IDAPython: The Wonder Woman of Embedded Reversing

Maddie Stone
@maddiestone

DerbyCon 2017

Who am I?

- Reverse engineer at the Johns Hopkins Applied Physics Lab
  - Mostly embedded devices
  - Merge of hardware and firmware reverse engineering
  - Lead of reverse engineering working group at JHU/APL
- BS in Computer Science, Russian, and Applied Math
- MS in Computer Science

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

Reduce the time required to analyze firmware of embedded devices by using IDAPython.

IDAPython Embedded Toolkit

https://github.com/maddiestone/IDAPythonEmbeddedToolkit

IDAPython

- "IDAPython is an IDA Pro plugin that integrates the Python programming language, allowing scripts to run in IDA Pro"
  - https://github.com/idapython/src/ – 6.95
  - Docs: https://www.hex-rays.com/products/ida/support/idapython_docs/
    - idc module contains 98% of the functions we use (in 6.95)

IDA 7.0

- Released last Thursday, Sept 14th.
- Moved to 64-bit
- Implications for IDAPython
  - New SDK: https://www.hex-rays.com/products/ida/7.0/docs/api70_porting_guide.shtml
  - Function names changed, removed, or moved
  - Compatibility layer in IDA 7.0 that supports MOST of 6.95 SDK
  - Documentation of new 7.0 SDK is still a work in progress

IDA 7.0– Implications for this Talk

- All scripts have not been fully tested for IDA 7.0
  - Some functionality in IDAPython Embedded Toolkit is not supported by the "compatibility layer"
  - Hope to release scripts with 7.0 testing in next week
  - Over next (maybe couple) month, hope to port everything to 7.0 SDK
    - Take advantage of new features too! ☺

- Live demo will be on IDA 6.95 instead of IDA 7.0

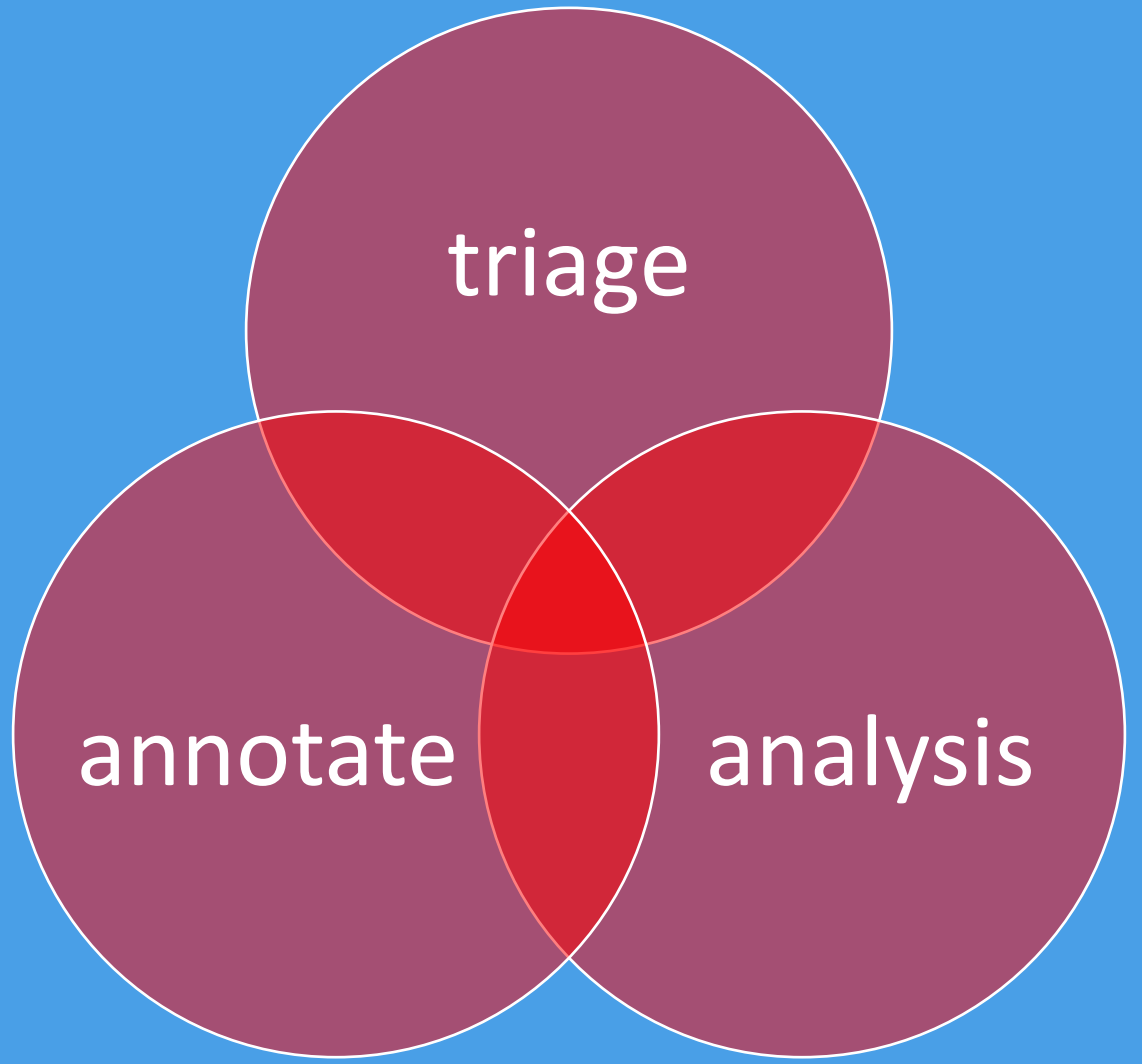
Why do you Care?

