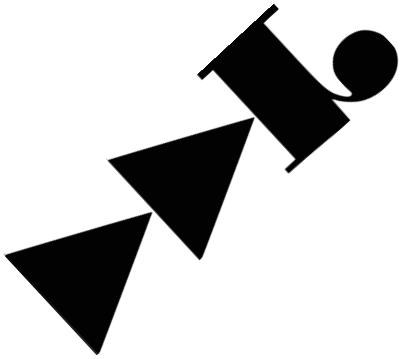
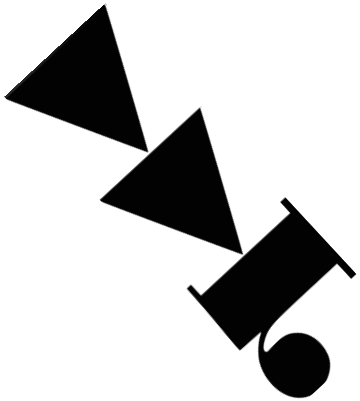
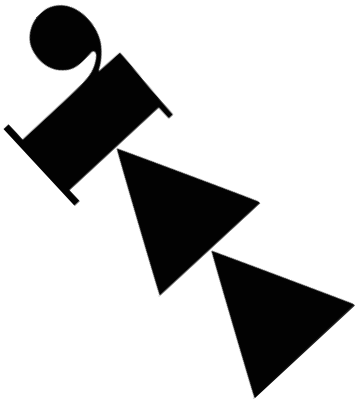
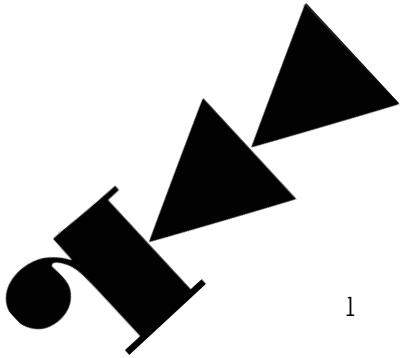



Function Argument Detection And Type Matching In Radare2

Oddcoder (@anoddcoder)
Xvilka (@akochkov)
R2Con 2016

GSoC

- Improving and fixing current types support
- Providing an uniform way to work with local variables
- Providing an uniform way to define argument types
- Very complex task involving a code refactoring
- http://radare.org/gsoc/2016/ideas.html#title_3

Task implications

- It required a change of the way to handle syscalls.
- Changing the '***t****'* commands syntax and set.
- Changing the '***af****'* commands syntax and set.
- It was required to simplify fetching types information to radeco.

Future directions

- It’s planned to add integration with signatures mechanisms.
- Supporting object-oriented programming data structures.
- Integration with demangling.
- Integration with DWARF/PDB/etc debug information.

$ whoami

- Computer and communication engineering student.
- Hobbyist system software developer, exploitation, etc..
- Active developer at Radare2 project.
- Participant in Google Summer Of Code 2016.

Google
Summer of Code

A word of gratitude

Problems I was trying to solve

- Auto detect formal arguments and local variables.

- Gather type information about them.

Rules of the game

No constraint solvers

No debugging.

Architecture independent

Modular and easy to extend.

Challenges

- A bit of everything is implemented.
- That was my first time to work on large code bases.
- I had to learn a lot on the fly.

Function Arguments/Locals

- Worked on x86 only.
- No 2 variables with same name in the whole binary.
- Only [ebp + stuff] could be manipulated.
- No stack based, or register based args/locals.

Function Arguments/Locals

NEW

- It is arch independent.
- variables are local to their function.
- Adding support for stack based, or register based args/ locals.

