

# Extending IDA with Scripts

---



**Dr. Jared DeMott**

CHIEF HACKING OFFICER

@jareddemott [www.vdalabs.com](http://www.vdalabs.com)



# Overview



IDC

IDAPy

SDK



# IDC Scripting

## C like

- No data types

## Declare Variables at Top of Script

- No globals
- Arrays are not fun

## From IDA help

- IDC Language
  - Expressions
  - Statements
  - Variables
  - Functions

IDC

## Variables

- Use 'auto' as the key word

## Functions

- All are defined with the "static"
- Argument list does not require any type info or the auto keyword
- Return type never specified



# Example IDC Function

```
static myCoolFunc(arg1, arg2) {  
    auto var1;  
    var1 = arg1 + arg2;  
    return var1;  
}
```



# IDC

## Statements

- Most C available
  - Loops
    - For, while, do. Break/continue
      - {} same as C
- No switch or goto

## Expressions which Differ from C

- Integers promote to float as required
- Add ('+') string concatenation
- Comparisons work for string operands
  - If ("frank" == "john")



# Interacting with IDA Database

## Read Data

- Return -1 if virtual addr is invalid
- long **Byte**(long addr);
- long **Word**(long addr);
- long **Dword**(long addr);

## Writing to IDA DB

- Useful for self-mod code
- void **PatchByte**(long addr, long val);
- void **PatchWord**(long addr, long val);
- void **PatchDword**(long addr, long val);



# Interacting with Reverse Engineer

***void Message (string format, ...)***

- Print to the message area
  - We'll use "print" in python

***void Warning (string format, ...)***

- Show warning dialog box

***long AskYN (long default, string prompt)***

- Ask a yes/no in dialog box

***string AskFile (bool forsave, string mask, string prompt)***





# Cursor Control and Address Operations

## ***long ScreenEA()***

- Read current cursor location
  - Return virtual address

## ***long Jump(long addr)***

- Set cursor to indicated location

## ***long LocByName(string name)***

- Return address for specified name

## ***long Name(long ea)***

- Return name of virtual address

# Iterating Cross References

## Iterate Code

- Rfirst, Rnext, RfirstB, RnextB
  - B is “to”; other is “from”

## Iterate Data

- Dfirst, Dnext, DfirstB, DnextB

## Iterate Segments

- FirstSeg, NextSeg

## Iterate Functions

- long NextFunction(long ea)
- long PrevFunction(long ea)
- long NextHead(long ea, long max)
  - Return the addr of the next item following the ea



```
#include <idc.idc>
```

```
static main() {  
    list_callers("strcpy");  
    list_callers("sprintf"); // use _name for some compilers  
}
```



```
static list_callers(bad_func) {
    auto func, addr, xref, source;
    func = LocByName(bad_func);
    if( func == BADADDR) {
        Warning("Sorry, %s, not found in database", bad_func);
    }
    else {
        for( addr = RfirstB(func); addr != BADADDR; addr = RnextB(func, addr)) {
            xref = XrefType();
            if( xref == fl_CN || xref == fl_CF) { //function calls only (near call or far call)
                source = GetFunctionName(addr);
                Message("%s is called from 0x%x in %s\n", bad_func, addr, source);
            }
        }
    }
}
```



# IDA python

## Extend IDA

- Plugin that allows scripts to be authored in Python
- Scripts have access to IDA and Python APIs
- Common amongst security researchers and malware analysts



# Demo



## Decode the secret message

- Pretend we're working with live malware that we don't want to actually run/execute
  - Find key
  - Reverse decoding algorithm
  - Write python to emulate that
  - Print the decoded message to the output window
- *1\_skeleton.py* → *2\_skeleton.py*



# SDK Extensions

---

## Exploring Plugins



# Example Extension

## CollabREate – Chris Eagle & Tim Vidas

- <http://www.idabook.com/collabreate>
- IDA Plugin is client
  - Copy the .plw into plugin dir
    - Must be compiled for version of IDA
- Java server w/ Postgres/Mysql backend
  - Commands are intercepted as they occur and sent to the server
  - Server caches and distributes commands
  - Authentication and project management





