KEYSTONE: Next Generation Assembler Framework

www.keystone-engine.org

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

HackInTheBox GSEC Singapore - August 25th, 2016
Bio

- **Nguyen Anh Quynh (aquynh @ 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](http://capstone-engine.org)
  - Unicorn emulator: [http://unicorn-engine.org](http://unicorn-engine.org)
  - Keystone assembler: [http://keystone-engine.org](http://keystone-engine.org)
Capstone: Next Generation Disassembler Engine

Blackhat USA 2014
Unicorn: Next Generation CPU Emulator

Blackhat USA 2015
Fundamental frameworks for Reverse Engineering

Binary → Keystone → Assembly

Emulator
Fundamental frameworks for Reverse Engineering

Binary → Assembly

Emulator
Assembler framework

**Definition**
- Compile assembly instructions & returns encoding as sequence of bytes
  - Ex: `inc EAX → 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: mnemonic + operands
    - Emit machine code accordingly
    - Instruction-Set-Architecture manual referenced is needed
Challenges of building assembler

- Huge amount of works for the core only!
  - 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  - Yasm: X86 only, no longer updated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 executable parser (ELF) 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
Problems

- No reasonable assembler framework even in 2016!
- Apparently nobody wants to fix the issues
- No light at the end of the dark tunnel
"If not now, then when? If not you, then who?" - Kailash Satyarthi
Keystone (architecture)

From Wikipedia, the free encyclopedia

This article is about the architectural element. For other uses, see Keystone (disambiguation).

A **keystone** is the wedge-shaped stone piece at the apex of a masonry arch, the generally round one at the apex of a **vault**. In both cases it is the final piece placed during construction and locks all the stones into position, allowing the arch or vault to bear weight.¹²³ In both arches and vaults, keystones are often enlarged beyond the structural requirements, and often decorated in some way. Keystones are often placed in the centre of the flat top of openings such as doors and windows, essentially for decorative effect.

![Keystone Diagram](image)
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
- Version 0.9.1 released on July 27th, 2016
  - 2 more bindings: Haskell & OCaml
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’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 implement 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?

FORK ALL THE THINGS

DONE!

imgflip.com
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
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] + libkeystone.a
  - keystone.dll + 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: \texttt{blx 0x86535200} $\rightarrow$ \texttt{0x35, 0xf1, 0x00, 0xe1}

Keystone job
- \texttt{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: \texttt{vaddpd zmm1, zmm1, zmm1, x} → "this is not an immediate"
- Returned \texttt{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 maintain 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 & OCaml
- Keep syncing with LLVM upstream for important changes & bug-fixes
Keystone flow

Assembly Parser
  \[\text{xxxAsmParser}\]
  \[\text{MatchAndEmitInstruction}\]
  \[\text{MatchInstructionImpl}\]

Code Emitter
  \[\text{xxxMCCodeEmitter}\]
  \[\text{getBinaryCodeForInstr}\]
  \[\text{encodeInstruction}\]

Object Writer
  \[\text{xxxELFOObjectWriter}\]
  \[\text{MCAsmBackend}\]
  \[\text{applyFixup}\]
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, OCaml as of August 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/OCaml bindings built around the core
Sample code in C

