__kingzone__的专栏 学习数据挖掘~

个人资料

kingzone_2008

访问： 296744次
积分： 5009
等级： BLOG 6
排名： 第2375名

原创： 109篇 转载： 15篇
译文： 19篇 评论： 86条

BitBlaze（四） - 动态分析组件TEMU

分类： Security&Privacy 2013-05-02 11:20 1727人阅读 评论(0) 收藏 举报

BitBlaze TEMU 动态分析 二进制代码分析

4 TEMU：动态分析组件

这一部分主要介绍TEMU，BItBlaze平台的动态分析组件，描述其提取操作系统级语义的组件，执行系统全局的动态污点分析，以及它的插件和实现。

4.1 TEMU概述

TEMU是一个基于全系统仿真器QEMU开发的全系统的动态二进制分析平台。在这个仿真器上运行一个完整的系统（包括操作系统和应用程序），并对相关二进制代码的执行进行细粒度的观察。TEMU是基于一下考量而采用全系统方法的：

-许多分析都需要对二进制代码进行细粒度的分析（如：指令级别）。通过动态的分析模拟代码，全系统模拟器确保了细粒度的分析。

-全系统仿真器为我们一个整个系统的视图。全系统的视图使我们能够分析操作系统内核以及多个进程间的交互。另一方面，许多其他的二进制分析工具(如，Valgrind，DynamoRIO，Pin)只提供了一个局部的视图（如，一个单用户模式进程的视图）。这对于分析恶意代码更为重要，因为许多攻击涉及到多个进程，而且诸如rootkits的内核攻击变得越来越普遍。

-全系统仿真器有效地隔离开了分析组件和待分析代码。因此，待分析代码更难干扰分析结果。

TEMU的设计得益于一下难点和考量：

-全系统仿真器只为我们提供硬件级别的视图，然而我们更希望得到一个软件级别的视图以获取有意义的结果。因此，我们需要一个从仿真系统中提取操作系统级语义的机制。例如，我们需要知道那个进程正在执行，一条指令来自那个模块。

-另外，许多分析过程需要推理特定的数据对其数据源的依赖性以及它如何在系统中传播。全系统的动态污点分析可以提供这项功能。

-我们需要提供一个涉及良好的编程接口（如API）以使用户能在此基础上开发他们自己的插件。

基于这些考量，我们设计的TEMU架构如图5所示。SemantisExtractor用于从Emulated System中提取操作系统级语义信息。Taint Analysis Engine用于进行动态污点分析。TEMU API可以方便用户开发自己的插件（TEMU plugins）。TEMU是在Linux系统下实现的，它可用于分析Windows 2000，Windows XP和Linux系统中的二进制代码。下面将分别介绍三个组成部分。

Security&Privacy (5)
笔试题 (7)
Python (0)
ACM (30)
Linux (3)
Maven (3)
汇编 (1)

文章存档

2015年05月 (1)
2015年04月 (2)
2015年03月 (4)
2015年01月 (1)
2014年12月 (1)

展开

阅读排行

GitHub入门：如何上传与 (16046)
Java：String、StringBuf (11815)
掌握VS2010调试 -- 入门 (10390)
jQuery动态添加删除sele (8884)
Java：Date、Calendar、 (7928)
Java List与数组之间的转 (6680)
Java连接SQL Server：j (6627)
Spring MVC视图层：thy (6152)
数据挖掘（六）：预测 (5912)
Nexus创建本地Maven仓 (5527)

Favorites

Java 6 Doc
Java SE 6 API
Java 6 中文Doc
C++ Reference
POJ
ZOJ
HDOJ
九度OJ
LeetCode OJ

编程语言

性能排名
TIOBE 流行度
LangPop 流行度
透明排行榜
Github top lang
Computer Programming Language Statistics
DB Engines

数据库与数据挖掘

Oracle Online Documentation
ACOUG
Ask Tom
ERI - Data Mining & Predictive Analytics

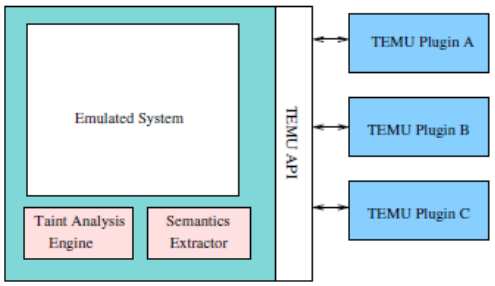

Fig. 5. TEMU Overview

4.2 语义提取器

语义提取器用于提取仿真系统的操作系统级语义信息，包括进程，模块，线程，以及符号信息。

进程和模块信息。我们需要知道当前执行的指令来自哪个进程，哪个线程以及哪个模块。某些情况下，指令可能是在堆上动态生成和执行的。

在内存中保存地址和模块的映射需要客户操作系统的信息。Windows和Linux采用了两种不同的解决方案来提取进程和模块信息。

Windows系统中，我们开发了一个称作module notifier的内核模块。通过把此模块加载到客户操作系统中来收集内存映射信息。Module notifier注册了两个回叫例程。第一个回叫例程进程创建或删除是被调用。第二个回叫例程在一个新模块加载时被调用，它会收集新模块在虚拟内存占用的地址范围。另外，module notifier获得每个进程的CR3寄存器的值。由于CR3寄存器保存着当前进程页表的物理地址，因而它们是各不相同的。上述所用信息都通过一个预定义的I/O端口传递给TEMU。

