Capstone: Next-Gen Disassembly Framework

www.capstone-engine.org

Nguyen Anh Quynh, Coseinc
<aquynh@gmail.com>

Blackhat USA, August 7th 2014

Agenda

1. Disassembly engines & their issues
2. Capstone: general ideas & design
   - Capstone goals
   - Capstone design
3. Capstone implementation
4. Some tricky X86 instructions
5. Applications
6. Conclusions

Story behind Capstone

Wanted a decent disassembly framework for my project (2013)

- X86 + ARM
- Windows + Linux
- Friendly license (no GPL)

Capstone is our solution with much more features!

Available disassembly frameworks & problems

Binary analysis & software exploit

Binary analysis

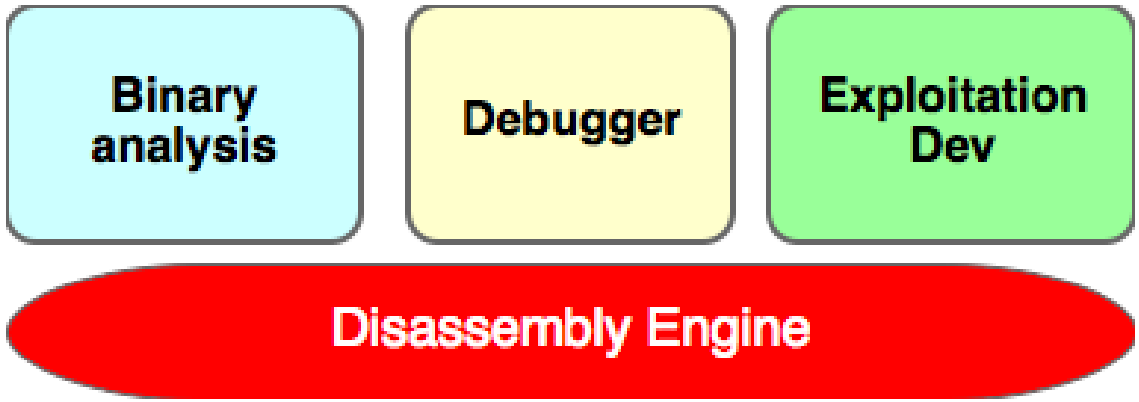
- Reverse binary code (like malware) for good internal understanding.
- Analyze binary code to find vulnerabilities.
- Debug machine code.
  - Machine level code is the only input → working with assembly code is the only choice

Software exploit

- Writing exploitation for software vulnerabilities.
- Building shellcode is an important part of the process.
  - Machine level shellcode is mandatory → working with assembly code is the only choice

Disassemble machine code

- Given binary code, decode and give back assembly code.
  - 01D8 = ADD EAX, EBX (x86)
  - 1160 = STR R1, [R2] (Arm's Thumb)
- Core part of all binary analysis/reverse tool/debugger/exploit development.
- Disassembly framework (or engine/library) is a lower layer in stack of architecture.

