RetDec: An Open-Source Machine-Code Decompiler

Jakub Křoustek
Peter Matula
Petr Zemek

Threat Labs

- Jakub Křoustek
  - founder of RetDec
  - Threat Labs lead @Avast (previously @AVG)
  - reverse engineer, malware hunter, security researcher
  - @JakubKroustek
  - jakub.kroustek@avast.com

- Jakub Křoustek
  - founder of RetDec
  - Threat Labs lead @Avast (previously @AVG)
  - reverse engineer, malware hunter, security researcher
  - @JakubKroustek
  - jakub.kroustek@avast.com
- Peter Matula
  - main developer of the RetDec decompiler
  - senior developer @Avast (previously @AVG)
  - rock climbing and
  - peter.matula@avast.com

Quiz Time

```
.text:004015BB          push    ebp
.text:004015BC          mov     ebp, esp
.text:004015BE          and     esp, 0FFFFFFF0h
.text:004015C1          sub     esp, 20h
.text:004015C4          call    ___main
.text:004015C9          mov     [esp+20h+var_4], 0
.text:004015D1          mov     [esp+20h+var_8], 0
.text:004015D9          mov     [esp+20h+var_C], 0
.text:004015E1          lea     eax, [esp+20h+var_C]
.text:004015E5          mov     [esp+20h+var_18], eax
.text:004015E9          lea     eax, [esp+20h+var_8]
.text:004015ED          mov     [esp+20h+var_1C], eax
.text:004015F1          mov     [esp+20h+Format], offset Format
.text:004015F8          call    _scanf
.text:004015FD          mov     edx, [esp+20h+var_C]
.text:00401601          mov     eax, [esp+20h+var_8]
.text:00401605          mov     [esp+20h+var_1C], edx
.text:00401609          mov     [esp+20h+Format], eax
.text:0040160C          call    _ack
.text:00401611          mov     [esp+20h+var_4], eax
.text:00401615          mov     edx, [esp+20h+var_C]
.text:00401619          mov     eax, [esp+20h+var_8]
.text:0040161D          mov     ecx, [esp+20h+var_4]
.text:00401621          mov     [esp+20h+var_14], ecx
.text:00401625          mov     [esp+20h+var_18], edx
.text:00401629          mov     [esp+20h+var_1C], eax
.text:0040162D          mov     [esp+20h+Format], offset aAckermanDDD
.text:00401634          call    _printf
.text:00401639          mov     eax, [esp+20h+var_4]
.text:0040163D          leave
.text:0040163E          retn
```

```
.text:000110E4    STMFD     SP!, {R11,LR}
.text:000110E8    ADD       R11, SP, #4
.text:000110EC    SUB       SP, SP, #0x14
.text:000110F0    STR       R0, [R11,#var_14]
.text:000110F4    STR       R1, [R11,#var_18]
.text:000110F8    BL        __gccmain
.text:000110FC    MOV       R3, #0
.text:00011100    STR       R3, [R11,#var_8]
.text:00011104    MOV       R3, #0
.text:00011108    STR       R3, [R11,#var_C]
.text:0001110C    MOV       R3, #0
.text:00011110    STR       R3, [R11,#var_10]
.text:00011114    SUB       R2, R11, #-var_C
.text:00011118    SUB       R3, R11, #-var_10
.text:0001111C    LDR       R0, =aDD
.text:00011120    MOV       R1, R2
.text:00011124    MOV       R2, R3
.text:00011128    BL        scanf
.text:0001112C    LDR       R2, [R11,#var_C]
.text:00011130    LDR       R3, [R11,#var_10]
.text:00011134    MOV       R0, R2
.text:00011138    MOV       R1, R3
.text:0001113C    BL        ack
.text:00011140    MOV       R3, R0
.text:00011144    STR       R3, [R11,#var_8]
.text:00011148    LDR       R2, [R11,#var_C]
.text:0001114C    LDR       R3, [R11,#var_10]
.text:00011150    LDR       R0, =aAckermanDDD
.text:00011154    MOV       R1, R2
.text:00011158    MOV       R2, R3
.text:0001115C    LDR       R3, [R11,#var_8]
.text:00011160    BL        printf
```

```
.text:08900428              addiu   $sp, -0x20
.text:0890042C              sw      $ra, 0x20+var_4($sp)
.text:08900430              sw      $fp, 0x20+var_8($sp)
.text:08900434              move    $fp, $sp
.text:08900438              sw      $a0, 0x20+var_10($fp)
.text:0890043C              sw      $a1, 0x20+var_C($fp)
.text:08900440              sw      $zero, 0x20+var_20($fp)
.text:08900444              sw      $zero, 0x20+var_1C($fp)
.text:08900448              sw      $zero, 0x20+var_18($fp)
.text:0890044C              addiu   $v1, $fp, 0x20+var_1C
.text:08900450              addiu   $a2, $fp, 0x20+var_18
.text:08900454              lui     $v0, 0x891
.text:08900458              addiu   $a0, $v0, (aDD - 0x8910000)
.text:0890045C              move    $a1, $v1
.text:08900460              jal     scanf
.text:08900464              nop
.text:08900468              lw      $v0, 0x20+var_1C($fp)
.text:0890046C              lw      $v1, 0x20+var_18($fp)
.text:08900470              move    $a0, $v0
.text:08900474              move    $a1, $v1
.text:08900478              jal     ack
.text:0890047C              nop
.text:08900480              sw      $v0, 0x20+var_20($fp)
.text:08900484              lw      $v1, 0x20+var_1C($fp)
.text:08900488              lw      $a2, 0x20+var_18($fp)
.text:0890048C              lui     $v0, 0x891
.text:08900490              addiu   $a0, $v0, (aAckermanDDD - 0x8910000)
.text:08900494              move    $a1, $v1
.text:08900498              lw      $a3, 0x20+var_20($fp)
.text:0890049C              jal     printf
```

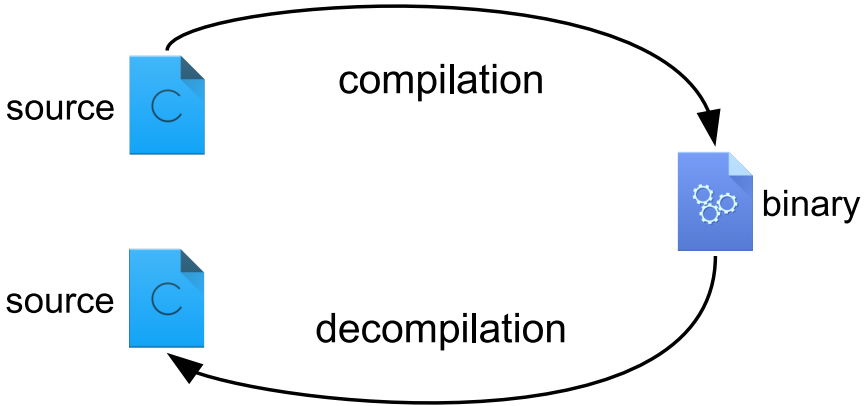
```
.text:1000056C                stwu      r1, back_chain(r1)
.text:10000570                mflr      r0
.text:10000574                stw       r0, 0x30+sender_lr(r1)
.text:10000578                stw       r31, 0x30+var_4(r1)
.text:1000057C                mr        r31, r1
.text:10000580                stw       r3, 0x18(r31)
.text:10000584                stw       r4, 0x1C(r31)
.text:10000588                li        r0, 0
.text:1000058C                stw       r0, 8(r31)
.text:10000590                li        r0, 0
.text:10000594                stw       r0, 0xC(r31)
.text:10000598                li        r0, 0
.text:1000059C                stw       r0, 0x10(r31)
.text:100005A0                lis       r0, 0x1000
.text:100005A4                addic     r11, r0, 0x82C # 0x1000082C
.text:100005A8                addi      r9, r31, 0xC
.text:100005AC                addi      r0, r31, 0x10
.text:100005B0                mr        r3, r11
.text:100005B4                mr        r4, r9
.text:100005B8                mr        r5, r0
.text:100005BC                crclr     4*cr1+eq
.text:100005C0                bl        __isoc99_scanf
.text:100005C4                lwz       r9, 0xC(r31)
.text:100005C8                lwz       r0, 0x10(r31)
.text:100005CC                mr        r3, r9
.text:100005D0                mr        r4, r0
.text:100005D4                bl        ack
.text:100005D8                stw       r3, 8(r31)
.text:100005DC                lis       r0, 0x1000
.text:100005E0                addic     r11, r0, 0x834 # 0x10000834
.text:100005E4                lwz       r9, 0xC(r31)
.text:100005E8                lwz       r0, 0x10(r31)
```

Disassembling vs. Decompilation

```
.text:004015BB          push    ebp
.text:004015BC          mov     ebp, esp
.text:004015BE          and     esp, 0FFFFFFF0h
.text:004015C1          sub     esp, 20h
.text:004015C4          call    ___main
.text:004015C9          mov     [esp+20h+var_4], 0
.text:004015D1          mov     [esp+20h+var_8], 0
.text:004015D9          mov     [esp+20h+var_C], 0
.text:004015E1          lea     eax, [esp+20h+var_C]
.text:004015E5          mov     [esp+20h+var_18], eax
.text:004015E9          lea     eax, [esp+20h+var_8]
.text:004015ED          mov     [esp+20h+var_1C], eax
.text:004015F1          mov     [esp+20h+Format], offset Format
.text:004015F8          call    _scanf
.text:004015FD          mov     edx, [esp+20h+var_C]
.text:00401601          mov     eax, [esp+20h+var_8]
.text:00401605          mov     [esp+20h+var_1C], edx
.text:00401609          mov     [esp+20h+Format], eax
.text:0040160C          call    _ack
.text:00401611          mov     [esp+20h+var_4], eax
.text:00401615          mov     edx, [esp+20h+var_C]
.text:00401619          mov     eax, [esp+20h+var_8]
.text:0040161D          mov     ecx, [esp+20h+var_4]
.text:00401621          mov     [esp+20h+var_14], ecx
.text:00401625          mov     [esp+20h+var_18], edx
.text:00401629          mov     [esp+20h+var_1C], eax
.text:0040162D          mov     [esp+20h+Format], offset aAckermanDDD
.text:00401634          call    _printf
.text:00401639          mov     eax, [esp+20h+var_4]
.text:0040163D          leave
.text:0040163E          retn
```

The same, but decompiled to C

```c
int main(int argc, char ** argv) {
    ___main();
    int32_t v1 = 0;
    int32_t v2 = 0;
    scanf("%d %d", &v1, &v2);
    int32_t result = _ack(v1, v2);
    printf("ackerman( %d , %d ) = %d\n", v1, v2, result);
    return result;
}
```


source
compilation
binary
source
decompilation

Decompilation? What good is it?

- Binary analysis
  - reverse engineering
  - malware analysis
  - vulnerability detection
  - verification
  - binary comparison
  - ...

Decompilation? What good is it?

avast

- Binary analysis
  - reverse engineering
  - malware analysis
  - vulnerability detection
  - verification
  - binary comparison
  - ...
- Binary recompilation (yeah, like that's ever gonna work)
  - porting
  - bug fixing
  - adding new features
  - original sources got lost
  - optimizations

Ok, why aren't we already using it?

- Multiple existing tools: Hex-Rays, Hopper, Snowman, etc.

Ok, why aren't we already using it?

- Multiple existing tools: Hex-Rays, Hopper, Snowman, etc.
- It is damn hard

Ok, why aren't we already using it?

- Multiple existing tools: Hex-Rays, Hopper, Snowman, etc.
- It is damn hard
  - compilation is not lossless
    - high-level constructions
    - data types
    - names
    - comments, macros, ...

Ok, why aren't we already using it?

- Multiple existing tools: Hex-Rays, Hopper, Snowman, etc.
- It is damn hard
  - compilation is not lossless
    - high-level constructions
    - data types
    - names
    - comments, macros, ...
  - compilers are optimizing

Ok, why aren't we already using it?

