

本科生毕业设计(论文)

基于模糊测试的协议脆弱性研究

AFLNetSpy: High-Throughput Fuzzing of

IoT Network Applications

|  |  |
| --- | --- |
| 学 院： | 网络空间安全学院 |
| 专 业： | 网络空间安全 |
| 班 级： | 12112002 |
| 学生姓名： | 常振轩 |
| 学 号： | 1120202439 |
| 指导教师： | 谭毓安 |

2024年5月10日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名： 日 期： 年 月 日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的,复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名： 日 期： 年 月 日

指导老师签名： 日 期： 年 月 日

基于模糊测试的协议脆弱性研究

摘　要

随着物联网的快速发展，基于Linux的固件数量迅速增长，其所面临的安全问题也日益严峻。对于传统的桌面程序，模糊测试是一项十分有用的漏洞发现技术。但由于固件程序特殊的执行环境，传统的模糊测试技术难以直接应用于固件程序。

本文从模糊测试的角度对协议脆弱性进行研究，提出利用QEMU插件系统获取灰盒模糊测试所需目标程序代码执行信息的方法，和利用网络请求探测和控制目标程序状态的方法。

基于上述两个方法，本文在AFLNet的基础上开发出针对固件网络应用的高效灰盒模糊测试框架AFLNetSpy，能够实现固件系统中HTTP和HTTPS服务程序的高效灰盒模糊测试，成功将AFLNet的灰盒模糊测试能力扩展到固件领域。

本文首先介绍模糊测试、固件安全、协议安全等领域的相关工作，之后介绍利用QEMU插件系统获取灰盒模糊测试所需目标程序代码执行信息和利用网络请求探测和控制目标程序状态的方法。然后介绍AFLNetSpy的系统架构和各模块设计要点，最后设计实验从正确性和效率两方面对AFLNetSpy进行分析。

关键词：灰盒模糊测试；固件安全；协议安全

AFLNetSpy: Greybox Fuzzing of IoT Network Applications

Abstract

In order to study……

Key Words: Greybox Fuzzing; IoT Security; Protocol Security

目　录

注：此处无需更改。阅后删除此文本框。

[摘　要 I](#_Toc8720747)

[Abstract II](#_Toc8720748)

[第1章 一级题目 1](#_Toc8720749)

[1.1 二级题目 1](#_Toc8720750)

[1.1.1 三级题目 1](#_Toc8720751)

[结　论 3](#_Toc8720752)

[参考文献 4](#_Toc8720753)

[附　录 6](#_Toc8720754)

[致　谢 7](#_Toc8720755)

注：在目录页面中点击鼠标右键，选择“更新域”，在弹出窗口中选择“更新整个目录”，确定即可自动生成目录。章、节标题和页码，字体：宋体，字号：小四，不加粗。阅后删除此文本框。

1. 绪论

**1.1** 研究意义

在当今信息技术高速发展的背景下，软件安全得到了广泛的关注。随着软件规模和复杂度的不断提高，依靠人工审查和手动测试来确保软件的正确性和安全性变得不再可行。因此，自动化软件测试技术尤其是模糊测试技术，应运而生，并在近年得到越来越多的关注。通过不断改进和利用模糊测试技术，人们在桌面应用如浏览器、网络服务、办公软件等程序中找到了大量有意义的错误和漏洞，证明了模糊测试技术的实用性。

同时，随着物联网的快速发展，基于Linux的固件数量迅速增长，这些固件广泛应用于各种物联网设备和嵌入式系统中。然而，固件程序面临的安全问题也日益严重。由于固件程序特殊的执行环境，传统的针对桌面应用的模糊测试技术难以直接应用于固件程序的测试。

...

