Unicorn：下一代CPU仿真器框架

www.unicorn-engine.org

NGUYEN Anh Quynh <aquynh-at- gamil.com>

Syscan360 Beijing –October 21st, 2015

自我介绍

- Nguyen Anh Quynh (aquynh -at- gmail.com)

  计算机科学博士，安全研究员

- 操作系统、虚拟机器、二进制分析、入侵取证等
- Capstone独立框架（capstone-engine.org）

议程

1. CPU 仿真器
   - 背景介绍
   - 现存CPU仿真器的问题

2. Unicorn引擎：要求、想法、设计和实现
   - Unicorn的目标
   - 设计和实现
   - 利用Unicorn API编写应用程序

3. 现场演示

4. 总结

CPU 仿真器

定义

- 模拟物理CPU— 仅使用软件
- 仅集中于CPU操作，但是忽略机器设备

应用程序

- 无需有真实的CPU去模拟代码
  - 跨构架的主机游戏模拟器
- 安全地分析恶意代码，检测病毒签名
- 逆向时确认代码语义

举例

模仿理解代码语义

```
mov eax, 0x30
mov esi, ecx
mov ebx, 0x45
add ecx, 0x78
sub ebx, 0x22
inc ecx
dec eax
mov ecx, eax
and ebx, 0x99
sub eax, 0x23
xor esi, esi
jz $_l0

_l0:
shl ecx, 1
add eax, ebx
xor edx, edx
inc ebx
jmp $_l1
nop

_l1:
shr ecx, 1
sub ecx, 0x11
inc eax
and esp, -0x10
dec eax
```

```
mov edx, 0x0
mov esi, 0x0
mov ebx, 0x2
and esp, -0x10
mov eax, 0xd
mov ecx, 0x1e
```

CPU仿真器的内部构造

给定的以二进制形式输入代码

- 解码二进制到单独的指令
- 每一条指令的精确模拟
  - 指令集和CPU构架的手册是必须的
  - 处理内存访问和I/O请求
- 每一步后更新CPU上下文（寄存器和内存等）

举例模拟X86 32位指令

- 例如：50 → push eax
  - 加载eax寄存器
  - 复制eax值到栈底
  - 减少esp除以4，和更新esp

- 例如：01D1 → 添加eax, ebx
  - 加载eax和ebx寄存器
  - 添加eax和ebx值，然后复制结果到eax
  - 相应的更新标志OF、SF、ZF、AF、CF、PF

创建CPU仿真器的挑战

大量的工作！

- 对CPU架构有很好的理解
- 对指令集有很好的理解
- 带有各种副作用的指令（有些是未文档的，例如：Intel X86）
- 很难支持现有的所有类型的代码

这个CPU仿真器好吗？

- 多架构？
  - X86、 Arm、Arm64、Mips、PowerPC、Sparc等
- 多平台？
  - 支持*nix、Windows、Android、iOS等
- 更新？
  - 紧跟最新的CPU扩展
- 独立？
  - 支持建立独立的工具
- 性能好吗？
  - JIT编译器技术vs还是翻译器技术？

现有的CPU仿真器

Features	libemu	PyEmu	IDA-x86emu	libCPU	Dream
Multi-arch	X	X	X	X [1]	✓
Updated	X	X	X	X	✓
Independent	X [2]	X [3]	X [4]	✓	✓
JIT	X	X	X	✓	✓

- 多架构：现有工具仅支持X86
- 已更新：现有工具不支持X86_64

1. 设计可以，但是不起作用
2. 仅集中于检测Windows shellcode
3. Python
4. 仅用于IDA

梦想一个好的仿真器

- 多架构
  - Arm、Arm64、Mips、PowerPC、Sparc、X86 (+X86_64) 以及更多
- 多平台：*nix、Windows、Android、 iOS等
- 更新：最新的所有硬件架构的扩展
- 同各种语言实现方式无关
  - 底层框架来支持各种操作系统和工具
  - 使用C语言为核心，支持多种绑定语言
- 良好的性能与JIT编译器技术
  - 动态编译vs翻译
- 允许不同级别的测量和追踪手段
  - 独步式/指令/内存访问

