FIRM-AFL: High-Throughput Greybox Fuzzing of IoT Firmware via Augmented Process Emulation

Lyu Jiuyang，Feb 10, 2022.

Infomation

Authors: Yaowen Zheng, Ali Davanian, Heng Yin,Chengyu Song, Hongsong Zhu, and Limin Sun. Usenix Security 2019.

Code: https://github.com/zyw-200/FirmAFL

一种提高IoT固件 Fuzz 性能的方案。

Background

fuzz

模糊测试。一种通过自动化生成并执行大量的随机测试用例来发现产品或协议的未知漏洞的技术。

- 黑盒：不利用任何反馈来做随机输入的变异
- 白盒：根据对目标程序有着深入的了解下，对随机输入进行变异
- 灰盒：检测目标程序以收集信息，用来指导样本的变异（如代码覆盖率）——能触发新代码的样本会被遗传算法选中，而不能触发新代码的样本会被丢弃。

AFL (American fuzzy lop)

Link: https://github.com/google/AFL

sota. 一种以代码覆盖率作为变异策略的灰盒fuzz。

启动AFL的主程序在 `afl-fuzz` 中。它从样本集中挑选一个原始样本执行一次随机变异，生成一个新的随机样本，并将这个样本喂给目标程序。然后先运行目标程序到特定位置（例如：main函数入口），使程序的代码和数据正确初始化，然后作为 `fork server` 重复地派生出子进程；

QEMU (Quick EMUlator)

模拟处理器软件。有两种主要运作模式。**用户模式（user-mode）**仅模拟 CPU 的运行（如wine），而**系统模式（full-system mode）**除了模拟 CPU 的运行，还要管理、模拟外围的设备（模拟整个系统）。

Introduction

Motivation

fuzz对硬件配置有较高的要求，使用软件进行系统模拟的效率要比使用IoT设备的效率高得多。进行系统仿真时，系统模式大约比用户模式慢十倍。

Solution

通过增强过程仿真进行灰盒模糊处理。结合了QEMU的两种模式，用系统模式提供用户模式无法运行时的环境，将用户模式作为主要的运行状态。

Target

Transparency：不需修改固件代码直接运行。

High efficiency: 快。

	Avatar [33]	IoTFuzzer [14]	Firmadyne [13]	Muench et al. [28]	AFL [34]
Technique	Whitebox fuzzing	Blackbox fuzzing	PoC	Blackbox fuzzing	Greybox fuzzing
Compatibility	High	High	High	High	Low
Hardware Support	Hybrid	Real	Emulation	Mixed	None
Code Coverage	Medium	Low	N/A	Low	High
Throughput	Very Low	Low	Medium	Low to Medium	High
Zero-day Detection	Yes	Yes	No	Yes	Yes

Table 1: Comparison of IoT firmware testing tools.

Firmadyne系统模式下，被限制吞吐量的原因

1. 内存地址转换

   系统模式下QEMU使用MMU（内存管理单元）对所有内存访问进行地址转换，用户模式要简单的多。

2. 动态代码翻译

   系统模式下块链接（block chaining）仅限于同一物理页中的基本块，用户模式会更快。

3. 系统调用模拟

系统模式下操作系统和硬件设备都需要模拟，用户模式系统调用是由主机操作系统和硬件直接处理所以更快。

Methodology

增强进程模拟（Augmented Process Emulation）

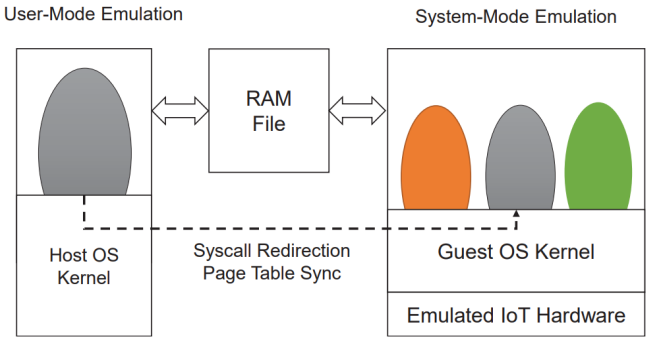

Figure 1: Overview of Augmented Process Emulation

IoT 固件在系统模式的仿真环境下启动，并启动各种需要的用户程序（包括fuzzer）。当要 fuzz 的程序运行到某个特定点的时候（比如 main 函数的入口、接收数据包的函数入口），该进程的执行就切换到用户模式的仿真环境下。极少数情况程序会回到系统模式以保证正确执行。

内存映射共享

系统模式的物理内存被分配为一个内存映射文件（RAM），这个文件同时也被映射到用户模式的地址空间，通过虚拟地址访问。在用户模式仿真环境下访问没有建立页面映射的页面时，就需要切换回系统模式仿真环境下执行来建立映射。

引导（Bootstrapping）

类似于AFL，在系统模式下启动固件，在监测到预定点时，遍历指定进程的页表得到映射信息。函数如下：

```
mmap(va, 4096, prot, MAP_FILE, ram_fd, pa);
```

系统模式暂停运行，CPU状态被发送到用户模式，然后在用户模式继续运行。

页面错误处理

若在用户模式访问时页面还没有建立映射时，需要向系统模式传递CPU状态。

> To capture the right moment when a mapping is established, we instrument the end of each basic block. If the execution is currently within the specified process (or thread), it means the execution has returned from the kernel to the user space to resume the faulting instruction. The mapping must be present in the software TLB. So we can just directly find the mapping there. At this moment, we pass the mapping information and the CPU state back to the user-mode emulation, which will create this new mapping by calling mmap and resume the execution.