**1.2** 研究意义

鉴于保证固件安全的重要性，将传统的用于桌面系统程序的模糊测试技术和工具，扩展到固件系统，对于固件安全领域的发展有极其重要的意义。本文提出基于QEMU的全系统仿真模式和插件功能，实现SUT代码执行信息收集和SUT状态控制的方法。并根据此方法，开发出支持对固件网络应用进行灰盒模糊测试的原型系统AFLNetSpy，成功将适用于桌面网络应用的灰盒模糊测试工具AFLNet，扩展到了固件网络应用领域，。

这一研究成果不仅为固件安全领域提供了新的测试工具和方法，有助于提高固件系统的安全性和可靠性，而且为后续该领域的研究提供了可供借鉴的思路，对于推动物联网技术和固件安全技术的进一步发展具有积极意义。

**1.3** 国内外研究现状

模糊测试领域，

固件模糊测试领域，

1.4 本论文组织结构

本论文的正文部分按照以下六个章节组织划分：

1. 第一章：绪论。重点介绍本论文的研究背景和研究意义，并阐明了模糊测试和固件模糊测试领域的国内外研究现状。同时介绍了本论文提出的新方法和开发出的AFLNetSpy原型系统。
2. 第二章：相关工作。本章重点介绍模糊测试领域的知名工具AFL，以及基于AFL二次开发得到的AFLNet和TriforceAFL。同时介绍DECAF和ISPRAS-QEMU两个涉及QEMU插件系统的项目。
3. 第三章：系统设计思路。本章重点介绍了本文提出的两个新方法：①利用QEMU插件系统获取进行灰盒模糊测试所需的目标程序代码执行信息；②利用网络请求探测和控制目标程序状态。同时介绍了系统设计相关的其它注意事项。
4. 第四章：系统详细设计。本章首先介绍了AFLNetSpy系统的整体架构，然后分别对各个模块的内部实现进行讲解。
5. 第五章：实验分析。本章从正确性和效率两个方面设计实验对AFLNetSpy原型系统进行分析。
6. 第六章：结论。本章根据前文的方法和实验结果，对研究成果进行总结，并指出不足之处，同时对未来进行展望。
7. 相关工作

2.1 AFL

AFL是Google开发的一个知名的灰盒模糊测试工具，通过收集测试进程的代码执行信息，并利用遗传算法，进行覆盖率引导的灰盒模糊测试。AFL支持两种使用模式，静态插桩模式和动态插桩模式。

静态插桩模式适用于有待测程序源码的情况。AFL对gcc等编译工具进行包装，得到afl-gcc等工具。使用AFL包装后的编译工具对待测程序进行编译时，AFL会在生成的程序中添加一些代码片段，用于在后续测试过程中收集代码执行信息，以及探测和控制程序运行状态，并在模糊测试过程中和afl-fuzz保持交互。静态插桩模式下，AFL的工作流程如下：

1. 首先使用afl-fuzz命令启动模糊测试后，afl-fuzz首先会创建一个子进程并执行待测程序，待测程序启动后作为Forkserver和afl-fuzz保持通信。
2. 之后的每一次测试前，afl-fuzz会通过管道请求Forkserver创建一个测试进程，并等待测试进程的进程号。
3. Forkserver接收到afl-fuzz的请求后，会创建测试进程，并将其进程号通过管道返回给afl-fuzz，实现同步。
4. afl-fuzz接收到测试进程号后，从测试队列选择或变异生成一个测试用例，并通过系统初始化阶段设置好的文件描述符将测试用例发送给测试进程。然后等到Forkserver返回测试进程的退出状态码，若定时器超时，则主动kill掉测试进程。测试进程执行过程中会不断更新trace\_bits数组，以记录代码执行情况。
5. Forkserver会等待其子进程即测试进程退出，并将退出状态码返回给afl-fuzz进行分析，实现同步。
6. afl-fuzz在接收到测试进程的退出状态码后，会检查通过共享内存和测试进程共享的trace\_bits数组的状态，并根据trace\_bits的状态和测试进程退出状态码对本次测试用例的价值进行评判，如果触发了新的执行路径，则将该测试用例保存至测试队列。然后回到第(2)步，重复(2)-(6)的过程，直到用户手动停止afl-fuzz。



动态插桩模式适用于没有待测程序源码，只有二进制程序的情况。AFL在动态插桩模式下的工作流程和静态插桩模式基本一致，但是这种情况下由于无法通过在待测程序中插入二进制片段，没办法探测和控制测试进程的状态，也无法测试进程的代码执行信息。为了解决这个问题，AFL引入并修改了用户模式的QEMU。修改后的QEMU可以发挥和静态插桩模式下Forkserver一样的作用，能够和afl-fuzz通过管道保持通信和状态同步。同时由于QEMU的翻译执行机制中翻译块的存在，代码执行信息的收集也变得可行。