问题

- 即使在2015年也没有合理的CPU仿真器！
- 显然没有人想去解决这个问题。
- 生命中没有光明。
- 直到Unicorn降临！

Unicorn == 新一代CPU仿真器

Unicorn的目标

- 多架构
  - Arm、Arm64、Mips、PowerPC、Sparc、X86 (+X86_64) 以及更多
- 多平台：*nix、Windows、Android、 iOS等
- 更新：最新的所有硬件架构的扩展
- 核心在于C语言，同时支持多种绑定语言
- 良好的性能与JIT编译器技术
- 允许各级仪器
  - 独步式/指令/内存访问

Unicorn vs其他

Features	libemu	PyEmu	IDA-x86emu	libCPU	Unicorn
Multi-arch	X	X	X	X	✓
Updated	X	X	X	X	✓
Independent	X	X	X	✓	✓
JIT	X	X	X	✓	✓

- 多架构：现有工具仅支持X86
- 更新：现有工具不支持X86_64

建立Unicorn引擎的挑战

大量的工作！

- 许多硬件架构
- 许多指令
- 带有各种副作用的指令（有些是未文档的，例如：Intel X86）
- 很难支持现有的所有类型的代码
- 资源有限
    - 开始作为一个私人的有趣业余的项目

Unicorn设计

野心和想法

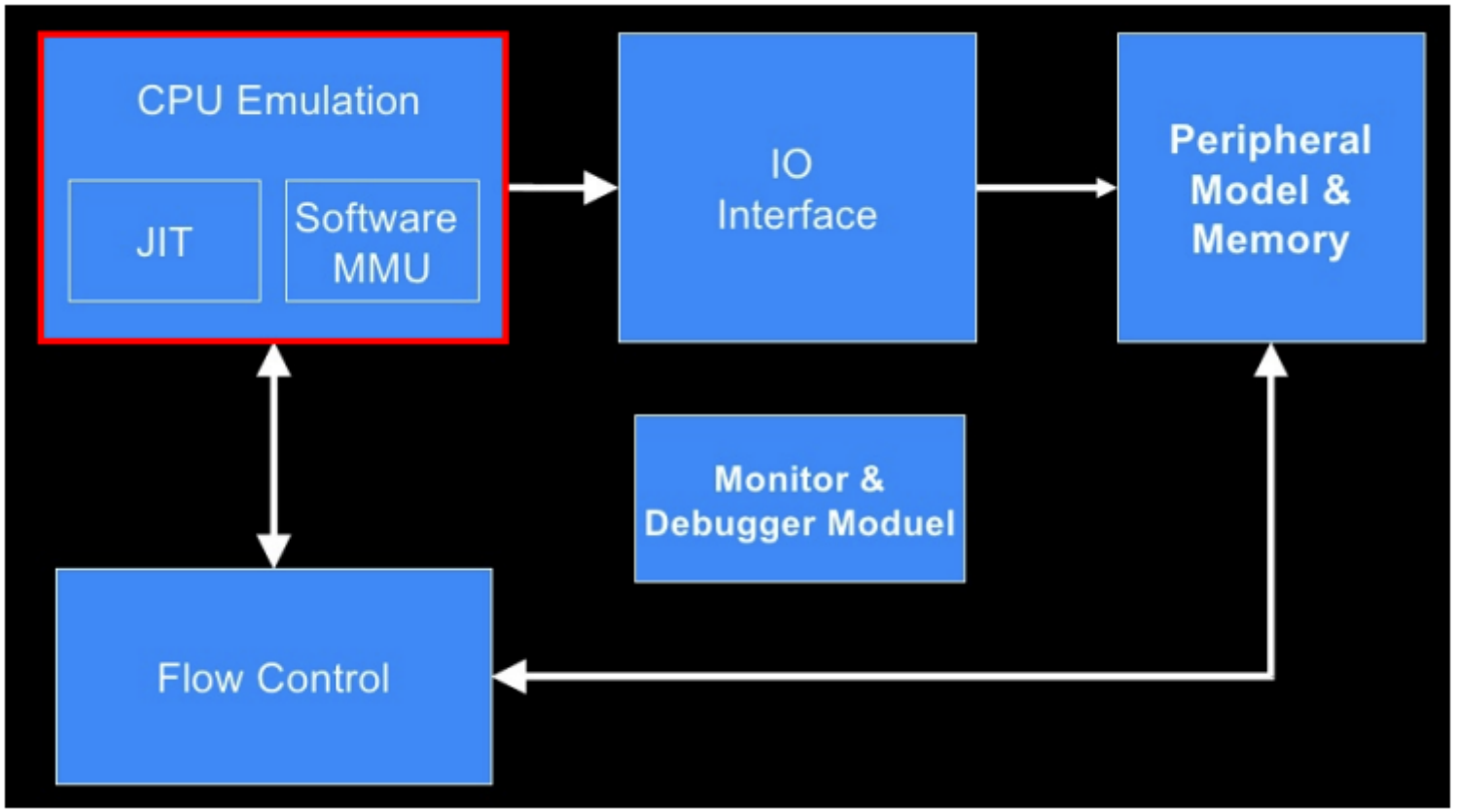
- 几个月内拥有所有的功能，而不是几年！
- 在初始阶段站在巨人的肩膀上。
- 开源项目去获得社区参与与贡献。
- 想法：Qemu！