<http://hex-rays.com>

Pseudo code in IDA  
Might not Properly  
Decompile Malware

1 Hexadecimal code: uncomprehensible for humans

```
00 00 C6 05 85 14 00 00 02 C6 05 89 14 00 00 00
74 28 A1 78 14 00 00 6A 08 50 E8 F5 10 00 00 83
C4 08 83 F8 FF 75 02 33 C0 83 E0 0F 0F B6 CB C1
E0 08 0B C1 A3 90 14 00 00 C3 0F B6 C3 3D 80 00
00 00 A3 90 14 00 00 72 0A 05 00 0F 00 00 A3 90
14 00 00 C3 53 8B D8 C1 F8 0A 83 F8 03 0F 87 C8
01 00 00 FF 24 85 90 05 00 00 83 FB 40 75 0B 66
C7 05 7C 14 00 00 04 00 5B C3 85 DB 75 0B 66 C7
05 7C 14 00 00 00 00 5B C3 83 FB 04 75 0B 66 C7
05 7C 14 00 00 1A 00 5B C3 83 FB 02 75 0B 66 C7
05 7C 14 00 00 24 00 5B C3 83 FB 03 75 0B 66 C7
05 7C 14 00 00 21 00 5B C3 F7 C3 F8 0F 00 00 75
11 8B C3 66 C7 05 7C 14 00 00 25 00 5B E9 B2 FE
FF FF F7 C3 80 0F 00 00 75 1F 8B C3 C1 F8 05 83
E0 03 66 8B 0C 45 D0 05 00 00 8B C3 66 89 0D 7C
14 00 00 5B E9 8B FE FF FF 8B D3 C1 FA 06 66 8B
04 55 B0 05 00 00 66 A3 7C 14 00 00 8B C3 E8 71
FE FF FF C1 E8 05 83 E3 01 66 89 1D A2 14 00 00
C6 05 9D 14 00 00 01 C6 05 A1 14 00 00 00 5B C3
8B CB C1 F9 03 83 E1 83 66 8B 14 4D A3 05 00 00
8B C3 66 89 15 7C 14 00 00 E8 36 FE FF FF C1 FB
05 83 E3 07 89 1D A4 14 00 00 C6 05 9D 14 00 00
05 C6 05 A1 14 00 00 00 5B C3 8B C3 C1 F8 05 83
E0 03 83 E8 00 74 79 83 E8 01 6A 0B 74 3A A1 78
14 00 00 50 E8 8B 0F 00 00 83 E0 60 81 E3 FF 01
00 00 83 C4 08 C1 E0 04 0B C3 66 C7 05 7C 14 00
00 1B 00 C6 05 85 14 00 00 07 A3 90 14 00 00 C6
05 89 14 00 00 09 5B C3 8B 0D 78 14 00 00 51 E8
50 0F 00 00 83 E0 60 0F B6 D3 83 C4 08 C1 E0 04
0B C2 66 C7 05 7C 14 00 00 19 00 C6 05 85 14 00
```

2 Disassembler output: makes sense but lengthy

```
mov     eax, dword ptr ds:?[cmd@@@3Vi
add     eax, 1
; int
push    eax
; unsigned
call    ?get_flags_ex@@YAKKH@Z ; ge
mov     ecx, eax
add     esp, 8
and     ecx, 600h
jz      short loc_2D1
cmp     ecx, 200h
jnz     short loc_2DE

; CODE XREF
shr     eax, 0Ch
test    al, 1
jnz     short loc_2DE
mov     eax, 1
retn

; CODE XREF
; may_grow(
xor     eax, eax
retn
```

3 Decompiler output: concise and familiar for programmers

```
if ( update )
{
    result = "+-";
}
else
{
    if ( add )
    {
        result = "+";
    }
    else
    {
        if ( dword_41F
```



# IDA SDK

## Required to Build Full Extensions

- Written in C++
- A .dll or shared object in UNIX
- Steeper learning curve
  - Worth it when more power is required

## Three Types of Extensions can be Created

- Plug-in
  - Extend IDA functionality
- Processor Module
  - Extend IDA to understand new instruction sets
- File Loader
  - Extend IDA to understand new executable/object file formats



# SDK

## Layout and API

- *Libxxx* directories contain IDA libs for building plugins/loaders/modules with various compilers
- *Bin* directory contains compiled plugins
- *Include* directory contains header files
- *Ldr* directory contains sample loaders for file formats
- *Module* directory contains sample processor modules
- *Plugins* directory contains sample plugins



# Plugin Architecture

## Plugins Export

- *plugin\_t* PLUGIN
  - Defined in loader.hpp

## Describes Plugin Options

- Name of the init function
- Name of the terminate function
- Name of the run function
- Desired hotkey to activate the plugin



More

[http://www.openrce.org/reference\\_library/files/ida/idapw.pdf](http://www.openrce.org/reference_library/files/ida/idapw.pdf)

Also

- IDC to SDK Reference
  - At the end of Eagle's book
- You can also use Windows API in plugin
  - [http://www.openrce.org/reference\\_library/ida\\_sdk](http://www.openrce.org/reference_library/ida_sdk)



# Summary



**Python Scripting in IDA**

**SDK**

**Next:**

- Course on Exploit Development

