Bio

About Me

Almost Every Weekend

With VN Security since year 2009

- CTF player
- Weekend gamer

Most of the time

Running xandora.net project.

- Soon
- Very Soon
- Brand New Online Sandbox

Once a year

Hack in The Box Crew

- Good friends
- CTF CTF and CTF

About Me

- 2008, Hack In The Box CTF Winner
- 2010, Hack In The Box Speaker, Malaysia
- 2012, Codegate Speaker, Korea
- 2015, VXRL Speaker, Hong Kong
- 2015, HITCON CTF, Prequal Top 10
- 2016, Codegate CTF, Prequal Top 5
- 2016, Qcon Speaker, Beijing

- OSX, Local Privilege Escalation
- Code commit for metasploit 3
- GDB Bug hunting
- Metasploit module
- Linux Randomization Bypass
- http://www.githiub.com/xwings/tuya

vnsec

Introduction

VN Security

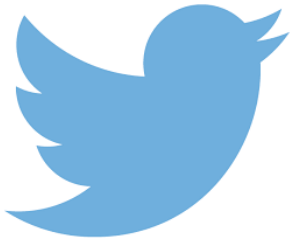
- Active CTF Player (CLGT)
- Active speaker at conferences
  - Blackhat USA
  - Tetcon
  - Hack In The Box
  - Xcon
- Our Tools
  - PEDA
  - Unicorn/ Capstone/ Keystone
  - Xandora
  - OllyDbg, Catcha!
  - ROPEME

Nguyen Anh Quynh

- Security Researcher
- Active speaker at conferences
  - Blackhat USA
  - Syscan
  - Hack In The Box
  - Xcon
- Research Topics
  - Emulators
  - Virtualization
  - Binary Analysis
  - Tools for Malware Analysis

Why This Talk

Virtual non Existence Website



The Framework

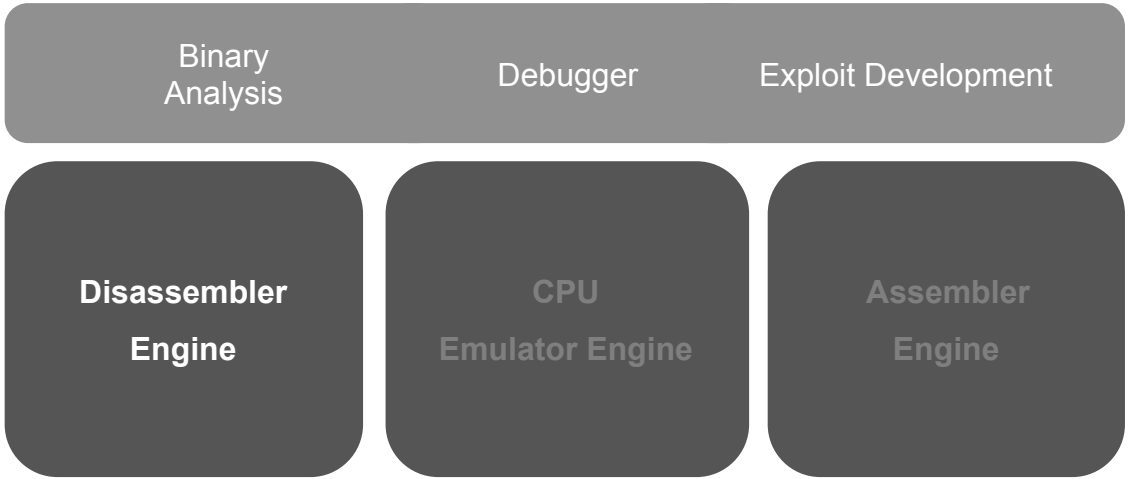
The Holy Trinity

What Is Disassembler

- From binary to assembly code
- Core part of all binary analysis/ reverse engineering / debugger and exploit development
- Disassembly framework (engine/library) is a lower layer in stack of architecture

Example

- 01D8 = ADD EAX,EBX (x86)
- 1169 = STR R1,[R2] (ARM's Thumb)

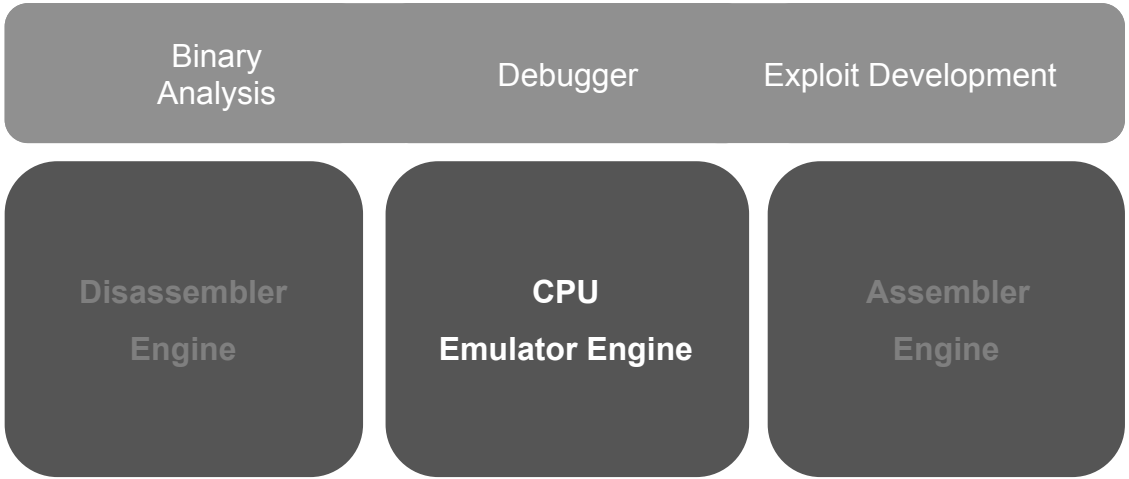


What Is Emulator

- Software only CPU Emulator
- Core focus on CPU operations.
- Design with no machine devices
- Safe emulation environment

Why CPU Emulator

- Safe Malware Analyzer
- Verify code semantics in reversing
- Real cross architecture without REAL CPU