介绍Qemu

Qemu 项目

- 开源项目（GPL 许可证）用于系统仿真器：
  - http://www.qemu.org
- 巨大的社区和高度活跃
- 多架构
  - X86、Arm、Arm64、Mips、PowerPC、Sparc等18个架构
- 多平台
  - Compile on *nix + cross-compile for Windows

Qemu架构



Courtesy of cmchao

为什么要Qemu？

- 支持各种架构和更新快
- 已经用纯C语言实现了，所以容易实现Unicorn的核心
- 已经支持JIT在CPU仿真下

我们做到了吗？

建立Unicorn的挑战 （1）

Qemu代码库是个挑战

- 不仅模拟CPU,而且设备模型和ROM / BIOS完全模拟物理机器
- Qemu代码库是超级大和杂乱的就像意大利面一样 :-(
- 难以阅读，因为代码是由许多不同的开发者贡献的

Unicorn的工作

- 只保留CPU仿真代码和移除其他（设备、ROM/BIOS、迁移等）
- 坚持支持像Qobject和Qom子系统
- 重写一些组件但是保持CPU仿真代码的完整性（以便将来与Qemu同步

建立Unicorn的挑战 （2）

Qemu是一组仿真器

- 用于单个架构的一组仿真器
  - 在编译时独立建造
  - 所有架构代码共享大量的内部数据结构和全局变量
- Unicorn想要一个仿真器，支持所有的架构 :-(

Unicorn的工作

- 隔离的常用变量和数据结构
  - 确保线程安全的设计
- 重构允许同时多个实例的Unicorn
- 改进构建系统去支持按需多个架构

建立Unicorn的挑战 （3）

Qemu没有测量和追踪手段

- 仅在静态编译时提供
- JIT利用大量快速路径技巧优化性能，使代码插装极其困难

Unicorn的工作

- 从头构建动态细粒度测量追踪层
- 支持各种级别的追踪和测量
  - 单步或特定指令（TCG级别）
  - 内存访问的测量和追踪（TLB级别）
  - 在模拟中动态读写寄存器或者内存。
  - 处理异常、中断、系统调用（arch-level）通过用户提供回调。

