KEYSTONE: the last missing framework for Reverse Engineering

www.keystone-engine.org

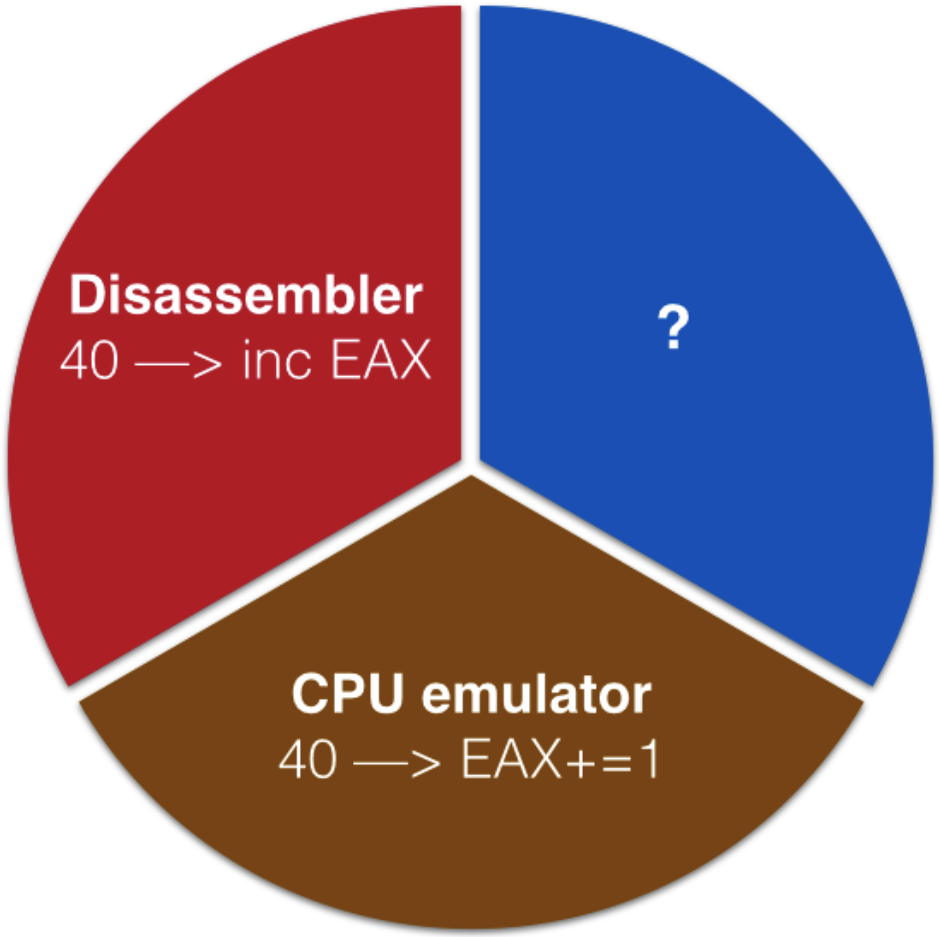
NGUYEN Anh Quynh <aquynh -at- gmail.com>

RECON - June 19th, 2016

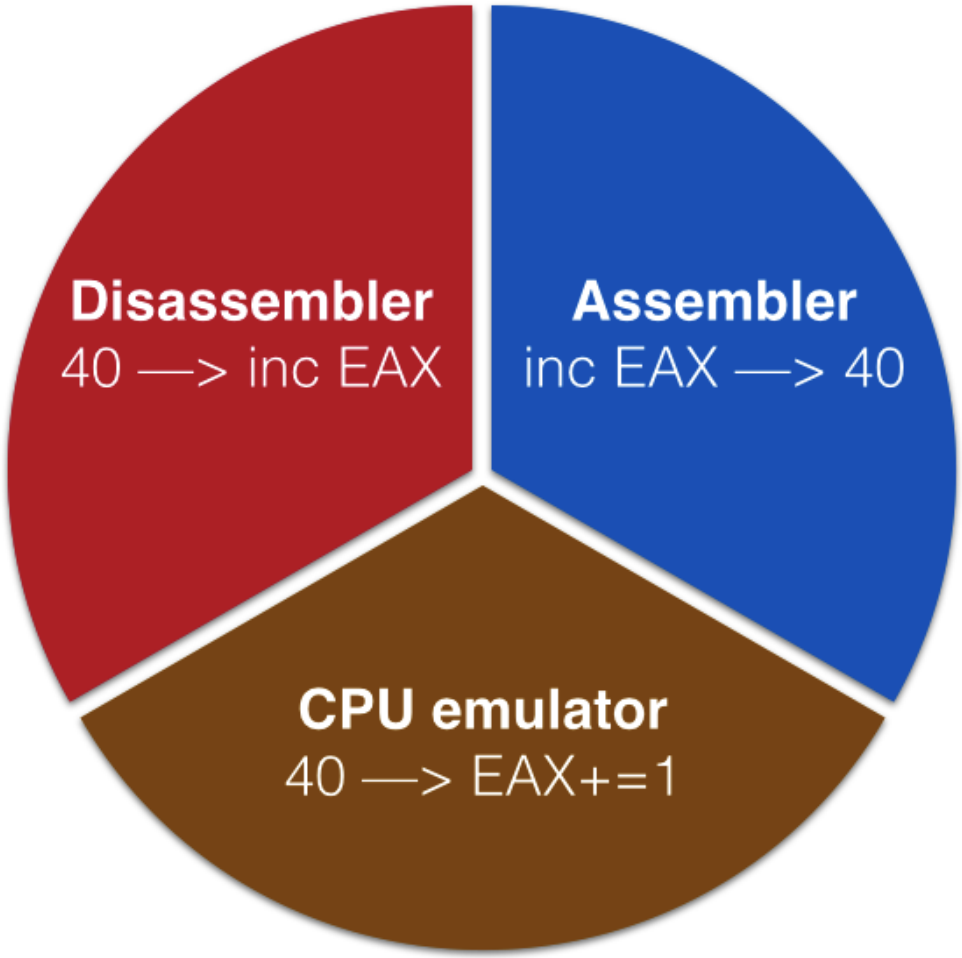
Bio

- Nguyen Anh Quynh (aquynh -at- gmail.com)
  - Nanyang Technological University, Singapore
  - Researcher with a PhD in Computer Science
  - Operating System, Virtual Machine, Binary analysis, etc
  - Capstone disassembler: http://capstone-engine.org
  - Unicorn emulator: http://unicorn-engine.org
  - Keystone assembler: http://keystone-engine.org

Fundamental frameworks for Reverse Engineering



Fundamental frameworks for Reverse Engineering



Assembler framework

Definition

- Compile assembly instructions & returns encoding as sequence of bytes
  - Ex: inc EAX $\rightarrow$ 40
- May support high-level concepts such as macro, function, etc
- Framework to build apps on top of it

Applications

- Dynamic machine code generation
  - Binary rewrite
  - Binary searching

Internals of assembler engine

Given assembly input code

- Parse assembly instructions into separate statements
- Parse each statement into different types
  - Label, macro, directive, etc
  - Instruction: menemonic + operands
    - ★ Emit machine code accordingly
    - ★ Instruction-Set-Architecture manual referenced is needed

Challenges of building assembler

Huge amount of works!

- Good understanding of CPU encoding
- Good understanding of instruction set
- Keep up with frequently updated instruction extensions.

Good assembler framework?

- True framework
  - Embedded into tool without resorting to external process
- Multi-arch
  - X86, Arm, Arm64, Mips, PowerPC, Sparc, etc
- Multi-platform
  - *nix, Windows, Android, iOS, etc
- Updated
  - Keep up with latest CPU extensions
- Bindings
  - Python, Ruby, Go, NodeJS, etc

Existing assembler frameworks

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  - Yasm: X86 only, no longer updated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

Life without assembler frameworks?

- People are very much struggling for years!
  - Use existing assembler tool to compile assembly from file
  - Call linker to link generated object file
  - Use ELF parser to parse resulted file for final encoding
- Ugly and inefficient
- Little control on the internal process & output
- Cross-platform support is very poor

Dream a good assembler

- Multi-architectures
  - Arm, Arm64, Mips, PowerPC, Sparc, X86 (+X86_64) + more
- Multi-platform: *nix, Windows, Android, iOS, etc
- Updated: latest extensions of all hardware architectures
- Independent with multiple bindings
  - Low-level framework to support all kind of OS and tools
  - Core in C++, with API in pure C, and support multiple binding languages

Timeline

- Indiegogo campaign started on March 17th, 2016 (for 3 weeks)
  - 99 contributors, 4 project sponsors
- Beta code released to beta testers on April 30th, 2016
  - Only Python binding available at this time
- Version 0.9 released on May 31st, 2016
  - More bindings by beta testers: NodeJS, Ruby, Go & Rust
- Haskell binding merged after v0.9 public

Keystone == Next Generation Assembler Framework

Goals of Keystone

- Multi-architectures
  - Arm, Arm64, Mips, PowerPC, Sparc, X86 (+X86_64) + more
- Multi-platform: *nix, Windows, Android, iOS, etc
- Updated: latest extensions of all hardware architectures
- Core in C/C++, API in pure C & support multiple binding languages

Challenges to build Keystone

Huge amount of works!

- Too many hardware architectures
- Too many instructions
- Limited resource
  - Started as a personal project

Keystone design & implementation

Ambitions & ideas

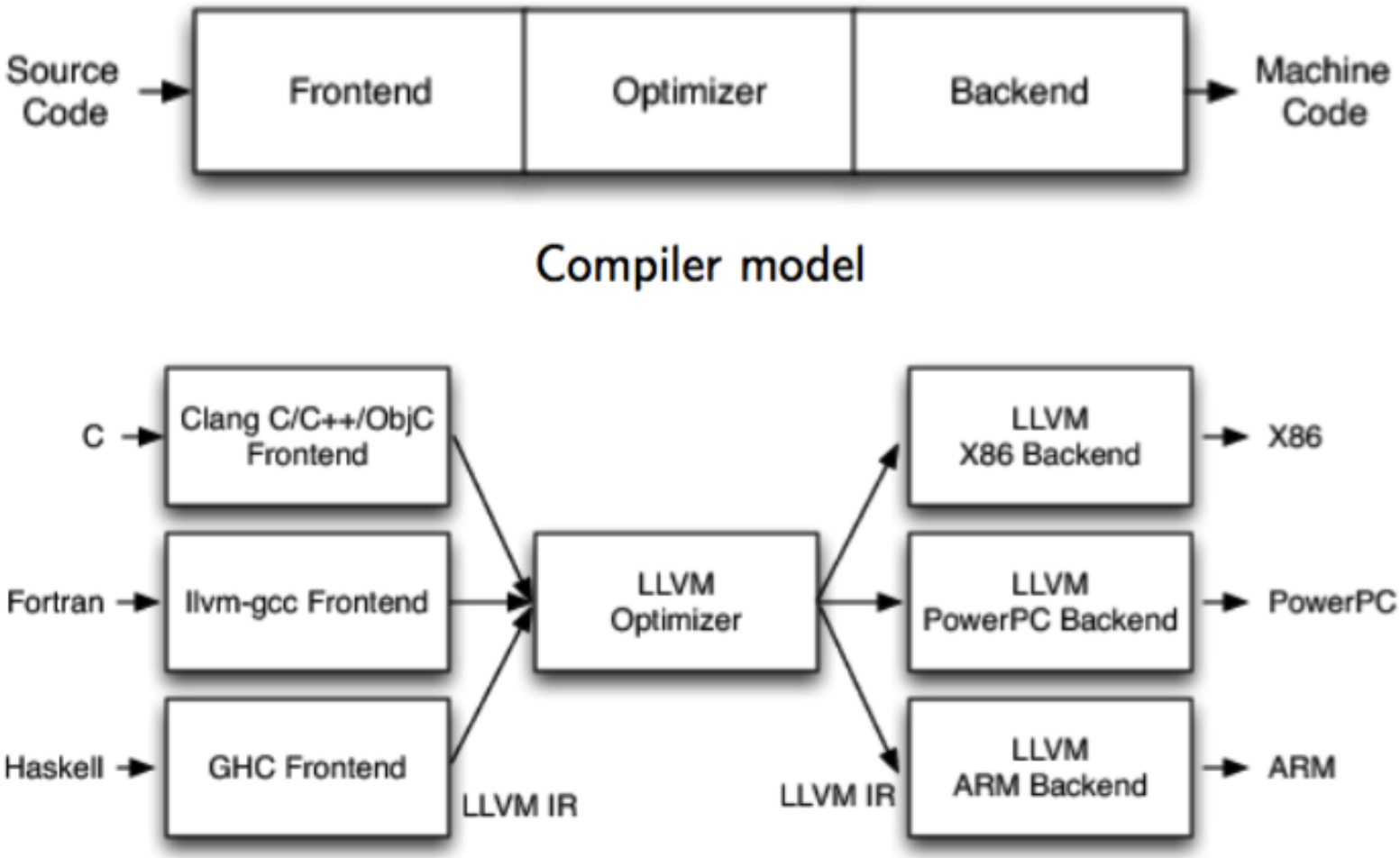
- Have all features in months, not years!
- Stand on the shoulders of the giants at the initial phase.
- Open source project to get community involved & contributed.
- Idea: LLVM!

Introduction on LLVM

LLVM project

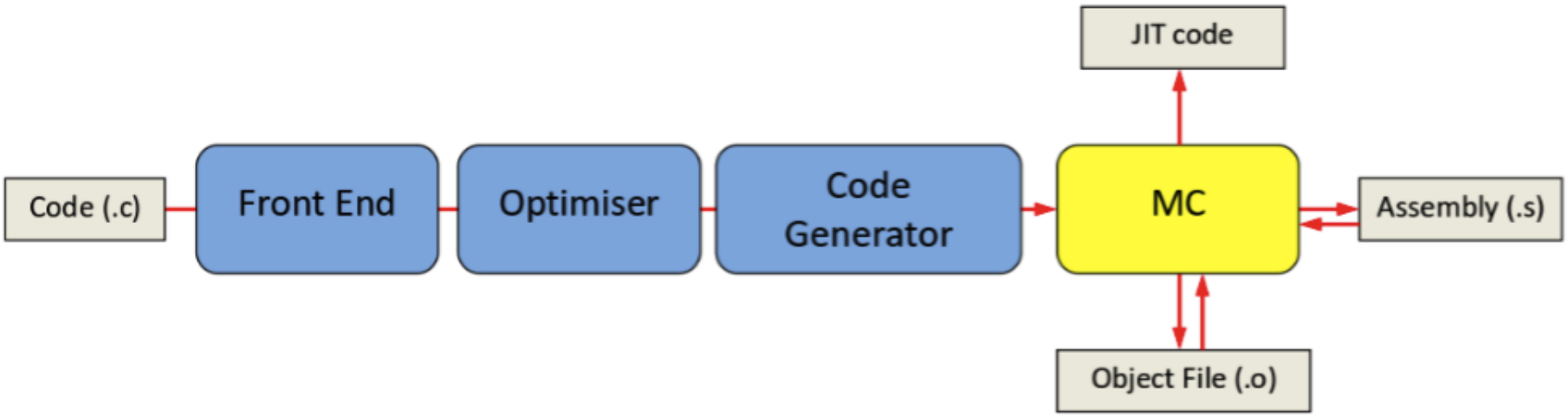
- Open source project on compiler: http://llvm.org
- Huge community & highly active
- Backed by many major players: AMD, Apple, Google, Intel, IBM, ARM, Imgtec, Nvidia, Qualcomm, Samsung, etc.
- Multi-arch
  - X86, Arm, Arm64, Mips, PowerPC, Sparc, Hexagon, SystemZ, etc
- Multi-platform
  - Native compile on Windows, Linux, macOS, BSD, Android, iOS, etc

LLVM architecture



LLVM's Machine Code (MC) layer

- Core layer of LLVM to integrate compiler with its internal assemblers
- Used by compiler, assembler, disassembler, debugger & JIT compilers
- Centralize with a big table of description (TableGen) of machine instructions
- Auto generate assembler, disassembler, and code emitter from TableGen (*.inc) - with llvm-tablegen tool.



Why LLVM?

- Available assembler internally in Machine Code (MC) module - for inline assembly support.
  - Only useable for LLVM modules, not for external code
  - Closely designed & implemented for LLVM
  - Very actively maintained & updated by a huge community
- Already implemented in C++, so easy to immplement Keystone core on top
- Pick up only those archs having assemblers: 8 archs for now.

LLVM advantages

- High quality code with lots of tested done using test cases
- Assembler maintained by top experts of each archs
  - X86: maintained by Intel (arch creator).
  - Arm+Arm64: maintained by Arm & Apple (arch creator & Arm64's device maker).
  - Hexagon: maintained by Qualcomm (arch creator)
  - Mips: maintained by Imgtec (arch creator)
  - SystemZ: maintained by IBM (arch creator)
  - PPC & Sparc: maintained by highly active community
- New instructions & bugs fixed quite frequently!
- Bugs can be either reported to us, or reported to LLVM upstream, then ported back.

Are we done?

Challenges to build Keystone (1)

LLVM MC is a challenge

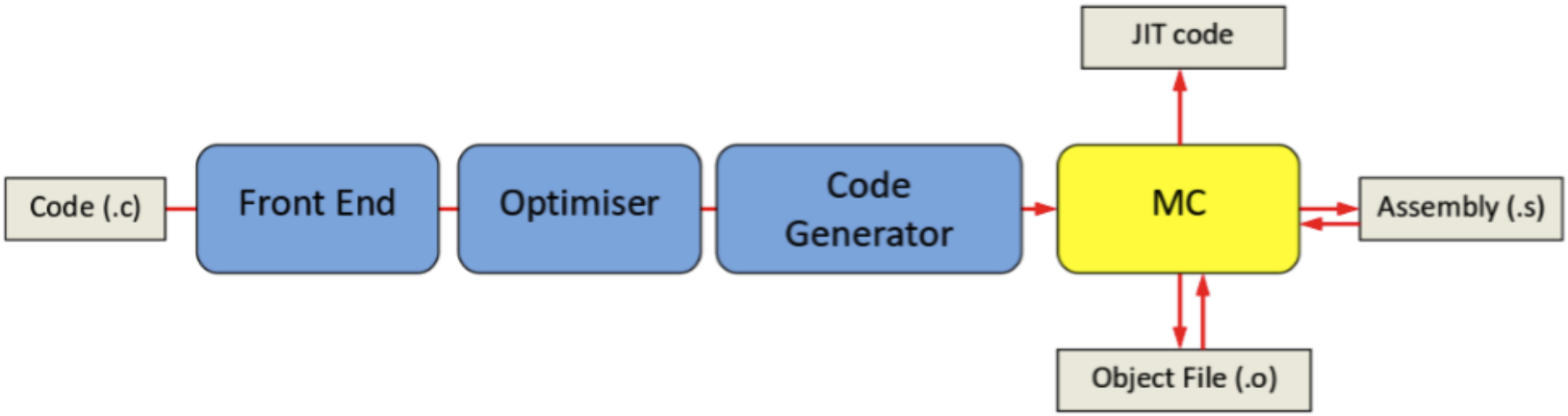
- Not just assembler, but also disassembler, Bitcode, InstPrinter, Linker Optimization, etc
- LLVM codebase is huge and mixed like spaghetti :-(