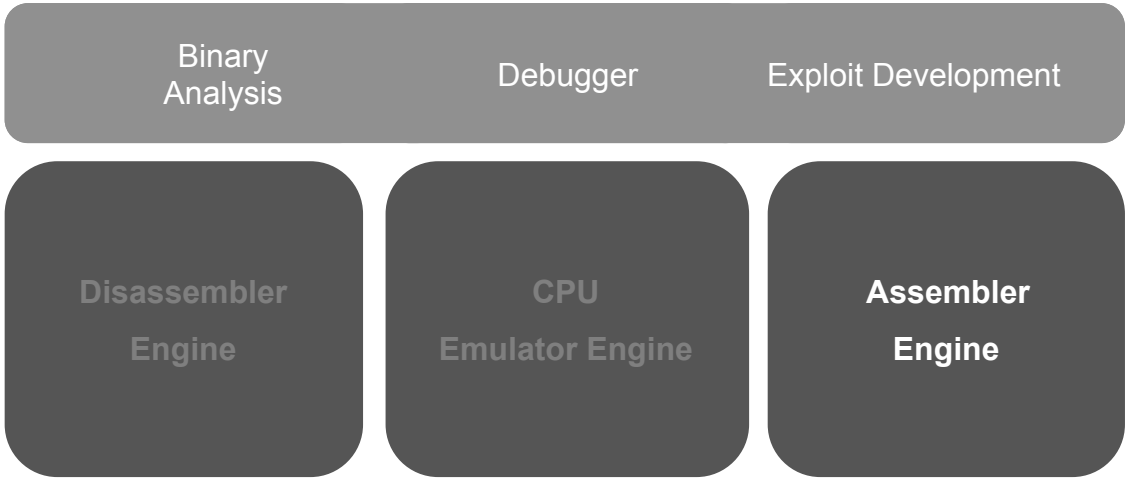


What Is Assembler

- From assembly to machine code
- Support high level concepts such as macro, functions and etc.
- Dynamic machine code generation

Example

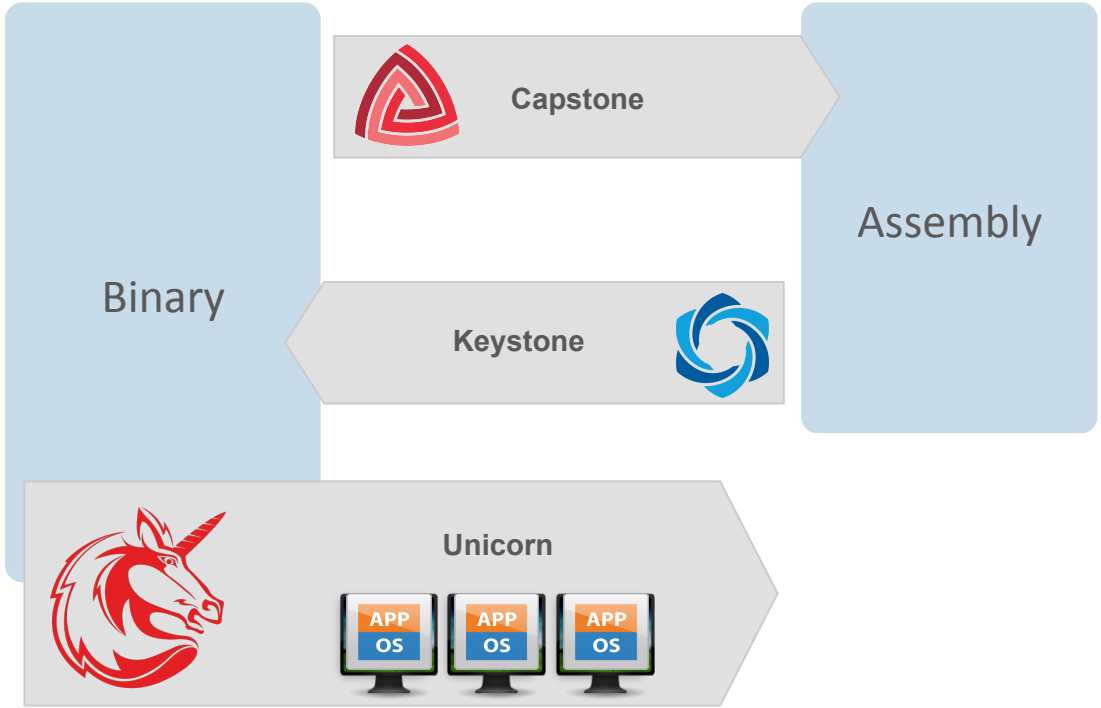
- ADD EAX,EBX = 01D8 (x86)
- STR R1,[R2] = 1169 (ARM's Thumb)



Born of The Trinity

Fundamental Frameworks for Reversing

- Components for a complete RE framework
- Interchange between assembler and disassembler
- A full CPU emulator always help when comes with obfuscated code



Capstone Engine

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

What's Wrong with Current Disassembler

Features	Distorm3	BeaEngine	Udis86	Libopcode
X86|Arm	✓|X	✓|X	✓|X	✓|✓ [1]
Linux|Windows	✓|✓	✓|✓	✓|✓	✓|X
Python|Ruby bindings	✓|X [2]	✓|X	✓|X	✓|X
Update	X	?	X	X
License	GPL	LGPL3	BSD	GPL

- Nothing works even up until 2013 (First release of Capstone Engine)
- Looks like no one take charge
- Industry stays in the dark side

Ideas

- Multiple archs: x86, ARM+ ARM64 + Mips + PPC and more
- Multiple platform: Windows, Linux, OSX and more
- Multiple binding: Python, Ruby, Java, C# and more