Old R2

```
|           ; DATA XREF from 0x0040056d (entry0)
|           0x00400646      55              push rbp
|           0x00400647      4889e5          mov rbp, rsp
|           0x0040064a      4883ec20        sub rsp, 0x20
|           0x0040064e      897dec          mov dword [rbp - 0x14], edi
|           0x00400651      488975e0        mov qword [rbp - 0x20], rsi
|           0x00400655      837dec01        cmp dword [rbp - 0x14], 1   ; [0x1:4]=0x2464c45
|       ,=< 0x00400659      7f11            jg 0x40066c
|       |   0x0040065b      bf70074000      mov edi, str.Usage_echo__string_ ; "Usage echo <string>" @ 0x400770
|       |   0x00400660      e89bfeffff      call sym.imp.puts
|       |   0x00400665      b800000000      mov eax, 0
|       ,==< 0x0040066a      eb64            jmp 0x4006d0
|       ||   ; JMP XREF from 0x00400659 (sym.main)
|       |`-> 0x0040066c      488b45e0        mov rax, qword [rbp - 0x20]
|       |   0x00400670      4883c008        add rax, 8
|       |   0x00400674      488b00          mov rax, qword [rax]
|       |   0x00400677      4889c7          mov rdi, rax
|       |   0x0040067a      e891feffff      call sym.imp.strlen
|       |   0x0040067f      8945fc          mov dword [rbp - 4], eax
|       |   0x00400682      8b45fc          mov eax, dword [rbp - 4]
|       |   0x00400685      4898            cdqe
|       |   0x00400687      4889c7          mov rdi, rax
|       |   0x0040068a      e8a1feffff      call sym.imp.malloc
|       |   0x0040068f      488945f0        mov qword [rbp - 0x10], rax
|       |   0x00400693      8b45fc          mov eax, dword [rbp - 4]
|       |   0x00400696      4863d0          movsxd rdx, eax
|       |   0x00400699      488b45e0        mov rax, qword [rbp - 0x20]
|       |   0x0040069d      4883c008        add rax, 8
|       |   0x004006a1      488b08          mov rcx, qword [rax]
|       |   0x004006a4      488b45f0        mov rax, qword [rbp - 0x10]
|       |   0x004006a8      4889ce          mov rsi, rcx
|       |   0x004006ab      4889c7          mov rdi, rax
|       |   0x004006ae      e83dfeffff      call sym.imp.strncpy
|       |   0x004006b3      488b45f0        mov rax, qword [rbp - 0x10]
|       |   0x004006b7      4889c7          mov rdi, rax
|       |   0x004006ba      e841feffff      call sym.imp.puts
|       |   0x004006bf      488b45f0        mov rax, qword [rbp - 0x10]
|       |   0x004006c3      4889c7          mov rdi, rax
|       |   0x004006c6      e815feffff      call sym.imp.free
|       |   0x004006cb      b800000000      mov eax, 0
|       |   ; JMP XREF from 0x0040066a (sym.main)
```

New R2

```
|           ; var int local_10h @ rbp-0x10
|           ; var int local_4h @ rbp-0x4
|           ; DATA XREF from 0x0040056d (entry0)
|           0x00400646      55              push rbp
|           0x00400647      4889e5          mov rbp, rsp
|           0x0040064a      4883ec20        sub rsp, 0x20
|           0x0040064e      897dec          mov dword [rbp - local_14h], edi
|           0x00400651      488975e0        mov qword [rbp - local_20h], rsi
|           0x00400655      837dec01        cmp dword [rbp - local_14h], 1 ; [0x1:4]=0x2464c45
|         ,=< 0x00400659      7f11            jg 0x40066c
|         |   0x0040065b      bf70074000      mov edi, str.Usage_echo__string_ ; "Usage echo <string>" @ 0x400770
|         |   0x00400660      e89bfeffff      call sym.imp.puts
|         |   0x00400665      b800000000      mov eax, 0
|        ,==< 0x0040066a      eb64            jmp 0x4006d0
|        ||   ; JMP XREF from 0x00400659 (sym.main)
|        |`-> 0x0040066c      488b45e0        mov rax, qword [rbp - local_20h]
|        |    0x00400670      4883c008        add rax, 8
|        |    0x00400674      488b00          mov rax, qword [rax]
|        |    0x00400677      4889c7          mov rdi, rax
|        |    0x0040067a      e891feffff      call sym.imp.strlen
|        |    0x0040067f      8945fc          mov dword [rbp - local_4h], eax
|        |    0x00400682      8b45fc          mov eax, dword [rbp - local_4h]
|        |    0x00400685      4898            cdqe
|        |    0x00400687      4889c7          mov rdi, rax
|        |    0x0040068a      e8a1feffff      call sym.imp.malloc
|        |    0x0040068f      488945f0        mov qword [rbp - local_10h], rax
|        |    0x00400693      8b45fc          mov eax, dword [rbp - local_4h]
|        |    0x00400696      4863d0          movsxd rdx, eax
|        |    0x00400699      488b45e0        mov rax, qword [rbp - local_20h]
|        |    0x0040069d      4883c008        add rax, 8
|        |    0x004006a1      488b08          mov rcx, qword [rax]
|        |    0x004006a4      488b45f0        mov rax, qword [rbp - local_10h]
|        |    0x004006a8      4889ce          mov rsi, rcx
|        |    0x004006ab      4889c7          mov rdi, rax
|        |    0x004006ae      e83dfeffff      call sym.imp.strncpy
|        |    0x004006b3      488b45f0        mov rax, qword [rbp - local_10h]
|        |    0x004006b7      4889c7          mov rdi, rax
|        |    0x004006ba      e841feffff      call sym.imp.puts
|        |    0x004006bf      488b45f0        mov rax, qword [rbp - local_10h]
|        |    0x004006c3      4889c7          mov rdi, rax
|        |    0x004006c6      e815feffff      call sym.imp.free
```