Keystone job

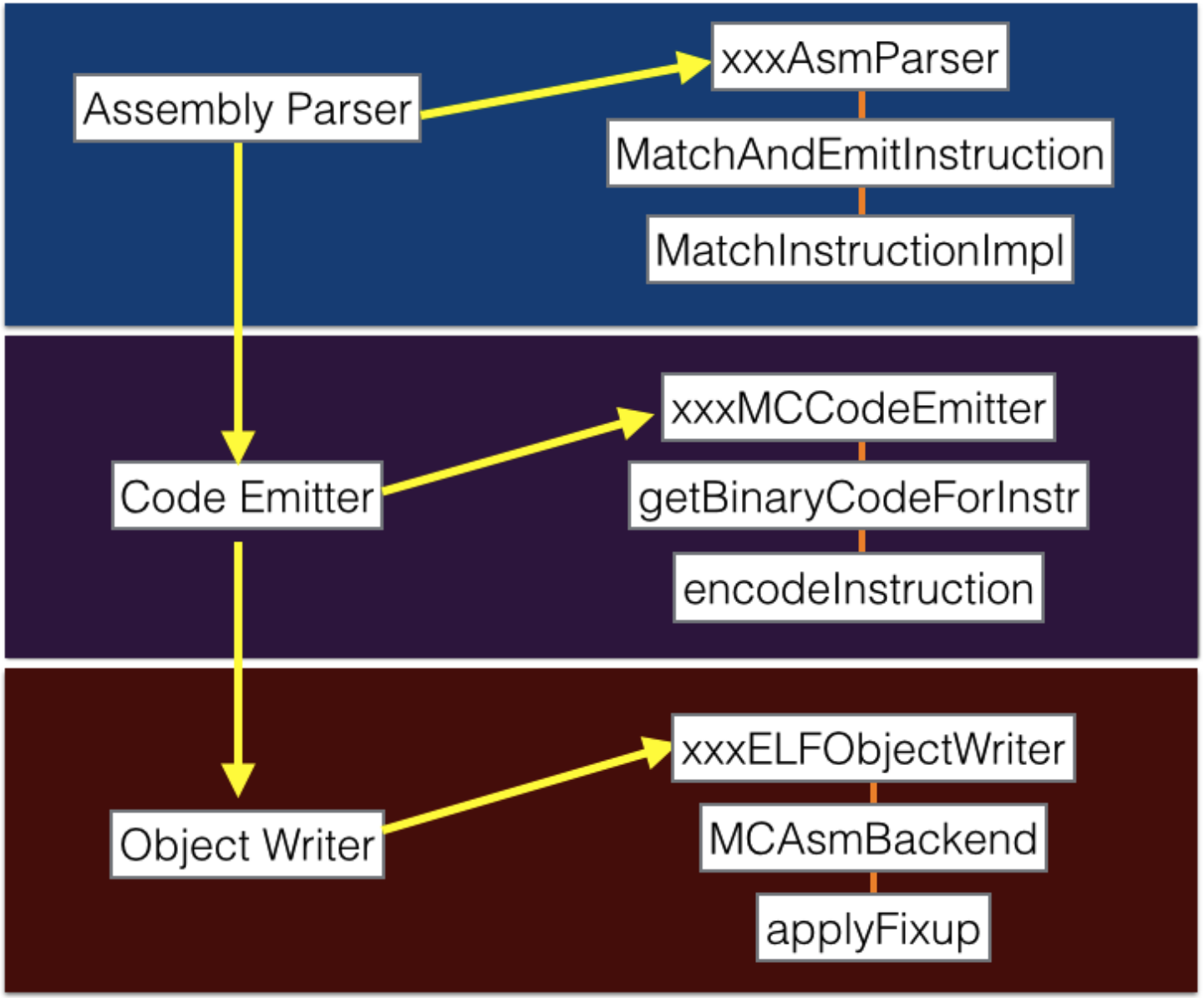
- Keep only assembler code & remove everything else unrelated
- Rewrites some components but keep AsmParser, CodeEmitter & AsmBackend code intact (so easy to sync with LLVM in future)
- Keep all the code in C++ to ease the job (unlike Capstone)
  - No need to rewrite complicated parsers
  - No need to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

Decide where to make the cut

- Where to make the cut?
  - Cut too little result in keeping lots of redundant code
  - Cut too much would change the code structure, making it hard to sync with upstream.
- Optimal design for Keystone
  - Take the assembler core & make minimal changes



Keystone flow



Challenges to build Keystone (2)

Multiple binaries

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

Keystone job

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

Challenges to build Keystone (3)

Code generated MC Assembler is only for linking

- 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

Keystone job

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

Challenges to build Keystone (4)

Unaware of relative branch targets

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

Keystone job

- 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.