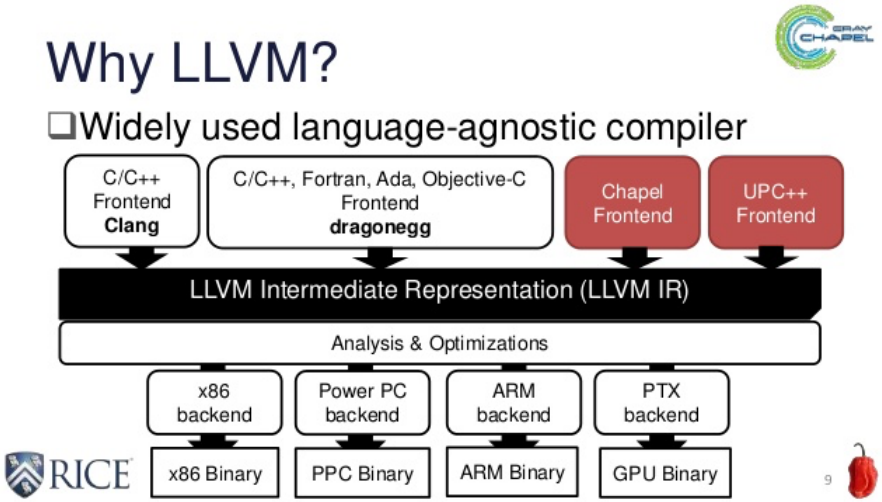
- Clean, simple, intuitive & architecture-neutral API
- Provide break-down details on instructions
- Friendly license: BSD

A Good Disassembler

- Multiple archs: x86, ARM
- Actively maintained & update within latest arch's change
- Multiple platform: Windows, Linux

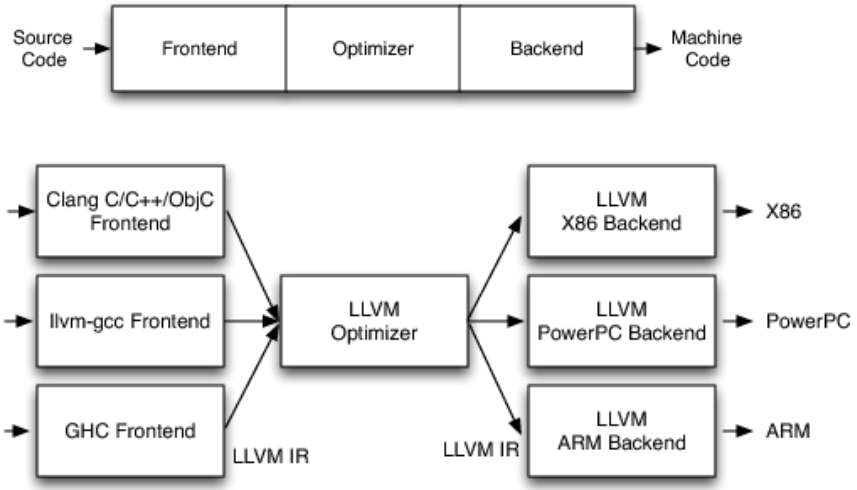
- Support python and ruby as binding languages
- Friendly license: BSD
- Easy to setup

Not Reinventing the Wheel



- Open source project compiler
- Sets of modules for machine code representing, compiling, optimizing
- Backed by many major players: AMD, Apple, Google, Intel, IBM, ARM, Imgtec, Nvidia, Qualcomm, Samsung, etc
- Incredibly huge (compiler) community around.

Fork from LLVM



- Multiple architectures ready
- In-disassembler (MC module)
  - Only, Only and Only build for LLVM
  - actively maintained
- Very actively maintained & updated by a huge community

Are We Done

Issues

- Cannot just reuse MC as-is without huge efforts.
  - LLVM code is in C++, but we want C code.
  - Code mixed like spaghetti with lots of LLVM layers.
  - Need to build instruction breakdown-details ourselves.
  - Expose semantics to the API.
  - Not designed to be thread-safe.
  - Poor Windows support.
- Need to build all bindings ourselves.
- Keep up with upstream code once forking LLVM to maitain ourselves.

Solutions

- Fork LLVM but must remove everything we do not need
- Replicated LLVM's MC
  - Build around MC and not changing MC
  - Replace C++ with C
- Extend LLVM's MC
  - Isolate some global variable to make sure thread-safe
- Semantics information
- cs_insn structure
  - Make arch-independent API

Capstone is not LLVM

More Superiors

- Zero dependency
- Compact in size
- More than assembly code
- Thread-safe design
- Able to embed into restricted firmware OS/ Environments
- Malware resistance (x86)
- Optimized for reverse engineers
- More hardware mode supported:- Big-Endian for ARM and ARM64
- More Instructions supported: 3DNow (x86)

More Robust

- Cannot always rely on LLVM to fix bugs
  - Disassembler is still conferred seconds-class LLVM, especially if does not affect code generation
  - May refuse to fix bugs if LLVM backed does not generate them (tricky x86 code)
- But handle all coner case properly is Capstone first piority
  - Handle all x86 malware ticks we aware of – Never Give Up

Demo

```c
/* test1.c */

#include <stdio.h>
#include <inttypes.h>

#include <capstone/capstone.h>

#define CODE "\x55\x48\x8b\x05\xb8\x13\x00\x00"

int main(void)
{
 csh handle;
 cs_insn *insn;
 size_t count;

 if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle) != CS_ERR_OK)
     return -1;
 count = cs_disasm(handle, CODE, sizeof(CODE)-1, 0x1000, 0, &insn);
 if (count > 0) {
     size_t j;
     for (j = 0; j < count; j++) {
         printf("0x%"PRIx64":\t%s\t\t%s\n", insn[j].address, insn[j].mnemoni
                 insn[j].op_str);
     }

     cs_free(insn, count);
 } else
     printf("ERROR: Failed to disassemble given code!\n");

 cs_close(&handle);

    return 0;
}
```

```
$ make
cc -c test1.c -o test1.o
cc test1.o -O3 -Wall -lcapstone -o test1

$ ./test1
0x1000: push        rbp
0x1001: mov     rax, qword ptr [rip + 0x13b8]
```

```python
# test1.py
from capstone import *

CODE = b"\x55\x48\x8b\x05\xb8\x13\x00\x00"

md = Cs(CS_ARCH_X86, CS_MODE_64)
for i in md.disasm(CODE, 0x1000):
    print("0x%x:\t%s\t%s" %(i.address, i.mnemonic, i.op_str))
```

```
$ python test1.py

0x1000: push    rbp
0x1001: mov rax, qword ptr [rip + 0x13b8]
```