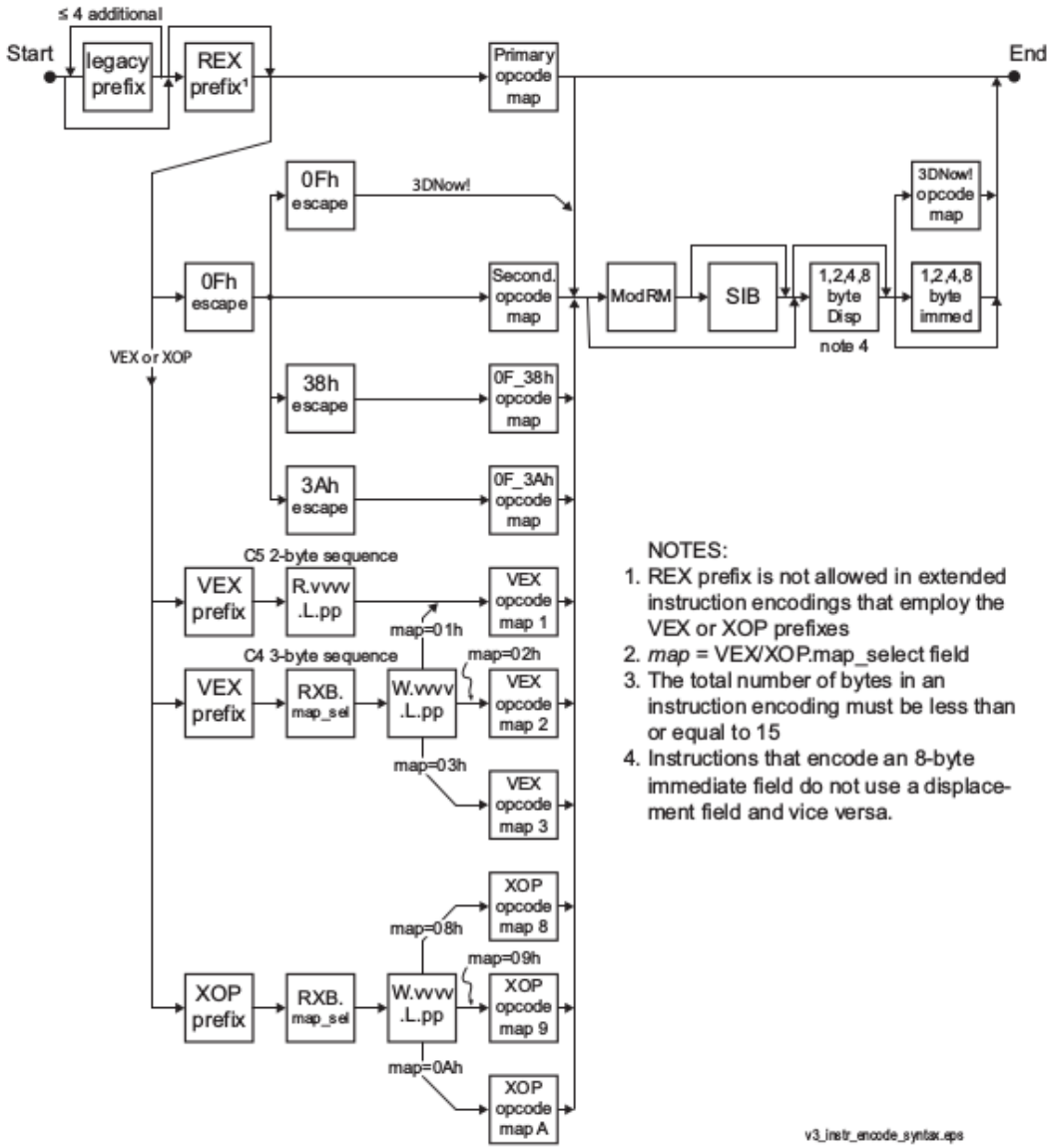


Building disassembly frameworks

- Need good understanding on hardware architectures + instruction sets.
- Decoding the binary code properly to return the assembly.
- Break down assembly in details to help applications to understand instruction internals

```
0x1000: add     dword ptr [ebx + eax*4 + 2], ebx
        Prefix: 0x00 0x00 0x00 0x00
        Opcode: 0x01 0x00 0x00 0x00
        rex: 0x0
        addr_size: 4
        modrm: 0x5c
        disp: 0x02
        sib: 0x83
                sib_base: ebx
                sib_index: eax
                sib_scale: 4
        op_count: 2
                operands[0].type: MEM
                        operands[0].mem.base: REG = ebx
                        operands[0].mem.index: REG = eax
                        operands[0].mem.scale: 4
                        operands[0].mem.disp: 0x02
                operands[0].size: 4
                operands[1].type: REG = ebx
                operands[1].size: 4
        Implicit registers modified: eflags
```

X86 instruction encoding



NOTES:

1. REX prefix is not allowed in extended instruction encodings that employ the VEX or XOP prefixes
2. *map* = VEX/XOP.map_select field
3. The total number of bytes in an instruction encoding must be less than or equal to 15
4. Instructions that encode an 8-byte immediate field do not use a displacement field and vice versa.

Building disassembly frameworks is tedious

- Lots of time spent on understanding instruction encoding schemes.
- Too many instructions to deal with.
- Too many corner cases & undocumented instructions (X86).
- Too many architectures: X86, Arm, Arm64, Mips, PPC, Sparc, etc.
- Language bindings hard to build: Python, Ruby, Java, C#, Javascript, etc

Demanding for a good disassembly framework

- Simple requirements
  - Multiple archs: X86 + Arm
  - Actively maintained & update with latest arch's changes
  - Multiple platforms: Windows + Linux
  - Support Python+Ruby as binding languages
  - Friendly license (GPL is bad!)
- Long standing issue for the security community - with no adequate solution even in 2013.

Available frameworks (2013)

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

[1] Poor quality

[2] Incomplete & unmaintained

Problems

- Nothing works even in 2013. Shame on this industry!
- Apparently nobody wanted to step up to fix the issues.
- No light at the end of the dark tunnel!
- Until Capstone came to rescue!

Capstone = the next generation disassembly framework!

Capstone's goals

- Multi-arch: X86 + Arm + Arm64 + Mips + PPC (surpassed eventually)
- Multi-platform: Windows + MacOSX + Linux (surpassed eventually).
- Multi-bindings: Python + Ruby + Java + C# (surpassed eventually).
- Clean, simple, intuitive & architecture-neutral API.
- Provide break-down details on instructions.
- Friendly license: BSD.

Problems

- Multi-arch: Too much works!
- Multi-platform: Too much works!
- Multi-bindings: Too much works!
- Only possible to finish in few years with very limited resource?

Miracle happened: Capstone made it!

Timeline

- August 2013: Started designing & implementing.
- November 2013: Called for beta test in public.
- December 2013: 1.0 & open source released (www.capstone-engine.org).
- January 2014: 2.0 released.
- March 2014: 2.1 released.
- April 2014: 2.1.2 released.
- August 2014: 3.0 RC1 released (tentative).
- Getting widely adopted by important tools, trainings & works everywhere.
- Packages readily available for all important Operating Systems (Windows, MacOSX, Linux, *BSD)

Capstone status at 7-month old

- Multi-arch: second only to Libopcode.
- Multi-platform: second to none (Windows, OSX, Linux, *BSD, iOS, Android, Solaris)
- Multi-bindings: second to none (9 languages).
- Provide more breakdown instruction details than others.
- Update: more than others.
- Mature: handle more tricky X86 instructions than others.
- Docs: lots of articles for compiling/installing/customizing/programming.

Capstone versus others

Features	Capstone	Distorm3	BeaEngine	Udis86	Libopcode
Multi-arch	✓	X	X	X	✓
X-platform	✓	?	?	?	X
Insn details	✓	✓	✓	X	X
Update	✓	X	?	X	X
License	BSD	GPL	LGPL3	BSD	GPL

- Capstone's archs: Arm, Arm64, Mips, PPC, Sparc, SystemZ, X86, XCore.
- Capstone's bindings: Python, Ruby, C++, C#, Java, NodeJS (JavaScript), GO, OCaml & Vala [3].
- Distorm3's bindings: Python, Ruby (poor quality), Java, C#.
- Others' bindings: Python.

[3] Python, Java & Ocaml maintained by Capstone. The rest made by community

Capstone design

Target

- Have all the desired features in under 1 year.
- With very limited resource available.
- Impossible dream?

Problems

- Multi-arch: Too much works!
- Multi-platform: Too much works!
- Multi-bindings: Too much works!
- Really possible to finish in few years - with very limited resource?

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.