How does Automatic detection work ?

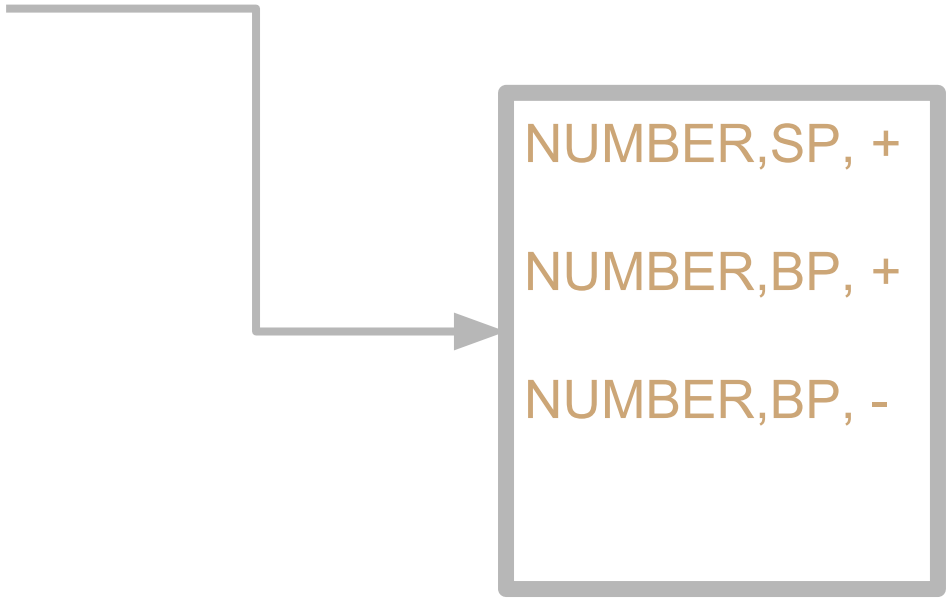
It is all strings matching on ESIL.

How does Automatic detection work ?

1. Locate candidate var.

How does Automatic detection work ?

1. Locate candidate var.

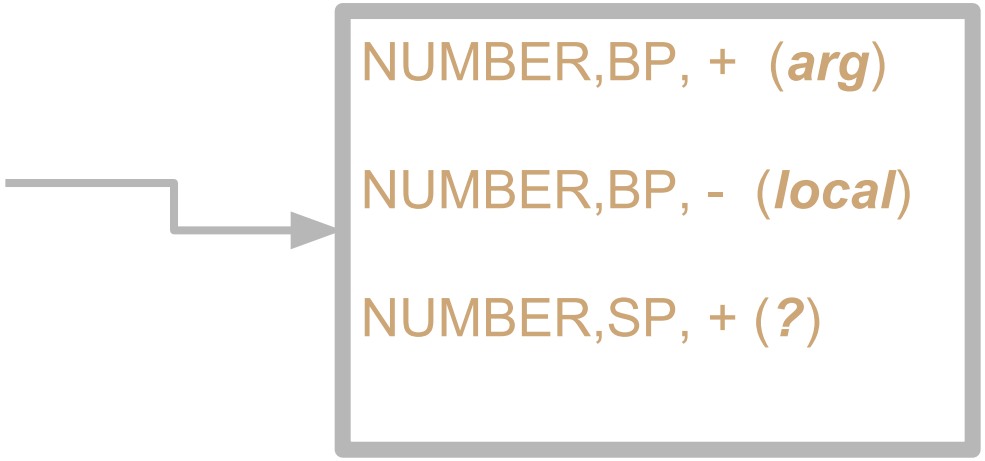


How does Automatic detection work ?