另外需要注意的是，动态插桩模式的效率要比静态插桩模式要低，且动态插桩模式无法像静态插桩模式那样，根据跳转指令切分程序代码块，所得到的代码执行信息的意义不如静态插桩模式直观。但QEMU的使用使得测试不同架构下的二进制程序变得可行，扩大了AFL的适用范围。

2.2 AFLNet

AFLNet是Thuan Pham等人针对网络应用测试开发出的AFL扩展版。AFL只支持测试以文件或命令行的方式进行输入的程序，如果想要使用AFL测试某个网络应用，首先必须修改该应用的输入方式，使其能够从文件中读取网络请求。当没有源代码时，AFL就对该网络应用束手无策了。而AFLNet在AFL的基础上添加了网络通信支持，能够通过网络套接字直接向待测网络应用发送测试请求。同时，考虑到网络应用的错误往往需要一个序列的请求才能够触发，因此AFLNet不像AFL一次测试只发送一个请求，它支持在一个测试用例中存放多个请求报文，并在一次测试中按顺序发送它们。

AFLNet的工作流程的如图所示，和AFL的工作流程基本一致。区别主要有两点：

1. 一是第(4)步中AFLNet采用网络套接字即socket通信的方式向测试进程发送测试请求，然后接受响应，并提取响应中的状态码，状态码不仅可以用于协议状态机的分析，还能够和代码执行信息一起用来评估测试用例的价值和指导新测试用例的生成。
2. 二是由于网络应用通常处于无限循环状态，处理完一次测试用例后并不会自动退出，为此AFLNet支持两种策略：①一个测试进程只处理一个测试用例，在一次测试结束后，由afl-fuzz主动向测试进程发送SIGTERM信号以关闭测试进程；②允许一个测试进程处理多个测试用例，直到出现超时或者崩溃，再启动一个新的测试进程。本文开发的原型系统AFLNetSpy就采用了第二种策略。



同时，AFLNet提出一个新的协议推断算法，通过观察AFLNet和待测网络应用间交换的请求响应数据，能够一定程度上还原出通信采用的协议格式。因此，作为一个协议测试工具，它并不需要用户手动指定协议规范和报文格式，可以直接将用户收集到的客户端和待测服务器的通信流量中的请求作为种子，在重放请求的过程中，自动学习协议知识。不过，当初始测试用例较少时，自动获取的报文格式难免会存在误差，导致变异生成的大部分测试用例仍然是无效的。

另外需要注意的是，AFLNet在发送一次测试用例后，会尝试接收响应。但是，以HTTP服务器为例，当测试请求无效时，服务器会直接丢弃该请求报文，并不会返回响应。因此，相比于AFL，AFLNet需要用户额外手动设置一个合适的等待响应的最大时间。

2.3 TriforceAFL

TriforceAFL是NccGroup针对固件系统调用测试开发出的AFL扩展版。它创新性地成功将QEMU的系统模式和AFL结合起来，使得TriforceAFL能够对QEMU模拟启动的固件系统中的系统调用进行高效的灰盒模糊测试。