建立Unicorn的挑战（4）

Qemu存在泄漏内存的现象

- 代码中到处都是对象打开后（内存分配）没有正常关闭（内存释放）的现象
- 作为工具来说还可以，但是对于框架是不能接受的

Unicorn的工作

- 发现和修复所有的内存泄露问题
- 重构各种子系统去跟踪和清理悬挂指针。

Unicorn vs Qemu

Qemu的分支，但是已远远超过了它

- 独立的框架
  - 可以在没有上下文的前提下模拟原始的二进制
- 在内存中规模更加紧凑、轻便
- 线程安全与多个架构支持在一个单一的二进制
- 为动态测量和追踪提供接口
- 提供绑定（Python, Java, Go, .NET as of version 0.9）
- 对漏洞攻击抵抗力更强（更安全）
  - CPU仿真成分从未被成功攻击过
  - 作为一个API容易去测试。

Qemu漏洞

CVE-2015-5165	QEMU leak of uninitialized heap memory in rtl8139 device model
CVE-2015-5166	Use after free in QEMU/Xen block unplug protocol
CVE-2015-5154	QEMU heap overflow flaw while processing certain ATAPI commands.
CVE-2015-3209	Heap overflow in QEMU PCNET controller, allowing guest->host escape
CVE-2015-4106	Unmediated PCI register access in qemu
CVE-2015-4105	Guest triggerable qemu MSI-X pass-through error messages
CVE-2015-4103	Potential unintended writes to host MSI message data field via qemu
CVE-2015-2756	Unmediated PCI command register access in qemu
CVE-2015-2152	HVM qemu unexpectedly enabling emulated VGA graphics backends
CVE-2013-4375	qemu disk backend (qdisk) resource leak
CVE-2013-4344	qemu SCSI REPORT LUNS buffer overflow
CVE-2013-2007	qemu guest agent (qga) insecure file permissions
CVE-2013-1922	qemu-nbd format-guessing due to missing format specification
CVE-2012-6075	qemu (e1000 device driver): Buffer overflow when processing large packets