Showcase

- CEnigma
- Unicorn
- CEbot
- Camal
- Radare2
- Pyew
- WinAppDbg
- PowerSploit
- MachOview
- RopShell
- ROPgadget
- Frida
- The-Backdoor-Factory
- Cuckoo
- Cerbero Profiler
- CryptoShark
- Ropper
- Snowman
- X86dbg
- Concolica
- Memtools Vita
- BARF
- rp++
- Binwalk
- MPRESS dumper
- Xipiter Toolkit
- Sonare
- PyDA
- Qira
- Rekall
- Inficere
- Pwntools
- Bokken
- Webkitties
- Malware_config_parsers
- Nightmare
- Catfish
- JSoS-Module-Dump
- Vitasploit
- PowerShellArsenal
- PyReil
- ARMSCGen
- Shwass
- Nrop
- lldb-capstone-arm
- Capstone-js
- ELF Unstrip Tool
- Binjitsu
- Rop-tool
- JitAsm
- OllyCapstone
- PackerId
- Volatility Plugins
- Pwndbg
- Lisa.py
- Many Other More

Unicorn Engine

NGUYEN Anh Quynh <aquynh -at- gmail.com>
DANG Hoang Vu <danghvu -at- gmail.com>

What's Wrong with Current Emulator

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

- Nothing works even up until 2015 (First release of Unicorn Engine)
- Limited bindings
- Limited functions, limited architecture

Ideas

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	✓	✓

- Multiple archs: x86, x86_64, ARM+ ARM64 + Mips + PPC
- Multiple platform: Windows, Linux, OSX, Android and more
- Multiple binding: Python, Ruby, Java, C# and more

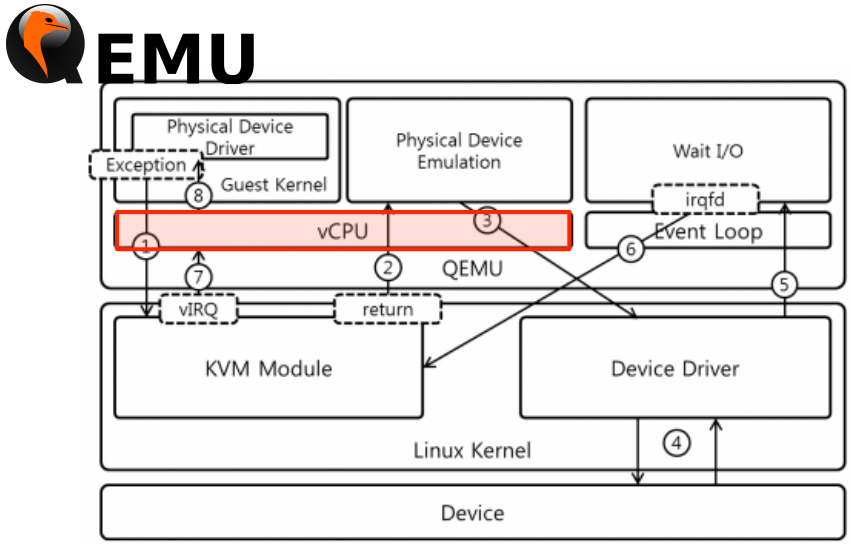
- Pure C implementation
- Latest and updated architecture
- With JIT compiler technique

A Good Emulator

- Multiple archs: x86, x86_64, ARM, ARM64, Mips and more
- Actively maintained & update within latest arch's change
- Multiple platform: Windows, Linux, OSX, Android and more

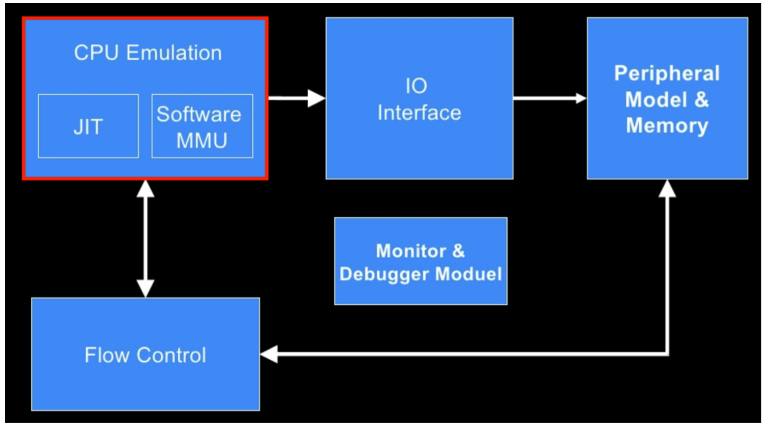
- Code in pure C
- Support python and ruby as binding languages
- JIT compiler technique
- Instrumentation at various level
  - Single step
  - Instruction
  - Memory Access

Not Reinventing the Wheel



- Open source project on system emulator
- Very huge community and highly active
- Multiple architecture: x86, ARM, ARM64, Mips, PowerPC, Sparc, etc (18 architectures)
- Multiple platform: *nix and Windows

Fork from LLVM



- Support all kind of architectures and very updated
- Already implemented in pure C, so easy to implement Unicorn core on top
- Already supported JIT in CPU emulation

Are We Done

Issues 1

- Not just emulate CPU, but also device models & ROM/BIOS to fully emulate physical machines
- Qemu codebase is huge and mixed like spaghetti
- Difficult to read, as contributed by many different people

Solutions

- Keep only CPU emulation code & remove everything else (devices, ROM/BIOS, migration, etc)
- Keep supported subsystems like Qobject, Qom
- Rewrites some components but keep CPU emulation code intact (so easy to sync with Qemu in future)

Issues 2

- Set of emulators for individual architecture
  - Independently built at compile time
  - All archs code share a lot of internal data structures and global variables
- Unicorn wants a single emulator that supports all archs

Solutions

- Isolated common variables & structures
  - Ensured thread-safe by design
- Refactored to allow multiple instances of Unicorn at the same time Modified the build system to support multiple archs on demand