TriforceAFL的整体工作流程如图所示，和AFL工作流程的主要区别，也即TriforceAFL重点解决的问题有三个：

1. 一是测试用例的输入。在使用AFL对桌面应用程序进行的模糊测试中，能够通过dup()函数在父子进程之间，主要是afl-fuzz进程和测试进程之间，共享所需的文件描述符，并通过该文件描述符直接传递测试用例。然而运行在主机系统上的afl-fuzz，无法与运行在QEMU内部的客户机系统及其中的应用程序共享文件描述符,也就无法通过这种方式来传递测试用例。出于进行固件系统调用测试的目的，TriforceAFL采用的策略是，在固件系统中添加一个用户编写的driver程序，driver程序能够通过执行特殊的HyperCall指令与QEMU进行协同，而QEMU进程作为afl-fuzz的子进程，能够通过管道和afl-fuzz进程保持通信并共享文件描述符。于是，以QEMU为桥梁，客户机系统中运行的driver程序便能够接收到afl-fuzz发送的测试用例。
2. 二是获取测试用例执行结果并退出测试系统。driver在接收到测试用例后，能够启动一个子进程作为测试进程执行测试用例对应的系统调用，并获取其执行状态，然后通过特定的HyperCall指令将执行状态以QEMU作为媒介，返回给afl-fuzz进程。同时，driver还会通过另一条特定的HyperCall指令，通知当前用于测试的QEMU进程退出，QEMU进程退出后，其内部用于本次测试的客户机系统自然也就实现了退出。
3. 三是测试系统的状态恢复。由于系统调用测试的特殊性，异常的系统调用很可能导致客户机操作系统的崩溃。为保证模糊测试的持续运行，每一次测试结束后，必须重启客户机系统或设法恢复客户机系统状态至执行测试用例前。重启客户机系统耗时过长，显然不是一个可行的方案。TriforceAFL通过实现QEMU-System级别的Forkserver，实现了客户机系统状态的快照和恢复。在客户机系统启动完成并运行driver进程后，driver会通过一个特定HyperCall指令告知QEMU-System客户机系统已经就绪，然后QEMU-System就会保存当前的虚拟CPU状态，后续创建的每一个测试系统都将以此刻为运行起点。

需要注意的是，虽然NccGroup宣称依靠TriforceAFL可以实现“Fuzz Everything”，TriforceAFL也的确提供了创新性的思路，但是目前依靠TriforceAFL仅仅只能够实现固件系统调用级别的灰盒模糊测试，要想实现固件内应用程序级别的灰盒模糊测试，还需要额外的工作。原因如下：

1. 代码执行信息的收集：QEMU系统模式下，启动的客户机系统中往往同时运行着大量的应用程序，而QEMU只会根据当前虚拟CPU的状态来获取二进制代码块，并进行翻译和执行，无法直接判断出当前代码块属于客户机系统中的哪个进程。为此，TriforceAFL采用的策略是，通过修改固件的文件系统镜像和初始化脚本，保证客户机系统启动后其内部只有driver进程和其子进程即测试进程处在运行状态，然后通过统计高地址空间即操作系统使用的地址空间中的指令执行情况，获取测试用例对应的代码执行信息。显然，这种方式并不适用于固件内的普通应用程序。
2. QEMU级别的Forkserver：TriforceAFL通过QEMU级别的Forkserver来保证每一次测试结束后，下一次测试执行前，能够恢复客户机系统的状态。然而，经过测试，这种方式并不能够在恢复客户机系统状态的同时恢复客户机内各个应用程序的执行状态。本文以HTTP服务器为例进行了测试，当客户机内的HTTP服务器接收到网络请求，即在QEMU层监测到该服务器进程调用accept系统调用时，仿照TriforceAFL的实现方式，退出vcpu线程，在主线程中执行fork()命令，得到的子进程无法继续处理刚才接收到的网络请求，且有可能触发各类段错误。因此，对于固件内应用程序级别的模糊测试，采用QEMU级别的Forkserver并不是可行的方案。本文将提供一种针对固件内网络应用的替代方案。

2.4 DECAF-QEMU

DECAF是Andrew Henderson等人开发的一个动态可执行代码分析框架。DECAF为QEMU提供了一套易于使用的、事件驱动的插件机制，并基于该机制实现了实时的虚拟机内省即VMI功能。

尽管DECAF的主要目的是实现比特粒度、高准确性的TCG指令污点分析，但具备VMI功能的DECAF-QEMU也可作为独立组件用于其它目的，如监测特定指令、监测API执行情况、监测键盘输入情况等。

另外，需要注意的是，DECAF-QEMU所使用的QEMU版本较低，难以启动较新的固件镜像如OpenBMC等镜像，且对操作系统的内核结构存在依赖，当特定数据结构出现变动时需要重新进行修改适配。但其实现方法仍然具有很大的参考价值，本文开发出的原型系统AFLNetSpy所需的系统调用监测功能就借鉴了其实现思路。