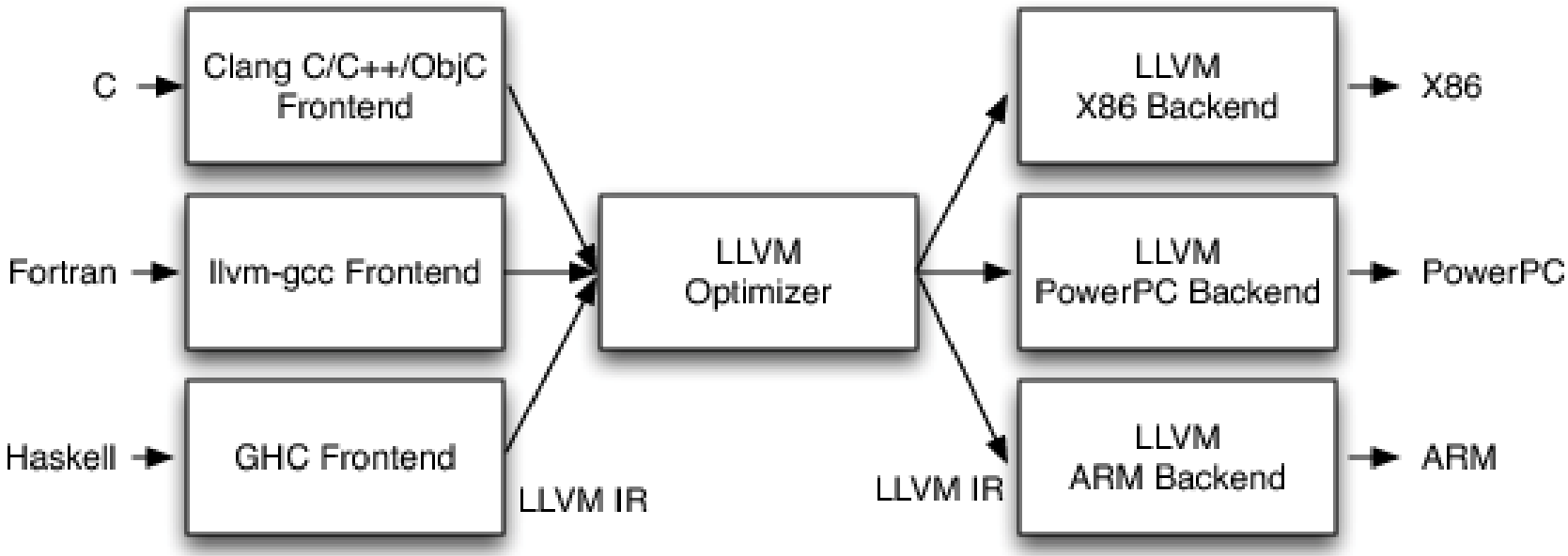
Introduction on LLVM

LLVM project

- Open source project on compiler: www.llvm.org
- A set of frameworks to build compiler
- Set 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.

LLVM model

Compiler model



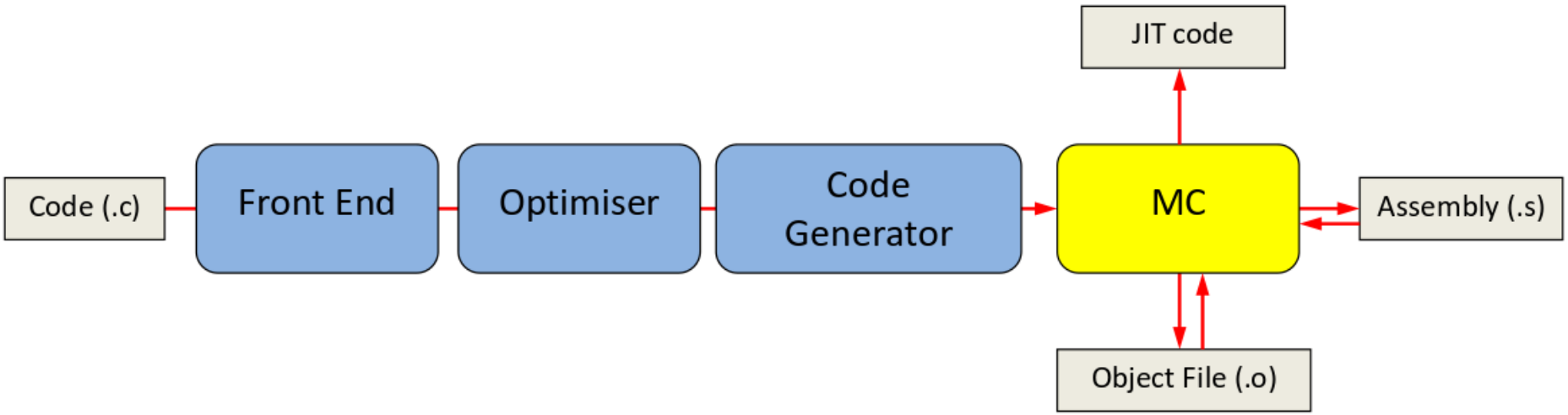
LLVM model: separate Frontend - Optimization - Backend

Why LLVM?

- Support multiple architectures.
- Available disassembler internally in Machine Code (MC) module
  - Only useable for LLVM modules, not for external code.
  - Closely designed & implemented for LLVM.
  - Very actively maintained & updated by a huge community.
- BSD license.
- Fork LLVM to build Capstone around MC!
- Pick up only those archs having disassemblers: 8 archs for now.

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.