Are We Done

Issues 3

- Instrumentation for static compilation only
- JIT optimizes for performance with lots of fast-path tricks, making code instrumenting extremely hard

Solutions

- Build dynamic fine-grained instrumentation layer from scratch Support various levels of instrumentation
  - Single-step or on particular instruction (TCG level)
  - Instrumentation of memory accesses (TLB level)
  - Dynamically read and write register
  - Handle exception, interrupt, syscall (arch-level) through user provided callback

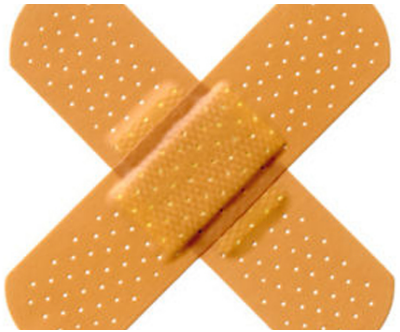
Issues 4

- Objects is open (malloc) without closing (freeing) properly everywhere
- Fine for a tool, but unacceptable for a framework

Solutions

- Find and fix all the memory leak issues
- Refactor various subsystems to keep track and cleanup dangling pointers

Unicorn Engine is not QEMU



- Independent framework
- Much more compact in size, lightweight in memory
- Thread-safe with multiple architectures supported in a single binary Provide interface for dynamic instrumentation
- More resistant to exploitation (more secure)
  - CPU emulation component is never exploited!
  - Easy to test and fuzz as an API.

Demo

```c
#include <unicorn/unicorn.h>

// code to be emulated
#define X86_CODE32 "\x41\x4a" // INC ecx; DEC edx

// memory address where emulation starts
#define ADDRESS 0x1000000

int main(int argc, char **argv, char **envp)
{
  uc_engine *uc;
  uc_err err;
  int r_ecx = 0x1234;     // ECX register
  int r_edx = 0x7890;     // EDX register

  printf("Emulate i386 code\n");

  // Initialize emulator in X86-32bit mode
  err = uc_open(UC_ARCH_X86, UC_MODE_32, &uc);
  if (err != UC_ERR_OK) {
    printf("Failed on uc_open() with error returned: %u\n", err);
    return -1;
  }

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

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

  // initialize machine registers
  uc_reg_write(uc, UC_X86_REG_ECX, &r_ecx);
  uc_reg_write(uc, UC_X86_REG_EDX, &r_edx);

  // emulate code in infinite time & unlimited instructions
  err=uc_emu_start(uc, 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
  printf("Emulation done. Below is the CPU context\n");

  uc_reg_read(uc, UC_X86_REG_ECX, &r_ecx);
  uc_reg_read(uc, UC_X86_REG_EDX, &r_edx);
  printf(">>> ECX = 0x%x\n", r_ecx);
  printf(">>> EDX = 0x%x\n", r_edx);

  uc_close(uc);

  return 0;
}
```

```
$ make
cc  test1.c -L/usr/local/Cellar/glib/2.44.1/lib -L/usr/local/opt/gettext/

$ ./test1
Emulate i386 code
Emulation done. Below is the CPU context
>>> ECX = 0x1235
>>> EDX = 0x788f
```

```python
from __future__ import print_function
from unicorn import *
from unicorn.x86_const import *

# code to be emulated
X86_CODE32 = b"\x41\x4a" # INC ecx; DEC edx

# memory address where emulation starts
ADDRESS = 0x1000000

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(UC_X86_REG_ECX, 0x1234)
    mu.reg_write(UC_X86_REG_EDX, 0x7890)

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

    # now print out some registers
    print("Emulation done. Below is the CPU context")

    r_ecx = mu.reg_read(UC_X86_REG_ECX)
    r_edx = mu.reg_read(UC_X86_REG_EDX)
    print(">>> ECX = 0x%x" %r_ecx)
    print(">>> EDX = 0x%x" %r_edx)

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