2.4 ISPRAS-QEMU

ISPRAS-QEMU是Ivan Vasiliev等人受ISPRAS组织支持开发出的一个基于QEMU的非入侵式虚拟机插桩和内省框架。它是一个用于对虚拟机内二进制程序进行动态分析的轻量级框架，具备跨系统和非入侵即不修改目标二进制程序及其执行流程两个主要特点。

ISPRAS-QEMU通过建立分层的事件驱动的插件机制，使得它实现的整个VMI系统结构清晰且易于扩展。然而和DECAF-QEMU一样，每个插件都需要在程序启动后手动开启，而不能直接在命令行参数中进行设置，在某些场景下使用起来不够方便。

另外，和DECAF-QEMU相比，ISPRAS-QEMU不需要依赖操作系统内核结构的信息，而是通过ABI规范创建了一些可以恢复内核级别信息的工具，从而实现各个事件的监测和定位。这种策略使得ISPRAS-QEMU具备了不依赖系统板的特性，但同时也损失了一些功能和便利性，如只能通过页目录地址标识进程，以至于无法通过指定进程名称和进程号的方式来筛选属于特定进程的各类事件。

2.5 本章小结

本章介绍了模糊测试和利用QEMU构建VMI系统的相关工作。其中，模糊测试方面，介绍了AFL、AFLNet、TriforceAFL三者的工作流程、各自特点及注意事项。利用QEMU构建VMI系统方面，介绍了DECAF-QEMU和ISPRAS-QEMU两项通过构建QEMU插件系统实现VMI功能的工作及其各自特点和注意事项。

本文将基于上述相关工作，进行一定的改进和扩展。上述相关工作中的AFLNet和TriforceAFL等工具对于固件内普通应用程序的灰盒模糊测试束手无策，而本文基于AFLNet和修改后的QEMU开发出的AFLSpyNet系统能够实现固件内网络应用的灰盒模糊测试。第三章和第四章将分别详细介绍AFLNetSpy的设计要点和模块细节。

另外相较于DECAF-QEMU和ISPRAS-QEMU实现的VMI系统，本文的主要改进点包括：

1. 基于最新稳定版的QEMU进行开发，保证系统能够支持较新的固件镜像如OpenBMC等镜像的启动。
2. 基于QEMU官方提供的插件机制进行开发。QEMU官方从4.2版本起开始提供一套插件机制，目前已经比较稳定，但功能较少，需要进行二次开发。使用QEMU官方的插件机制，而不是像DECAF-QEMU或ISPRAS-QEMU一样自行实现或使用第三方的插件机制，可以保证系统稳定性和对QEMU版本的向后兼容性。
3. 系统设计要点

本章将讲解AFLNetSpy系统的设计要点：①目标进程代码执行信息的获取；②目标进程的状态探测和控制。

3.1 代码执行信息的获取

要想进行固件内应用程序的灰盒模糊测试，首先需要能够获取目标进程的代码执行信息。

要实现这个任务，第一步是设法区分不同进程的指令。Linux系统下，每启动一个进程，会为其分配一个页目录，页目录地址和进程是一对一的对应关系，因此可以通过页目录地址来标识不同的进程。且当前正在执行的进程的页目录地址会被存放在VCPU的特定页目录地址寄存器内，根据翻译块执行时VCPU页目录地址寄存器的值，即可判断当前翻译块中的指令属于哪一个进程。

第二步是要设法获取目标进程对应的页目录地址。考虑到目标进程为网络应用，通过向客户及系统内的目标进程发送一个网络请求，然后在QEMU中监测接下来触发Accept系统调用时VCPU的页目录地址寄存器，即可获得目标进程对应的页目录地址。之后在QEMU中就能够根据当前执行的翻译块对应的页目录地址是否和目标进程的页目录地址相同，来判断当前翻译块中的指令是否属于目标进程，从而筛选和获取目标进程的代码执行信息了。