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

// separate assembly instructions by ; or 
#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

```python
from keystone import *

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

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
Keypatch

- Open source IDA plugin [https://keystone-engine.org/keypatch](https://keystone-engine.org/keypatch)
- Tool for assembling & patching in IDA
- Co-developed with Thanh Nguyen (VNSecurity.net)
Shellcode compilation with Pwnypack

- Open source tool [https://github.com/edibledinos/pwnypack](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
**Metame: Metamorphic engine**

- **Open source tool** [https://github.com/a0rtega/metame](https://github.com/a0rtega/metame)
- **Analyze executable binaries:** Windows, Linux, MacOS
- **Randomly replace instructions with equivalences in logic and size**
- **Bypass signature matching of AntiVirus**

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x080cb9d</td>
<td>eb0d</td>
<td>jmp 0x080cbda0</td>
<td>0x080cb9f</td>
<td>eb0d</td>
<td>jmp 0x080cbda0</td>
</tr>
<tr>
<td>0x080cb9e</td>
<td>90</td>
<td>nop</td>
<td>0x080cb9f</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb9d</td>
<td>90</td>
<td>nop</td>
<td>0x080cb9d</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb9c</td>
<td>90</td>
<td>nop</td>
<td>0x080cb9c</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb9b</td>
<td>90</td>
<td>nop</td>
<td>0x080cb9b</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb9a</td>
<td>90</td>
<td>nop</td>
<td>0x080cb9a</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb99</td>
<td>90</td>
<td>nop</td>
<td>0x080cb99</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb98</td>
<td>90</td>
<td>nop</td>
<td>0x080cb98</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb97</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
<td>0x080cb97</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
</tr>
<tr>
<td>0x080cb96</td>
<td>90</td>
<td>nop</td>
<td>0x080cb96</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb95</td>
<td>90</td>
<td>nop</td>
<td>0x080cb95</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb94</td>
<td>90</td>
<td>nop</td>
<td>0x080cb94</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb93</td>
<td>90</td>
<td>nop</td>
<td>0x080cb93</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb92</td>
<td>90</td>
<td>nop</td>
<td>0x080cb92</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb91</td>
<td>90</td>
<td>nop</td>
<td>0x080cb91</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb90</td>
<td>90</td>
<td>nop</td>
<td>0x080cb90</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb8f</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
<td>0x080cb8f</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
</tr>
<tr>
<td>0x080cb8e</td>
<td>90</td>
<td>nop</td>
<td>0x080cb8e</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb8d</td>
<td>90</td>
<td>nop</td>
<td>0x080cb8d</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb8c</td>
<td>90</td>
<td>nop</td>
<td>0x080cb8c</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb8b</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
<td>0x080cb8b</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
</tr>
<tr>
<td>0x080cb8a</td>
<td>90</td>
<td>nop</td>
<td>0x080cb8a</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb89</td>
<td>90</td>
<td>nop</td>
<td>0x080cb89</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb88</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
<td>0x080cb88</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
</tr>
<tr>
<td>0x080cb87</td>
<td>90</td>
<td>nop</td>
<td>0x080cb87</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb86</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
<td>0x080cb86</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
</tr>
<tr>
<td>0x080cb85</td>
<td>90</td>
<td>nop</td>
<td>0x080cb85</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb84</td>
<td>90</td>
<td>nop</td>
<td>0x080cb84</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb83</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
<td>0x080cb83</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
</tr>
<tr>
<td>0x080cb82</td>
<td>90</td>
<td>nop</td>
<td>0x080cb82</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x080cb81</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
<td>0x080cb81</td>
<td>90</td>
<td>jmp 0x080cbda0</td>
</tr>
<tr>
<td>0x080cb80</td>
<td>90</td>
<td>nop</td>
<td>0x080cb80</td>
<td>90</td>
<td>nop</td>
</tr>
</tbody>
</table>
```

**KEYSTONE: Next Generation Assembler Framework**
Other applications from around internet

- **Radare2**: Unix-like reverse engineering framework and commandline tools
- **Ropper**: Rop gadget and binary information tool
- **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
- **Demovfuscator**: Deobfuscator for movfuscated binaries.
- More from [http://keystone-engine.org/showcase](http://keystone-engine.org/showcase)
Status & future works

Status

- Version 0.9 went public on May 31st, 2016
- Version 0.9.1 was out on July 27th, 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

![Diagram showing the relationship between Binary, Assembly, and Emulator with logos for Capstone Engine and Keystone Engine.](image-url)
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
References

- Keystone assembler
  - Homepage: http://keystone-engine.org
  - Twitter: @keystone_engine
  - Github: http://github.com/keystone-engine/keystone
  - Mailing list: http://freelists.org/list/keystone-engine

- Keypatch: http://keystone-engine.org/keypatch

- 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!
- Indiegogo contributors for amazing financial support!
- Code contributors!
- Community for great encouragement!
<table>
<thead>
<tr>
<th>Questions and answers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KEYSTONE: next generation assembler engine</strong></td>
</tr>
<tr>
<td><a href="http://keystone-engine.org">http://keystone-engine.org</a></td>
</tr>
<tr>
<td>NGUYEN Anh Quynh &lt;aquynh -at- gmail.com&gt;</td>
</tr>
</tbody>
</table>