Advantages

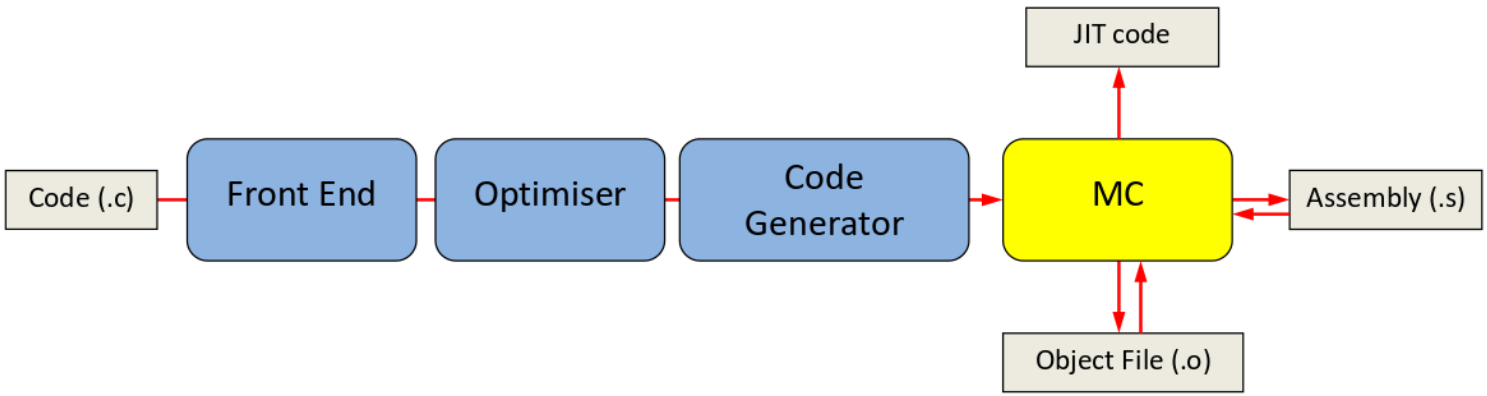
- High quality code with lots of tested done using test cases.
- Disassembler maintained by top experts of each archs.
  - X86: maintained by Intel (arch creator).
  - Arm+Arm64: maintained by Arm & Apple (arch creator & Arm64's device maker).
  - Mips: maintained by Imgtec (arch creator).
  - SystemZ: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

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.

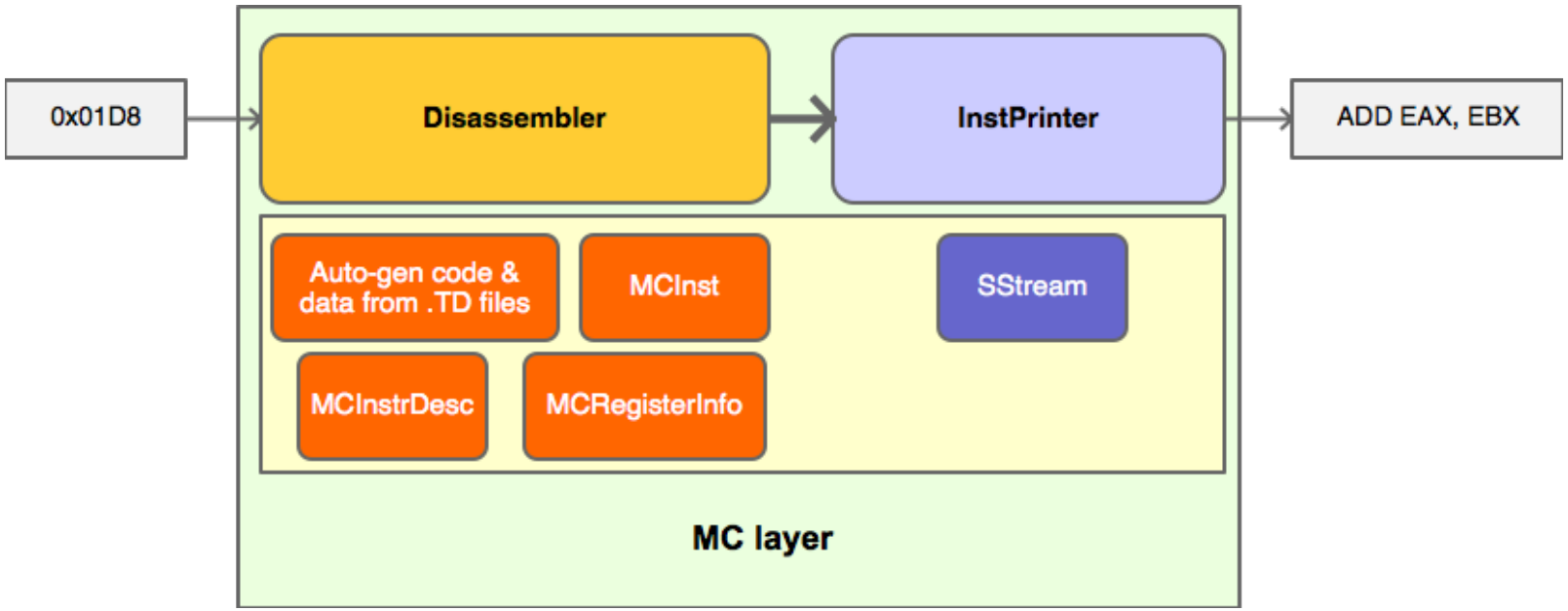
Decide where to make the cut

- Fork LLVM but must remove everything we do not need
- 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 port changes from upstream.
- Optimal design for Capstone chosen.
  - Take the disasm core & make minimal changes.
  - Reimplement required dependent layers ourselves.