```
$ python test1.py

Emulate i386 code
Emulation done. Below is the CPU context
>>> ECX = 0x1235
>>> EDX = 0x788f
```

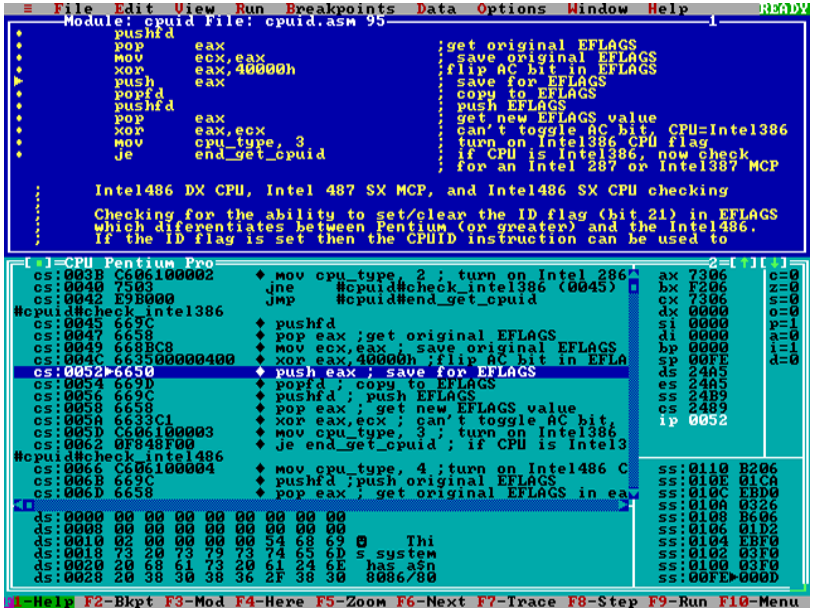
Showcase

- UniDOS: Microsoft DOS emulator.
- Radare2: Unix-like reverse engineering framework and commandline tools.
- Usercorn: User-space system emulator.
- Unicorn-decoder: A shellcode decoder that can dump self-modifying-code.
- Univm: A plugin for x64dbg for x86 emulation.
- PyAna: Analyzing Windows shellcode.
- GEF: GDB Enhanced Features.
- Pwndbg: A Python plugin of GDB to assist exploit development.
- Eli.Decode: Decode obfuscated shellcodes.
- IdaEmu: an IDA Pro Plugin for code emulation.
- Roper: build ROP-chain attacks on a target binary using genetic algorithms.
- Sk3wlDbg: A plugin for IDA Pro for machine code emulation.
- Angr: A framework for static & dynamic concolic (symbolic) analysis.
- Cemu: Cheap EMUlator based on Keystone and Unicorn engines.
- ROPMEMU: Analyze ROP-based exploitation.
- BroIDS_Unicorn: Plugin to detect shellcode on Bro IDS with Unicorn.
- UniAna: Analysis PE file or Shellcode (Only Windows x86).
- ARMSCGen: ARM Shellcode Generator.
- TinyAntivirus: Open source Antivirus engine designed for detecting & disinfecting polymorphic virus.
- Patchkit: A powerful binary patching toolkit.

Keystone Engine

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

What's Wrong with Assembler



- Nothing is up to our standard, even in 2016!
  - Yasm: X86 only, no longer updated
  - Intel XED: X86 only, miss many instructions & closed-source
  - Use assembler to generate object files
  - Other important archs: Arm, Arm64, Mips, PPC, Sparc, etc?

Ideas

- Multiple archs: x86, ARM+ ARM64 + Mips + PPC and more
- Multiple platform: Windows, Linux, OSX and more
- Multiple binding: Python, Ruby, Java, C# and more

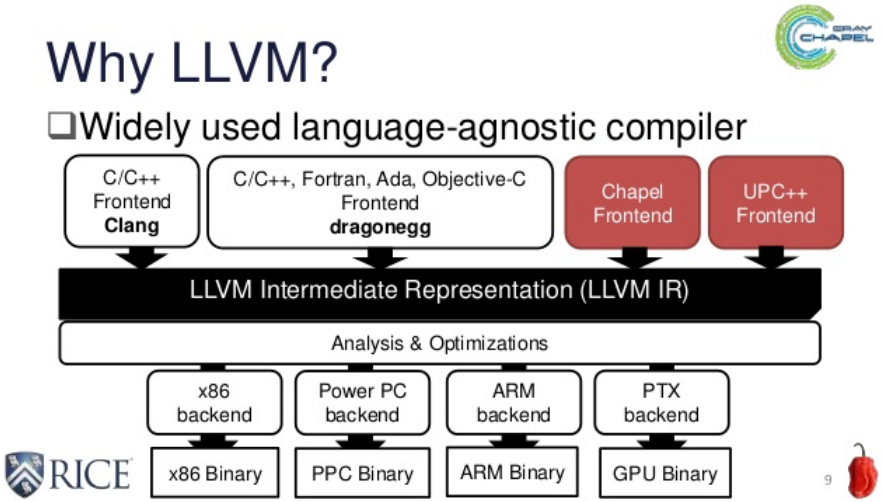
- Clean, simple, intuitive & architecture-neutral API
- Provide break-down details on instructions
- Friendly license: BSD

A Good Assembler

- Multiple archs: x86, ARM
- Actively maintained & update within latest arch's change
- Multiple platform: Windows, Linux

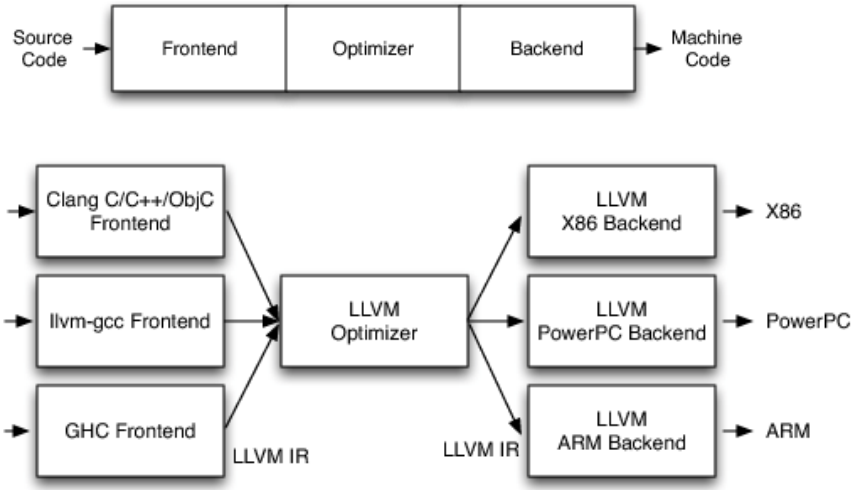
- Support python and ruby as binding languages
- Friendly license (BSD)
- Easy to setup

Not Reinventing the Wheel



- Open source project compiler
- Sets of modules for machine code representing, compiling, optimizing
- Backed by many major players: AMD, Apple, Google, Intel, IBM, ARM, Imgtec, Nvidia, Qualcomm, Samsung, etc
- Incredibly huge (compiler) community around.

Fork from LLVM



- Multiple architectures ready
- In-build assembler (MC module)
  - Only, Only and Only build for LLVM
  - actively maintained
- Very actively maintained & updated by a huge community

Are We Done

Issue 1

- Not just assembler, but also disassembler, Bitcode, InstPrinter, Linker Optimization, etc
- LLVM codebase is huge and mixed like spaghetti

Solutions

- Keep only assembler code & remove everything else unrelated
- Rewrites some components but keep AsmParser, CodeEmitter & AsmBackend code intact (so easy to sync with LLVM in future)
- Keep all the code in C++ to ease the job (unlike Capstone)
  - No need to rewrite complicated parsers
  - No need to fork llvm-tblgen

Issue 2

- LLVM compiled into multiple libraries
  - Supported libs
  - Parser
  - TableGen and etc
- Keystone needs to be a single library

Solutions

- Modify linking setup to generate a single library
  - libkeystone.[so, dylib] + libkeystone.a
  - keystone.dll + keystone.lib

Are We Done

Issue 3

- Relocation object code generated for linking in the final code generation phase of compiler
- Ex on X86:
  - inc [_var1] → 0xff, 0x04, 0x25, A, A, A, A

Solutions

- Make fixup phase to detect & report missing symbols
- Propagate this error back to the top level API ks_asm()

Issue 4

- Ex on ARM: blx 0x86535200 → 0x35, 0xf1, 0x00, 0xe1

Solutions

- ks_asm() allows to specify address of first instruction
- Change the core to retain address for each statement
- Find all relative branch instruction to fix the encoding according to current & target address

Are We Done

Issue 5

- Ex on X86: vaddpd zmm1, zmm1, zmm1, x → "this is not an immediate"
- Returned llvm_unreachable() on input it cannot handle

Solutions

- Fix all exits & propagate errors back to ks_asm()
  - Parse phase
  - Code emit phase

Issue 6

- LLVM does not support non-LLVM syntax
  - We want other syntaxes like Nasm, Masm, etc
- Bindings must be built from scratch
- Keep up with upstream code once forking LLVM to maintain ourselves

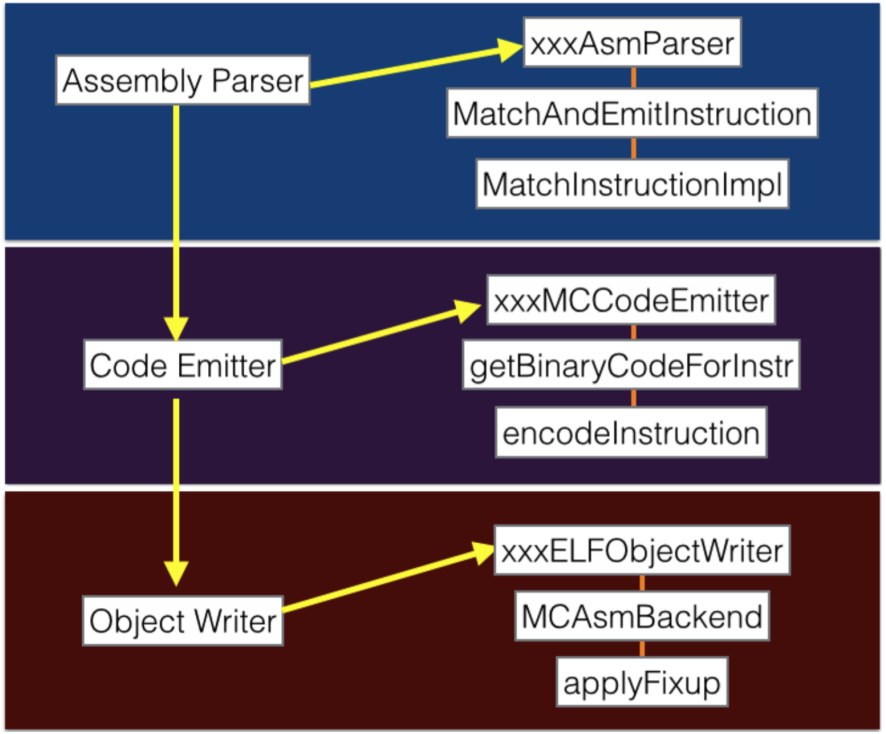
Solutions

- Extend X86 parser for new syntaxes: Nasm, Masm, etc
- Built Python binding
- Extra bindings came later, by community: NodeJS, Ruby, Go, Rust, Haskell & OCaml
- Keep syncing with LLVM upstream for important changes & bug-fixes

Keystone is not LLVM

Fork and Beyond

- Independent & truly a framework
  - Do not give up on bad-formed assembly
- Aware of current code position (for relative branches)
- Much more compact in size, lightweight in memory
- Thread-safe with multiple architectures supported in a single binary More flexible: support X86 Nasm syntax
- Support undocumented instructions: X86
- Provide bindings (Python, NodeJS, Ruby, Go, Rust, Haskell, OCaml as of August 2016)



Demo

