# CSTE v1.0 动态分析工具设计文档

内存安全基准原型系统

### 工具简介

动态分析工具是基于QEMU开发的一个原型验证工具。

QEMU是一套由Fabrice Bellard编写的模拟处理器的开源软件。它通过动态翻译执行快速的模拟一个处理器，并达到很好的模拟速度。QEMU有两种操作模式：

* 全系统模拟：在此模式，QEMU模拟整个操作系统，包括一个或者多个处理器和多个外设。在这个模式QEMU可以运行不同的操作系统不需要重启电脑或者调试系统代码。
* 用户态模拟：在此模式，QEMU运行不同于宿主机CPU架构的不同架构的进程。因此，QEMU用户态可以简化不同架构程序的交叉编译和交叉调试。

CSTE v1.0版本的动态分析工具是基于QEMU的用户态模拟模式开发而来，此动态分析工具可以运行不同功能需求的程序，并且可以根本不同的需求监控程序的运行。此动态分析工具中开发实现了如粗粒度CFI、细粒度CFI、影子栈等多种防御机制，用户可以根据需求监控程序运行得到程序控制流、不同转移指令的跳转及源地址和目的地址等、各种指令执行数量和匹配配件以及配件链等信息。用户可以选择不同的防御机制来测试用户程序的攻击和漏洞，并且组合输出程序运行过程中的各种信息，方便分析程序行为。

### 安装环境

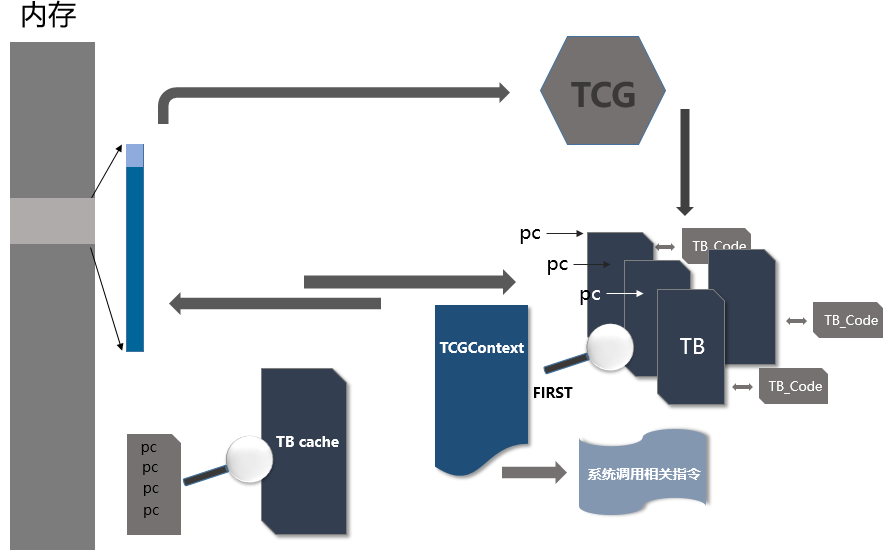
QEMU是一个跨平台的开源软件，因此QEMU可以安装在所有的Linux平台，OS-X，Win32和多种UNIX平台。

此动态分析工具的实验环境为Ubuntu 16.04 64bit，Intel Core i5-3470 CPU，4GB RAM。

### 基本原理

QEMU模拟处理器动态执行的核心是动态翻译模块TCG（Tiny Code Generator）。TCG可以使架构不同的程序跨平台执行，例如ARM架构编译链接的程序通过TCG可以在X86、MIPS和OpenRisc等架构平台上执行。TCG也分为前端和后端，目标指令通过TCG前端转换成TCG IR指令，之后这些指令通过TCG后端转换为 对应宿主机架构的机器指令。

QEMU动态执行时是以代码块的粒度来翻译执行用户程序，每一个代码块称之为TB块，而TB块是按照ret、call、jmp，syscall转移指令和最大指令条数来划分成块。TCG的动态翻译执行原理如下图一所示。程序载入QEMU的用户空间后，QEMU会模拟处理器的执行过程取指执行，QEMU在取出程序执行的第一条指令后首先到TB cache结构（专门存储程序动态翻译后的TB块）中查找是否存在TB块首地址为该指令地址的TB块，若没有找到则通过TCG动态翻译生成一个TB块，并将TB块的相关信息记录在TCGContext中。TCG在翻译执行后将生成的TB块加到TB cache中，并将TB块交给QEMU执行。若TB cache中存在翻译好的TB块则直接取出执行。