3.2 目标进程的状态探测和控制

要想进行固件内应用程序的灰盒模糊测试，还要实现目标进程的状态探测和控制。

目标进程的状态探测，包括：

1. 判断客户机系统和目标进程是否启动完毕：通过监测execve等用于创建新进程的系统调用及其参数，即可根据进程名称判断目标进程是否启动。同时，可以通过进程名称选定一个大概率最后启动的进程，当该进程启动时，认为客户机启动完成。
2. 每次测试结束后检查目标进程是否仍处于运行状态：由于目标进程是网络服务器，除了检查它返回给afl-fuzz的响应中的状态码，还应在每次测试结束后检查目标进程的存活状态，以保证当目标进程崩溃时及时得到通知。本文采取的策略是通过发送由用户编写的针对目标进程的TEST\_ALIVE网络请求，根据返回情况判断目标进程是否仍处于运行状态。

目标进程的控制，包括：

1. 目标进程的退出：经过测试，当外部请求触发客户机内目标进程的一个段错误时，会触发异常信号，目标进程会等待客户机系统处理该信号，并关闭目标进程。然而，由于QEMU的实现机制，异常信号不会立刻得到处理，目标进程也不会迅速被关闭，从触发段错误到目标进程成功退出，需要4分钟左右。这是难以接受的。因此，本文提出一种主动探测式的方法，通过TEST\_ALIVE请求探测目标进程状态，若为成功返回，则认为目标进程出现了CRASH，并向Spy-Agent程序发送强制杀掉目标进程的命令，从而实现目标进程的快速退出。其中，Spy-Agent是本文实现的一个需要内置于客户机系统内部运行，负责接收外部请求实现远程命令执行的网络程序。其实现细节会在第四章进行详细介绍。
2. 目标进程的重新启动：为保证模糊测试的持续运行，在触发目标进程的崩溃并退出目标进程后，需要重新启动新的目标进程来继续模糊测试。这可以通过向客户及系统内的代理程序Spy-Agent发送重启目标进程的shell命令来实现，如systemctl restart $TARGET\_SERVICE等。另外，还需要注意的是，现代操作系统及服务管理程序如systemd等，可能存在避免服务器频繁异常重启的保护机制，因此可能需要在强制退出目标进程后，重启目标进程前，重置该网络服务的错误状态信息，如使用systemd的systemctl reset-failed命令等。
3. 系统详细设计

4.1 整体架构

AFLNetSpy的整体架构如图所示。其中trace\_enabled和next\_step与trace\_bits一样，是afl-fuzz和QEMU-SPY通过共享内存方式共享的两个变量，用于控制和同步trace\_bits的记录。QEMU-Spy为经过扩展的QEMU和AFL-SPY插件组成的具有VMI功能的系统。SpyAgent进程为本文实现的能够进行远程命令执行的网络服务进程。



结合上图，AFLNetSpy的整体工作流程如下：

1. 首先afl-fuzz进程创建一个子进程用来运行QEMU，QEMU启动客户机系统前会加载通过命令行参数指定的AFL-SPY插件libaflspy.so，并注册所需的各类回调函数。启动后的客户及系统内，包含待测试的目标网络进程，和提前内置或后续传输进客户机系统的SpyAgent进程，该进程能够接收外部的网络请求并执行请参数中的shell命令，可以实现快速关闭和重启目标进程。
2. 初始化测试目标，包含三个步骤：①detect\_agent，向SpyAgent进程发送用户编写的TEST\_AGENT请求，AFL-SPY插件在监测到该请求时，会将SpyAgent进程的页目录地址设为AgentCtx，并将该值通过管道返回给afl-fuzz进程；②detect\_target，向目标测试进程发送用户编写的TEST\_ALIVE请求，AFL-SPY插件在监测到该请求时，会将目标进程的页目录地址设为TargetCtx，并将该值通过管道返回给afl-fuzz进程；③restart\_target，由于重启后的目标进程的页目录地址会发生改变，需要重新确定。因此该函数分为两步，一是向SpyAgent进程发送用户编写的RESTART\_TARGET请求，SpyAgent接收到该请求后，会提取请求参数中的shell命令并执行，从而实现目标进程的重启，同时AFL-SPY插件在监测到该请求时会将TargetCtx的值设为0，表明原TargetCtx已失效，需要重新定位。二是执行②中提到的detect\_target函数，从此实现TargetCtx的更新。
3. 发送测试请求并接收响应，包含四个步骤：①激活trace\_enabled，通知AFL-SPY插件，对接下来的目标进程的翻译块进行记录；②send\_over\_request是AFLNet提供的发送测试请求并接受响应的函数，本文对其进行了扩展，以支持HTTPS通信；③等待next\_step信号，AFL-SPY插件在监测到目标进程执行特定系统调用如sendto、sendmsg时，将next\_step的值设为1，表示此次测试请求已经处理完毕可以开始准备下一次测试了。afl-fuzz进程会轮询next\_step的值，直到该值为1或达到最大轮询次数，则进入下一步；④关闭trace\_enable，通知AFL-SPY停止记录翻译块的执行信息，保证得到的trace\_bits的数据足够干净。
4. 发送TEST\_ALIVE请求，检查处理上次测试请求后目标测试进程的存活状态，若未正常返回，则认为上次的测试请求触发了测试进程的CRASH，然后使用restart\_target函数重启一个新的测试进程，保证测试能够继续进行。若正常返回，说明上次的测试请求没能触发测试进程的CRASH，出于性能考虑，可以不重启测试进程。
5. afl-fuzz获取AFL-SPY插件更新过的trace\_bits，进行后续分析。然后生成新的测试请求，重复(3)-(5)步，直到用户手动停止测试。

