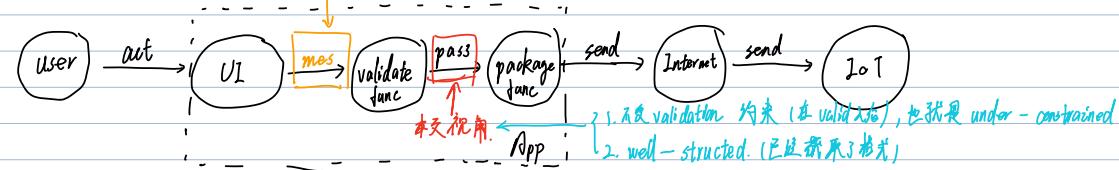


- DIANE. fuzz the first function's parameters

视角对称: IoTFuzzer 视角: 只是生成样例



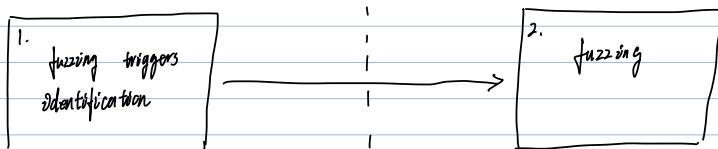
以前的方法, IoTFuzzer 生成 APP 可处理的 mes.

- 但面临 2 个问题:
 1. App 会对输入 sanitize, 即输入验证 (51% 的都会被过滤). IoTFuzzer 的效率会变低
 2. App 会由输入验证产生不同的结果触发不同的行为 (这篇论文很像有点废话, 可以大概看一下)

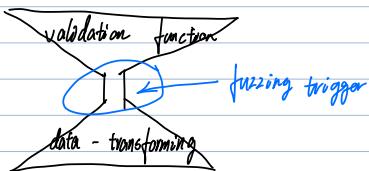
本文直觉: 将 App 执行视作 a sequence of functions. 其中, top - of - seq 需要将用户输入转换为内部 struct.
end - of - seq 会将 struct 转成 encode.

想法: 寻找特定函数进行 fuzz 称为 (fuzzing triggers)

整体布局: 2 steps



core: 寻找 optimally produce inputs 的 functions. 這些 funcs | core |: 利用第 1 步中找到的 fuzzing triggers 他们 fuzz 以生成 valid 且 under-constrained 的输入.



- 修改对象: 修改 trigger 的参数
- mutate strategies:
 - String length: 生成具有不同长度的随机字符串, 以触发缓冲区溢出和非绑定访问 (out-of-bound access).
 - Numerical Values: 更改数值, 如很大的值、负数、零等.
 - Empty Values: 即置空, 有指针引用等.
 - Array length: 通过删、增数组元素来观察.

提出了新的 analysis pipeline, 基于:

- 1. static call-graph
- 2. network traffic analysis
- 3. static data-flow
- 4. dynamic analysis func arguments

探索的方向: "bottom-up", 分为 4 步:

Step 1. send Message Candidates Identification

即向 IoT 设备发送消息的功能模块功能

难点: 有些函数可能在独立的进程中执行, 造成数据难以跟踪, 因而难以分析

解决方案: 利用 "border functions", 位于 App 的核心功能组件和外部组件之间 (如 Android framework, native lib)

当其执行时, 最终会触发一条发送到 IoT 设备的消息

最终方法: 静态分析. 作者收集了调用关键网络功能的本地函数 / Android 框架的函数

Core 2: 求解 crash.

通过流量进行监控. 如果满足以下任意一个, 都认为是 crash

- Connection dropped.
- HTTP Internal Server Error (500).
- Irregular network traffic size. 如果 App 和设备之间交换的数据量低于某个阈值, 则很有可能导致 crash 的输入. 直观是: 当 crash 时, 流量会突然无法正常使用, 从而导致数据交换量减少. 具体来说, 是当小于 50% 时, 可认为是 crash.
- Heartbeat Monitoring. IP 监测响应时间. 超时时认为 crash

Step 2. send Message Validation

目的: 看 Step 1 生成函数是否满足

① 多次动态执行函数, 看是否每次都能产生流量

② 阻止 App 执行该函数，再看是否产生

难点/问题：无论是①还是②，都会引起程序 crash

解决方式：使用基于 time stamps 和 machine learning 的方式



机制：记录时间间隔，计算 mean、标准差等

用 K-means 算法进行聚类。(2类)

选择执行时间间隔小的 cluster，作为后续分析对象

原理：导致网络活动的 Func 的幅值和方差比较小，因为其受噪声的影响较小

Step 3. Data - Transforming Function Identification (没有太懂)

目的：识别此类 data-transforming function.



如果在夜里 fuzz，可能应该让函数在解码时会无法识别而抛弃。

因此要在夜里 fuzz

难点：① 将变量的 data 位于 class 中，理论上涨字段就会被任何线程设置，包括其他线程。

② 要考虑溢出情况（可能 data 来自父类，布告被不同的类继承）

解决方式：① 复查源代码：

先静态识别，看哪些变量可能是，并记录其可能被 set 的位置。为把创建一个集合 Su，元素为 (v, cl)，v 为 variable，cl 为 code location.

② 切片。

对每个 (v, cl) 做 static inter-procedural backward slicing 从 cl 回溯到从 UI objects 中检索值的 function.

③ 划分 scope：

之后将这些 slices 划分为 function scopes. (说明：对于一个给定的 slice，function scope 是其中属于同一个 function 的指定序列形而上的一个 subsequence)

④ 变量区间：

对于每个 scope，存储其中变量（如 local 和 class field）的活跃区间（生命周期）。将 Lif 作为 scope 中开始部分活跃期变量集合；将 Log 作为 scope 结束部分活跃期变量集合。

⑤ 计算熵：

为了识别 data-transforming function，作者利用之前研究中做一个结论：此类函数会增加他们所消耗的数据的熵。

因此，计算在运行时分配给 Lif 和 Log 中每个变量的熵。

解决方法：判断 v 是原始类型（如 int），或已知类型（如 String, Float, Integer, and Double）。用转换包围在字节表示中的数据（原文是 we convert the data it contains in its byte representation）并计算熵。

如果 v 是类中的字段，则去掉类头看其类型，之后同上。

⑥ 判断。

看 Log 中最大的熵和 Log 中最小的熵的比值，如果大于指定的阈值（本文为 2.0）则认为是 data-transforming function

Step 7. Top - Chain Functions Collection

Data-chain: data-transforming 的序列，即会对最终以改变的数据序列

Top: 第一个（即改完了他就会改变后续的）

核心想法：数据的 transforming 有一定的执行顺序，只要控制了第一个，后续就同样会受影响，所以 Top 是精准的 fuzzing trigger 选择。

难点：怎么识别是个转变。

解决方法：构建上一步中检测的 data-transforming 以树的 dominance

tree（树状结构，每个节点的子节点是它立即影响的节点）

注：如果没有 data-transforming function 立即 send Messages

则将 send message 视为 fuzzing trigger。同时，原附上

app-side sanitization 它们也可能出现在 chain 中。本

文暂未想到解决方法。

Limitation:

- 1.
2. 代码覆盖率有限，不能识别没有被 App 调用的 trigger
3. 不能 fuzz 嵌套的 Java 对象。