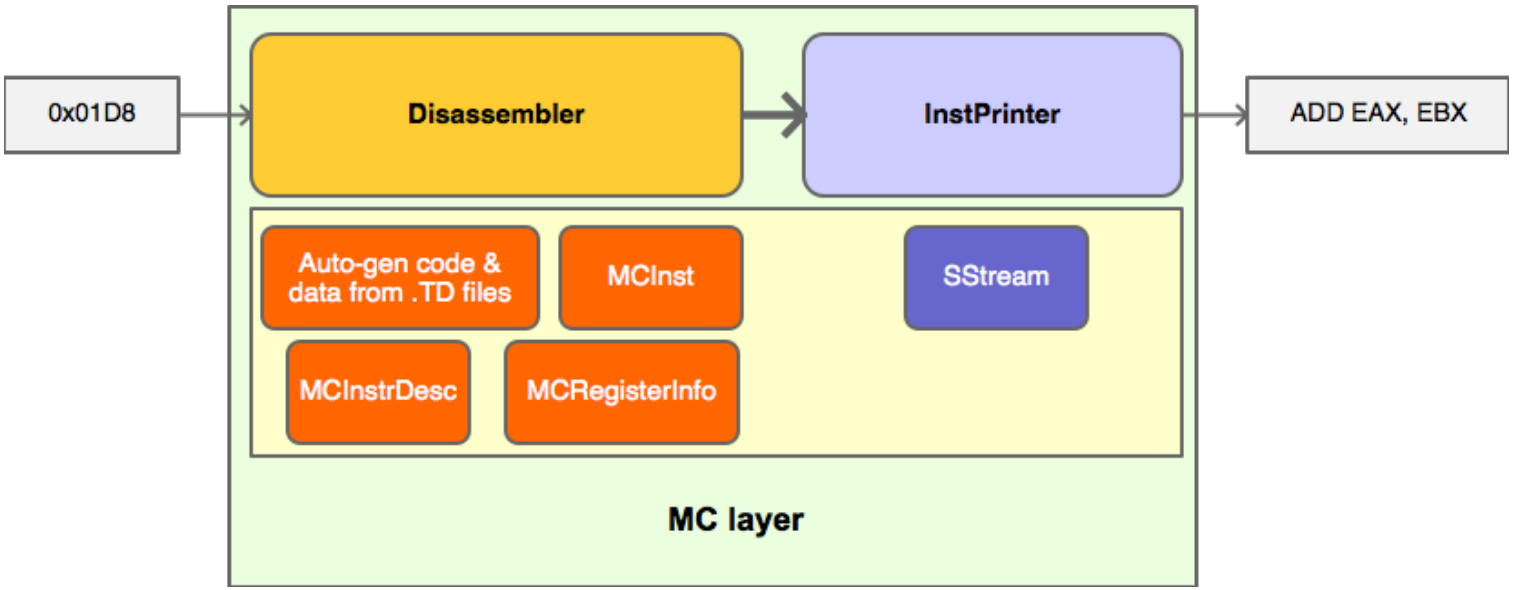
Implementation 1 - replicate LLVM's MC

- Build our core around Disassembler/InstPrinter layers of MC with minimal changes.
  - Rewrite dependent layers of Disassembler: MCInst, MCInstrDesc, MCRegisterInfo.
  - Rewrite dependent layers of InstPrinter: SStream.
- Replace C++ class/method with pure C function pointers + struct/union.
- Fork llvm-tablegen to produce pure C code (*.inc files).



Implementation 2 - extend LLVM's MC

- Hook into InstPrinter layer to build instruction's details (cs_insn struct)
  - Instruction ID, size, mnemonic, operand-string.
  - Operands (Immediate, Register, Memory types)
  - Arch-dependent info for each arch (ex: Prefix, ModRM, SIB, etc for X86)
- Isolate some global variables to make Capstone thread-safe.



Implementation 3 - semantics information

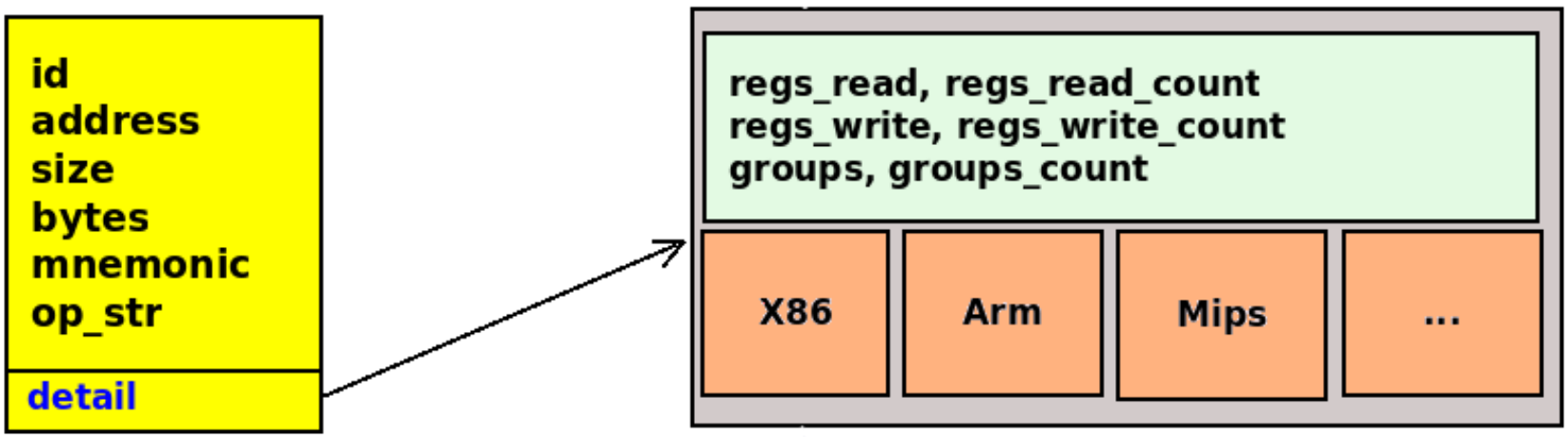
- Take instruction semantics info from *.TD files
  - Available for code analysis & generator.
  - Implicit registers read/written.
  - Instruction's groups.
- Extract these info to put them in mapping tables & copy to cs_insn struct in InstPrinter layer.

```
let Defs = [AL,EFLAGS,AX], Uses = [AL] in
def IMUL8r  : I<0xF6, MRM5r, (outs),  (ins GR8:$src), "imul{b}\t$src", [],
              IIC_IMUL8>, Sched<[WriteIMul]>;
```

```
def CVTSD2SSrm  : I<0x5A, MRMSrcMem, (outs FR32:$dst), (ins f64mem:$src),
                    "cvtsd2ss\t{$src, $dst|$dst, $src}",
                    [(set FR32:$dst, (fround (loadf64 addr:$src)))],
                    IIC_SSE_CVT_Scalar_RM>,
                    XD,
                  Requires<[UseSSE2, OptForSize]>, Sched<[WriteCvtF2FLd]>;
```

cs_insn structure

- Grouped into arch-independent + arch-dependent info.
  - API is arch-independent.
- Grouped in basic mode (default) + detail mode.



Capstone is superior to LLVM's disassembler

- Independent framework - with zero dependency.
- Much more compact in size.
- Provide much more information than just assembly code.
- Thread-safe design.
- Able to embed into restricted firmware/OS environments.
- Malware resistence (X86).
- More optimization towards disassembling/reversing tasks.
- More hardware modes supported: Big-Endian for Arm+Arm64
- More instructions supported: 3DNow (X86).
- More at www.capstone-engine.org/beyond_llvm.html

Robustness of Capstone

- Cannot always rely on LLVM to fix bugs
  - Disassembler is still considered second-class in LLVM, especially if does not affect code generation.
  - May refuse to fix bugs if LLVM backend does not generate them.
    - ★ Tricky & corner cases of X86 code are example.
- But handle all corner cases properly is Capstone's first priority.
  - Handled all X86 malware tricks we are aware of - more than any others.

Embedding Capstone into firmware/OS

- Only build archs you really need.
- Build engine in "diet" mode.
- Build X86 engine in "reduced" mode.
- Special APIs designed to support embedding.
- Find examples for Windows kernel driver + OSX kext in source/docs/README.

Some tricky X86 instructions

Tricky X86 instructions[4]

Hexcode & assembly	Capstone	Distorm3	Beaengine	Udis86	Libopcode	IDA
678B0510000000 (64-bit) mov eax, [eip+10h]	√	X	X	X	√	X
0F1A00 nop dword ptr [eax]	√	X	√	√	X	√
F3F2660F58C0 addpd xmm0, xmm0	√	X	X	X	X	X
F788000000000000000 test dword ptr [eax], 0	√	X	√	√	X	√
D9D8 fstpnce st0, st0	√	X	X	X	X	X
DFDF fstp st0, st7	√	X	X	X	X	X
0F2040 mov eax, cr0	√	√	√	√	X	√

[4]Tested with BeaEngine 3.1, IDA 6.5 & latest versions for others

Write applications with Capstone

Write your tools with Capstone

- Introduce Capstone's API.
- Sample code in C.
- Sample code in Python.
- More tutorials in source/docs/README.

Sample code in C

```c
#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_ex(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].mnemonic,
					insn[j].op_str);
		}

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

	cs_close(&handle);

    return 0;
}
```

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

Sample code in Python

```
from capstone import *

CODE = "\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)
```