```c
/* test1.c */
#include <stdio.h>
#include <keystone/keystone.h>

// separate assembly instructions by ; or \n
#define CODE "INC ecx; DEC edx"

int main(int argc, char **argv)
{
    ks_engine *ks;
    ks_err err;
    size_t count;
    unsigned char *encode;
    size_t size;

    err = ks_open(KS_ARCH_X86, KS_MODE_32, &ks);
    if (err != KS_ERR_OK) {
        printf("ERROR: failed on ks_open(), quit\n");
        return -1;
    }

    if (ks_asm(ks, CODE, 0, &encode, &size, &count) != KS_ERR_OK) {
        printf("ERROR: ks_asm() failed & count = %lu, error = %u\n",
                        count, ks_errno(ks));
    } else {
        size_t i;

        printf("%s = ", CODE);
        for (i = 0; i < size; i++) {
            printf("%02x ", encode[i]);
        }
        printf("\n");
        printf("Compiled: %lu bytes, statements: %lu\n", size, count);
    }

    // NOTE: free encode after usage to avoid leaking memory
    ks_free(encode);

    // close Keystone instance when done
    ks_close(ks);

    return 0;
}
```

```
$ make
cc -o test1 test1.c -lkeystone -lstdc++ -lm

$ ./test1
INC ecx; DEC edx = 41 4a
Compiled: 2 bytes, statements: 2
```

```python
from keystone import *

# separate assembly instructions by ; or \n
CODE = b"INC ecx; DEC edx"

try:
  # Initialize engine in X86-32bit mode
  ks = Ks(KS_ARCH_X86, KS_MODE_32)
  encoding, count = ks.asm(CODE)
  print("%s = %s (number of statements: %u)" %(CODE, encoding, count))
except KsError as e:
  print("ERROR: %s" %e)
```

