无法正常运行(无人机将被迫降落并无法再次起飞)。由于该漏洞可导致机器任务失败、设备损毁并引发次生安全风险,按照CVSS v3基础度量指标,该漏洞可获得8.2/10的高危评分等级。

若想缓解该漏洞造成的影响,可以在无人系统的数据采集和协议交互等不同环节引入缓解措施,提升系统运行的稳定性与可靠性:

1) 构建电池状态的多源交叉验证机制,将本机传感器采集的电池剩余电量设定为最高可信度数据源,外部来源仅作为辅助信息。并在多个来源的数据之间执行数值偏差检测,若某一来源与其余来源差异超过阈值,则视为异常数据并触发告警而非立即执行降落操作。

2) 可在MAVLink通信中增加消息发送端的身份校验或签名验证,避免非授权设备伪造电池数据,但可能会提升无人系统的计算开销。

6 结 语

本研究以智能无人系统作为研究平台,针对覆盖率引导模糊测试性能退化问题,通过理论分析与

实验验证,揭示了智能无人系统多线程异步执行环境下覆盖率引导机制失效及其解决路径。研究结果表明,基于强制同步机制、拓展改进后的选择性插桩、消息过滤和时间变量替换策略构建的IUSFUZZ系统,能够有效抑制控制程序中异步线程执行、内外部信号交互及时空状态残留等核心干扰因素,这些策略组合相较于AFL++稳定性实现了16.5个百分点的提升,漏洞挖掘效率实现了约7倍的提升,验证了IUSFUZZ稳定性提升机制与覆盖率引导机制的有效性。

需要指出的是,强制同步策略中使用的阻塞式管道通信速率限制了模糊测试的吞吐量,未来将考虑使用更加高效的进程间通信方式以提升测试性能。与现有覆盖率引导模糊测试工具关注单线程程序执行异常与多线程程序条件竞争漏洞不同,本工具在保持程序多线程运行的前提下挖掘由外界输出触发的安全漏洞并解决了由稳定性下降引起的测试性能退化问题。本研究不仅证明了覆盖率引导模糊测试在智能无人系统安全验证中的技术可行性,更为智能无人系统的形式化验证提供了新的方法论参考,其发现的低电量警报DoS漏洞已在实机环境中进行复现,具有显著的工程实践价值。未来研究将从以下两个维度进一步深化:其一,通过优化系统测试的吞吐量以提升测试效率;其二,探索可视化驱动测试用例生成技术,以同时提高测试覆盖率和漏洞定位精度。

参考文献:

- [1] Alphabet Company. Wing autonomous, lightweight delivery drones to transport goods[EB/OL]. [2025-08-07]. <https://x.company/projects/wing>.
- [2] MÜLLER C. World robotics 2024 - service robots[EB/OL]. [2025-08-07]. https://ifr.org/img/worldrobotics/Executive_Summary_WR_2024_Service_Robots.pdf.
- [3] SINGH I. Super Bowl weekend: 1000 strong drone show to light up Vegas sky[EB/OL]. [2024-02-08]. <https://dronedj.com/2024/02/08/super-bowl-drone-show-vegas>.
- [4] KOUBÂA A, ALLOUCH A, ALAJLAN M, *et al.* Micro air vehicle link (MAVlink) in a nutshell: A survey [J]. *IEEE Access*, 2019, **7**: 87658-87680. DOI: 10.1109/ACCESS.2019.2924410.
- [5] MEIER L, AGAR D, KÜNG B, *et al.* PX4-Autopilot [EB/OL]. [2025-08-07]. <https://github.com/PX4/PX4-Autopilot>.
- [6] PRATAP A, BONNEY B, PUROHIT S B, *et al.* Ardupilot[EB/OL]. [2025-08-07]. <https://github.com/ArduPilot/ardupilot>.
- [7] BENOWITZ D. The rise of open-source drones[EB/OL]. [2021-05-30]. <https://droneanalyst.com/2021/05/30/rise-of-open-source-drones#os-size>.
- [8] MALVIYA V K, MINN W, SHAR L K, *et al.* Fuzzing drones for anomaly detection: A systematic literature review [J]. *Computers & Security*, 2025, **148**: 104157. DOI: 10.1016/j.cose.2024.104157.
- [9] CHEN H, GUO S, XUE Y, *et al.* MUZZ: Thread-aware grey-box fuzzing for effective bug hunting in multithreaded programs[EB/OL]. [2025-03-10]. <https://www.usenix.org/conference/usenixsecurity20/presentation/chen-hongxu>.
- [10] LI J Q, ZHOU Y J, WU L. Improving multitasking DBMS fuzzing with more accurate coverage and testcase trimming [J]. *IEEE Transactions on Dependable and Secure Computing*, 2025, **22**(3): 2756-2770. DOI: 10.1109/TDSC.2024.3521591.
- [11] 王丽娜, 赵敬昌, 余荣威, 等. 针对 ARM 安全监视器的模糊测试方法研究[J]. 武汉大学学报(理学版), 2024, **70**(4): 453-462. DOI: 10.14188/j.1671-8836.2023.0009.
- [12] WANG L N, ZHAO J C, YU R W, *et al.* Research on fuzz testing method for ARM secure monitor [J]. *Journal of Wuhan University (Natural Science Edition)*, 2024, **70**(4): 453-462. DOI: 10.14188/j.1671-8836.2023.0009(Ch).
- [12] CHANG Z W, ZHANG H F, YANG Y, *et al.* Fuzzing drone control system configurations based on quality-diversity enhanced genetic algorithm[C]//Artificial Intelligence Security and Privacy. Singapore: Springer, 2024: 499-512. DOI: 10.1007/978-981-99-9785-5_35.
- [13] KIM T, KIM C H, RHEE J J, *et al.* RVFuzzer: Finding input validation bugs in robotic vehicles through control-guided testing[EB/OL]. [2025-03-12]. <https://friends.cs.purdue.edu/pubs/USENIXSec19.pdf>.
- [14] LI G P, LI Y R, JHA S, *et al.* AV-FUZZER: Finding safety violations in autonomous driving systems[C]//2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE). New York: IEEE Press, 2020: 25-36. DOI: 10.1109/issre5003.2020.00012.
- [15] WANG Y, YANG C, ZHANG X D, *et al.* DPFuzzer: Discovering safety critical vulnerabilities for drone path planners[C]//2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE). New York: IEEE Press, 2025: 1973-1985. DOI: 10.1109/ICSE55347.2025.00051.
- [16] DOMIN K, MARIN E, SYMEONIDIS I. Security analysis of the drone communication protocol: Fuzzing the MAVLink protocol[EB/OL]. [2025-08-07]. <https://orbilu.uni.lu/bitstream/10993/37613/1/article-2667.pdf>.
- [17] BOB4Drone. 4D-Fuzzer [EB/OL]. [2025-08-07]. <https://github.com/BOB4Drone/4D-Fuzzer>.
- [18] KIM H, OZMEN M O, BIANCHI A, *et al.* PGFUZZ: Policy-guided fuzzing for robotic vehicles[EB/OL]. [2025-09-27]. https://www.ndss-symposium.org/wp-content/uploads/ndss2021_6A-1_24096_paper.pdf.
- [19] MEIER L, AGAR D, KÜNG B, *et al.* Simulation[EB/OL]. [2025-08-07]. <https://docs.px4.io/main/en/simulation/#sitr-simulation-environment>.
- [20] FIORALDI A, MAIER D, EIBFELDT H, *et al.* AFL++: Combining incremental steps of fuzzing research[EB/OL]. [2025-09-28]. <https://www.usenix.org/system/files/woot20-paper-fioraldi.pdf>.
- [21] DROBILLA D R. Llvm_mode persistent mode[EB/OL]. [2025-08-07]. https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/README.persistent_mode.md.
- [22] MEIER L, HONEGGER D, POLLEFEYS M. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms[C]//2015 IEEE International Conference on Robotics and Automation (ICRA). New York: IEEE Press, 2015: 6235-6240. DOI: 10.1109/ICRA.2015.7140074.
- [23] VANHAUSER-THC. Using AFL++ with partial instrumentation[EB/OL]. [2025-08-07]. https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/README.instrument_list.md.
- [24] HOLDEN R. Qgroundcontrol [EB/OL]. [2025-08-07]. <https://github.com/mavlink/qgroundcontrol>.

□