下面介绍QEMU-SPY系统的详细设计和实现方法。

4.2 QEMU-SPY

本节介绍QEMU-SPY系统的详细设计。QEMU-SPY系统由QEMU和AFL-SPY插件组成。



通过在QEMU官方的插件机制基础上进行扩展，实现全系统模拟模式下的系统调用监测，并将系统调用信息传递给AFL-SPY插件进行处理。

并在每个翻译块中注入回调函数



1. 实验分析

1.1.1 三级题目

正文……

正文部分：宋体、小四；正文行距：22磅；间距段前段后均为0行。【阅后删除此段】

图、表居中，图注标在图下方，表头标在表上方，宋体、五号、居中，1.25倍行距，间距段前段后均为0行，图表与上下文之间各空一行。【阅后删除此段】

图-示例：【阅后删除此段】



图1-1 标题序号

表-示例：【阅后删除此段】

表1-1 统计表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 项目 | 产量 | 销量 | 产值 | 比重 |
| 手机 | 1000 | 10000 | 500 | 50% |
| 计算机 | 5500 | 5000 | 220 | 22% |
| 笔记本电脑 | 1100 | 1000 | 280 | 28% |
| 合计 | 17600 | 16000 | 1000 | 100% |

公式标注应于该公式所在行的最右侧。对于较长的公式只可在符号处（+、-、\*、/、≤≥等）转行。在文中引用公式时，在标号前加“式”，如式（1-2）。【阅后删除此段】

公式-示例：【阅后删除此段】

 (1-1)

结　论

注：此处无需更改。阅后删除此文本框。

本文结论……。

结论作为毕业设计（论文）正文的最后部分单独排写，但不加章号。结论是对整个论文主要结果的总结。在结论中应明确指出本研究的创新点，对其应用前景和社会、经济价值等加以预测和评价，并指出今后进一步在本研究方向进行研究工作的展望与设想。结论部分的撰写应简明扼要，突出创新性。【阅后删除此段】

结论正文样式与文章正文相同：宋体、小四；行距：22磅；间距段前段后均为0行。【阅后删除此段】

参考文献

注：此部分蓝色字体为注释，阅后可删除；黑色字体为具体示例。

阅后删除此文本框。

注：此处无需更改。阅后删除此文本框。

参考文献书写规范

参考国家标准《信息与文献参考文献著录规则》【GB/T 7714—2015】，参考文献书写规范如下：

1. 文献类型和标识代码

普通图书：M 会议录：C 汇编：G 报纸：N

期刊：J 学位论文：D 报告：R 标准：S

专利：P 数据库：DB 计算机程序：CP 电子公告：EB

档案：A 舆图：CM 数据集：DS 其他：Z

2. 不同类别文献书写规范要求

**期刊**

[序号] 主要责任者. 文献题名[J]. 刊名, 出版年份, 卷号(期号): 起止页码.