Sample Python code to disassemble binary.

```
$ python test1.py

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

Sample Python code - output.

Applications from around internet

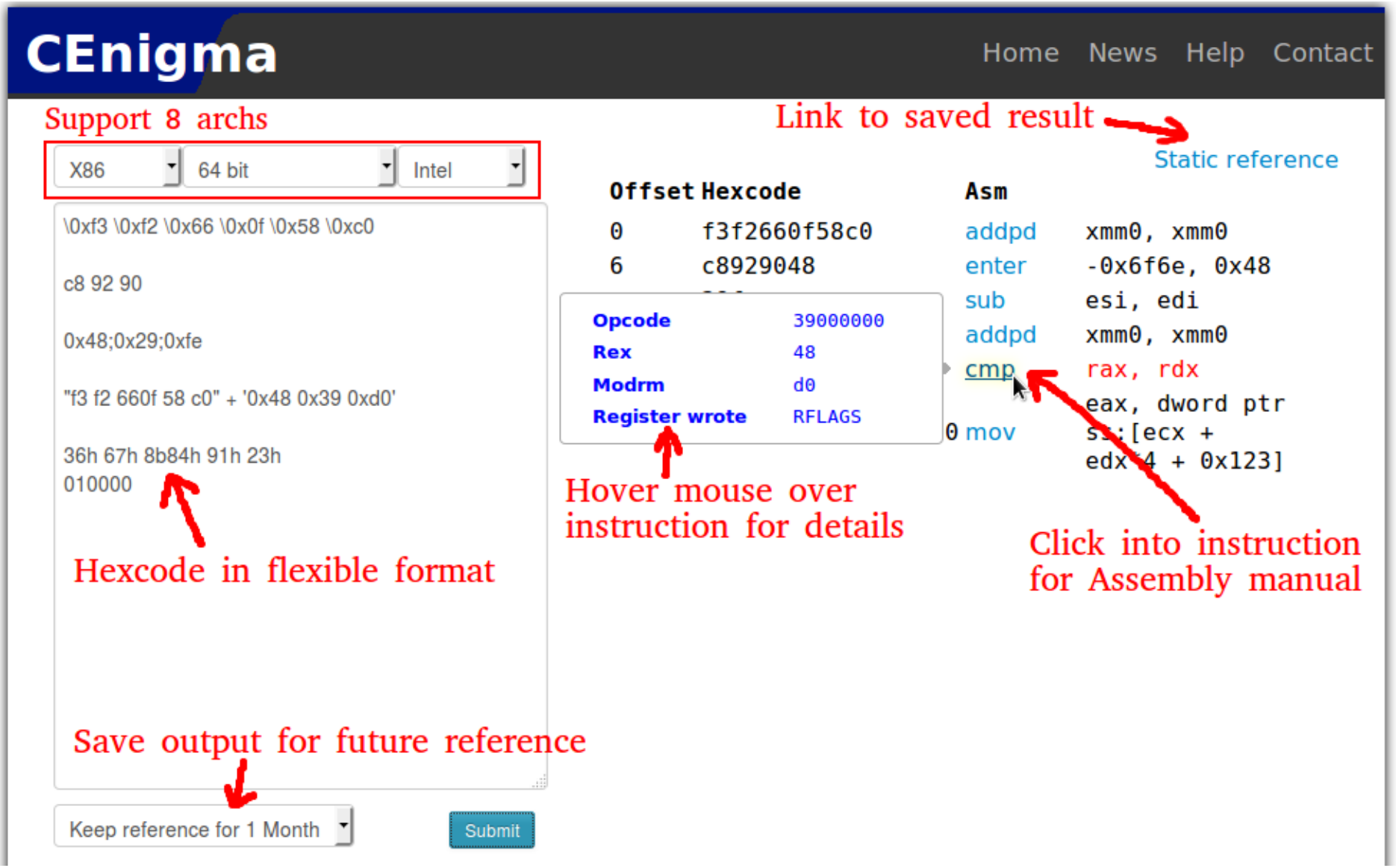
- Camal: Coseinc automated malware analysis lab.
- Pyew: Python tool for static malware analysis.
- Radare2: Unix-like reverse engineering framework and commandline tools.
- ROPGadget: ROP gadgets finder and auto-roper.
- Frida: Inject JavaScript code into native apps on Windows, Mac, Linux and iOS.
- WinAppDbg: Code instrumentation scripts in Python under a Windows environment.
- Cuckoo sandbox: Automated malware analysis.
- PowerSploit: PowerShell Post-Exploitation Framework.
- More at www.capstone-engine.org/showcase.html

CEbot


Rick Flores @nanotechz9l · Mar 25
x32 "\x31\xc0\x40\x89\xc3\xcd\x80\x00\xff\x0a\x0d\x20\x40" #2ce
Details
Reply Retweet Favorite More
Capstone Engine Bot
@ceb0t
Follow
@nanotechz9l
xor eax, eax
inc eax
mov ebx, eax
int 0x80
add bh, bh
Reply Retweet Favorite More
RETWEET
1
FAVORITE
1
11:25 PM - 25 Mar 2014

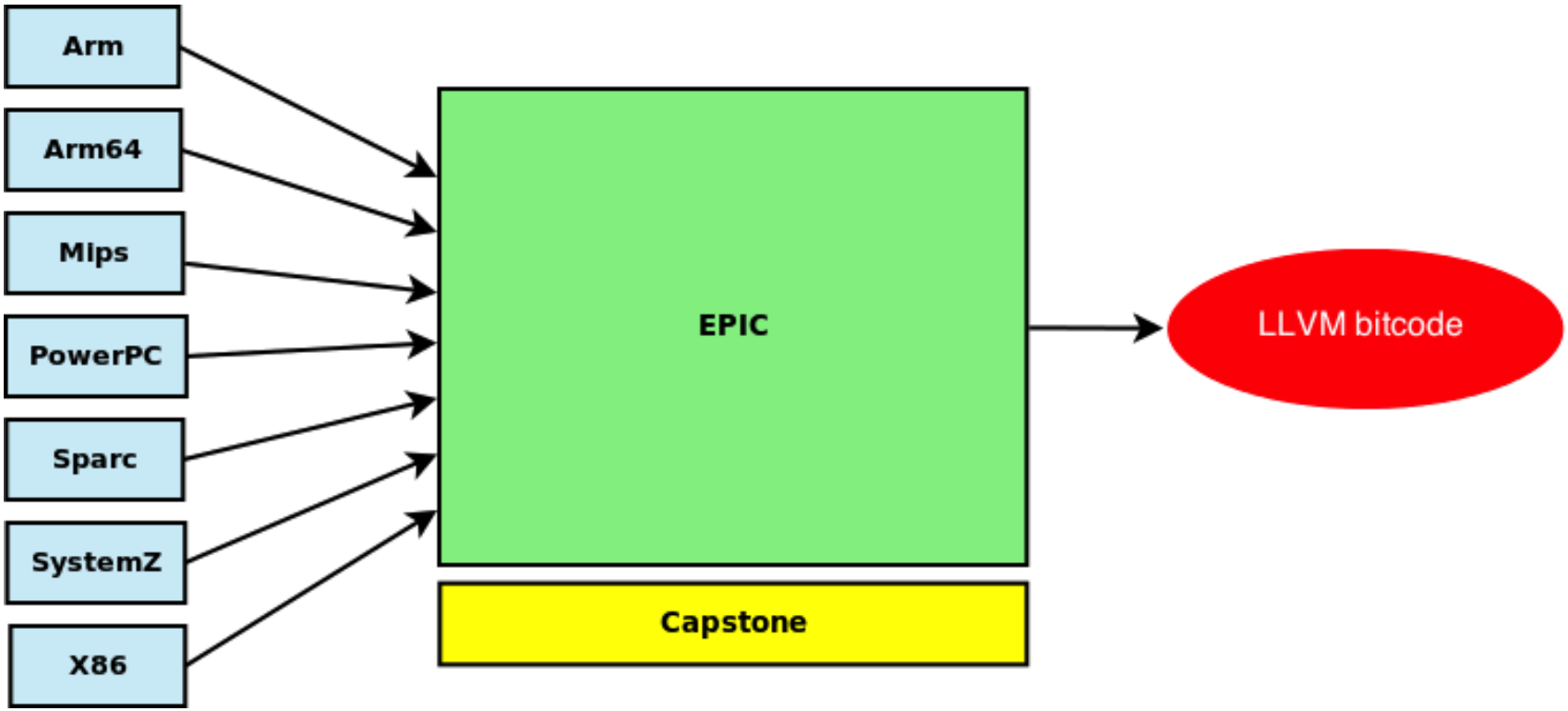
CEnigma

- www.cenigma.org: disassemble hexcode online.



Epic

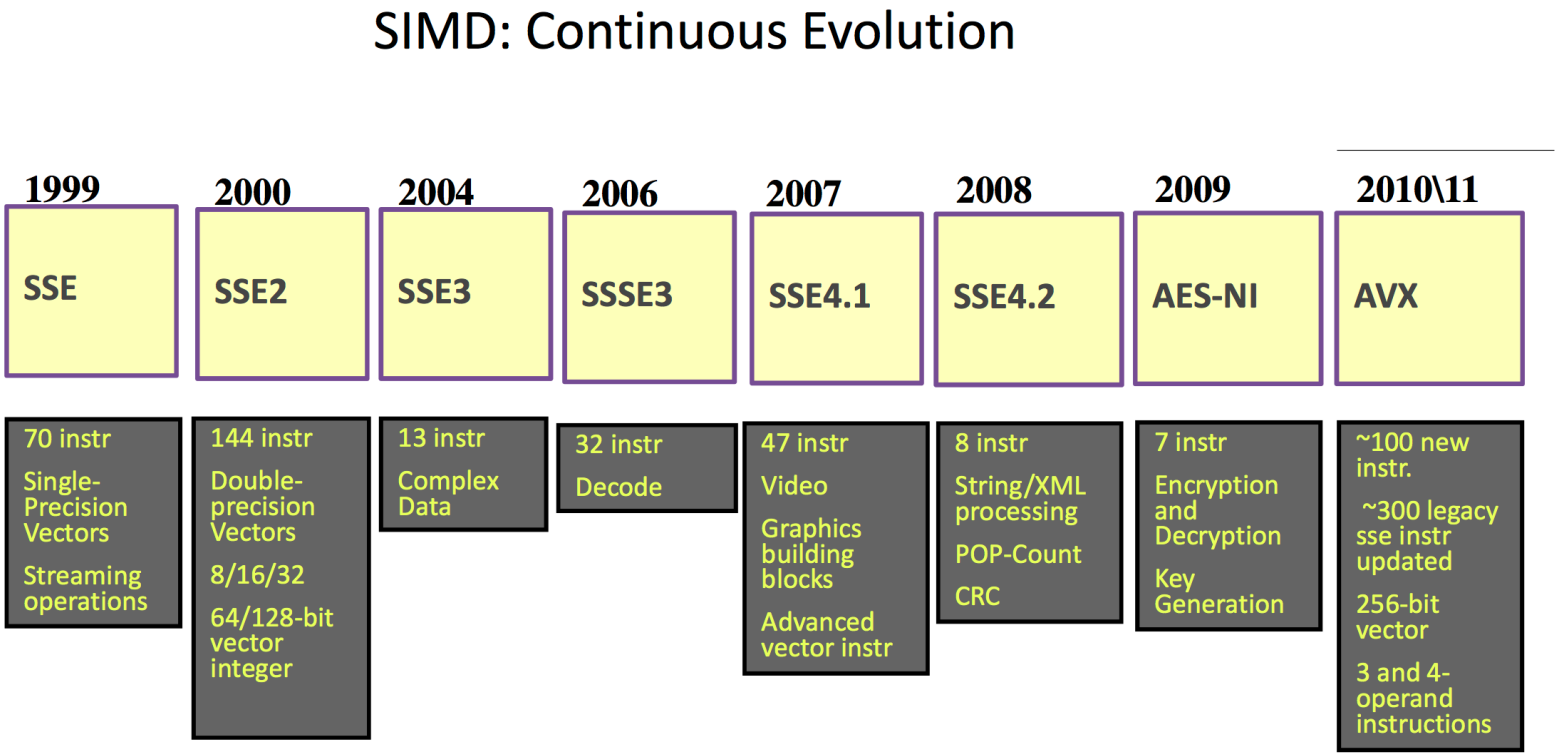
- Framework to translate binaries of any arch to LLVM bitcode
- Enable arch-independent binary analysis using existing LLVM-based tools.



Future works

- More malware resistence: X86.
- More architectures: Hexagon, M68K, etc ?
  - Using code from outside LLVM?
- Provide more semantics of instructions?