1. Locate candidate var.
2. Identify type of candidate var.

How does Automatic detection work ?

1. Locate candidate var.
2. Identify type of candidate var.

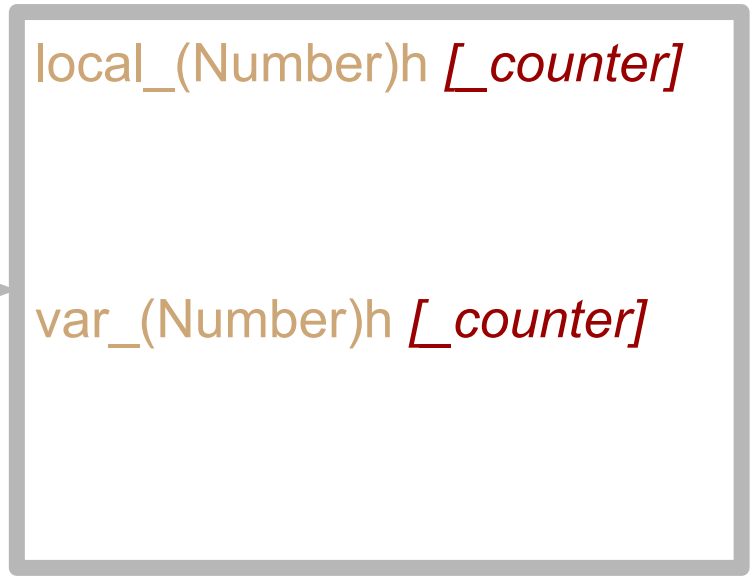


How does Automatic detection work ?

1. Locate candidate var.
2. Identify type of candidate var.
3. Naming and registering var.

How does Automatic detection work ?

1. Locate candidate var.
2. Identify type of candidate var.
3. Naming and registering var.



Type Matching

- Implementing type hints for standard function.
- Depends on calling convention of that function.
- Needs emulation to avoid confusion with stack & registers tracing.

What is Calling Convention?

- It is the way functions call another another function.
- How arguments are arranged in the stack ?
- Who cleans the callee stack ?
- Where is the return value placed ?

Cdecl

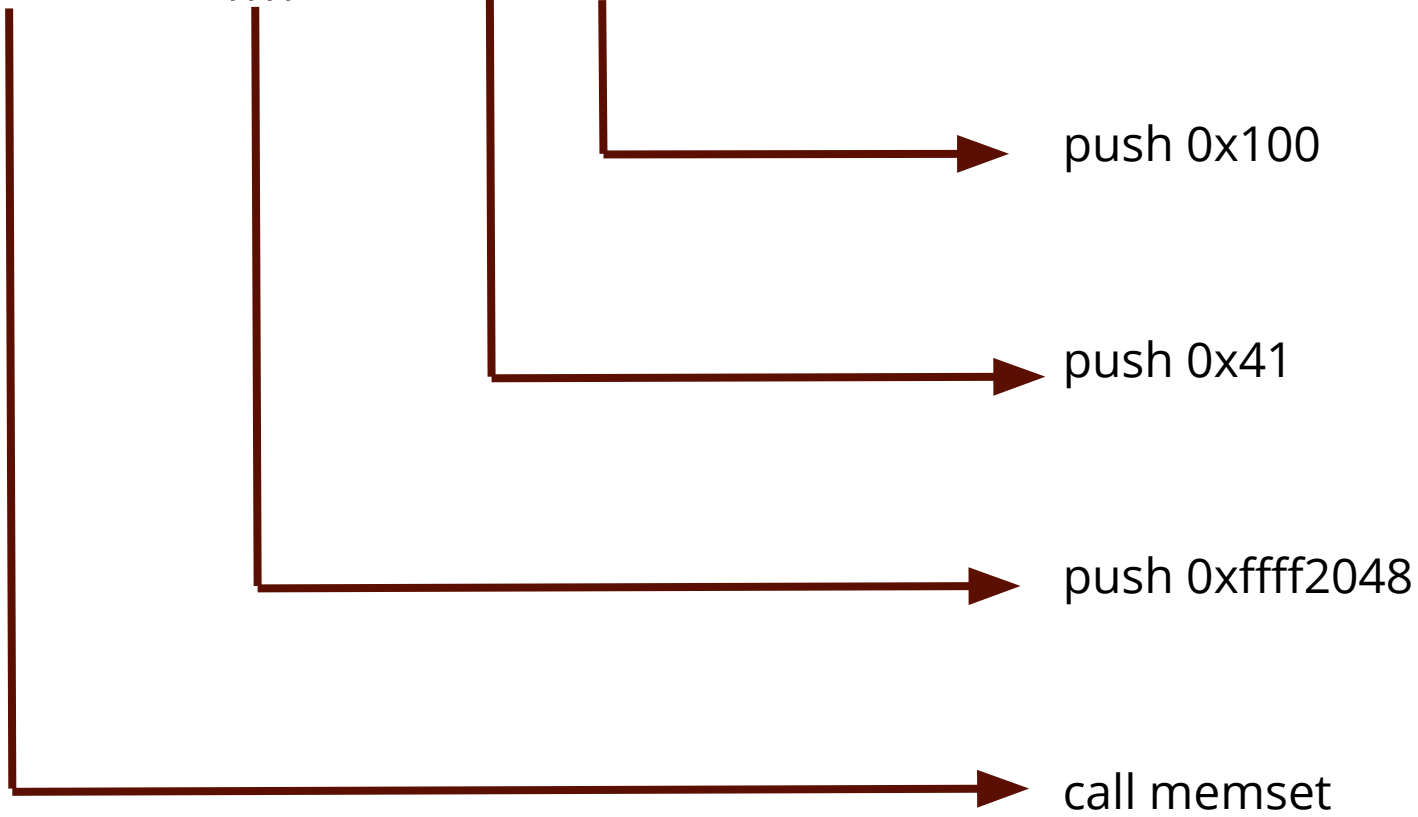
- Most common calling convention in linux i386.
- Arguments are pushed right to left on the stack.
- Return value is put in eax.
- The caller is the one who cleans the stack.

Cdecl

memset (0xffff2048, 'A', 0x100);

Cdecl

memset (0xffff2048, 'A', 0x100);