Challenges to build Keystone (5)

Give up when failing to handle craft input

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

Keystone job

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

Challenges to build Keystone (6)

Other issues

- 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 maitain ourselves

Keystone job

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

Keystone vs LLVM

Forked LLVM, but go far beyond it

- 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 as of June 2016)

Write applications with Keystone

Introduce Keystone API

- Clean/simple/lightweight/intuitive architecture-neutral API.
- Core implemented in C++, but API provided in C
  - open & close Keystone instance
  - customize runtime instance (allow to change assembly syntax, etc)
  - assemble input code
  - memory management: free allocated memory
- Python/NodeJS/Ruby/Go/Rust/Haskell bindings built around the core

Sample code in C

```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 = KS_ERR_ARCH;
    size_t count;
    unsigned char *encode;
    size_t size, i;

    ks_open(KS_ARCH_X86, KS_MODE_32, &ks);
    ks_asm(ks, CODE, 0, &encode, &size, &count);
    printf("%s = ", CODE);
    for (i = 0; i < size; i++) {
        printf("%02x ", encode[i]);
    }
    printf("\n");

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

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

    return 0;
}
```

Sample code in Python

```
from keystone import *

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

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

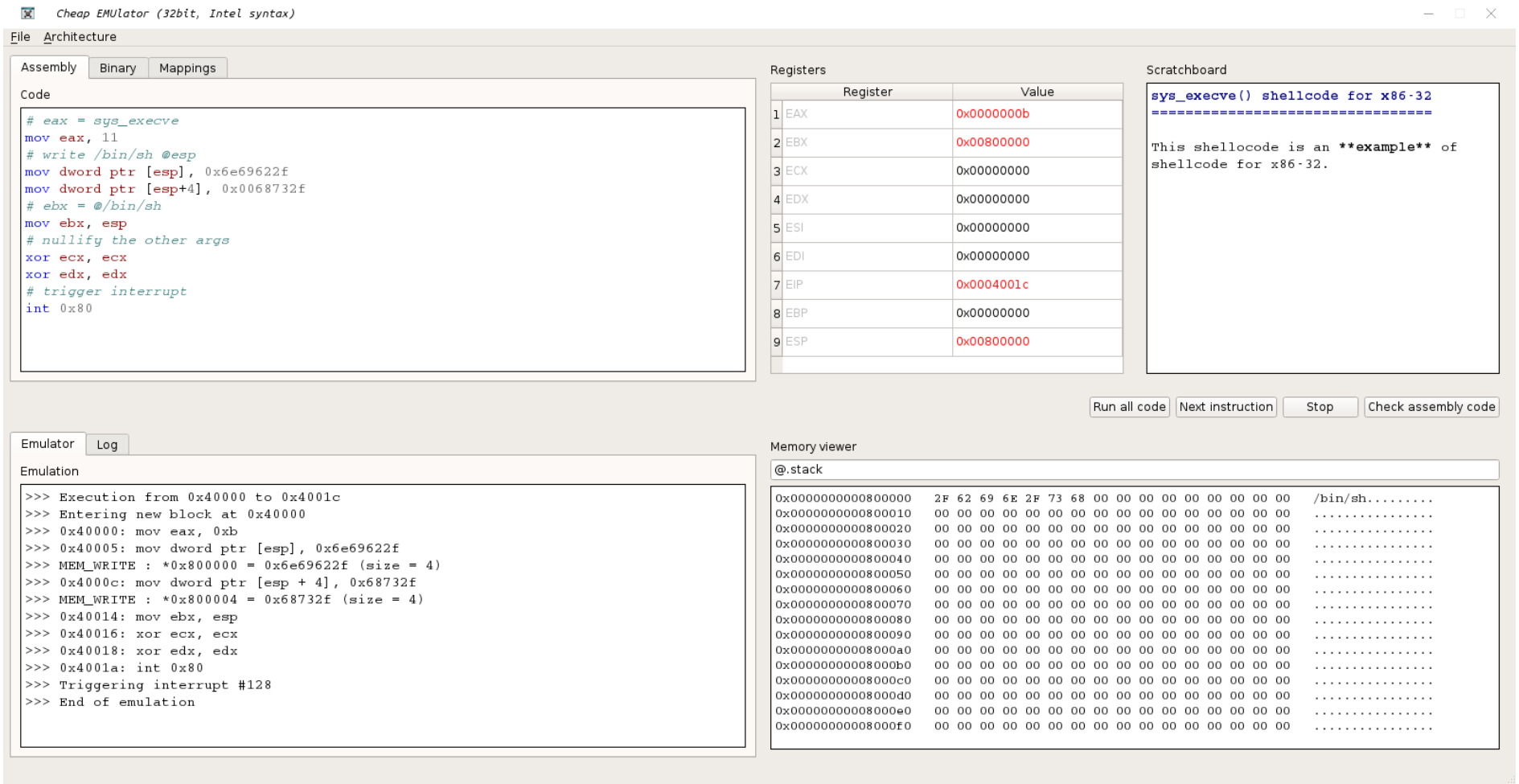
Demo

Shellcode compilation with Pwnypack

- Open source tool https://github.com/edibledinos/pwnypack
- Describe high level operations of shellcode
- Translate operations to low level assembly
- Cross-compile assembly to machine code using Keystone

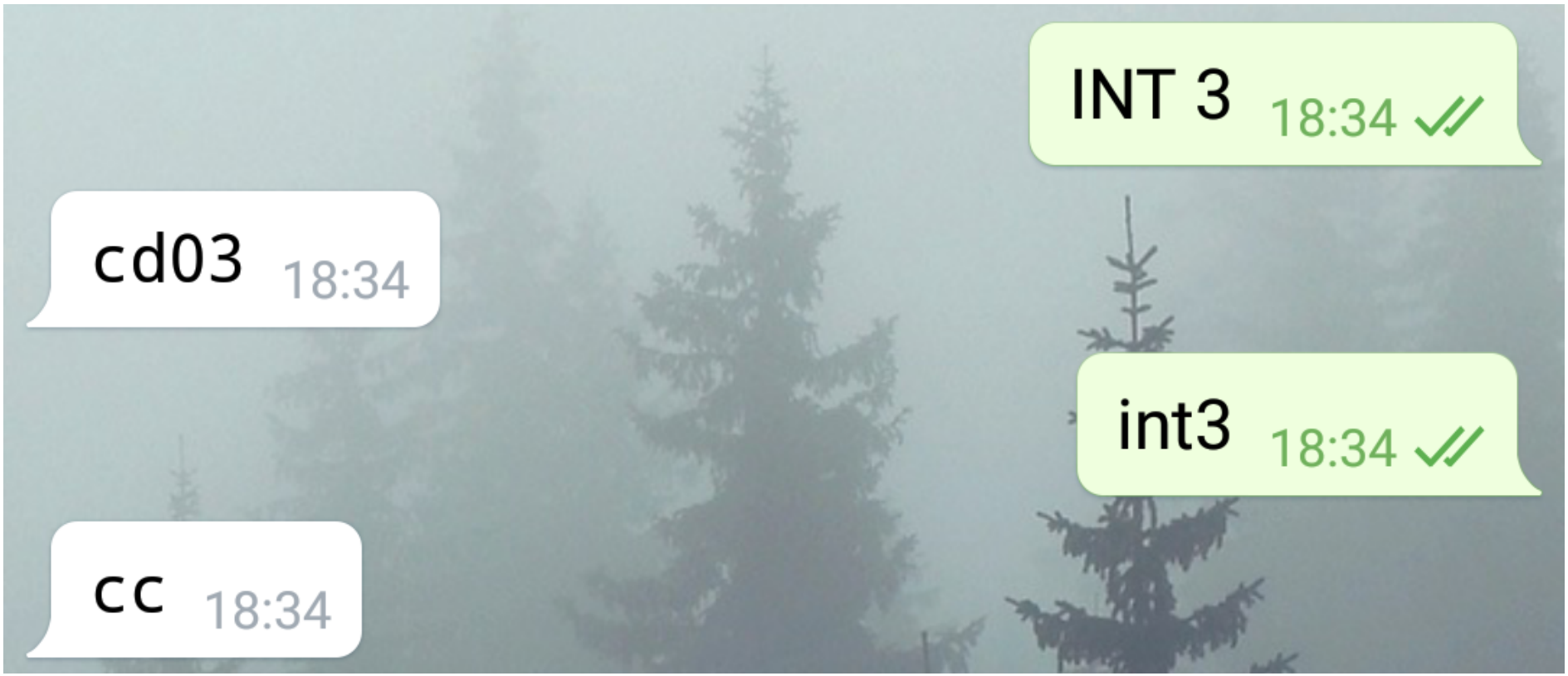
CEMU emulator

- Open source tool https://github.com/hugsy/cemu
- Emulate input assembly instructions
  - Compile assembly input with Keystone
  - Feed the output encoding to Unicorn for emulation



Telegram bot

- Open source bot for Telegram
  https://github.com/mbikovitsky/AssemblyBot
- Receive request on Telegram, and return the result
  - Encode assembly with Keystone
  - Decode hexcode to with Capstone



Other applications from around internet

- Radare2: Unix-like reverse engineering framework and commandline tools
- Ropper: Rop gadget and binary information tool
- Keystone.js: embedding Keystone into Javascript
- GEF: GDB plugin with enhanced features
- Usercorn: Versatile kernel+system+userspace emulator
- X64dbg: An open-source x64/x32 debugger for windows
- Liberation: code injection library for iOS
- More from http://keystone-engine.org/showcase

Status & future works

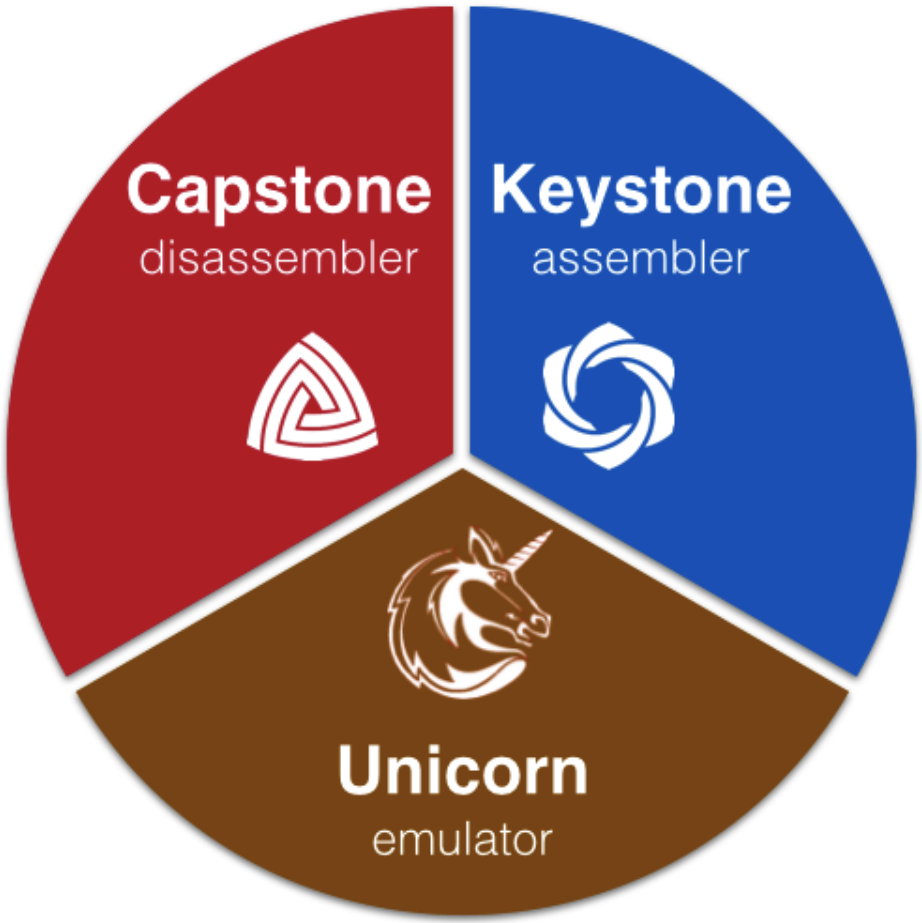
Status

- Version 0.9 went public on May 31st, 2016
- Based on LLVM 3.9
- Version 1.0 will be released as soon as all important bugs get fixed

Future works

- More refined error code returned by parser?
- Find & fix all the corner cases where crafted input cause the core exit