如果因为某些原因进程被杀死，则终止两边的执行。

预加载页面映射

在引导过程中模拟系统模式对每个程序代码页的访问，以强制操作系统将每个页映射到进程的地址空间。

系统调用重定向

系统在用户模式完成代码翻译和执行，在遇到系统调用时，会暂停执行并保存当前的CPU状态，将其发送给系统模式。当系统调用返回时，再把执行流切换回用户模式仿真环境下执行。

许多系统调用都与文件系统有关。方案从固件中映射文件系统，并将其作为物理机的一个目录挂载，便于用户模式直接访问；同样地，用户模式就可以直接将文件系统相关的系统调用传递到物理机操作系统，而不是重定向到系统模式。

Firm-AFL 灰盒测试系统

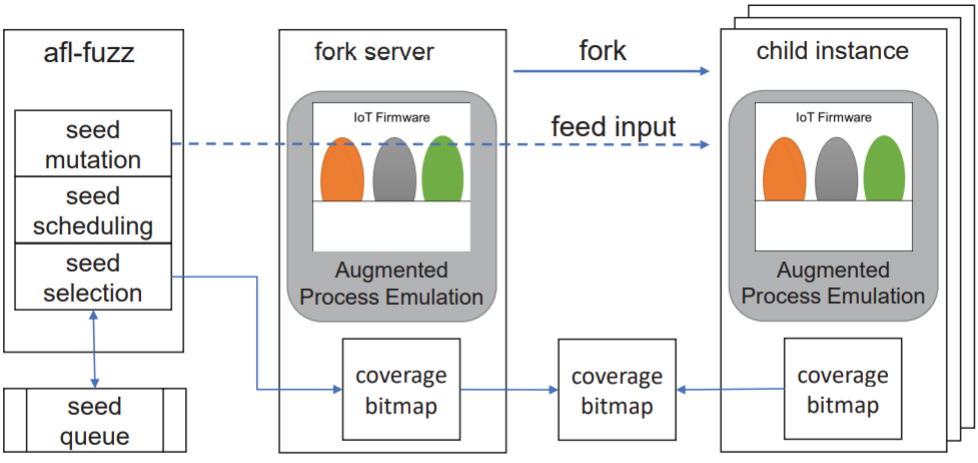

Figure 2: Overview of FIRM-AFL

将QEMU user-mode替换为增强进程模拟，其余组件保持不变。

引导

Firmadyne（模拟固件）+DECAF（实时获取系统中所有进程的状态信息）

Forking

hook网络相关的系统调用函数，当这些函数被第一次调用时就会创建分支点，用来派生一个子进程，并**为系统模式创建一个虚拟机的快照**以保证两个模式的同步。fuzz完成后恢复快照。

基于写时复制（Copy-on-Write）的原理，作者实现了一个轻量级的快照机制。首先将映射到系统模式的RAM文件标记为只读；复制这个页面，然后把这个页面标记为可写。因此我们记录在一次 fuzz 期间修改的所有内存页；当恢复快照时，只需要将这些记录下来的页写回即可。

样本输入

系统对网络相关的系统调用进行插桩并实现输入注入。

对于代码覆盖信息的收集，由于系统仿真端只进行内存映射查找和系统调用执行，因此只在用户模式下进行插桩收集。

Evaluation

Transparency

本系统与系统模式仿真在nbench测试机和120个 IoT HTTP程序下结果一致。

High efficiency

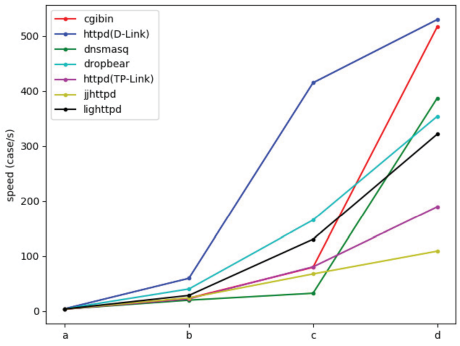

Figure 3: Fuzzing throughput of FIRM-AFL under different optimization level. The x-axis is the optimization level: (a) baseline, (b) w/ lightweight snapshot, (c) w/ augmented process emulation, and (d) w/ selective syscall redirection. Fuzzing throughput for each program is shown in a different color.

Baseline: TriforceAFL.