- Multiple existing tools: Hex-Rays, Hopper, Snowman, etc.
- It is damn hard
  - compilation is not lossless
    - high-level constructions
    - data types
    - names
    - comments, macros, ...
  - compilers are optimizing
  - computer science goodies
    - undecidable problems
    - complex algorithms
    - exponential complexities

Ok, why aren't we already using it?

- Multiple existing tools: Hex-Rays, Hopper, Snowman, etc.
- It is damn hard
  - compilation is not lossless
    - high-level constructions
    - data types
    - names
    - comments, macros, ...
  - compilers are optimizing
  - computer science goodies
    - undecidable problems
    - complex algorithms
    - exponential complexities
  - obfuscation, packing, anti-debugging

Generic decompilation? Even harder

- Many architectures
  - x86, ARM, MIPS, PowerPC, . . .
  - CISC vs. RISC
  - bit length, endianness, floating points
  - versions & extensions

Generic decompilation? Even harder

- Many architectures
  - x86, ARM, MIPS, PowerPC, . . .
  - CISC vs. RISC
  - bit length, endianness, floating points
  - versions & extensions
- Many ABIs

Generic decompilation? Even harder

avast

- Many architectures
  - x86, ARM, MIPS, PowerPC, ...
  - CISC vs. RISC
  - bit length, endianness, floating points
  - versions & extensions
- Many ABIs
- Many OFFs (object-file formats)
  - ELF, PE, Mach-O, ...

Generic decompilation? Even harder

avast

- Many architectures
  - x86, ARM, MIPS, PowerPC, ...
  - CISC vs. RISC
  - bit length, endianness, floating points
  - versions & extensions
- Many ABIs
- Many OFFs (object-file formats)
  - ELF, PE, Mach-O, ...
- Many programming languages

Generic decompilation? Even harder

- Many architectures
  - x86, ARM, MIPS, PowerPC, ...
  - CISC vs. RISC
  - bit length, endianness, floating points
  - versions & extensions
- Many ABIs
- Many OFFs (object-file formats)
  - ELF, PE, Mach-O, ...
- Many programming languages
- Many compilers & optimizations

Generic decompilation? Even harder

avast

- Many architectures
  - x86, ARM, MIPS, PowerPC, ...
  - CISC vs. RISC
  - bit length, endianness, floating points
  - versions & extensions
- Many ABIs
- Many OFFs (object-file formats)
  - ELF, PE, Mach-O, ...
- Many programming languages
- Many compilers & optimizations
- Statically linked code

Generic decompilation? Even harder

- Many architectures
  - x86, ARM, MIPS, PowerPC, ...
  - CISC vs. RISC
  - bit length, endianness, floating points
  - versions & extensions
- Many ABIs
- Many OFFs (object-file formats)
  - ELF, PE, Mach-O, ...
- Many programming languages
- Many compilers & optimizations
- Statically linked code
- ...

- Goal
  - generic decompilation of binary code

Retargetable Decompiler (RetDec)

avast

- Goal
  - generic decompilation of binary code
- History
  - 2011–2013 (AVG + BUT FIT via TAČR TA01010667 grant)
  - 2013–2016 (AVG + BUT FIT students via diploma theses)
  - 2016–* (Avast + BUT FIT students)

Retargetable Decompiler (RetDec)

avast

- Goal
  - generic decompilation of binary code
- History
  - 2011–2013 (AVG + BUT FIT via TAČR TA01010667 grant)
  - 2013–2016 (AVG + BUT FIT students via diploma theses)
  - 2016–* (Avast + BUT FIT students)
- People
  - 3-4 core developers
  - $\approx$ 20 BSc/MSc/PhD students

Retargetable Decompiler (RetDec)

avast

- Goal
  - generic decompilation of binary code
- History
  - 2011–2013 (AVG + BUT FIT via TAČR TA01010667 grant)
  - 2013–2016 (AVG + BUT FIT students via diploma theses)
  - 2016–* (Avast + BUT FIT students)
- People
  - 3-4 core developers
  - ≈ 20 BSc/MSc/PhD students
- Lines of code
  - 419,451 code
  - 205,222 comments, etc.
  - 624,673 total

Supports

- architectures (32-bit): x86, ARM, PowerPC, MIPS
- OFFs: ELF, PE, COFF, Mach-O, Intel HEX, AR, raw
- compilers (we test with): gcc, Clang, MSVC

RetDec? What does it do

avast

- Supports
  - architectures (32-bit): x86, ARM, PowerPC, MIPS
  - OFFs: ELF, PE, COFF, Mach-O, Intel HEX, AR, raw
  - compilers (we test with): gcc, Clang, MSVC
- Does
  - compiler/packer detection
  - statically linked code detection
  - OS loader simulation
  - recursive traversal disassembling
  - high-level constructions/types reconstruction
  - pattern detection
  - ...

- Supports
  - architectures (32-bit): x86, ARM, PowerPC, MIPS
  - OFFs: ELF, PE, COFF, Mach-O, Intel HEX, AR, raw
  - compilers (we test with): gcc, Clang, MSVC
- Does
  - compiler/packer detection
  - statically linked code detection
  - OS loader simulation
  - recursive traversal disassembling
  - high-level constructions/types reconstruction
  - pattern detection
  - ...
- Runs on (hopefully)

Windows

Linux

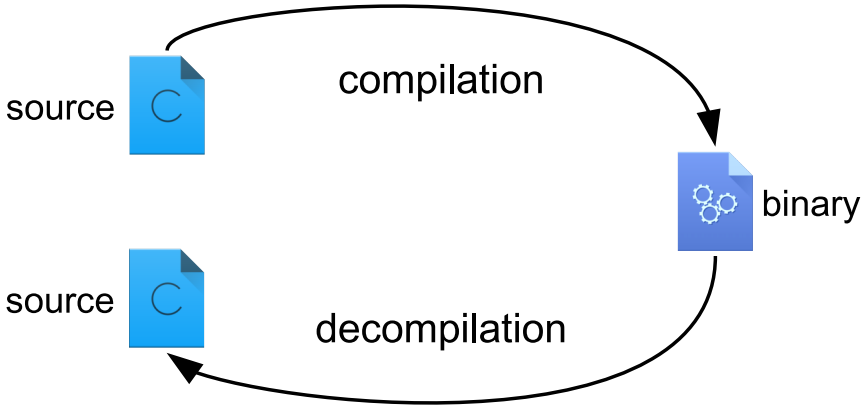
Good news everyone!

- RetDec goes open-source under the MIT license
  - december 2017, shortly after the conference

Good news everyone!

- RetDec goes open-source under the MIT license
  - december 2017, shortly after the conference
- Repositories
  - 11 core
  - 6 support
  - 8 third party
- Contacts
  - https://retdec.com/
  - https://github.com/avast-tl
  - https://twitter.com/retdec
  - https://retdec.com/rss/
  - info@retdec.com

How to get from EXE to C . . .