- Improve performance further (already very fast).

Capstone's future is guaranteed!



- Story continues: AVX-512 extensions proposed in 2013 to be supported in 2015 (Intel's Knights Landing processor)
- Intel already took care of that for Capstone!

Conclusions

- Capstone is a superior disassembly framework
  - Multi-arch + multi-platform + multi-bindings.
  - Clean/simple/lightweight/intuitive architecture-neutral API.
  - Provide details + semantics on disassembled instruction.
  - Rich choices of options to customize engine at run-time.
  - Special support for embedding into firmware/OS kernel.
  - Future update guaranteed for all archs.
  - Open source BSD license.
- We are seriously committed to this project to make it the best disasm engine.
- More applications building on top of Capstone - soon.

References

- Website: www.capstone-engine.org
- Github source: github.com/aquynh/capstone/tree/next (latest)
- Docs: github.com/aquynh/capstone/blob/next/docs/README
- CEbot: www.capstone-engine.org/bot.html
- CEnigma: www.cenigma.org

Acknowledgements

- Capstone was forked from & will continue to get "supported" by the almighty LLVM project.
- Community support is incredible, thanks!
- Special thanks to all binding authors!
- Shouts to all code contributors & bug reporters!
- Ange Albertini & Dang Hoang Vu for reviewing slides!

Questions and answers

Capstone: Next Generation Disassembly Framework

www.capstone-engine.org

Twitter: @capstone_engine

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