**[项目名称]**

|  |  |
| --- | --- |
| **申请者姓名** |  |
| **所属机构** |  |
| **提交日期** |  |
| **联系方式** |  |

**目录**

**1. 项目介绍**

1.1. 题目

1.2. 项目背景及研究意义

1.3. 研究目标

1.4. 研究方法

**2. 研究计划**

2.1. 项目期限

2.2. 进度安排

**3. 预期成果**

**1. 项目介绍**

* 1. 题目

对基于Linux的IoT平台应用的Fuzzing漏洞挖掘

* 1. 项目背景及研究意义

物联网（Internet of Things，IoT）是万物互联的网络，利用现代信息传送载体的高速及容量将现实世界数字化，应用范围十分广泛。在“互联网+”的背景下，物联网发展迅猛，物联网设备正在以极高的速度渗透进人们生活中的方方面面，从个人智能设备，到传统的交通和物流领域，再到智能医疗的领域，到处都可以看到物联网设备的身影。

在物联网设备规模呈现爆发性增长的同时，物联网设备的安全性也就成为了非常重要的一个问题。大量智能设备存在于人们的生活中，如果这些设备被黑客攻破利用就能造成非常严重的后果。比如如果连入互联网的汽车系统被攻破而被黑客远程控制，就可以随意控制目标车辆刹车或者加速，这对物联网设备使用者的安全造成了极大地威胁。除此之外，上至国家级工厂的设备，下到个人家庭使用的传感器网络，都有可能收到物联网安全问题的威胁。

在这个背景下，对物联网设备的安全性进行研究就成了非常重要的一个课题。传统物联网设备大致分成两个类别，第一类设备拥有非常少的能源供给，或者需要比较高实时性，所以需要运行低功耗的系统或者高响应的系统，此类设备通常会运行RTOS作为底层的操作系统，RTOS往往会与应用程序打包在一起，并且缺乏明显的系统界线，因此这一类设备的安全性往往需要研究人员大量的人工分析，很难做到自动化的测试。第二类设备就是采用传统的Linux系统的IoT设备，这类设备往往有持续的能源供给，不太关心能耗的问题，并且不需要极高的实时性。因为Linux设备的开发门槛更低，很多传统PC已经存在的工具都可以直接套用，所以有大量的设备采用了Linux作为底层操作系统，这使得这一类设备上的应用与传统PC上的应用非常类似，很多软件测试方法也可以借鉴。

模糊测试（Fuzzing）是传统PC平台上近些年非常流行的软件测试方法，相比较于传统的人工漏洞挖掘，Fuzzing可以提供更高的自动化，在软件测试过程中，持续性的对软件进行测试，并且Fuzzing可以利用随机或者半随机的数据，以覆盖率等作为回馈指导，从而达到挖掘出深层次软件漏洞的目的。在Fuzzing技术逐渐成熟的过程中，AFL（American Fuzzy lop）就是一个代表之作，其通过在软件编译时在程序中插入插桩代码，以统计程序执行的分支覆盖率，并根据覆盖率的反馈决定输入的质量，并进一步的更改输入从而使程序探索更多的分支，如此就有更大的概率可以触发深层次的漏洞。在实际使用过程中，AFL在开源软件的测试中发挥了极其重要的作用，其编译产生的代码运行效率极高，加上采用Fork-Server等技术提速使得AFL测试的速度可以达到接近原生执行的速度。

但是在IoT的背景下，这种模糊测试技术并不能直接被采用，原因有二。首先，IoT背景下的软件测试大多是已经编译完成的二进制文件，安全测试人员并不能取得软件的源代码，也就无法使用AFL的源码编译功能来插入插桩代码。AFL针对无法编译的情况，提供了基于QEMU模拟执行的功能，通过在运行时插入动态代码来实现与源码编译一样的功能。QEMU是一个模拟器，可以模拟执行多种架构的代码，通过对QEMU进行一定的定制化可以实现动态插桩的功能。但是这就引出来这个功能的另一个弊端，QEMU模拟执行将会极大的降低程序执行的速度，从而使Fuzzing的效率降低，大大限制了AFL的能力。除了这两点之外，还有一个也非常重要的问题，当今IoT设备使用的硬件平台大多是基于MIPS或者ARM的，而传统的模糊测试大多是传统PC平台环境下的X86架构，比如AFL用来源代码插桩的代码也只提供了X86与X86-64两种架构的，所以就使传统工具很难被使用到IoT平台的安全研究上。

在这样的背景下，我们希望可以找到一种针对AFL的改进，使其能更加快速的、针对基于ARM的IoT平台进行Fuzzing。通过这种改进，使得AFL能够适用于IoT平台的安全测试上。

* 1. 研究目标

综合上面提到的难点，为了达到类似于AFL编译执行的效果，我们需要解决的问题就有三个，第一是在目标的二进制找到AFL所使用的所有插桩点，这些点代表了程序的执行路径，只有找到这些路径才可以记录程序执行的覆盖率，AFL插桩的代码点就是程序的Basic Blocks的开头。第二是将AFL的插桩代码由X86翻译到ARM平台，上面提到AFL只提供了X86平台的插桩代码，因为底层操作系统都是Linux，我们可以借鉴其插桩代码来实现ARM平台的插桩代码。第三就是将插桩代码Patch进目标二进制文件，目标二进制是编译完成的并且我们没有对应的源代码，这个时候对其的修改就只能直接修改目标文件。

完成这三个目标，基本可以实现近似于目标程序在AFL下编译的效果，从而将传统PC平台上的测试方法迁移到IoT平台上，实现IoT平台上的应用Fuzzing测试。

* 1. 研究方法

通过传统程序逆向分析技术实现对程序Basic Blocks的识别，然后将翻译之后的插桩代码通过Patch手段打包进目标二进制文件中，实现AFL Fuzzing的近似Native执行。

1. **研究计划**

2.1．项目期限：

2.2．进度安排：

|  |  |  |  |
| --- | --- | --- | --- |
| **阶段** | **研究内容** | **时间** | **提交结果** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
|  |  |  |  |

1. **预期成果**

开发实现Patch框架，通过Patch框架实现插桩平台，完成插桩后实现高速的自动化Fuzzing测试。