- More bindings promised by community!
- Synchronize with latest LLVM version
  - Future of Keystone is guaranteed by LLVM active development!

Reverse Engineering Trilogy



Conclusions

- Keystone is an innovative next generation assembler
  - Multi-arch + multi-platform
  - Clean/simple/lightweight/intuitive architecture-neutral API
  - Implemented in C++, with API in C language & multiple bindings available
  - Thread-safe by design
  - Open source in dual license
  - Future update guaranteed for all architectures
- We are seriously committed to this project to make it the best assembler engine


KEYSTONE
ENGINE
UNICORN ENGINE
CAPSTONE
ENGINE

References

- Yasm: http://yasm.tortall.net
- LLVM: http://llvm.org
- Keystone assembler
  - Homepage: http://keystone-engine.org
  - Github: http://github.com/keystone-engine/keystone
  - Mailing list: http://freelists.org/list/keystone-engine
  - Twitter: @keystone_engine
- Available apps using Keystone:
  http://keystone-engine.org/showcase
- Capstone disassembler: http://capstone-engine.org
- Unicorn emulator: http://unicorn-engine.org

Acknowledgement

- FX for the inspiration of the Keystone name!
- Ingmar Steen for insight on Pwnypack!
- Indiegogo contributors for amazing financial support!
- Beta testers helped to improve our code for first public release!
- Community for great encouragement!

Questions and answers

KEYSTONE: the last missing framework for RE

http://keystone-engine.org

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