Stdcall

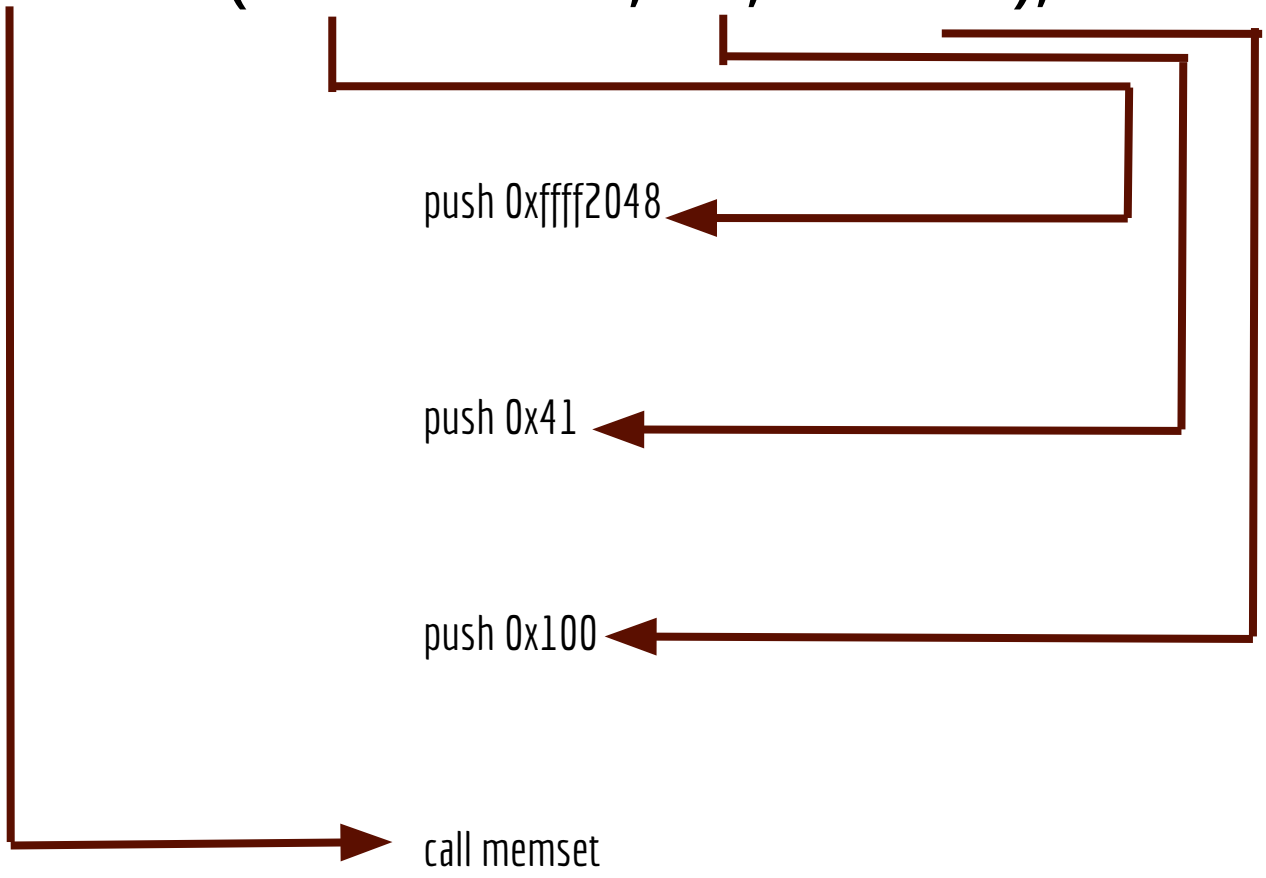
- Popular on i386 windows machines.
- Arguments pushed left to right on the stack.
- Return value is also put in eax.
- Callee cleans its own stack frame.

Stdcall

memset (0xffff2048, ‘A’, 0x100);

Stdcall

memset (0xffff2048, ‘A’, 0x100);



X64 Windows ABI

- Arguments passed into RCX, RDX, R8, R9, then the remaining arguments are put on stack right to left.
- Return value is put into rax

And much more

- And there are lots of other calling conventions.
- Calling conventions for each architecture, and for each Operating system.
- Even some programming languages and some compilers got their own calling conventions (ABI)

Why would we care ?

We already had this.

```
|           ; DATA XREF from 0x0040056d (entry0)
|           0x00400646      55              push rbp
|           0x00400647      4889e5          mov rbp, rsp
|           0x0040064a      4883ec20        sub rsp, 0x20
|           0x0040064e      897dec          mov dword [rbp - 0x14], edi
|           0x00400651      488975e0        mov qword [rbp - 0x20], rsi
|           0x00400655      837dec01        cmp dword [rbp - 0x14], 1   ; [0x1:4]=0x2464c45
|       ,=< 0x00400659      7f11            jg 0x40066c
|       |   0x0040065b      bf70074000      mov edi, str.Usage_echo__string_ ; "Usage echo <string>" @ 0x400770
|       |   0x00400660      e89bfeffff      call sym.imp.puts
|       |   0x00400665      b800000000      mov eax, 0
|      ,==< 0x0040066a      eb64            jmp 0x4006d0
|      ||   ; JMP XREF from 0x00400659 (sym.main)
|      |`-> 0x0040066c      488b45e0        mov rax, qword [rbp - 0x20]
|      |    0x00400670      4883c008        add rax, 8
|      |    0x00400674      488b00          mov rax, qword [rax]
|      |    0x00400677      4889c7          mov rdi, rax
|      |    0x0040067a      e891feffff      call sym.imp.strlen
|      |    0x0040067f      8945fc          mov dword [rbp - 4], eax
|      |    0x00400682      8b45fc          mov eax, dword [rbp - 4]
|      |    0x00400685      4898            cdqe
|      |    0x00400687      4889c7          mov rdi, rax
|      |    0x0040068a      e8a1feffff      call sym.imp.malloc
|      |    0x0040068f      488945f0        mov qword [rbp - 0x10], rax
|      |    0x00400693      8b45fc          mov eax, dword [rbp - 4]
|      |    0x00400696      4863d0          movsxd rdx, eax
|      |    0x00400699      488b45e0        mov rax, qword [rbp - 0x20]
|      |    0x0040069d      4883c008        add rax, 8
|      |    0x004006a1      488b08          mov rcx, qword [rax]
|      |    0x004006a4      488b45f0        mov rax, qword [rbp - 0x10]
|      |    0x004006a8      4889ce          mov rsi, rcx
|      |    0x004006ab      4889c7          mov rdi, rax
|      |    0x004006ae      e83dfeffff      call sym.imp.strncpy
|      |    0x004006b3      488b45f0        mov rax, qword [rbp - 0x10]
|      |    0x004006b7      4889c7          mov rdi, rax
|      |    0x004006ba      e841feffff      call sym.imp.puts
|      |    0x004006bf      488b45f0        mov rax, qword [rbp - 0x10]
|      |    0x004006c3      4889c7          mov rdi, rax
|      |    0x004006c6      e815feffff      call sym.imp.free
|      |    0x004006cb      b800000000      mov eax, 0
|      |    ; JMP XREF from 0x0040066a (sym.main)
```