KEYSTONE
ENGINE


LLVM
Bleeding Edge Compiler Technology


yara


{JSON}

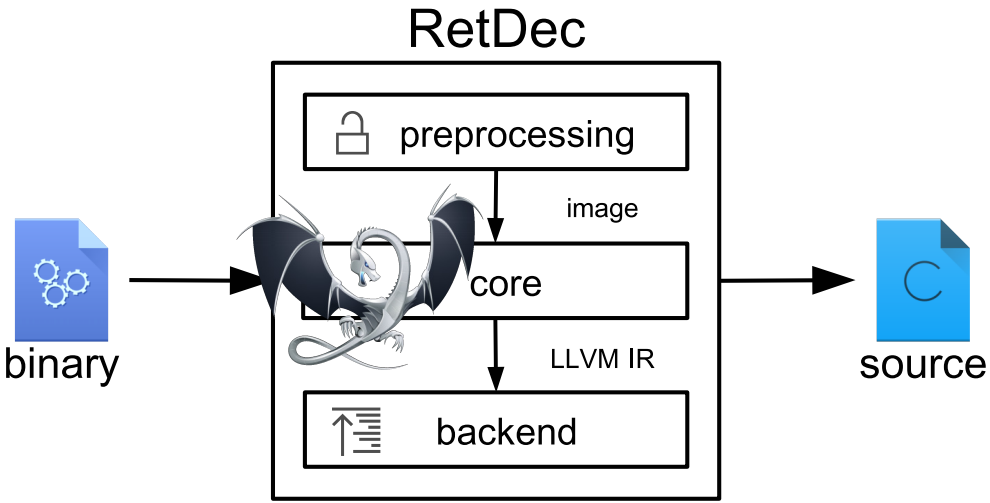

googletest
Google C++ Testing Framework

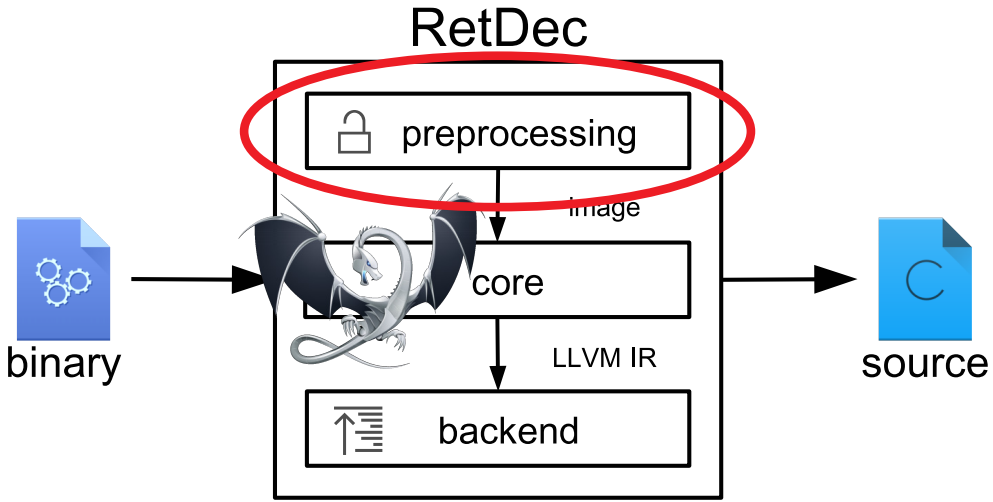

OpenSSL


C++


BASH
THE BOURNE-AGAIN SHELL

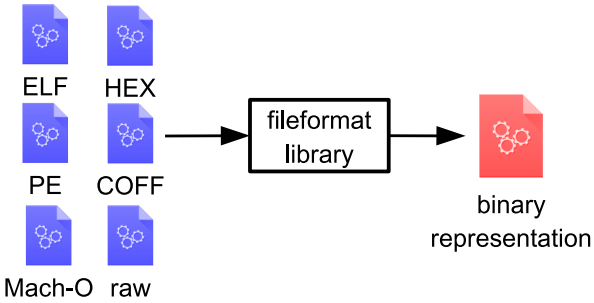

python™

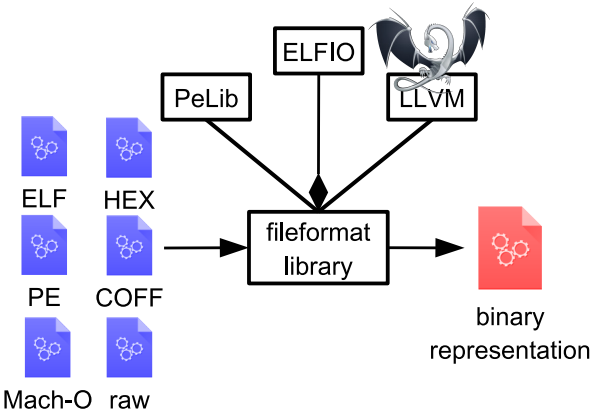

RetDec
preprocessing
image
core
LLVM IR
backend
binary
source

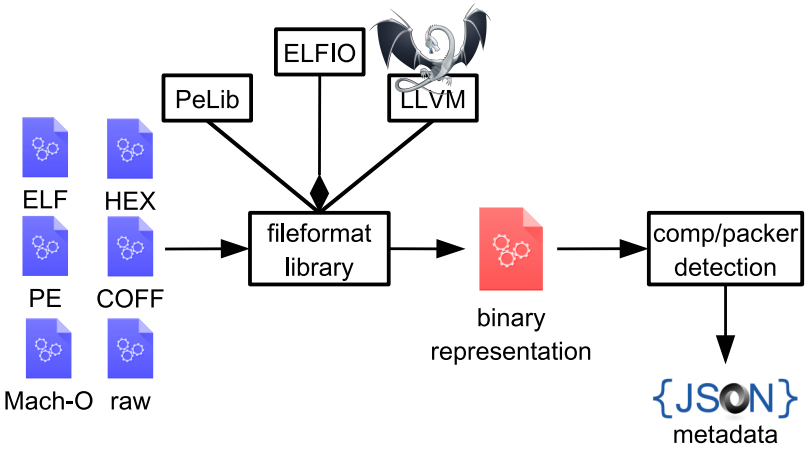

RetDec
preprocessing
image
core
LLVM IR
backend
binary
source

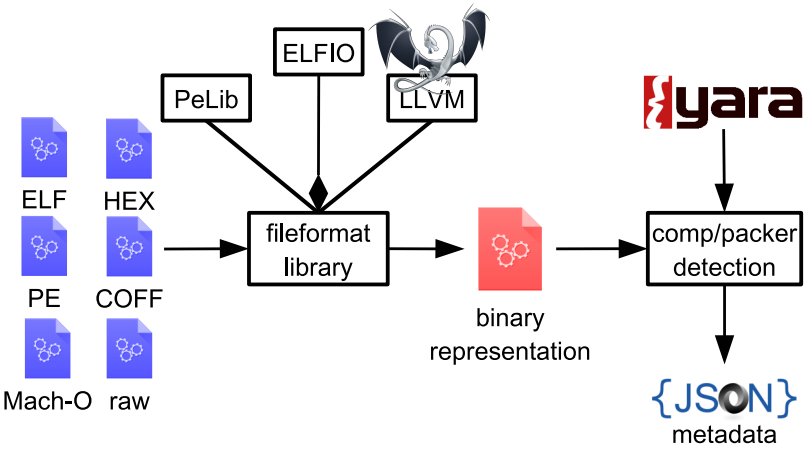

binary

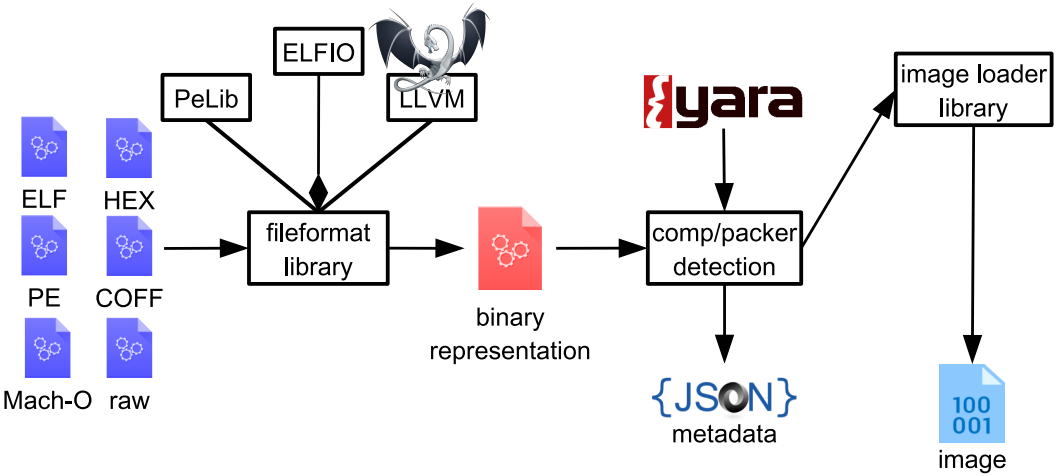

ELF
HEX
PE
COFF
Mach-O
raw

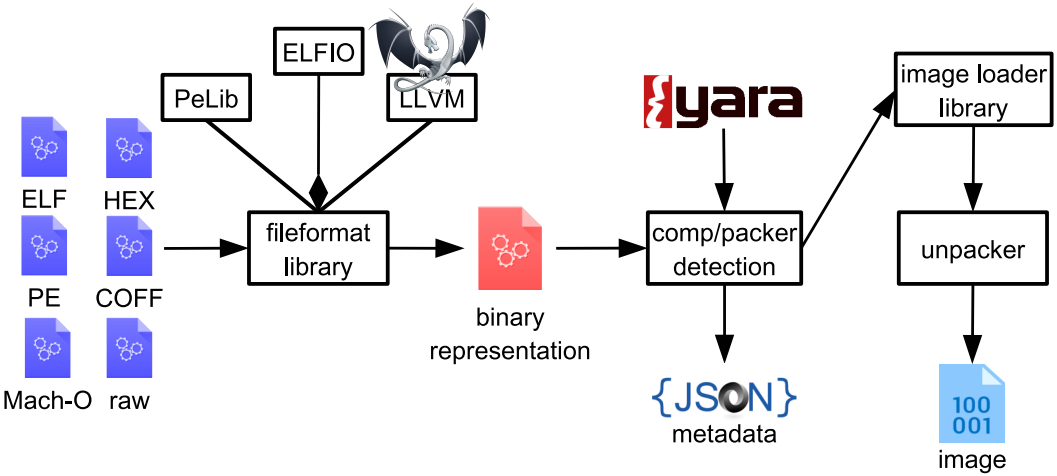

ELF
HEX
PE
COFF
Mach-O
raw
fileformat
library
binary
representation

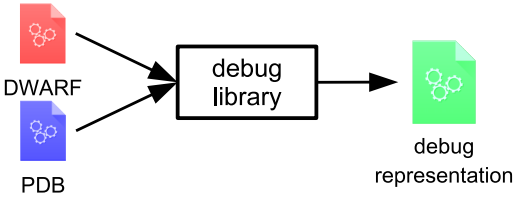

ELFIO
PeLib
LLVM
ELF
HEX
PE
COFF
Mach-O
raw
fileformat
library
binary
representation

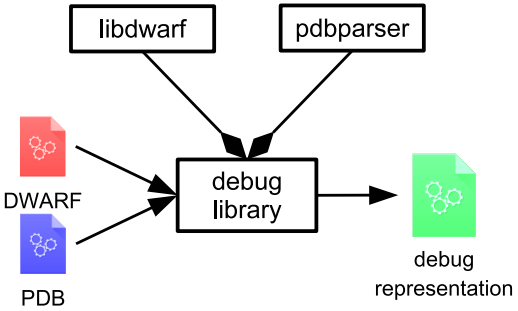

ELFIO
PeLib
LLVM
ELF
HEX
PE
COFF
Mach-O
raw
fileformat library
binary representation
comp/packer detection
{JSON}
metadata


ELFIO
PeLib
LLVM
yara
ELF
HEX
PE
COFF
Mach-O
raw
fileformat
library
binary
representation
comp/packer
detection
{JSON}
metadata


ELFIO
PeLib
LLVM
yara
image loader library
ELF
HEX
PE
COFF
Mach-O
raw
fileformat library
binary representation
comp/packer detection
{JSON}
metadata
100
001
image


ELFIO
PeLib
LLVM
ELF
HEX
PE
COFF
Mach-O
raw
fileformat library
binary representation
yara
comp/packer detection
{JSON}
metadata
image loader library
unpacker
100
001
image


DWARF
PDB
debug
library
debug
representation


libdwarf
pdbparser
DWARF
PDB
debug
library
debug
representation

git Fileformat

- fileformat, loader, cpdetect, fileinfo, unpacker
- ar-extractor, macho-extractor, . . .

Preprocessing repos

git Fileformat

- fileformat, loader, cpdetect, fileinfo, unpacker
- ar-extractor, macho-extractor, . . .

git PeLib

- strengthened
- new modules (rich header, delayed imports, security dir, . . .)

Preprocessing repos

git Fileformat
- fileformat, loader, cpdetect, fileinfo, unpacker
- ar-extractor, macho-extractor, . . .

git PeLib
- strengthened
- new modules (rich header, delayed imports, security dir, . . .)

git ELFIO
- strengthened

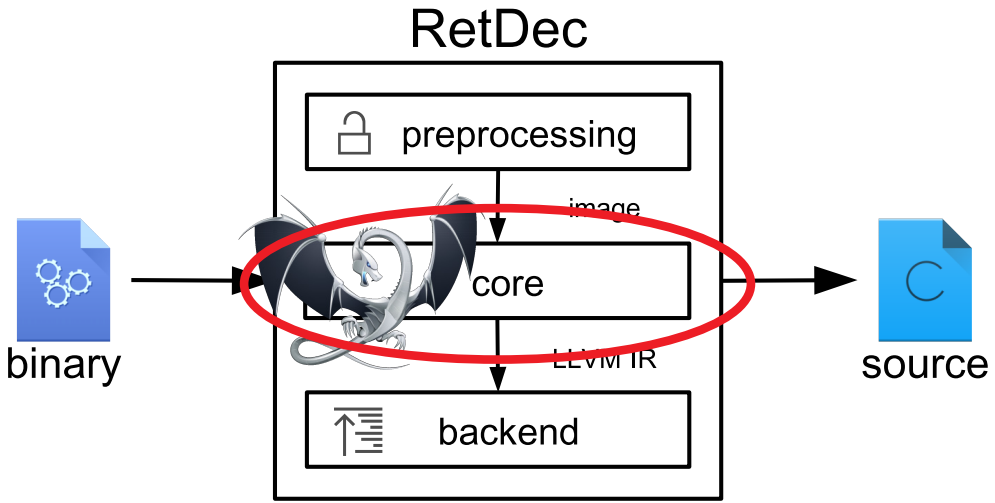
Preprocessing repos

avast

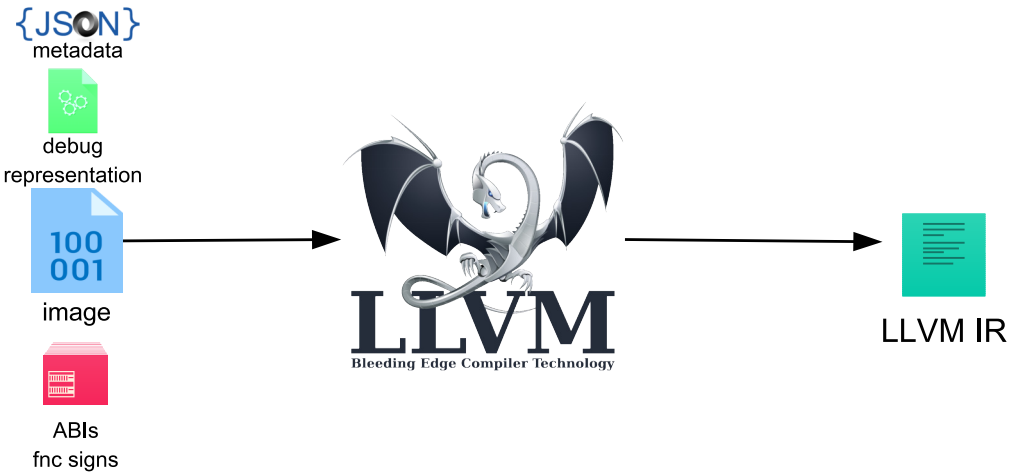
- **git** Fileformat
  - fileformat, loader, cpdetect, fileinfo, unpacker
  - ar-extractor, macho-extractor, . . .
- **git** PeLib
  - strengthened
  - new modules (rich header, delayed imports, security dir, . . .)
- **git** ELFIO
  - strengthened
- **git** PDBparser
  - will hopefully be replaced by LLVM parsers

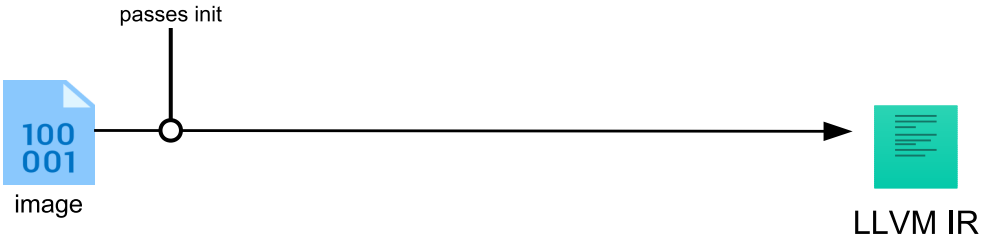
Preprocessing repos

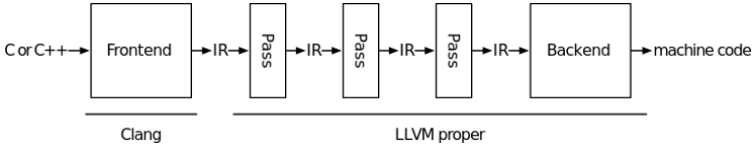
- **git** Fileformat
  - fileformat, loader, cpdetect, fileinfo, unpacker
  - ar-extractor, macho-extractor, ...
- **git** PeLib
  - strengthened
  - new modules (rich header, delayed imports, security dir, ...)
- **git** ELFIO
  - strengthened
- **git** PDBparser
  - will hopefully be replaced by LLVM parsers
- **git** Yaracpp
  - YARA C++ wrapper


RetDec
preprocessing
image
core
LLVM IR
backend
binary
source

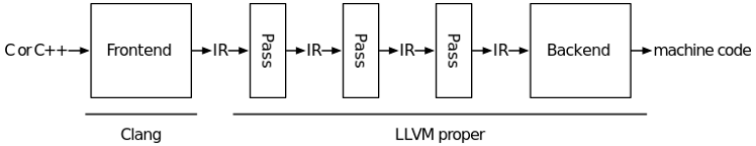

100
001
image
LLVM
Bleeding Edge Compiler Technology
LLVM IR


{JSON}
metadata
debug
representation
100
001
image
ABIs
fnc signs
LLVM
Bleeding Edge Compiler Technology
LLVM IR


passes init
100
001
image
LLVM IR



- dozens of analysis & transform & utility passes
  - dead global elimination, constant propagation, inlining, reassociation, loop optimization, memory promotion, dead store elimination, . . .



- dozens of analysis & transform & utility passes
  - dead global elimination, constant propagation, inlining, reassociation, loop optimization, memory promotion, dead store elimination, ...
- `clang -o hello hello.c -O3`
  - 217 passes
  - `-targetlibinfo -tti -tbaa -scoped-noalias -assumption-cache-tracker -profile-summary-info -forceattrs -inferattrs -ipsccp -globalopt -domtree -mem2reg -deadargelim -domtree -basicaa -aa -instcombine -simplifycfg -basiccg -globals-aa -prune-eh -inline -functionattrs -argpromotion -domtree -sroa -basicaa -aa -memoryssa -early-cse-memssa -speculative-execution -domtree -basicaa -aa -lazy-value-info -jump-threading...`

Core: LLVM IR

- LLVM IR = LLVM Intermediate Representation
  - kind of assembly language / three address code

```
@global = global i32
define i32 @fnc(i32 %arg) {
    %x = load i32, i32* @global
    %y = add i32 %x, %arg
    store i32 %y, @global
    return i32 %y
}
```

Core: LLVM IR

- LLVM IR = LLVM Intermediate Representation
  - kind of assembly language / three address code

```
@global = global i32
define i32 @fnc(i32 %arg) {
    %x = load i32, i32* @global
    %y = add i32 %x, %arg
    store i32 %y, @global
    return i32 %y
}
```

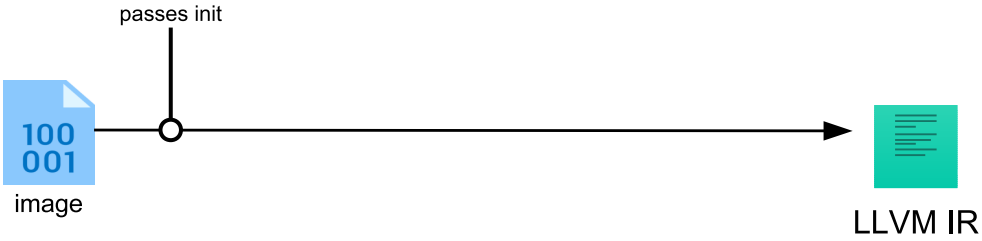
- SSA = Static Single Assignment
  - `%y = add i32 %x, %arg`
- Load/Store architecture
  - `%x = load i32, i32* @global`
- Functions, arguments, returns, data types
- (Un)conditional branches, switches

- LLVM IR = LLVM Intermediate Representation
  - kind of assembly language / three address code

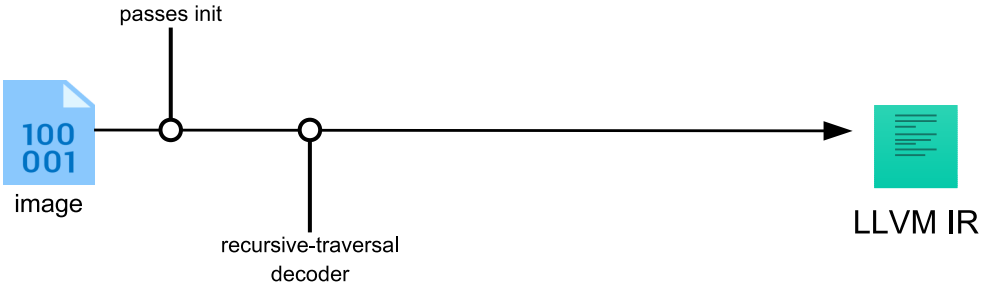
```
@global = global i32
define i32 @fnc(i32 %arg) {
    %x = load i32, i32* @global
    %y = add i32 %x, %arg
    store i32 %y, @global
    return i32 %y
}
```

- SSA = Static Single Assignment
  - `%y = add i32 %x, %arg`
- Load/Store architecture
  - `%x = load i32, i32* @global`
- Functions, arguments, returns, data types
- (Un)conditional branches, switches
- 👍 Universal IR for efficient compiler transformations and analyses

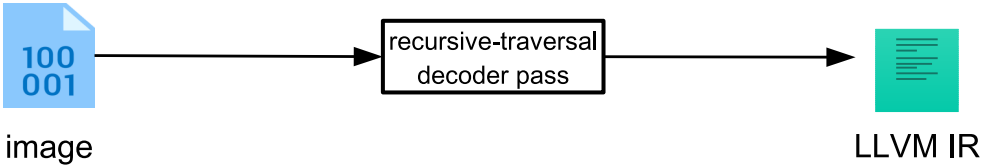
Core: decoder

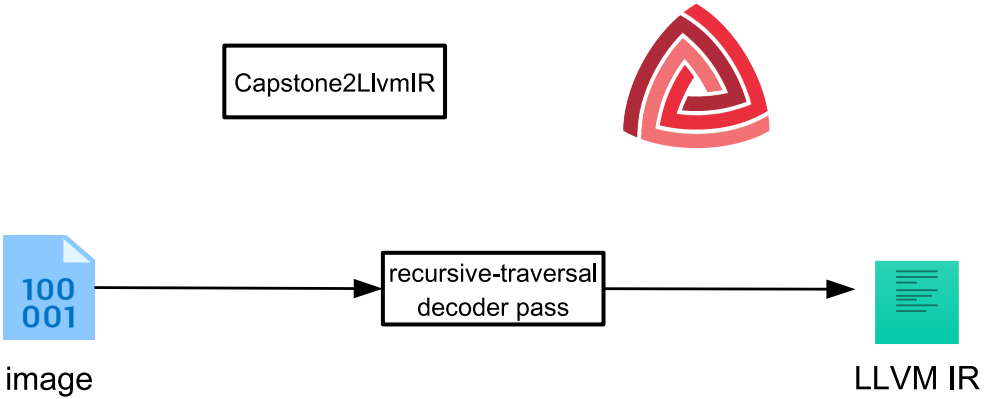


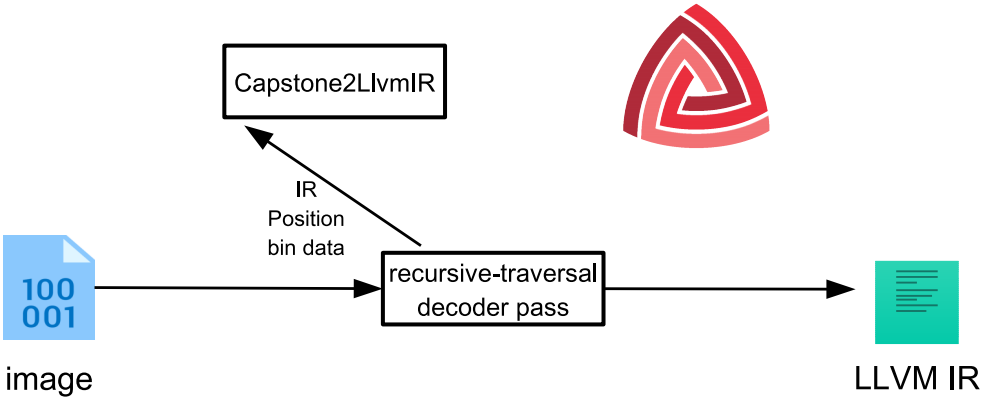
Core: decoder

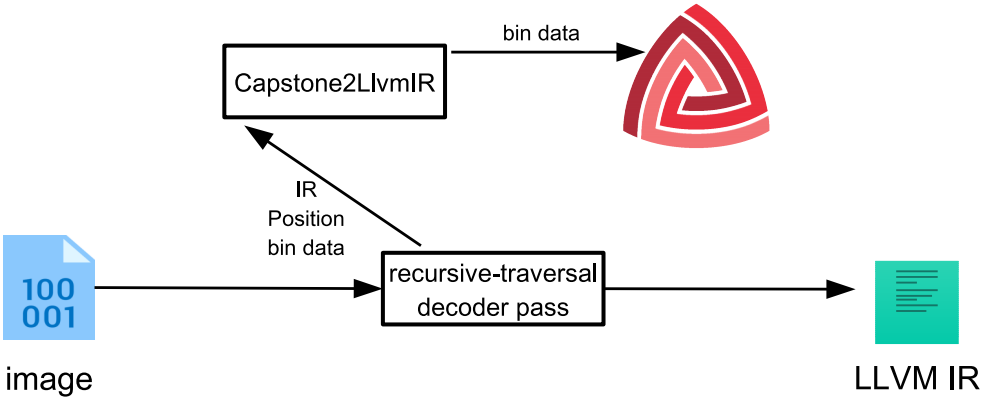


Core: decoder

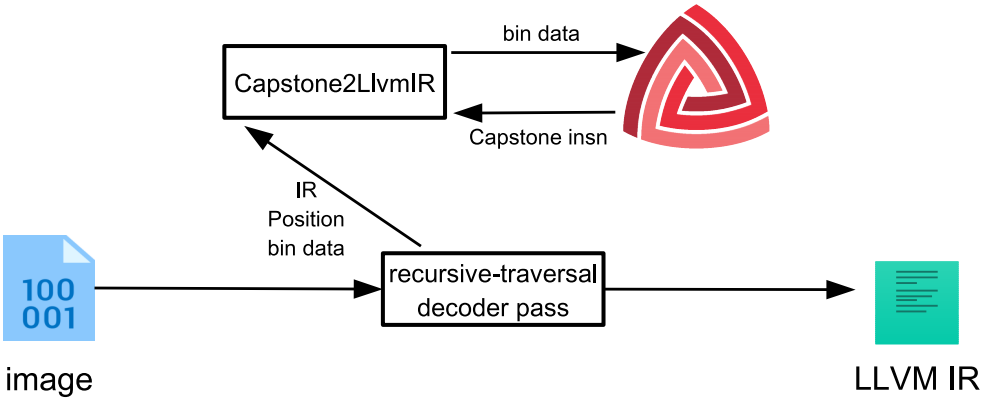


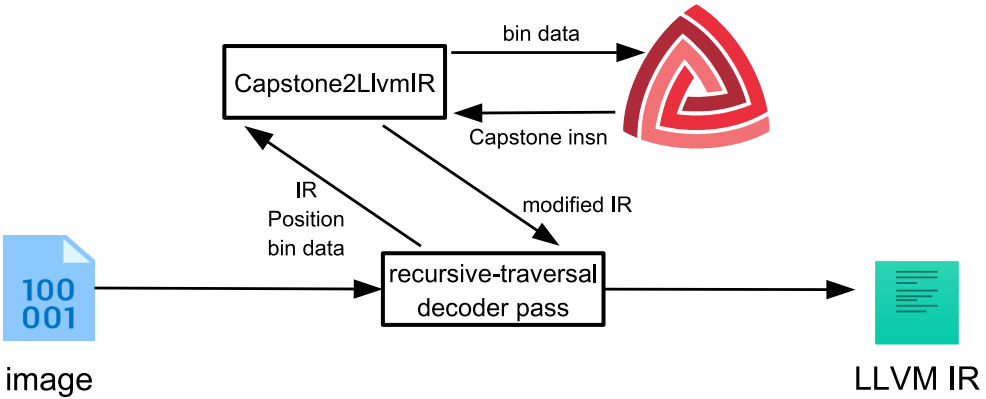

Capstone2LlvmIR
100
001
image
recursive-traversal
decoder pass
LLVM IR


Capstone2LlvmIR
IR
Position
bin data
100
001
recursive-traversal
decoder pass
image
LLVM IR


bin data
Capstone2LlvmIR
IR
Position
bin data
100
001
recursive-traversal
decoder pass
image
LLVM IR

Core: decoder




bin data
Capstone2LlvmIR
Capstone insn
IR
Position
bin data
modified IR
100
001
recursive-traversal
decoder pass
image
LLVM IR

Core: Capstone2LlvmIR

- Capstone insn $\rightarrow$ sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person $\approx$ 2-3 weeks

Core: Capstone2LlvmIR

- Capstone insn $\rightarrow$ sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person $\approx$ 2-3 weeks
- Architectures (core instruction sets):
  - ARM + Thumb extension – 32-bit
  - MIPS – 32/64-bit
  - PowerPC – 32/64-bit
  - x86 – 32/64-bit
- Capstone: 64-bit ARM, SPARC, SYSZ, XCore, m68k, m680x, TMS320C64x

Core: Capstone2LlvmIR

- Capstone insn $\rightarrow$ sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person $\approx$ 2-3 weeks
- Architectures (core instruction sets):
  - ARM + Thumb extension – 32-bit
  - MIPS – 32/64-bit
  - PowerPC – 32/64-bit
  - x86 – 32/64-bit
- Capstone: 64-bit ARM, SPARC, SYSZ, XCore, m68k, m680x, TMS320C64x
- Decompilation & advanced insns

- PMULHUW
  - Multiply Packed Unsigned Integers and Store High Result

```
if (OperandSize == 64) {
//PMULHUW instruction with 64-bit operands:
    Tmp0[0..31] = Dst[0..15] * Src[0..15];
    Tmp1[0..31] = Dst[16..31] * Src[16..31];
    Tmp2[0..31] = Dst[32..47] * Src[32..47];
    Tmp3[0..31] = Dst[48..63] * Src[48..63];
    Dst[0..15] = Tmp0[16..31];
    Dst[16..31] = Tmp1[16..31];
    Dst[32..47] = Tmp2[16..31];
    Dst[48..63] = Tmp3[16..31];
}
else {
//PMULHUW instruction with 128-bit operands:
    // Even longer ...
}
```

```
__asm_PMULHUW(mm1, mm2);
```

Core: Capstone2LlvmIR

- Capstone insn $\rightarrow$ sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person $\approx$ 2-3 weeks
- Architectures (core instruction sets):
  - ARM + Thumb extension – 32-bit
  - MIPS – 32/64-bit
  - PowerPC – 32/64-bit
  - x86 – 32/64-bit
- Capstone: 64-bit ARM, SPARC, SYSZ, XCore, m68k, m680x, TMS320C64x
- Decompilation & advanced insns

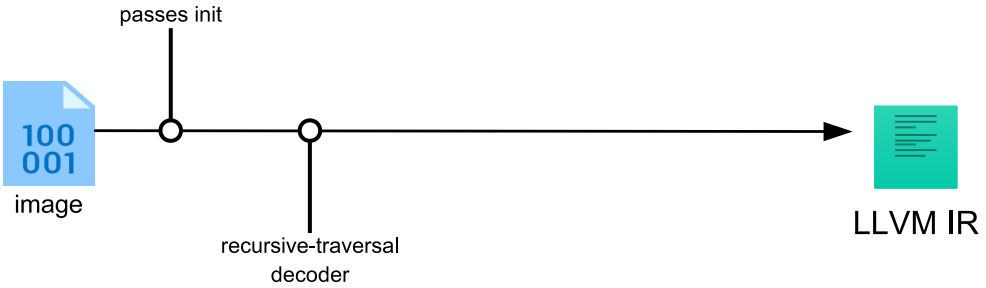
Core: Capstone2LlvmIR

- Capstone insn $\rightarrow$ sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person $\approx$ 2-3 weeks
- Architectures (core instruction sets):
  - ARM + Thumb extension – 32-bit
  - MIPS – 32/64-bit
  - PowerPC – 32/64-bit
  - x86 – 32/64-bit
- Capstone: 64-bit ARM, SPARC, SYSZ, XCore, m68k, m680x, TMS320C64x
- Decompilation & advanced insns
  - full semantics only for simple instructions

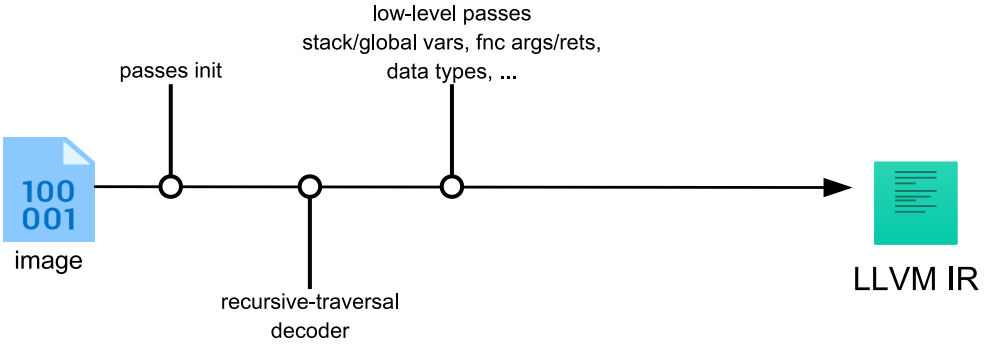
- Capstone insn $\rightarrow$ sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person $\approx$ 2-3 weeks
- Architectures (core instruction sets):
  - ARM + Thumb extension – 32-bit
  - MIPS – 32/64-bit
  - PowerPC – 32/64-bit
  - x86 – 32/64-bit
- Capstone: 64-bit ARM, SPARC, SYSZ, XCore, m68k, m680x, TMS320C64x
- Decompilation & advanced insns
  - full semantics only for simple instructions
- Implementation details, testing framework (Keystone Engine + LLVM emulator), keeping LLVM IR $\leftrightarrow$ ASM mapping, ...

- Capstone insn $\rightarrow$ sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person $\approx$ 2-3 weeks
- Architectures (core instruction sets):
  - ARM + Thumb extension – 32-bit
  - MIPS – 32/64-bit
  - PowerPC – 32/64-bit
  - x86 – 32/64-bit
- Capstone: 64-bit ARM, SPARC, SYSZ, XCore, m68k, m680x, TMS320C64x
- Decompilation & advanced insns
  - full semantics only for simple instructions
- Implementation details, testing framework (Keystone Engine + LLVM emulator), keeping LLVM IR $\leftrightarrow$ ASM mapping, ...

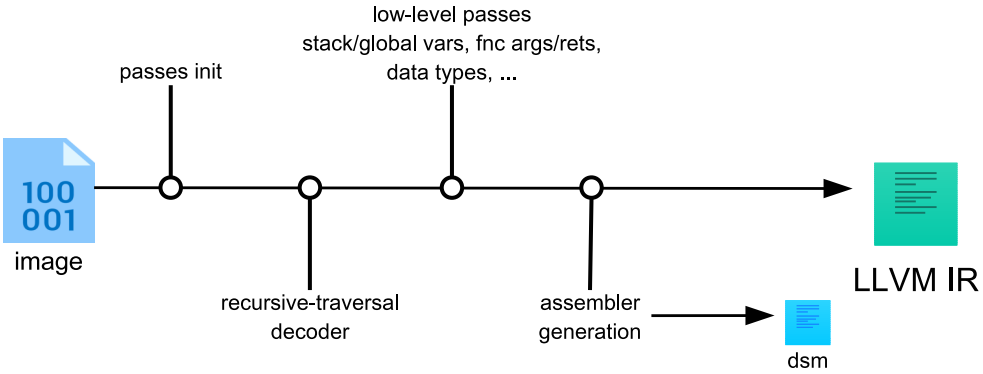
Core: low-level passes



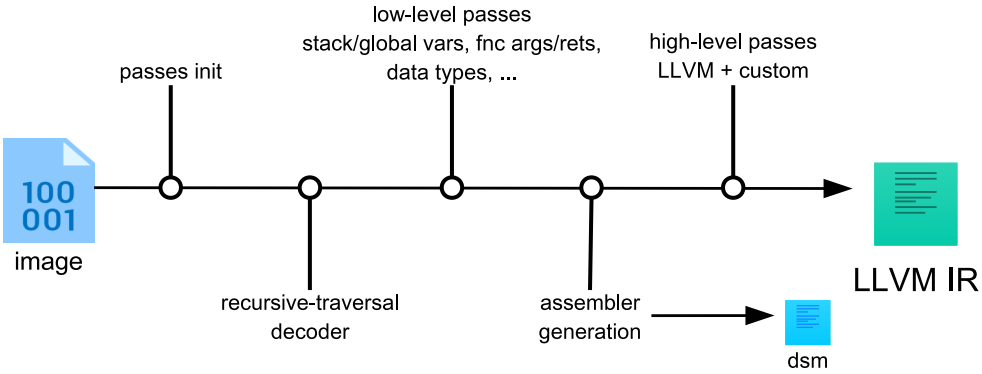
Core: low-level passes



Core: assembly generation



Core: high-level passes



Core repos

avast

git RetDec

- bin2llvmir library
- bin2llvmirtool

Core repos

avast

- git RetDec
  - bin2llvmir library
  - bin2llvmirtool
- git Capstone2LlvmIR
  - Capstone instruction to LLVM IR translation

Core repos

- **git** RetDec
  - bin2llvmir library
  - bin2llvmirtool
- **git** Capstone2LlvmIR
  - Capstone instruction to LLVM IR translation
- **git** Capstone-dumper
  - what does Capstone know about any instruction

Core repos

- git RetDec
  - bin2llvmir library
  - bin2llvmirtool
- git Capstone2LlvmIR
  - Capstone instruction to LLVM IR translation
- git Capstone-dumper
  - what does Capstone know about any instruction
- git Fnc-patterns
  - statically linked function pattern creation and detection

Core repos

- git RetDec
  - bin2llvmir library
  - bin2llvmirtool
- git Capstone2LlvmIR
  - Capstone instruction to LLVM IR translation
- git Capstone-dumper
  - what does Capstone know about any instruction
- git Fnc-patterns
  - statically linked function pattern creation and detection
- git Yaramod
  - YARA to AST parsing & C++ API to build new YARA rulesets

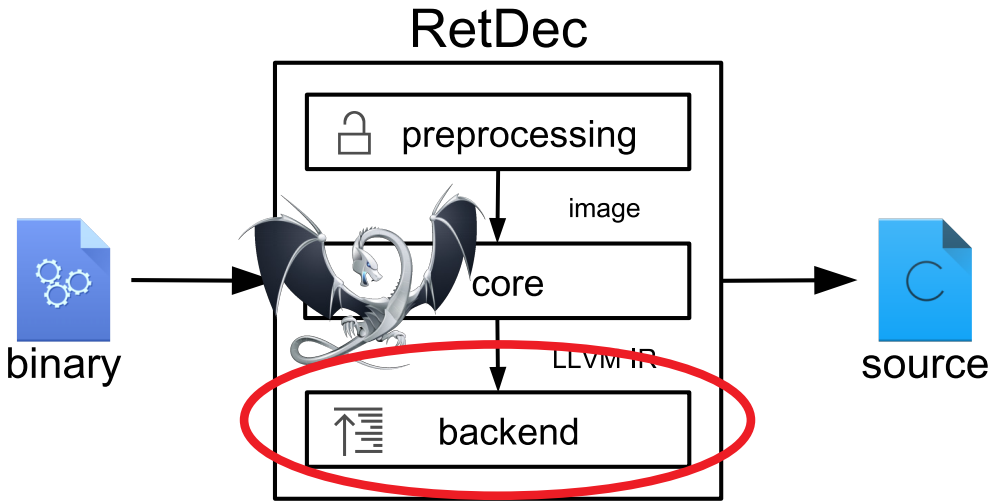
Core repos

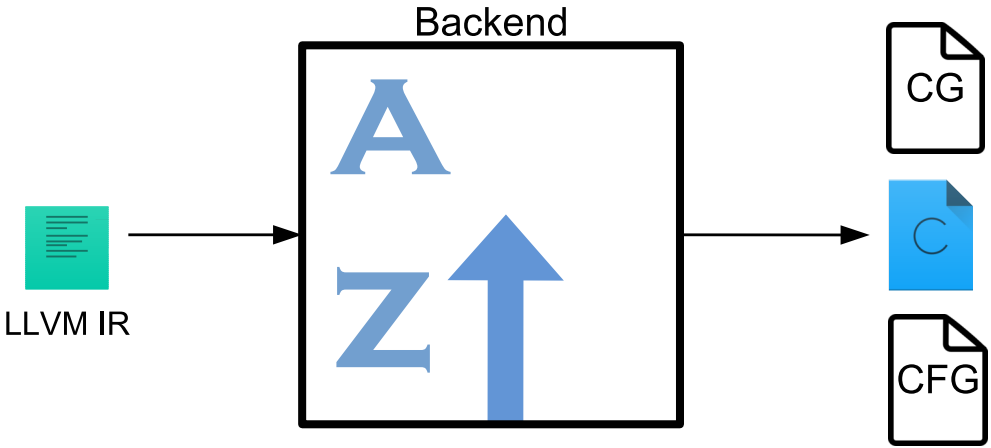
- git RetDec
  - bin2llvmir library
  - bin2llvmirtool
- git Capstone2LlvmIR
  - Capstone instruction to LLVM IR translation
- git Capstone-dumper
  - what does Capstone know about any instruction
- git Fnc-patterns
  - statically linked function pattern creation and detection
- git Yaramod
  - YARA to AST parsing & C++ API to build new YARA rulesets
- git Ctypes
  - extraction and presentation of C function data types

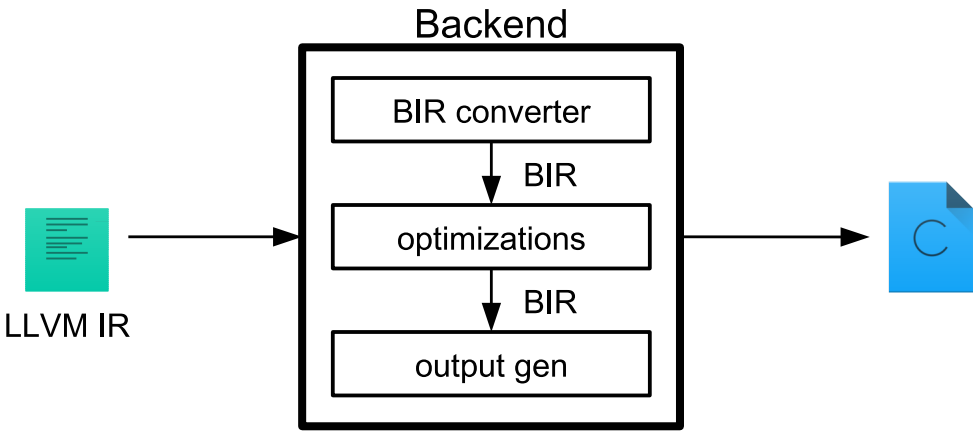
Core repos

avast

- git RetDec
  - bin2llvmir library
  - bin2llvmirtool
- git Capstone2LlvmIR
  - Capstone instruction to LLVM IR translation
- git Capstone-dumper
  - what does Capstone know about any instruction
- git Fnc-patterns
  - statically linked function pattern creation and detection
- git Yaramod
  - YARA to AST parsing & C++ API to build new YARA rulesets
- git Ctypes
  - extraction and presentation of C function data types
- git Demangler
  - gcc/Clang, Microsoft Visual C++, and Borland C++

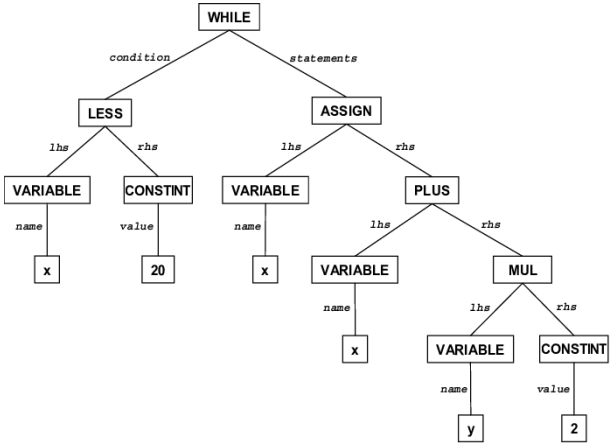

RetDec
preprocessing
image
core
LLVM IR
backend
binary
source


Backend
LLVM IR
CG
CFG


Backend
BIR converter
BIR
optimizations
BIR
output gen
LLVM IR

Backend: BIR is an AST

- BIR = Backend IR
- AST = Abstract syntax tree
- `while (x < 20){ x = x + (y * 2); }`

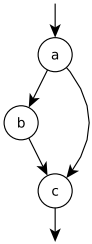


Backend: code structuring

- LLVM IR: only (un)conditional branches & switches
- identify high-level control-flow patterns
- restructure BIR: if-else, for-loop, while-loop, switch, break, continue

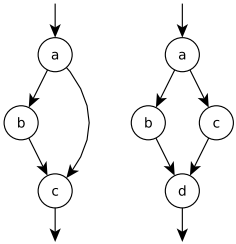
Backend: code structuring

- LLVM IR: only (un)conditional branches & switches
- identify high-level control-flow patterns
- restructure BIR: if-else, for-loop, while-loop, switch, break, continue

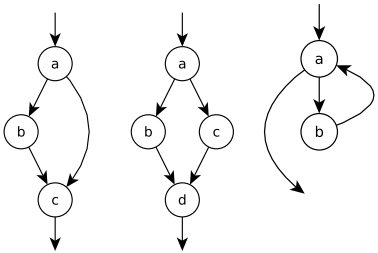


Backend: code structuring

- LLVM IR: only (un)conditional branches & switches
- identify high-level control-flow patterns
- restructure BIR: if-else, for-loop, while-loop, switch, break, continue

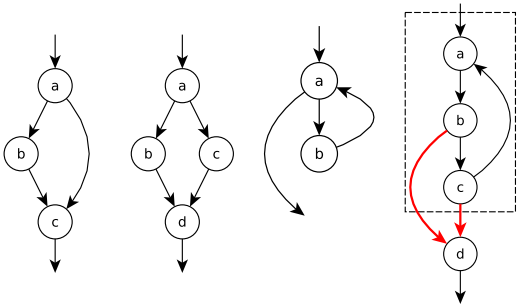


- LLVM IR: only (un)conditional branches & switches
- identify high-level control-flow patterns
- restructure BIR: if-else, for-loop, while-loop, switch, break, continue



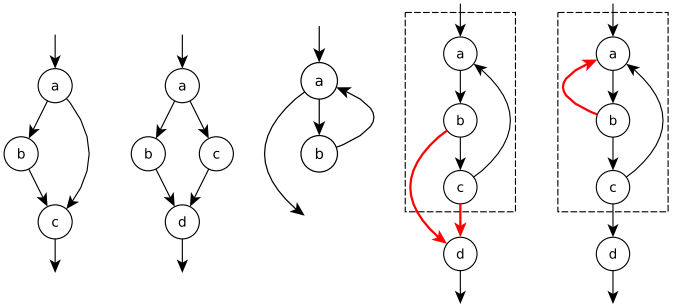
Backend: code structuring

- LLVM IR: only (un)conditional branches & switches
- identify high-level control-flow patterns
- restructure BIR: if-else, for-loop, while-loop, switch, break, continue



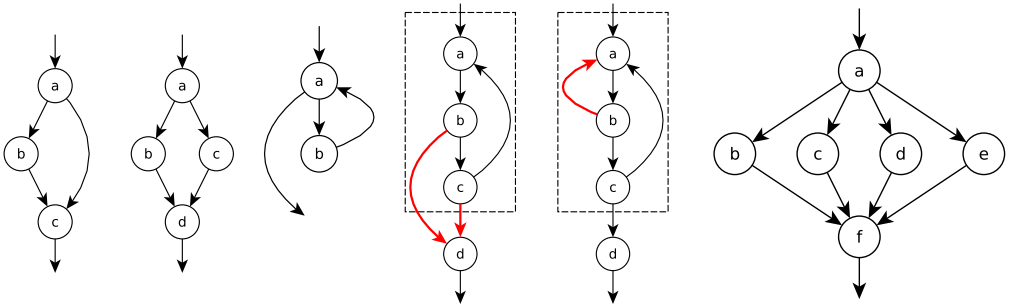
Backend: code structuring

- LLVM IR: only (un)conditional branches & switches
- identify high-level control-flow patterns
- restructure BIR: if-else, for-loop, while-loop, switch, break, continue



Backend: code structuring

- LLVM IR: only (un)conditional branches & switches
- identify high-level control-flow patterns
- restructure BIR: if-else, for-loop, while-loop, switch, break, continue



- copy propagation
  - reducing the number of variables

Backend: optimizations

- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - `a + -1 - -4` ⇒ `a + 3`

Backend: optimizations

- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - `a + -1 - -4` ⇒ `a + 3`
- negation optimization
  - `if (!(a == b))` ⇒ `if (a != b)`

Backend: optimizations

- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - `a + -1 - -4` ⇒ `a + 3`
- negation optimization
  - `if (!(a == b))` ⇒ `if (a != b)`
- pointer arithmetic
  - `*(a + 4)` ⇒ `a[4]`

Backend: optimizations

- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - `a + -1 - -4` ⇒ `a + 3`
- negation optimization
  - `if (!(a == b))` ⇒ `if (a != b)`
- pointer arithmetic
  - `*(a + 4)` ⇒ `a[4]`
- conversion of `while (true){ ... if (cond) break; ... }`
  - `for (cond){ ... }`
  - `while (cond){ ... }`

Backend: optimizations

- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - `a + -1 - -4` ⇒ `a + 3`
- negation optimization
  - `if (!(a == b))` ⇒ `if (a != b)`
- pointer arithmetic
  - `*(a + 4)` ⇒ `a[4]`
- conversion of `while (true){ ... if (cond) break; ... }`
  - `for (cond){ ... }`
  - `while (cond){ ... }`
- conversion of `if`/`else`-`if`/`else` chains to switch

Backend: optimizations

- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - `a + -1 - -4` ⇒ `a + 3`
- negation optimization
  - `if (!(a == b))` ⇒ `if (a != b)`
- pointer arithmetic
  - `*(a + 4)` ⇒ `a[4]`
- conversion of `while (true){ ... if (cond) break; ... }`
  - `for (cond){ ... }`
  - `while (cond){ ... }`
- conversion of `if`/`else`-`if`/`else` chains to switch
- ...

Backend: code generation

- variable name assignment
  - induction variables: `for (i = 0; i < 10; ++i)`
  - function arguments: `a1, a2, a3, ...`
  - general context names: `return result;`
  - stdlib context names: `int len = strlen();`

Backend: code generation

- variable name assignment
  - induction variables: `for (i = 0; i < 10; ++i)`
  - function arguments: `a1, a2, a3, ...`
  - general context names: `return result;`
  - stdlib context names: `int len = strlen();`
- stdlib context literals
  - `var_ffff7dc6 = socket(2, 3, 255)`

Backend: code generation

- variable name assignment
  - induction variables: `for (i = 0; i < 10; ++i)`
  - function arguments: `a1, a2, a3, ...`
  - general context names: `return result;`
  - stdlib context names: `int len = strlen();`
- stdlib context literals
  - `var_ffff7dc6 = socket(2, 3, 255)`
  - `sock_id = socket(PF_INET, SOCK_RAW, IPPROTO_RAW)`

Backend: code generation

- variable name assignment
  - induction variables: `for (i = 0; i < 10; ++i)`
  - function arguments: `a1, a2, a3, ...`
  - general context names: `return result;`
  - stdlib context names: `int len = strlen();`
- stdlib context literals
  - `var_ffff7dc6 = socket(2, 3, 255)`
  - `sock_id = socket(PF_INET, SOCK_RAW, IPPROTO_RAW)`
  - `flock(sock_id, 7)`
  - `flock(sock_id, LOCK_SH | LOCK_EX | LOCK_NB)`

Backend: code generation

- variable name assignment
  - induction variables: `for (i = 0; i < 10; ++i)`
  - function arguments: `a1, a2, a3, ...`
  - general context names: `return result;`
  - stdlib context names: `int len = strlen();`
- stdlib context literals
  - `var_ffff7dc6 = socket(2, 3, 255)`
  - `sock_id = socket(PF_INET, SOCK_RAW, IPPROTO_RAW)`
  - `flock(sock_id, 7)`
  - `flock(sock_id, LOCK_SH | LOCK_EX | LOCK_NB)`
- output generation
  - C
  - CFG = Control-Flow Graph
  - Call Graph

Backend repos

git RetDec

- llvmir2hll library
- llvmir2hlltool

- Online decompilation service
  https://retdec.com/decompilation/

- Online decompilation service
  https://retdec.com/decompilation/
- REST API
  https://retdec.com/api/

How to use RetDec

- Online decompilation service
  https://retdec.com/decompilation/
- REST API
  https://retdec.com/api/
- Build it yourself
  - CMake, gcc/Clang, Visual Studio 2015 Update 2
  - Perl, GNU Bison, Flex, GNU Tar, scp, GNU bash, UPX, dot
  - Recursively clone the main RetDec repository

```
mkdir build && cd build
cmake ..
make && make install
```

How to use RetDec

- Online decompilation service
  https://retdec.com/decompilation/
- REST API
  https://retdec.com/api/
- Build it yourself
  - CMake, gcc/Clang, Visual Studio 2015 Update 2
  - Perl, GNU Bison, Flex, GNU Tar, scp, GNU bash, UPX, dot
  - git Recursively clone the main RetDec repository
  - `mkdir build && cd build`
  - `cmake ..`
  - `make && make install`
- Run it yourself
  - `decompile.sh binary.exe`

How to use RetDec

- Online decompilation service
  https://retdec.com/decompilation/
- REST API
  https://retdec.com/api/
- Build it yourself
  - CMake, gcc/Clang, Visual Studio 2015 Update 2
  - Perl, GNU Bison, Flex, GNU Tar, scp, GNU bash, UPX, dot
  - git Recursively clone the main RetDec repository
  - `mkdir build && cd build`
  - `cmake ..`
  - `make && make install`
- Run it yourself
  - `decompile.sh binary.exe`
- Get RetDec IDA plugin

What is RetDec IDA plugin

```
.text:004015BB          push    ebp
.text:004015BC          mov     ebp, esp
.text:004015BE          and     esp, 0FFFFFFF0h
.text:004015C1          sub     esp, 20h
.text:004015C4          call    ___main
.text:004015C9          mov     [esp+20h+var_4], 0
.text:004015D1          mov     [esp+20h+var_8], 0
.text:004015D9          mov     [esp+20h+var_C], 0
.text:004015E1          lea     eax, [esp+20h+var_C]
.text:004015E5          mov     [esp+20h+var_18], eax
.text:004015E9          lea     eax, [esp+20h+var_8]
.text:004015ED          mov     [esp+20h+var_1C], eax
.text:004015F1          mov     [esp+20h+Format], offset Format
.text:004015F8          call    _scanf
.text:004015FD          mov     edx, [esp+20h+var_C]
.text:00401601          mov     eax, [esp+20h+var_8]
.text:00401605          mov     [esp+20h+var_1C], edx
.text:00401609          mov     [esp+20h+Format], eax
.text:0040160C          call    _ack
.text:00401611          mov     [esp+20h+var_4], eax
.text:00401615          mov     edx, [esp+20h+var_C]
.text:00401619          mov     eax, [esp+20h+var_8]
.text:0040161D          mov     ecx, [esp+20h+var_4]
.text:00401621          mov     [esp+20h+var_14], ecx
.text:00401625          mov     [esp+20h+var_18], edx
.text:00401629          mov     [esp+20h+var_1C], eax
.text:0040162D          mov     [esp+20h+Format], offset aAckermanDDD
.text:00401634          call    _printf
.text:00401639          mov     eax, [esp+20h+var_4]
.text:0040163D          leave
.text:0040163E          retn
```

```
//
// This file was generated by the Retargetable Decompiler
// Website: https://retdec.com
// Copyright (c) 2017 Retargetable Decompiler <info@retdec.com>
//

#include <stdint.h>
#include <stdio.h>

// ------------------------ Functions -------------------------
int32_t _ack(int32_t a1, int32_t a2) {
    if (a1 == 0) {
        return a2 + 1;
    }
    int32_t result;
    if (a2 == 0) {
        result = _ack(a1 - 1, 1);
    } else {
        result = _ack(a1 - 1, _ack(a1, a2 - 1));
    }
    return result;
}
int main(int argc, char ** argv) {
    ___main();
    int32_t v1 = 0;
    int32_t v2 = 0;
    scanf("%d %d", &v1, &v2);
    int32_t result = _ack(v1, v2);
    printf("ackerman( %d , %d ) = %d\n", v1, v2, result);
    return result;
}
```

How does RetDec IDA plugin work

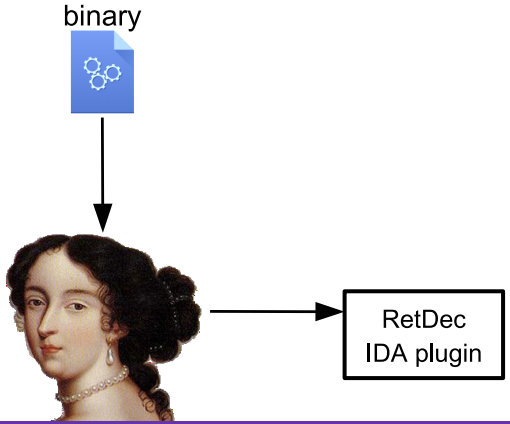
- Goals
  - look & feel native
  - same object names as IDA
  - interactive

How does RetDec IDA plugin work

- Goals
  - look & feel native
  - same object names as IDA
  - interactive

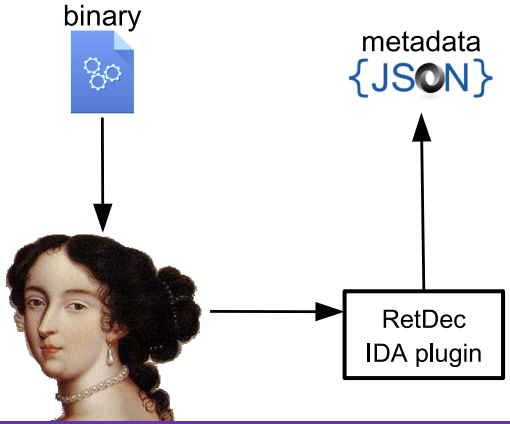
How does RetDec IDA plugin work

- Goals
  - look & feel native
  - same object names as IDA
  - interactive



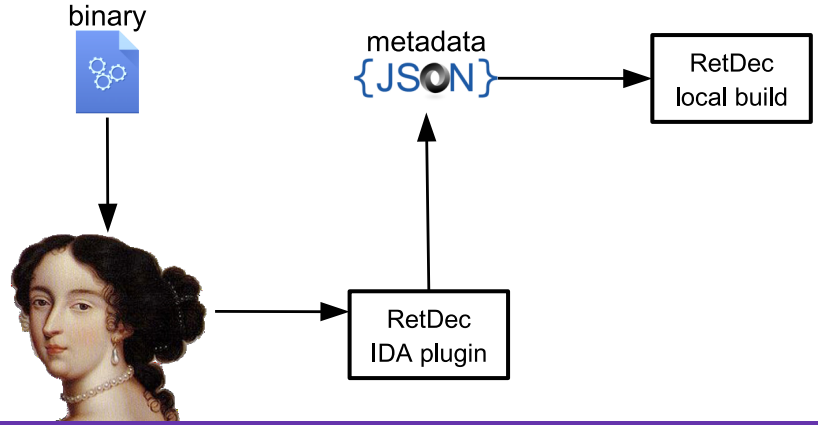
How does RetDec IDA plugin work

- Goals
  - look & feel native
  - same object names as IDA
  - interactive



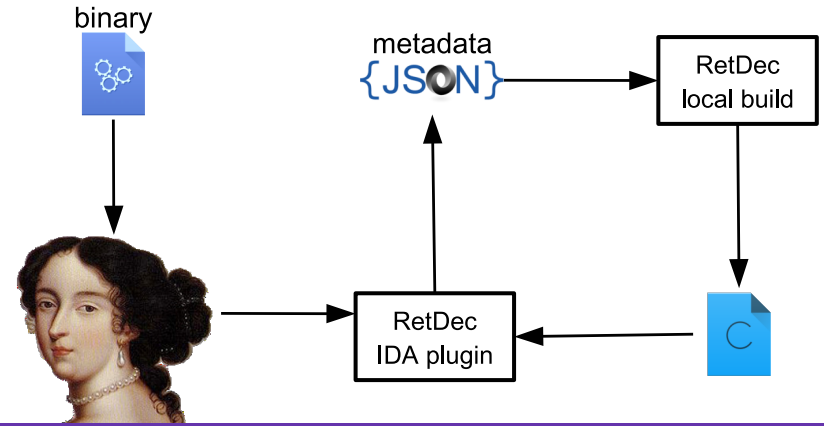
How does RetDec IDA plugin work

- Goals
  - look & feel native
  - same object names as IDA
  - interactive



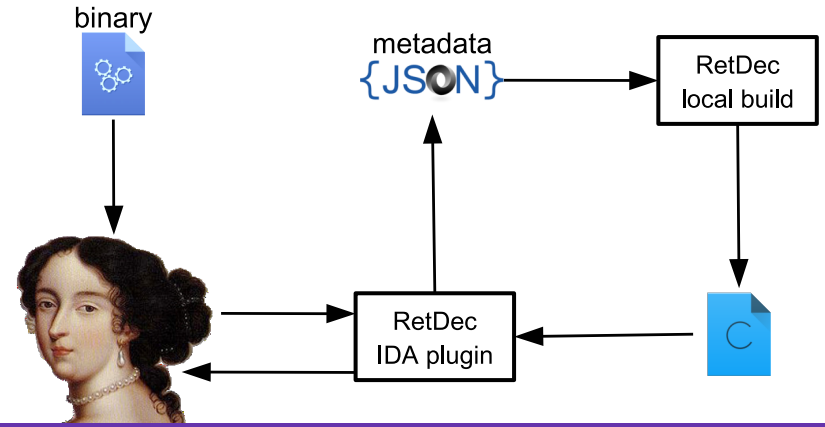
How does RetDec IDA plugin work

- Goals
  - look & feel native
  - same object names as IDA
  - interactive



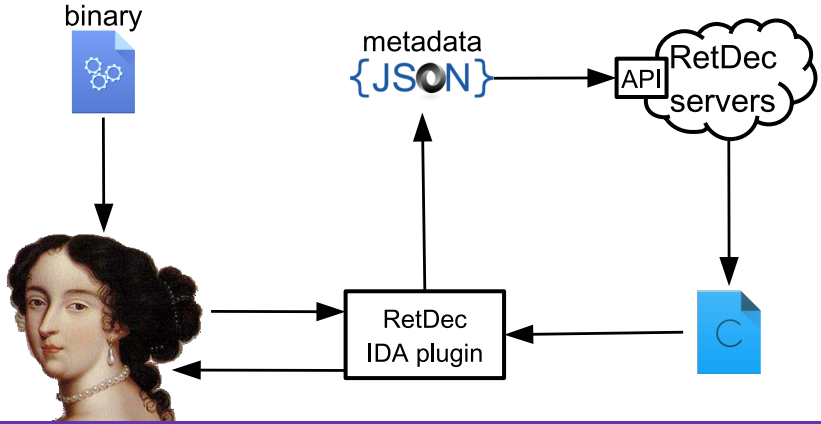
How does RetDec IDA plugin work

- Goals
  - look & feel native
  - same object names as IDA
  - interactive



How does RetDec IDA plugin work

- Goals
  - look & feel native
  - same object names as IDA
  - interactive



RetDec IDA plugin is interactive

```
// From module:   /home/peter/decompiler/decompiler
// Address range: 0x804851c - 0x8048576
// Line range:    4 - 11
int32_t ack(int32_t m, int32_t n) {
    // 0x8
    if (m
        //
        //
        //
        re
    }
    // 0x8
    int32_t result; // 0x8048576_11
    if (n == 0) {
        // 0x8048536
        result = ack(m - 1, 1);
        // branch -> 0x8048575
    } else {
        // 0x804854e
        result = ack(m - 1, ack(m, n - 1));
        // branch -> 0x8048575
    }
    // 0x8048575
    return result;
```

Jump to ASM	A
Rename function	N
Change type declaration	Y
Open xrefs window	X
Open calls window	C
Edit func comment	/
Move backward	Esc
Move forward	Ctrl+Enter

```
// ---------------------- Global Variables ------------

int32_t __CTOR_LIST__ = -1; // 0x80497f4

// ------------------------- Functions ----------------

// Address range: 0x8048680 - 0x80486a9
int32_t __do_global_ctors_aux(void) {
    // 0x8048680
    if (__CTOR_LIST__ == -1) {
        // 0x80486a4
        return -1;
    }
    int32_t v1 = 0x8
    unknown_ffffffff
    // branch -> 0x8048698
    while (*(int32_t *)(v1 - 4) != -1) {
        // 0x8048698
        v1 -= 4;
        unknown_ffffffff();
        // continue -> 0x8048698
    }
    // 0x80486a4
    return -1;
```

Jump to ASM	A
Rename global variable	N
Edit func comment	/
Move backward	Esc
Move forward	Ctrl+Enter

How was RetDec used so far

avast

- retdec.com launched on 2015-02-05

How was RetDec used so far

retdec.com launched on 2015-02-05

12,000 registered users

How was RetDec used so far

- retdec.com launched on 2015-02-05
- 12,000 registered users
- 423,000 decompilations
  - 350,000 Web
  - 73,000 API
- 410 decompilations daily

Real example #1: Vawtrak (x86)

avast

DSM

```
.text:1000DD3E ; HKEY __usercall getNextRegValueOfFCkey@<eax>(CHAR *outSubKey@<edx>, char **filename, DWORD *pcbData)
.text:1000DD3E getNextRegValueOfFCkey proc near
.text:1000DD3E                 push    ebp
.text:1000DD3F                 mov     ebp, esp
.text:1000DD41                 sub     esp, 14h
```

Hex-Rays

```
HKEY __usercall getNextRegValueOfFCkey@<eax>(CHAR *outSubKey@<edx>, char **filename, DWORD *pcbData)
{
  v3 = outSubKey;
  v11 = 0;
  lstrcpyA(outSubKey, "SOFTWARE\\");
  lstrcatA(v3, byte_10032D70);
```

RetDec

```
struct HKEY__ * getNextRegValueOfFCkey(char * outSubKey, char ** filename, int32_t * pcbData) {
    int32_t lpValueName = (int32_t)outSubKey; // esi
    lstrcpyA(outSubKey, "SOFTWARE\\");
    lstrcatA(outSubKey, byte_10032D70);
```

Real example #2: Vawtrak (x86)

Hex-Rays

```
int unregisterAutorun3()
{
  CHAR pszPath; // [esp+0h] [ebp-118h]@1
  int v2; // [esp+104h] [ebp-14h]@3
  int v3; // [esp+108h] [ebp-10h]@3
  CHAR *v4; // [esp+10Ch] [ebp-Ch]@3
  int v5; // [esp+114h] [ebp-4h]@1

  v5 = getApplicationDataFullPath(26, 0, 0, &pszPath);
  if ( !v5 )
    return 0;
  v2 = -2147483647;
  v3 = 0;
  v4 = &pszPath;
  regOpenKeyAndCallProc(
    HKEY_CURRENT_USER,
    "Software\\Microsoft\\Windows\\CurrentVersion\\Run",
    (int)sub_10018797,
    (int)&v2);
  v2 = -2147483646;
  regOpenKeyAndCallProc(
    HKEY_LOCAL_MACHINE,
    "Software\\Microsoft\\Windows\\CurrentVersion\\Run",
    (int)sub_10018797,
    (int)&v2);
  return v3;
}
```

RetDec

```
int32_t unregisterAutorun3(void) {
    int32_t v1 = 0; // bp-284
    if (getApplicationDataFullPath(26, NULL, NULL, (char *)&v1) != 0) {
        int32_t v2 = -0x7fffffff; // bp-24
        int32_t v3 = &v2; // 0x10018849
        regOpenKeyAndCallProc((struct HKEY__ *)-0x7fffffff,
            "Software\\Microsoft\\Windows\\CurrentVersion\\Run",
            0x10018797,
            v3);
        v2 = -0x7ffffffe;
        regOpenKeyAndCallProc((struct HKEY__ *)-0x7ffffffe,
            "Software\\Microsoft\\Windows\\CurrentVersion\\Run",
            0x10018797,
            v3);
    }
    // 0x10018889
    return 0;
}
```

Real example #3: CryproWall (x86)

avast

DSM

```
dd offset loc_4096B8; jump table
; indirect table for switch
byte_4096FC db      0,     5,     1,     1
            db      5,     2,     5,     5
            db      3,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     5
            db      5,     5,     5,     4
```

Hex-Rays

```
v4 = 1;
switch ( GetLastError() )
{
  case 0u:
    v4 = 0;
    break;
  case 2u:
  case 3u:
    v4 = 6;
    break;
  case 5u:
    v4 = 4;
    break;
  case 8u:
    v4 = 3;
    break;
  case 0x57u:
    v4 = 2;
    break;
  default:
    break;
}
```

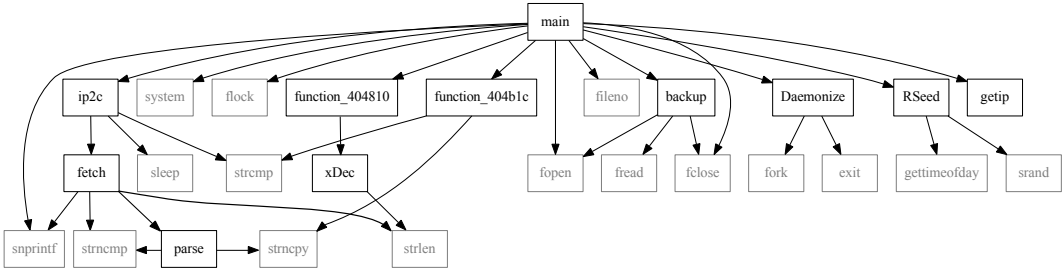
RetDec

```
int32_t result = 1; // esi
switch (GetLastError()) {
    case 0: {
        result = 0;
        break;
    }
    case 2: {
        result = 6;
        break;
    }
    case 3: {
        result = 6;
        break:
    }
    case 5: {
        result = 4;
        break;
    }
    case 8: {
        result = 3;
        break;
    }
    case 87: {
        result = 2;
        break;
    }
}
```

Real example #4: Psyb0t (mips) RetDec

```
// Address range: 0x419378 - 0x41946f
int32_t uptime(void) {
    struct _IO_FILE * file = fopen("/proc/uptime", "r"); // 0x4193bc
    int32_t str = 0; // bp-56
    fgets((char *)&str, 32, file);
    fclose(file);
    int32_t result = 0; // bp-64
    sscanf((char *)&str, "%d", &result);
    return result;
}
```

Real example #5: Psyb0t (mips) RetDec



Should you throw away your Hex-Rays?

NO!

- IDA and Hex-Rays are great
  - output quality
  - interactive
  - seamlessly integrated
  - mature
  - many plugins
  - official support

NO!

- IDA and Hex-Rays are great
  - output quality
  - interactive
  - seamlessly integrated
  - mature
  - many plugins
  - official support
- IDA and Hex-Rays have flaws
  - not free
  - proprietary
  - big monolithic GUI app

RetDec is handy because . . .

- Obvious reasons
  - ฿ it is free
  - ✚ MIPS architecture
  - ⚖ MIT license
  - 🔧 you can play with the sources

RetDec is handy because . . .

- Obvious reasons
  - ฿ it is free
  - ✚ MIPS architecture
  - MIT license
  - 🔧 you can play with the sources
- Not so obvious reasons
  - 💎 LLVM is awesome

RetDec is handy because . . .

- Obvious reasons
  - it is free
  - MIPS architecture
  - MIT license
  - you can play with the sources
- Not so obvious reasons
  - LLVM is awesome
  - different basic designs: interactive GUI vs. pipeline

RetDec is handy because . . .

- Obvious reasons
  - it is free
  - MIPS architecture
  - MIT license
  - you can play with the sources
- Not so obvious reasons
  - LLVM is awesome
  - different basic designs: interactive GUI vs. pipeline
  - LLVM is OP (don't worry, it won't be nerfed)

RetDec is not only decompiler

- git RetDec – the decompiler
- git RetDec IDA plugin – Hex-Rays impersonation

RetDec is not only decompiler

- git RetDec – the decompiler
- git RetDec IDA plugin – Hex-Rays impersonation
- git Fileformat – generic OFF parsing and analysis
- git Capstone2LlvmIR – binary to LLVM translation
- git Fnc-patterns – statically linked code detection in YARA (IDA F.L.I.R.T.)
- git Yaramod – hack YARA rules in C++
- git Yaracpp – YARA C++ wrapper
- git Ctypes – info on function types

- Release the sources on Github shortly after the conference

What's next

- Release the sources on Github shortly after the conference
- Throw a release party

What's next

- Release the sources on Github shortly after the conference
- Throw a release party
- Solve some inevitable "hey guys, I'm unable to build your repo"

What's next

- Release the sources on Github shortly after the conference
- Throw a release party
- Solve some inevitable "hey guys, I'm unable to build your repo"
- Write more technical documentation on how it all works

What's next

- Release the sources on Github shortly after the conference
- Throw a release party
- Solve some inevitable "hey guys, I'm unable to build your repo"
- Write more technical documentation on how it all works
- Present it somewhere (maybe LLVM dev meeting)

- Release the sources on Github shortly after the conference
- Throw a release party
- Solve some inevitable "hey guys, I'm unable to build your repo"
- Write more technical documentation on how it all works
- Present it somewhere (maybe LLVM dev meeting)
- Continue improving the implementation
  - make it more portable: Bash ⇒ Python, ...
  - 64-bit architectures supports
  - replace some libraries & modules

- Release the sources on Github shortly after the conference
- Throw a release party
- Solve some inevitable "hey guys, I'm unable to build your repo"
- Write more technical documentation on how it all works
- Present it somewhere (maybe LLVM dev meeting)
- Continue improving the implementation
  - make it more portable: Bash ⇒ Python, . . .
  - 64-bit architectures supports
  - replace some libraries & modules
- We will see. . .

Thanks!

- Contacts
  - https://retdec.com/
  - https://github.com/avast-tl
  - https://twitter.com/retdec
  - https://retdec.com/rss/
  - info@retdec.com