```
$ ./test1.py
INC ecx; DEC edx = [65, 74] (number of statements: 2)
```

Showcase

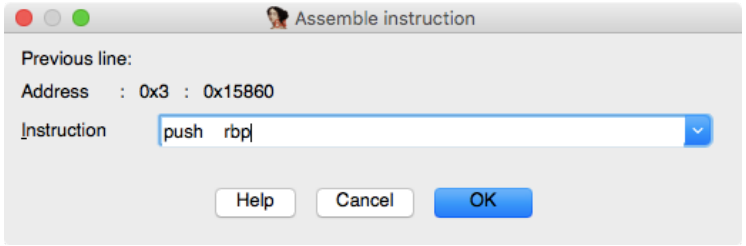
- Keypatch: IDA Pro plugin for code assembling & binary patching.
- Radare2: Unix-like reverse engineering framework and commandline tools.
- GEF: GDB Enhanced Features.
- Ropper: Rop gadget and binary information tool.
- Cemu: Cheap EMUlator based on Keystone and Unicorn engines.
- Pwnypack: Certified Edible Dinosaurs official CTF toolkit.
- Keystone.JS: Emscripten-port of Keystone for JavaScript.
- Usercorn: Versatile kernel+system+userspace emulator.
- x64dbg: An open-source x64/x32 debugger for windows.
- Liberation: a next generation code injection library for iOS cheaters everywhere.
- Strongdb: GDB plugin for Android debugging.
- AssemblyBot: Telegram bot for assembling and disassembling on-the-go.
- demovfuscator: Deobfuscator for movfuscated binaries.
- Dash: A simple web based tool for working with assembly language.
- ARMSCGen: ARM Shellcode Generator.
- Asm_Ops: Assembler for IDA Pro (IDA Plugin).
- Binch: A lightweight ELF binary patch tool.
- Metame: Metamorphic code engine for arbitrary executables.
- Patchkit: A powerful binary patching toolkit.
- Pymetamorph: Metamorphic engine in Python for Windows executables.

One More Thing

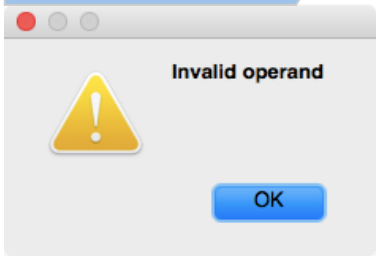
The IDA Pro

IDA Pro

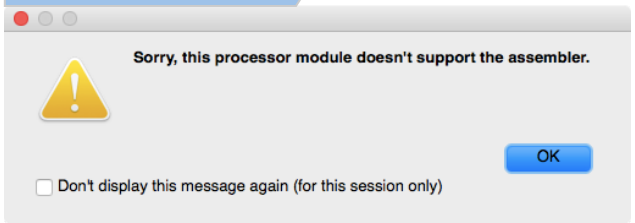
- RE Standard
- Patching on the fly is always a must
- Broken “Edit\Patch Program\ Assembler” is always giving us problem



PUSH RAX



ARM



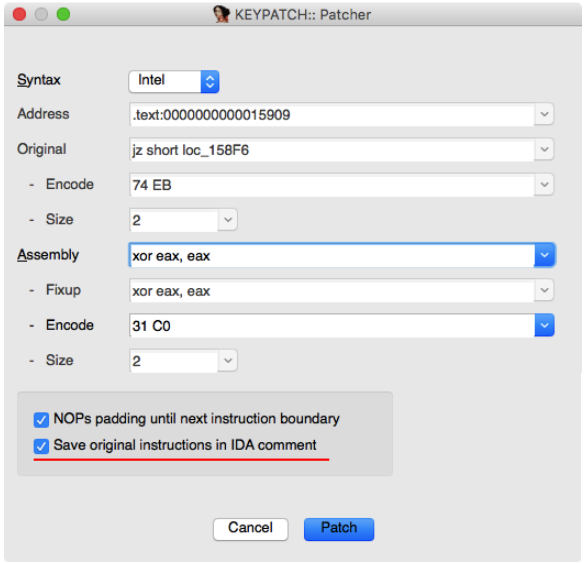
PUSH ESI

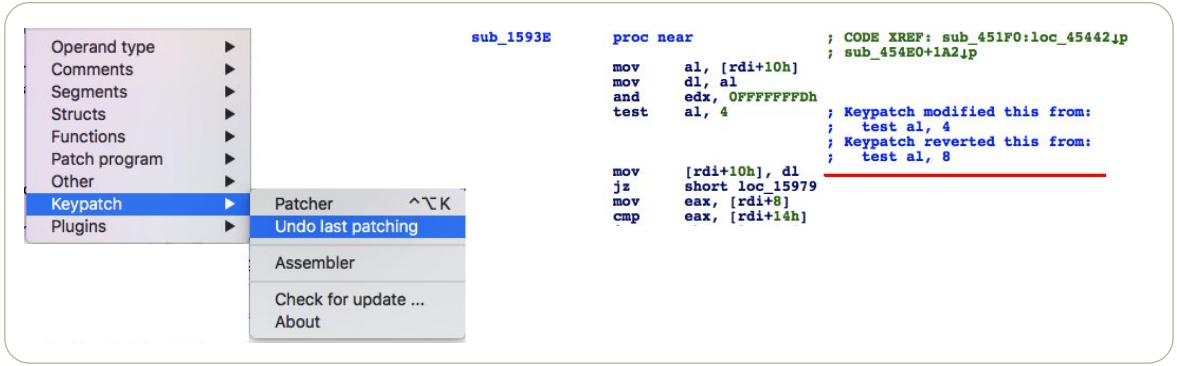
```
15860 55                                    push    rbp
15861 48 8D 57 10                           lea     rdx, [rdi+10h]
15865 56                                    push    rsi
15866 48 89 FB                              mov     rbx, rdi
15869 50                                    push    rax
```

Keypatch

A binary editor plugin for IDA Pro

- Fully open source @ https://keystone-engine.org/keypatch
- On the fly patching in IDA Pro with Multi Arch
- Base on Keystone Engine
- By Nguyen Anh Quynh & Thanh Nguyen (rd) from vnsecurity.net






```
                        ; CODE XREF: sub_158D3+211↑j
shl     esi, 4
jz      short loc_158F6
mov     edi, esi        ; size
call    _malloc
test    rax, rax
xor     eax, eax        ; Keypatch modified this from:
                        ;   jz short loc_158F6
mov     ecx, 800h
mov     rdi, rax
mov     rsi, rbp
```

T H A N K S

[Hacker@KCon]