对于Linux系统，我们可以直接从外部读取进程和模块信息，因为我们知道相关的内核数据结构，而且相关符号的地址也被导出到system.map文件中。为了在执行时保存进程和模块信息，我们钩住了几个内核函数，如do_fork和do_exec。

线程信息。在Windows系统中，需要获取当前线程信息以支持对多线程应用程序和操作系统内核的分析。Windows中当前线程的数据结构映射到了一个众所周知的虚拟地址处，因而这比较直观。目前，还没有支持Linux的版本，可能后续版本会实现这一功能。

符号信息。对于PE（Windows）格式的二进制文件，也要分析它们的PE头并提取导出符号和偏移。确定了所有模块的位置后，我们就可以通过基址加偏移计算出每个符号的绝对地址。这一特征是非常有用的，因为所有的windows API和内核API都是由其宿主模块导出的。符号信息传递了重要的语义信息，因为从一个函数名，我们可以得知它的用途，输入参数，输出参数以及返回值。符号信息也使监视一个函数变得更加方便-只需要给出模块名和函数名，而不用指明其实际地址。TEMU映射到其实际地址。

目前，此功能只支持PE文件。后续版本会支持ELF（Linux）二进制文件。

4.3 污点分析引擎

我们的动态污点分析在本质上与先前的一些系统是相似的。但是我们的目的是支持多种不同的应用，因而我们的设计和实现是最完整的。例如，以往的方案要么只支持单进程，要么不支持内存交换和磁盘。

影子内存。影子内存用来存储物理内存、CPU寄存器、硬盘和网络接口缓冲区每个字节的污点状态。每个标记的字节都与一个存储污点源和其他TEMU插件用到的信息的小型数据结构相关联。影子内存组织成一个类似页表的结构以提高内存利用率。通过对硬盘使用影子内存，系统可以在标记数据被换出后继续跟踪它，也可以跟踪存入文件后又被读入的标记数据。

污点源。一个TEMU插件负责把污点源导入系统。TEMU从硬件开始支持污点输入，比如键盘，网络接口，硬盘。TEMU也可以标记高级抽象的数据对象（如，函数调用的输出，特定应用程序或操作系统内核的数据结构）。

污点传播。当数据源被标记后，污点分析引擎会监控操作这个数据的每条CPU指令和DMA操作以判定污点的转播过程。污点分析引擎通过数据转移指令、DMA操作、算术操作和查表来传播污点。有些指令（如，xor eax，eax）产生的结果与操作数无关，污点分析引擎并不传播这些指令中的污点。

注意，TEMU插件可能根据其需求遵守不同的策略。例如，对某些程序，不需要传播查表产生的污点。因此，在

污点传播期间，污点分析引擎让TEMU插件决定采用何种策略。

这种设计使TEMU更加灵活。TEMU插件可以指定不同的污点源，为每个标记的字节保存任意的记录，跟踪多个污点源，采用多种策略。

4.4 TEMU API和插件

为了便于用户使用TEMU提供的功能，我们定义了一系列函数和回叫。利用这些接口，用户可以开发自己的插件并在运行时加载到TEMU来进行分析。目前，TEMU提供以下功能：

-查询和设置内存区和CPU寄存器的值。

-查询和设置内存或寄存器的污点信息。

-在一个函数进入和退出时设置一个钩子，删除钩子。TEMU插件可以使用此接口监控用户和内核函数。

-查询操作系统级语义信息，如当前进程，模块，线程。

-保存和加载仿真系统状态。这个接口用于实现不同机器状态间的转换以实现更高效的分析。例如，此接口使多路搜索更加高效，因为我们可以保存分支点的状态，然后搜索一条路径，之后可以加载先前保存的分支点再执行另一条分支，而不必重新执行程序。

TEMU为多种事件定义了回叫，包括（1）基本块地进入和退出；（2）一条指令的进入和退出；（3）污点传播时；（4）内存被读或写时；（5）寄存器读或写时；（6）诸如网络和磁盘输入输出的硬件事件。

下列TEMU插件是使用这些接口和回叫实现的：

-Panorama

-HookFinder

-Renovo:从加壳的可执行文件中提取未加壳的代码的插件。

-Polyglot

-Tracecap：记录指令执行踪迹的插件。

-MineSweeper

-BitScope

-HookScout

4.5 TEMU的实现

TEMU由C和C++实现。由于C的高效性，性能要求高的代码由C实现，而面向分析的代码由C++编写，这样能很好的利用C++ STL中的抽象数据类型和类型检查。例如，污点分析引擎把代码片段插入到QEMU微操作中以检查和传播污点信息。由于污点分析对性能要求很高，因此它用C语言实现。另一方面，我们用C++ STL中的string，list，map等抽象数据类型实现语义提取器，使映射保持在操作系统级视图和硬件视图之间。TEMUAPI由C定义。这使得用户可以在开发自己的插件时既可以使用C,也可以使用C++。TEMU内核由约37000行代码组成，不包括QEMU的代码（约306000行）。TEMU插件包含约134000行代码。

上一篇 数据仓库之三：数据仓库环境--Inmon
下一篇 BitBlaze（五） - 应用及相关工作

BitBlaze五 - 应用及相关工作
使用gdb调试temu插件
插桩技术
用Visual studio2012在Windows8上开发内核中隐藏进程
我的AIX入门之路完整版

【精品课程】HTML 5视频教程系列之JavaScript学习篇
【精品课程】从此不求人:自主研发一套PHP前端开发框架
【精品课程】HTML 5移动开发从入门到精通
【精品课程】C语言及程序设计初步
【精品课程】HTML 5全掌控

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap