RetDec: An Open-Source Machine-Code Decompiler

Jakub Křouštek
Peter Matula
Petr Zemek

Threat Labs
Jakub Křouštek

- founder of RetDec
- Threat Labs lead @Avast (previously @AVG)
- reverse engineer, malware hunter, security researcher
- @JakubKroustek
- jakub.kroustek@avast.com
Jakub Křouštěk
- 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
push ebp
mov ebp, esp
and esp, 0FFFFFFF0h
sub esp, 20h
call ___main
mov [esp+20h+var_4], 0
mov [esp+20h+var_8], 0
mov [esp+20h+var_C], 0
lea eax, [esp+20h+var_C]
lea eax, [esp+20h+var_18], eax
lea eax, [esp+20h+var_8]
lea eax, [esp+20h+var_1C], eax
mov [esp+20h+Format], offset Format
call __scanf
mov edx, [esp+20h+var_C]
mov eax, [esp+20h+var_0]
mov [esp+20h+var_1C], edx
mov [esp+20h+Format], eax
call __ack
mov [esp+20h+var_4], eax
mov edx, [esp+20h+var_C]
mov eax, [esp+20h+var_0]
mov ecx, [esp+20h+var_4]
mov [esp+20h+var_14], ecx
mov [esp+20h+var_18], edx
mov [esp+20h+var_1C], eax
mov [esp+20h+Format], offset aAckermanDDD
call __printf
mov eax, [esp+20h+var_4]
leave
ret
Quiz Time

```
STMFD SP!, {R11,LR}
ADD R11, SP, #4
SUB SP, SP, #0x14
STR R0, [R11,#var_14]
STR R1, [R11,#var_10]
BL __gccmain
MOV R3, #0
STR R3, [R11,#var_8]
MOV R3, #0
STR R3, [R11,#var_C]
MOV R3, #0
STR R3, [R11,#var_10]
SUB R2, R11, #-var_C
SUB R3, R11, #-var_10
LDR R0, =aDD
MOV R1, R2
MOV R2, R3
BL scanf
LDR R2, [R11,#var_C]
LDR R3, [R11,#var_10]
MOV R0, R2
MOV R1, R3
BL ack
MOV R3, R0
STR R3, [R11,#var_8]
LDR R2, [R11,#var_C]
LDR R3, [R11,#var_10]
LDR R0, =aAckermanDDD
MOV R1, R2
MOV R2, R3
LDR R3, [R11,#var_8]
BL printf
```
addiu $sp, -0x20
sw $ra, 0x20+var_4($sp)
sw $fp, 0x20+var_8($sp)
move $fp, $sp
sw $a0, 0x20+var_10($fp)
sw $a1, 0x20+var_C($fp)
sw $zero, 0x20+var_20($fp)
sw $zero, 0x20+var_1C($fp)
sw $zero, 0x20+var_18($fp)
addiu $v1, $fp, 0x20+var_1C
addiu $a2, $fp, 0x20+var_18
lui $v0, 0x891
addiu $a0, $v0, (aDD - 0x8910000)
move $a1, $v1
jal scanf
nop
lw $v0, 0x20+var_1C($fp)
lw $v1, 0x20+var_18($fp)
move $a0, $v0
move $a1, $v1
jal ack
nop
sw $v0, 0x20+var_20($fp)
lw $v1, 0x20+var_1C($fp)
lw $a2, 0x20+var_18($fp)
lui $v0, 0x891
addiu $a0, $v0, (aAckermanDDD - 0x8910000)
move $a1, $v1
lw $a3, 0x20+var_20($fp)
jal printf
```assembly
.stext:1000056C    stwu    r1, back_chain(r1)
.stext:10000570   mflr    r0
.stext:10000574   stw     r0, 0x30+sender_lr(r1)
.stext:10000578   stw     r31, 0x30+var_4(r1)
.stext:1000057C   mr      r31, r1
.stext:10000580   stw     r3, 0x18(r31)
.stext:10000584   stw     r4, 0x1C(r31)
.stext:10000588   li      r0, 0
.stext:1000058C   stw     r0, 0(r31)
.stext:10000590   li      r0, 0
.stext:10000594   stw     r0, 0xC(r31)
.stext:10000598   li      r0, 0
.stext:1000059C   stw     r0, 0x10(r31)
.stext:100005A0   lis     r0, 0x1000
.stext:100005A4   addic   r11, r0, 0x82C # 0x1000682C
.stext:100005A8   addi    r9, r31, 0xC
.stext:100005AC   addi    r0, r31, 0x10
.stext:100005B0   mr      r3, r11
.stext:100005B4   mr      r4, r9
.stext:100005B8   mr      r5, r0
.stext:100005BC   crcr    4*cr1+eq
.stext:100005C0   bl      __isoc99_scanf
.stext:100005C4   lwz     r9, 0xC(r31)
.stext:100005C8   lwz     r0, 0x10(r31)
.stext:100005CC   mr      r3, r9
.stext:100005D0   mr      r4, r0
.stext:100005D4   bl      aCk
.stext:100005D8   stw     r3, 0(r31)
.stext:100005DC   lis     r0, 0x1000
.stext:100005E0   addic   r11, r0, 0x834 # 0x10006834
.stext:100005E4   lwz     r9, 0xC(r31)
.stext:100005E8   lwz     r0, 0x10(r31)
```
Disassembling vs. Decompilation

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;
}
```
Decompilation? What is it?
Decomposition? What good is it?

- Binary analysis
  - reverse engineering
  - malware analysis
  - vulnerability detection
  - verification
  - binary comparison
  - ...
Decomposition? What good is it?

.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

- 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

- 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

- 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
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)
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
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
RetDec? What does it do

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

**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
- ...

Botconf 2017
RetDec? What does it do

**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 ...
... by using cool technologies
We need to go deeper!

RetDec

binary

preprocessing

image

core

LLVM IR

backend

source
Preprocessing

RetDec

Preprocessing

binary

RetDec

core

image

LLVM IR

backend

source
binary
Preprocessing

ELF
HEX
PE
COFF
Mach-O
raw
Preprocessing

fileformat library

binary representation

ELF  HEX  PE  COFF  Mach-O  raw
Preprocessing

ELFIO

PeLib

LLVM

fileformat library

binary representation

ELF

HEX

PE

COFF

Mach-O

raw
Preprocessing

ELFIO

PeLib

LLVM

fileformat library

binary representation

comp/packer detection

{JSON}

metadata

ELF

HEX

PE

COFF

Mach-O

raw
Preprocessing

- ELF
- HEX
- PE
- COFF
- Mach-O
- raw

ELFIO → PeLib → LLVM → fileformat library

→ binary representation

→ comp/packer detection

→ JSON metadata
Preprocessing

- ELF
- HEX
- PE
- COFF
- Mach-O
- raw

ELFIO
LLVM

fileformat library

binary representation

comp/packer detection

metadata

image loader library

unpacker

{JSON}

image

100 001
Preprocessing

libdwarf  \rightarrow  pdbparser  \rightarrow  debug library

\rightarrow  debug representation

DWARF

PDB
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

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

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

**ELFIO**
- strengthened
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
Preprocessing repos

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

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

**ELFIO**
- strengthened

**PDBparser**
- will hopefully be replaced by LLVM parsers

**Yaracpp**
- YARA C++ wrapper
Core: LLVM

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

- 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

```llvm
@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

```assembly
@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
Core: LLVM IR

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

```llvm
@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

image

passes init

LLVM IR
Core: decoder

100 001

image

passes init

recursive-traversal decoder

LLVM IR
Core: decoder

```
Capstone2LlvmIR
```

```
IR
Position
bin data
```

```
recursive-traversal
decoder pass
```

```
100
001
```

```
image
```

```
LLVM IR
```

Botconf 2017
Core: decoder

Image

100 001

bin data

IR
Position
bin data

Capstone2LlvmIR

Recursive-traversal decoder pass

LLVM IR
Core: Capstone2LlvmlIR

- Capstone insn $\rightarrow$ sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person $\approx$ 2-3 weeks
Core: Capstone2LlvmlIR

- Capstone insn → sequence of LLVM IR
- Handcoded sequences
  - 32/64-bit x86 – 1 person ≈ 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
• Capstone insn → sequence of LLVM IR
• Handcoded sequences
  • 32/64-bit x86 – 1 person ≈ 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

```c
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);
```
• 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
• Capstone insn → sequence of LLVM IR
• Handcoded sequences
  • 32/64-bit x86 – 1 person ≈ 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 → sequence of LLVM IR
• Handcoded sequences
  • 32/64-bit x86 – 1 person ≈ 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 ↔ ASM mapping, ...
Core: low-level passes

100001

image

passes init

recursive-traversal decoder

LLVM IR
Core: low-level passes

image

recursive-traversal decoder

low-level passes
stack/global vars, fnc args/rets,
data types, ...

passes init

LLVM IR
Core: assembly generation

100 001

image

recursive-traversal decoder

low-level passes
stack/global vars, fnc args/rets, data types, ...

passes init

assembler generation

dsm

LLVM IR
Core: high-level passes

100 001

image

passes init

recursive-traversal decoder

low-level passes
stack/global vars, fnc args/rets, data types, ...

assembler generation

high-level passes
LLVM + custom

dsm

LLVM IR
git RetDec
  • bin2llvmir library
  • bin2llvmirtool
Core repos

```
- **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
<table>
<thead>
<tr>
<th>Core repos</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>git</strong> RetDec</td>
</tr>
<tr>
<td>• bin2llvmir library</td>
</tr>
<tr>
<td>• bin2llvmirtool</td>
</tr>
<tr>
<td><strong>git</strong> Capstone2LlvmlIR</td>
</tr>
<tr>
<td>• Capstone instruction to LLVM IR translation</td>
</tr>
<tr>
<td><strong>git</strong> Capstone-dumper</td>
</tr>
<tr>
<td>• what does Capstone know about any instruction</td>
</tr>
<tr>
<td><strong>git</strong> Fnc-patterns</td>
</tr>
<tr>
<td>• statically linked function pattern creation and detection</td>
</tr>
<tr>
<td><strong>git</strong> Yaramod</td>
</tr>
<tr>
<td>• YARA to AST parsing &amp; C++ API to build new YARA rulesets</td>
</tr>
</tbody>
</table>
Core repos

- **git** RetDec
  - `bin2llvmir` library
  - `bin2llvmirtool`

- **git** Capstone2LlvmlIR
  - 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

**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

binary -> preprocessing

image -> core

LLVM IR -> backend

backend -> source
Backend

LLVM IR

BIR converter

optimizations

output gen

C
- BIR = Backend IR
- AST = Abstract syntax tree
- While loop: 
  \[\text{while } (x < 20) \{ x = x + (y \times 2); \} \]
• 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
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
• copy propagation
  • reducing the number of variables
Backend: optimizations

- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - \( a + -1 - -4 \Rightarrow a + 3 \)
• copy propagation
  • reducing the number of variables
• arithmetic expression simplification
  • \( a + -1 - -4 \) \( \Rightarrow \) \( a + 3 \)
• negation optimization
  • \textbf{if} \((!(a == b))\) \(\Rightarrow\) \textbf{if} \((a != b)\)
Backend: optimizations

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

- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - $a + -1 - -4 \Rightarrow a + 3$
- negation optimization
  - `if (!(a == b)) \Rightarrow if (a != b)`
- pointer arithmetic
  - `*(a + 4) \Rightarrow 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
  - \(a + -1 - -4 \Rightarrow a + 3\)
- negation optimization
  - ```
if (!a == b) \Rightarrow if (a != b)
```  
- pointer arithmetic
  - ```
*(a + 4) \Rightarrow a[4]
```  
- conversion of `while (true){ ... if (cond) break; ... }`
  - ```
for (cond){ ... }  
while (cond){ ... }
```  
- conversion of `if/else-if/else` chains to switch
- copy propagation
  - reducing the number of variables
- arithmetic expression simplification
  - \( a + -1 - -4 \Rightarrow a + 3 \)
- negation optimization
  - \( \text{if} \ (! (a == b)) \Rightarrow \text{if} \ (a != b) \)
- pointer arithmetic
  - \( *(a + 4) \Rightarrow a[4] \)
- conversion of while (true){ ... if (cond) break; ... }:
  - for (cond){ ... }
  - while (cond){ ... }
- conversion of if/else-if/else chains to switch
- ...
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_fffff7dc6 = socket(2, 3, 255)`
• 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_fffff7dc6` = `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_fffff7dc6 = 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

- RetDec
  - llvmir2hll library
  - llvmir2hlltool
How to use RetDec

Online decompilation service
https://retdec.com/decompilation/
How to use RetDec

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
- 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
- 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

// 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

![Diagram showing the process of how RetDec IDA plugin works](image)
How does RetDec IDA plugin work

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

Diagram:
- Binary
- Metadata (JSON)
- RetDec IDA plugin
How does RetDec IDA plugin work

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

Diagram:
- Binary
- Metadata (JSON)
- RetDec local build
- RetDec IDA plugin
How does RetDec IDA plugin work

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

[Diagram showing the process from binary to metadata to RetDec IDA plugin to C]

RetDec local build

Botconf 2017
How does RetDec IDA plugin work

◎ Goals

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

Diagram:

- Binary
- Metadata (JSON)
- RetDec local build
- RetDec IDA plugin
- C

Botconf 2017
How does RetDec IDA plugin work

◎ Goals

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

![Diagram showing the flow of data from binary to metadata, then to RetDec servers via API, and back to the IDA plugin]

RetDec IDA plugin
RetDec IDA plugin is interactive

```c
int32_t ack(int32_t m, int32_t n) {
    if (m == 0) {
        return n + 1;
    } else {
        return ack(m - 1, ack(m, n - 1));
    }
}
```

---

Global Variables

```
int32_t CTOR_LIST = -1; // 0x80497f4
```

Functions

```
int32_t __do_global_ctors_aux(void) {
    int32_t v1 = 0x8048698;
    while (*(int32_t *)&v1 -= 4)
        unknown_ffffffff();
    // continue -> 0x8048698
    return -1;
}
```
How was RetDec used so far

📅 **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
 здоровья 350,000 Web
👥 73,000 API
📊 410 decompilations daily
Real example #1: Vawtrak (x86)

**DSM**
```
.text:1000DD3E   ; HKEY__usercall getNextRegValue0fFKey@<eax>(CHAR *outSubKey@<edx>, char **filename, DWORD *pcbData)
.text:1000DD3E   getNextRegValue0fFKey proc near
.text:1000DD3E       push   ebp
.text:1000DD3F       mov    ebp, esp
.text:1000DD41       sub    esp, 14h
```

**Hex-Rays**
```
HKEY__usercall getNextRegValue0fFKey@<eax>(CHAR *outSubKey@<edx>, char **filename, DWORD *pcbData)
{
    v3 = outSubKey;
    v11 = 0;
    lstrcpyA(outSubKey, "SOFTWARE\");
    lstrcatA(v3, byte_10032D70);
}

**RetDec**
```
struct HKEY__ * getNextRegValue0fFKey(char * outSubKey, char ** filename, int32_t * pcbData) {
    int32_t lpValueName = (int32_t)outSubKey; // esi
    lstrcpyA(outSubKey, "SOFTWARE\");
    lstrcatA(outSubKey, byte_10032D70);
```
Hex-Rays

```c
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-CCh]+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_10618797,
        (int)&v2);
    v2 = -2147483646;
    regOpenKeyAndCallProc(
        HKEY_LOCAL_MACHINE,
        "Software\Microsoft\Windows\CurrentVersion\Run",
        (int)sub_10618797,
        (int)&v2);
    return v3;
}
```

RetDec

```c
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 = -0x7fffffff;
        regOpenKeyAndCallProc((struct HKEY__ *)-0x7fffffff,
            "Software\Microsoft\Windows\CurrentVersion\Run",
            0x10018797,
            v3);
    }
    return v3;
}
```
Real example #3: CryproWall (x86)

**DSM**

```assembly
dd offset loc_409688; jump table
; indirect table for switch

byte_4096FC
```

```assembly
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
```

**Hex-Rays**

```c
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**

```c
int32_t result = 1; // esi
switch (GetLastError())
{
    case 0:
        result = 0;
        break;
    case 2:
    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

```c
// 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
- refresh
- seamlessly integrated
- mature
- many plugins
- official support

- IDA and Hex-Rays have flaws
- not free
- proprietary
- big monolithic GUI app
Should you throw away your Hex-Rays?

NO!

- IDA and Hex-Rays are great
  - output quality
  - interactive
  - seamlessly integrated
  - mature
  - many plugins
  - official support
Should you throw away your Hex-Rays?

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

- **Not so obvious reasons**
  - ℹ️ LLVM is awesome
  - 🌐 different basic designs: interactive GUI vs. pipeline
  - 🐳 LLVM is OP (don’t worry, it won’t be nerfed)
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
  - Bitcoin
  - 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

- RetDec – the decompiler
- RetDec IDA plugin – Hex-Rays impersonation
RetDec is not only decompiler

- **RetDec** – the decompiler
- **RetDec IDA plugin** – Hex-Rays impersonation
- **Fileformat** – generic OFF parsing and analysis
- **Capstone2LlvmIR** – binary to LLVM translation
- **Fnc-patterns** – statically linked code detection in YARA (IDA F.L.I.R.T.)
- **Yaramod** – hack YARA rules in C++
- **Yaracpp** – YARA C++ wrapper
- **Ctypes** – info on function types
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)
- Continue improving the implementation
- Make it more portable: Bash ⇒ Python, ...
- 64-bit architectures supports
- Replace some libraries & modules
- We will see.
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)
- Continue improving the implementation
- Make it more portable: Bash ⇒ Python, ...
- 64-bit architectures supports
- Replace some libraries & modules
- We will see.
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
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)
- 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