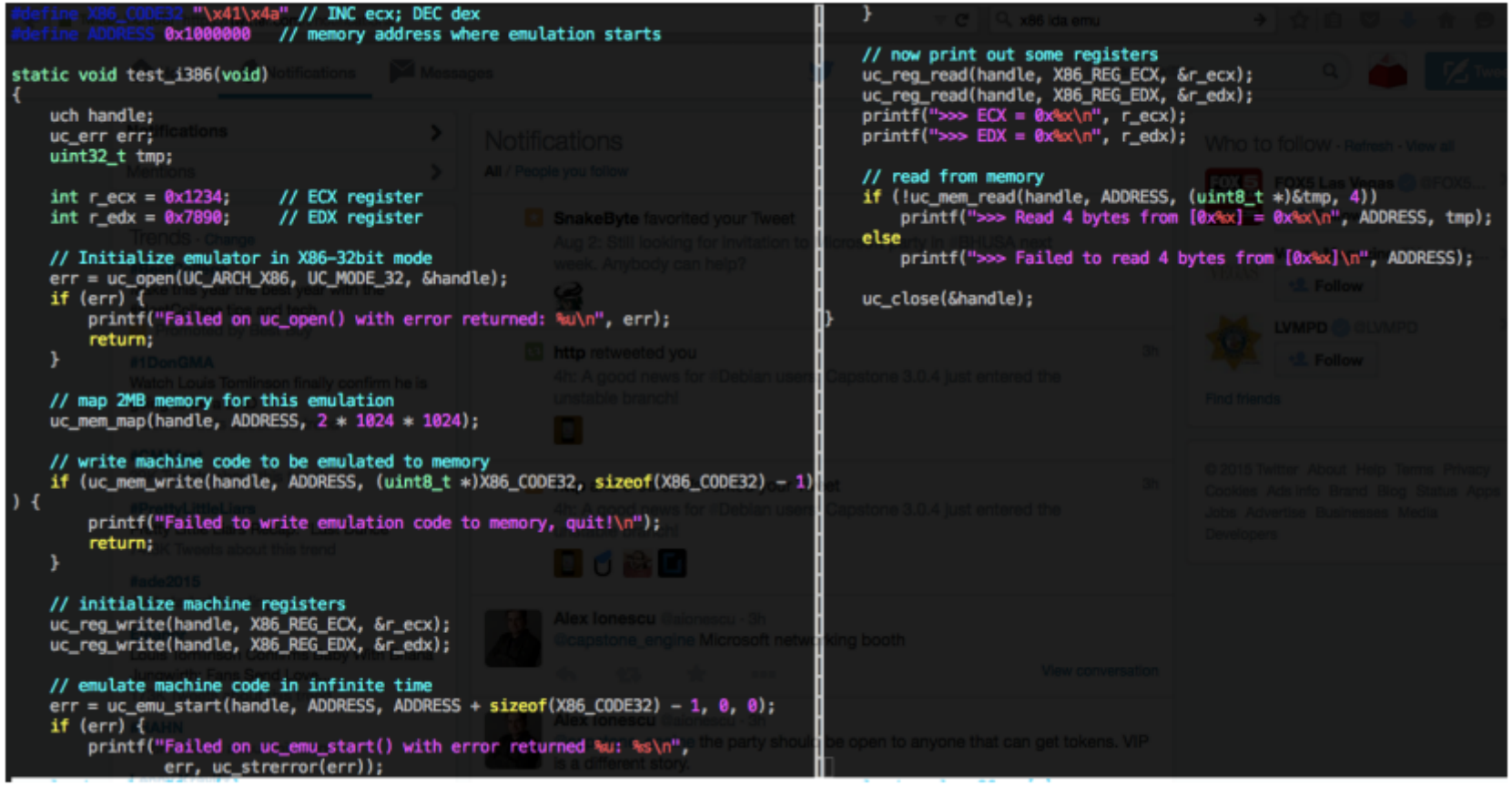
运用Unicorn编写应用程序

介绍Unicorn API

- 干净、简单、轻便和直观的构架无关API。
- 提供C格式的核心API
  - 打开和关闭Unicorn实例
  - 启动和停止仿真（基于结束地址、时间或者指令数）
  - 读取和写入内存
  - 读取和写入寄存器
  - 内存管理：挂钩内存事件、在运行时动态dump内存
    - 挂钩无效内存访问的内存事件
    - 在运行时动态dump内存（处理无效的/失踪的内存）
  - 可以使用用户自定义的回调来测量指令、单步以及内存事件等

- 核心外围可由Python/Java/Go/.NET构建

用C语言示例代码



```c
#define X86_CODE32 "\x41\x4a" // INC ecx; DEC dex
#define ADDRESS 0x1000000   // memory address where emulation starts

static void test_i386(void)
{
    uch handle;
    uc_err err;
    uint32_t tmp;

    int r_ecx = 0x1234;     // ECX register
    int r_edx = 0x7890;     // EDX register

    // Initialize emulator in X86-32bit mode
    err = uc_open(UC_ARCH_X86, UC_MODE_32, &handle);
    if (err) {
        printf("Failed on uc_open() with error returned: %u\n", err);
        return;
    }

    // map 2MB memory for this emulation
    uc_mem_map(handle, ADDRESS, 2 * 1024 * 1024);

    // write machine code to be emulated to memory
    if (uc_mem_write(handle, ADDRESS, (uint8_t *)X86_CODE32, sizeof(X86_CODE32) - 1)
) {
        printf("Failed to write emulation code to memory, quit!\n");
        return;
    }

    // initialize machine registers
    uc_reg_write(handle, X86_REG_ECX, &r_ecx);
    uc_reg_write(handle, X86_REG_EDX, &r_edx);

    // emulate machine code in infinite time
    err = uc_emu_start(handle, ADDRESS, ADDRESS + sizeof(X86_CODE32) - 1, 0, 0);
    if (err) {
        printf("Failed on uc_emu_start() with error returned %u: %s\n",
                err, uc_strerror(err));
    }

    // now print out some registers
    uc_reg_read(handle, X86_REG_ECX, &r_ecx);
    uc_reg_read(handle, X86_REG_EDX, &r_edx);
    printf(">>> ECX = 0x%x\n", r_ecx);
    printf(">>> EDX = 0x%x\n", r_edx);

    // read from memory
    if (!uc_mem_read(handle, ADDRESS, (uint8_t *)&tmp, 4))
        printf(">>> Read 4 bytes from [0x%x] = 0x%x\n", ADDRESS, tmp);
    else
        printf(">>> Failed to read 4 bytes from [0x%x]\n", ADDRESS);

    uc_close(&handle);
}
```