[1] 余雄庆. 飞机总体多学科设计优化的现状与发展方向[J]. 南京航空航天大学学报, 2008, 40(4): 417-426.

[2] Hajela P, Bloebaumj C L, Sobieszczanski-Sobieski J. Application of Global Sensitivity Equations in Multidisciplinary Aircraft Synthesis[J]. Journal of Aircraft, 1990, 27(12): 1002-110.

**普通图书**

[序号] 主要责任者. 文献题名[M]. 出版地: 出版者, 出版年: 起止页码.

[3] 张伯伟. 全唐五代诗格会考[M]. 南京: 江苏古籍出版社, 2002: 288.

[4] O’BRIEN J A. Introduction to information systems[M]. 7th ed. Burr Ridge, III: Irwin, 1994.

**会议论文集**

[序号] 主要责任者．题名:其他题名信息[C]. 出版地: 出版者, 出版年.

[5] 雷光春. 综合湿地管理: 综合湿地管理国际研讨会论文集[C]. 北京: 海洋出版社, 2012.

**专著中析出的文献**

[序号] 析出文献主要责任者. 析出题名[M]//专著主要责任者. 专著题名. 出版地: 出版者, 出版年: 起止页码.

[6] 白书农. 植物开花研究[M]//李承森. 植物科学进展. 北京: 高等教育出版社, 1998: 146-163.

**学位论文**

[序号] 主要责任者. 文献题名[D]. 保存地: 保存单位, 年份.

[7] 张和生. 嵌入式单片机系统设计[D]. 北京: 北京理工大学, 1998.

[8] Sobieski I P. Multidisciplinary Design Using Collaborative Optimization[D]. United States -- California: Stanford University, 1998.

**报告**

[序号] 主要责任者. 文献题名[R]. 报告地: 报告会主办单位, 年份.

[9] 冯西桥. 核反应堆压力容器的LBB分析[R]. 北京: 清华大学核能技术设计研究院, 1997.

[10] Sobieszczanski-Sobieski J. Optimization by Decomposition: A Step from Hierarchic to Non-Hierarchic Systems[R]. NASA CP-3031, 1989.

**专利文献**

[序号] 专利所有者. 专利题名:专利号[P]. 公告日期或公开日期[引用日期]. 获取和访问路径. 数字对象唯一标识符.

[11] 姜锡洲. 一种温热外敷药制备方案: 881056078 [P]. 1983-08-12.

**国际、国家标准**

[序号] 主要责任人. 题名: 其他题名信息[S]. 出版地: 出版者, 出版年: 引文页码.

[12] 全国信息与文献标准化技术委员会. 文献著录: 第4部分 非书资料: GB/T 3792.4-2009[S]. 北京: 中国标准出版社, 2010: 3.

**报纸文章**

[序号] 主要责任者. 文献题名[N]. 报纸名, 年(期): 页码.

[13] 谢希德. 创造学习的思路[N]. 人民日报, 1998-12-25(10).

**电子文献**

[序号] 主要责任者. 电子文献题名[文献类型/载体类型]. (发表或更新日期) [引用日期]. 获取和访问路径. 数字对象唯一标识符.

[14] 姚伯元. 毕业设计(论文)规范化管理与培养学生综合素质[EB/OL]. [2005-02-02]. 中国高等教育网教学研究.

关于参考文献的未尽事项可参考国家标准《信息与文献参考文献著录规则》（GB/T 7714—2015）

附　录

注：此处无需更改。阅后删除此文本框。

附录相关内容…

附录是毕业设计（论文）主体的补充项目，为了体现整篇文章的完整性，写入正文又可能有损于论文的条理性、逻辑性和精炼性，这些材料可以写入附录段，但对于每一篇文章并不是必须的。附录依次用大写正体英文字母A、B、C……编序号，如附录A、附录B。【阅后删除此段】

附录正文样式与文章正文相同：宋体、小四；行距：22磅；间距段前段后均为0行。【阅后删除此段】

致　谢

注：此处无需更改。阅后删除此文本框。

值此论文完成之际，首先向我的导师……

致谢正文样式与文章正文相同：宋体、小四；行距：22磅；间距段前段后均为0行。【阅后删除此段】