And we turned it into that

```
|           ; var int local_10h @ rbp-0x10
|           ; var int local_4h @ rbp-0x4
|           ; DATA XREF from 0x0040056d (entry0)
|           0x00400646      55              push rbp
|           0x00400647      4889e5          mov rbp, rsp
|           0x0040064a      4883ec20        sub rsp, 0x20
|           0x0040064e      897dec          mov dword [rbp - local_14h], edi
|           0x00400651      488975e0        mov qword [rbp - local_20h], rsi
|           0x00400655      837dec01        cmp dword [rbp - local_14h], 1 ; [0x1:4]=0x2464c45
|       ,=< 0x00400659      7f11            jg 0x40066c
|       |   0x0040065b      bf70074000      mov edi, str.Usage_echo__string_ ; "Usage echo <string>" @ 0x400770
|       |   0x00400660      e89bfeffff      call sym.imp.puts
|       |   0x00400665      b800000000      mov eax, 0
|      ,==< 0x0040066a      eb64            jmp 0x4006d0
|      ||   ; JMP XREF from 0x00400659 (sym.main)
|      |`-> 0x0040066c      488b45e0        mov rax, qword [rbp - local_20h]
|      |    0x00400670      4883c008        add rax, 8
|      |    0x00400674      488b00          mov rax, qword [rax]
|      |    0x00400677      4889c7          mov rdi, rax
|      |    0x0040067a      e891feffff      call sym.imp.strlen
|      |    0x0040067f      8945fc          mov dword [rbp - local_4h], eax
|      |    0x00400682      8b45fc          mov eax, dword [rbp - local_4h]
|      |    0x00400685      4898            cdqe
|      |    0x00400687      4889c7          mov rdi, rax
|      |    0x0040068a      e8a1feffff      call sym.imp.malloc
|      |    0x0040068f      488945f0        mov qword [rbp - local_10h], rax
|      |    0x00400693      8b45fc          mov eax, dword [rbp - local_4h]
|      |    0x00400696      4863d0          movsxd rdx, eax
|      |    0x00400699      488b45e0        mov rax, qword [rbp - local_20h]
|      |    0x0040069d      4883c008        add rax, 8
|      |    0x004006a1      488b08          mov rcx, qword [rax]
|      |    0x004006a4      488b45f0        mov rax, qword [rbp - local_10h]
|      |    0x004006a8      4889ce          mov rsi, rcx
|      |    0x004006ab      4889c7          mov rdi, rax
|      |    0x004006ae      e83dfeffff      call sym.imp.strncpy
|      |    0x004006b3      488b45f0        mov rax, qword [rbp - local_10h]
|      |    0x004006b7      4889c7          mov rdi, rax
|      |    0x004006ba      e841feffff      call sym.imp.puts
|      |    0x004006bf      488b45f0        mov rax, qword [rbp - local_10h]
|      |    0x004006c3      4889c7          mov rdi, rax
|      |    0x004006c6      e815feffff      call sym.imp.free
```