用Python示例代码

```
X86_CODE32 = b"\x41\x4a" # INC ecx; DEC dex
ADDRESS = 0x1000000 # memory address where emulation starts

print("Emulate i386 code")
try:
    # Initialize emulator in X86-32bit mode
    mu = Uc(UC_ARCH_X86, UC_MODE_32)

    # map 2MB memory for this emulation
    mu.mem_map(ADDRESS, 2 * 1024 * 1024)

    # write machine code to be emulated to memory
    mu.mem_write(ADDRESS, X86_CODE32)

    # initialize machine registers
    mu.reg_write(X86_REG_ECX, 0x1234)
    mu.reg_write(X86_REG_EDX, 0x7890)

    # emulate machine code in infinite time
    mu.emu_start(ADDRESS, ADDRESS + len(X86_CODE32))

    # done. now print out some registers
    r_ecx = mu.reg_read(X86_REG_ECX)
    r_edx = mu.reg_read(X86_REG_EDX)
    print(">>> ECX = 0x%x" %r_ecx)
    print(">>> EDX = 0x%x" %r_edx)

    # read from memory
    tmp = mu.mem_read(ADDRESS, 2)
    print(">>> Read 2 bytes from [0x%x] =" %(ADDRESS), end="")
    for i in tmp:
        print(" %02x" %i, end="")
    print("")

except UcError as e:
    print("ERROR: %s" % e)
```

现场演示

现状和未来工作

现状

- 2015年10月15日，发布0.9版本。
- 支持Arm, Arm64, Mips, M68K, Sparc, X86 (+X86_64)
  Python/Java/Go/.NET绑定可用
- 基于Qemu2.2.1

未来工作

- 支持Qemu所有剩余的架构
- (PPC/alpha/s360x/microblaze/sh4/etc - 总共 18个)
  社区承诺所有的bindings！
- 与Qemu2.4.1同步（目前最新版）
  - Unicorn的未来由Qemu积极发展来保证！

总结

- Unicorn是一个创新的新一代CPU仿真器
  - 多架构+多平台
  - 干净、简单、轻便和直观的构架无关API
  - 通过纯正C语言已经实现，多种绑定可用。
  - 高性能与JIT编译器技术
  - 支持细粒度各种级别的测量和追踪。
  - 设计级的线程安全。
  - 开源GPL许可证。
  - 未来更新保证所有的架构。

- 我们非常重视致力于这个项目使它成为最好的CPU仿真器。

提问和回答

Unicorn：新一代CPU仿真器框架

http://www.unicorn-engine.org

NGUYEN Anh Quynh <aquynh -at- gmail.com>

- 参考文献

- Qemu: http://www.qemu.org
- libemu: http://libemu.carnivore.it
- PyEmu: http://code.google.com/p/pyemu
- libcpu: https://github.com/libcpu/libcpu
- IDA-x86emu: http://www.idabook.com/x86emu/index.html
- Unicorn engine
  - Homepage: http://www.unicorn-engine.org
  - Mailing list:http://www.freelists.org/list/unicorn-engine
  - Twitter: @unicorn_engine

致谢

- 感谢Nguyen Tan Cong帮助我演示shellcode！
- 感谢其他测试人员帮助改进我们的代码！