图一 TCG翻译执行原理

### 需求分析

CSTE v1.0版本动态分析工具需满足粗粒度CFI的设计规则，因此此动态分析工具的设计必须满足一下几点：

* 监控并标记jmp指令，并判断jmp的跳转地址必须为所有函数头地址或本函数内部；
* 监控并标记call指令，并判断call的跳转地址必须为所有函数头地址；
* 监控并标记ret指令，并判断ret的跳转地址必须为所有call指令的下一条地址；
* 记录每一对间接转移指令间的指令数，若指令数小于x，则此代码片段判定为一个配件，若连续出现y个这样的代码片段则判定为一个配件链；
* 输出所有间接跳转地址的源地址和目标地址，间接指令间隔的指令数量，各个类型的间接指令的数量。

### 动态分析实现

QEMU翻译执行的过程中，TCG当遇到跳转指令如jmp、call等就会将指令流截断为一个TB块再将此TB块翻译成宿主机架构机器码的TB块。因此，QEMU生成的TB块中唯一且一定存在一个跳转指令，因此，在TCG翻译TB块的过程中标记带有间接jmp、call、ret的TB块，并存在TB cache中。

QEMU从TB cache查找TB块执行时，检测要执行的TB块是否含有标记，若检测到标记则在执行下一个TB块的时候得到间接指令的源地址和目标地址，并统计与上一个间接指令的间隔的指令数，若指令数量小于x（5），则判定代码片段为配件，检测若连续出现y（10）个这样的配件认定为配件链并退出程序。

根据静态分析生成的函数信息表（.func）和指令信息表（.call），在执行含有标记的TB块的下一个TB块的时判断跳转目标地址和信息表中的数据是否一致，不一致则退出程序。

此动态分析工具提供了如下表1的参数，用户可以根据自己的需求配置不同的参数：

|  |  |  |
| --- | --- | --- |
| **OPTION** | **ARGS** | **DESCRIPTION** |
| -M | instruction | monitor jmp/call/ret instruction and output information |
| -jpath | Path | set jmp CFG file path |
| -cpath | path | set call CFG file path |
| -rpath | path | set ret CFG file path |
| -enable-coarse-CFI |  | enable Coarse-grained CFI mechanism |
| -enable-fine-CFI |  | enable Fine-grained CFI mechanism |
| -ss |  | set shadow stack mechanism |

表1 参数表

假设用户执行demo程序，需要监控jmp指令并使用粗粒度CFI机制来进行防御，因此配置的参数如下：

*qemu-x86\_64 -M jmp -jpath ~/test/demo.func -enable-coarse-CFI demo*

假设用户想要分析demo程序的间接跳转指令，则配置参数如下：

*qemu-x86\_64 -M jmp -M call -M ret demo*

### 测试

用此动态分析工具结合静态分析的结果运行CSTE漏洞库提供的16个漏洞攻击程序，在此次测试中，我们开启了粗粒度CFI防御机制，最后的结果如表2所示：



表2 测试结果

从表中我们可以看到动态分析工具的粗粒度CFI机制可以防御大部分的cop、rop、jop等攻击，但是无法防御double free、hoverflow、ret to lib等攻击，因为它们可以欺骗或者并没有触发粗粒度CFI的防御规则，因此动态分析工具在执行漏洞程序时无法防御此攻击。

### 结论

基于QEMU的动态分析工具实现了多种防御机制，并且通过测试验证了防御机制的效用，并且动态分析工具提供许多参数，可以满足用户分析程序行为的多种需求，因此满足CSTE v1.0版本的设计要求。

### 讨论

此动态分析工具在分析程序行为时目前只记录了间接跳转指令的源地址和目标地址及配件信息，对于分析程序更多的行为提供的信息仍然比较少，因此还要提供记录执行指令的总数量和间接跳转指令的数量，记录分支跳转指令的数量和对应的条件参数，记录库函数调用的次数和参数，系统调用的次数和参数。