Effectiveness of optimization

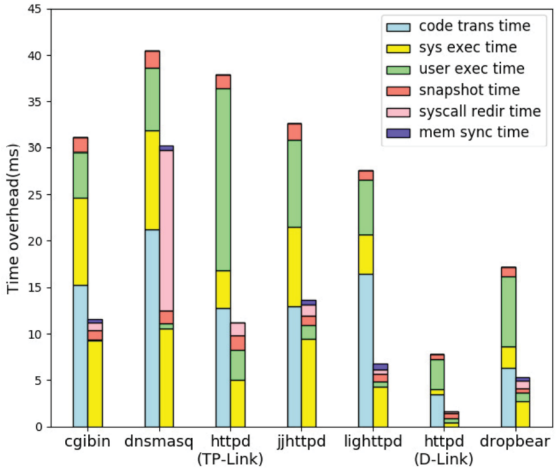

Figure 5: Execution time breakdown: augmented process emulation vs. full-system emulation.

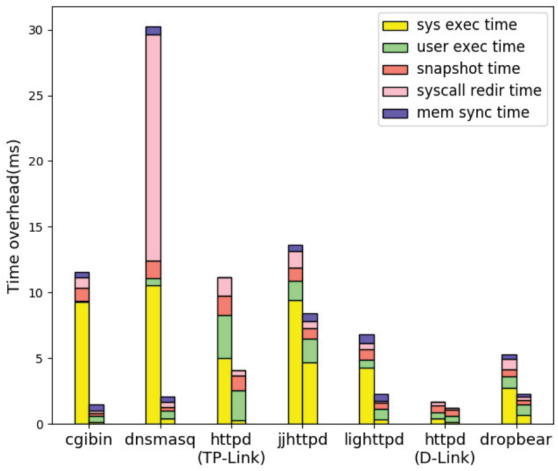

Figure 6: Execution time breakdown: augmented process emulation w/o and w/ selective syscall redirection.

三个导致系统模式下性能瓶颈的问题都得到了有效缓解。

Effectiveness in vulnerability discovery

与系统模式仿真相比，在Firmadyne数据集上漏洞挖掘的性能有非常大的提升。

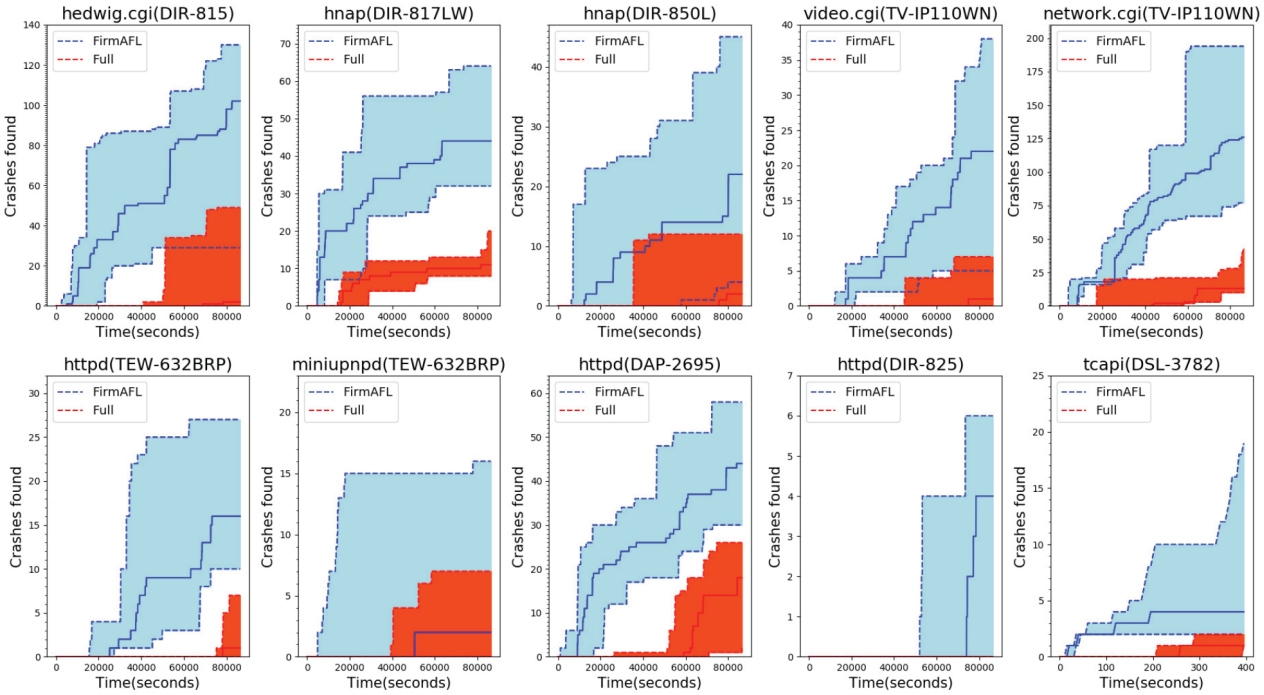

Figure 7: Crashes found over time