And this is our final goal

```
|           ; DATA XREF from 0x0040056d (entry0)
|           0x00400646      55              push rbp
|           0x00400647      4889e5          mov rbp, rsp
|           0x0040064a      4883ec20        sub rsp, 0x20
|           0x0040064e      897dec          mov dword [rbp - local_14h], edi
|           0x00400651      488975e0        mov qword [rbp - local_20h], rsi
|           0x00400655      837dec01        cmp dword [rbp - local_14h], 1 ; [0x1:4]=0x2464c45
|       ,=< 0x00400659      7f11            jg 0x40066c                 ;[2]
|       |   0x0040065b      bf70074000      mov edi, str.Usage_echo__string_ ; "Usage echo <string>" @ 0x400770 ; con
|       |   0x00400660      e89bfeffff      call sym.imp.puts           ;[3]
|       |   0x00400665      b800000000      mov eax, 0
|       ,==< 0x0040066a      eb64            jmp 0x4006d0                ;[4]
|       |`-> 0x0040066c      488b45e0        mov rax, qword [rbp - local_20h]
|       |   0x00400670      4883c008        add rax, 8
|       |   0x00400674      488b00          mov rax, qword [rax]
|       |   0x00400677      4889c7          mov rdi, rax                ; const char * s
|       |   0x0040067a      e891feffff      call sym.imp.strlen         ;[5]
|       |   0x0040067f      8945fc          mov dword [rbp - local_4h], eax
|       |   0x00400682      8b45fc          mov eax, dword [rbp - local_4h]
|       |   0x00400685      4898            cdqe
|       |   0x00400687      4889c7          mov rdi, rax                ; size_t size
|       |   0x0040068a      e8a1feffff      call sym.imp.malloc         ;[6]
|       |   0x0040068f      488945f0        mov qword [rbp - local_10h], rax
|       |   0x00400693      8b45fc          mov eax, dword [rbp - local_4h]
|       |   0x00400696      4863d0          movsxd rdx, eax             ; size_t n
|       |   0x00400699      488b45e0        mov rax, qword [rbp - local_20h]
|       |   0x0040069d      4883c008        add rax, 8
|       |   0x004006a1      488b08          mov rcx, qword [rax]
|       |   0x004006a4      488b45f0        mov rax, qword [rbp - local_10h]
|       |   0x004006a8      4889ce          mov rsi, rcx                ; const char * src
|       |   0x004006ab      4889c7          mov rdi, rax                ; char * dest
|       |   0x004006ae      e83dfeffff      call sym.imp.strncpy        ;[7]
|       |   0x004006b3      488b45f0        mov rax, qword [rbp - local_10h]
|       |   0x004006b7      4889c7          mov rdi, rax                ; const char * s
|       |   0x004006ba      e841feffff      call sym.imp.puts           ;[3]
|       |   0x004006bf      488b45f0        mov rax, qword [rbp - local_10h]
|       |   0x004006c3      4889c7          mov rdi, rax
|       |   0x004006c6      e815feffff      call sym.imp.free           ; sym.imp.__libc_start_main-0x40 ;[8]
```

Implementation details

1. Read next instruction.

Implementation details

1. Read next instruction.

2. Record that instruction for traceback.

Implementation details

1. Read next instruction.
2. Record that instruction for traceback.
3. Record stack and register accesses.

Implementation details

1. Read next instruction.
2. Record that instruction for traceback.
3. Record stack and register accesses.
4. Go to step 1  .

Implementation details

1. Read next instruction.
2. Record that instruction for traceback.
3. Record stack and register accesses.
4. Go to step 1 .

→ This loop is interrupted when we find a call instruction

Implementation details

Interesting stuff starts when we catch a call instruction.

Implementation details

1- Grab function's definition from types database.

Implementation details

1- Grab function's definition from types database.

2- Do the math to calculate the location of all arguments.

Implementation details

1- Grab function's definition from types database.

2- Do the math to calculate the location of all arguments.

3- Search the trace log for writing to these locations.

Implementation details

Now it is up to us what to do with all that information.

SHOW TIME

Let’s add types SDB(s)

See also

GitHub issue https://github.com/radare/radare2/issues/3655

RT (Radare Today) article http://radare.today/posts/GSOC-The-last-commit-213c6f/

Calling convention db docs:
https://github.com/radare/radare2/blob/master/doc/calling-conventions.md

Types db docs: https://github.com/radare/radare2/blob/master/doc/types.md