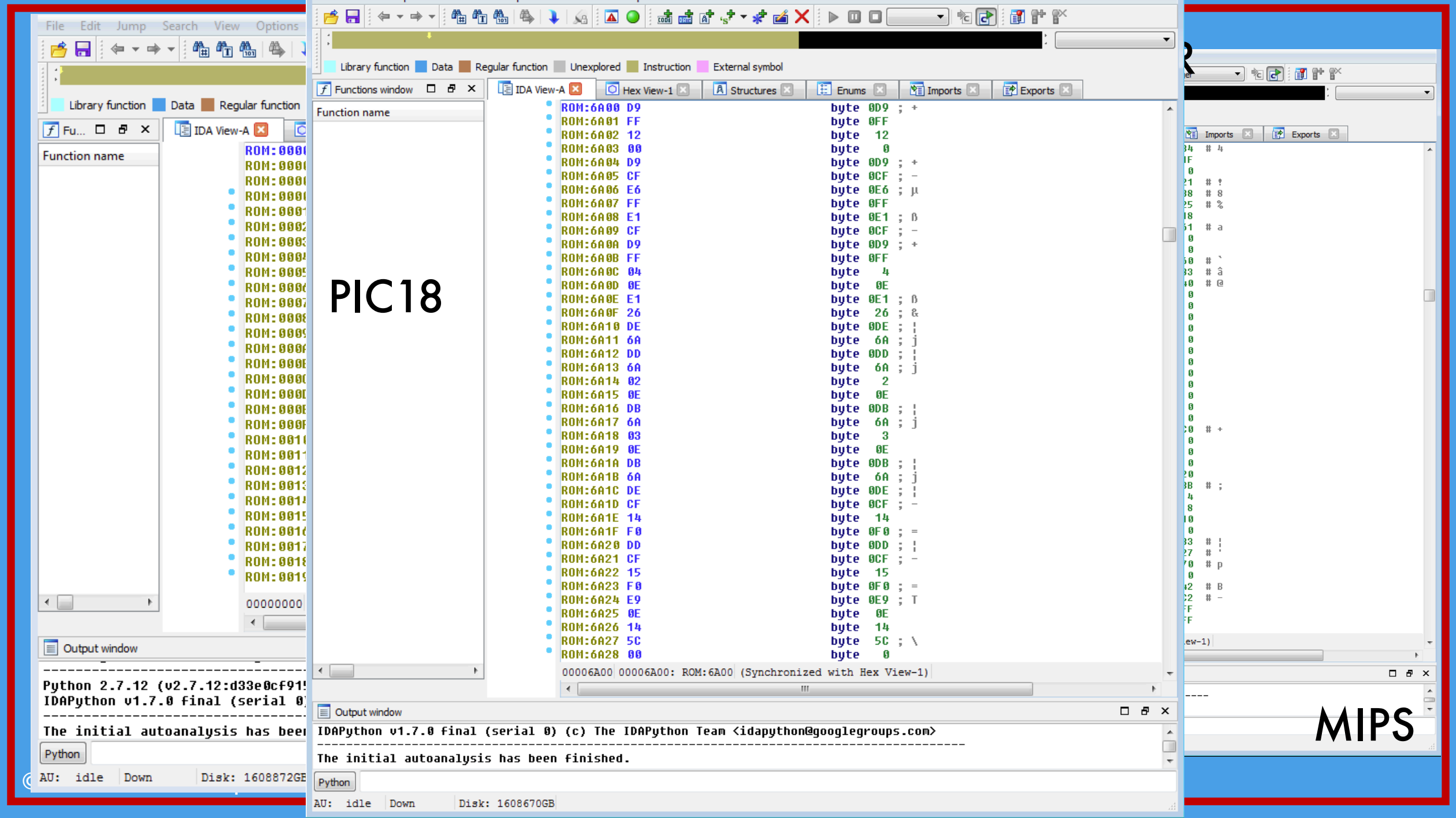
- Current Resources for IDAPython
  - Mostly x86 or ARM-based (PC applications or malware)
  - Palo Alto Networks: http://researchcenter.paloaltonetworks.com/2015/12/using-idapython-to-make-your-life-easier-part-1/
  - "The Beginner's Guide to IDAPython" by Alexander Hanel (@nullandnull)
- More embedded devices (hello, Internet of Things!)
  - Microcontroller/microprocessor architectures
  - Different goals of analysis than malware/application RE

Important Differences for Firmware Images

- Purpose of analysis
- Entire firmware image vs. application
- Memory structure
- Many different architectures

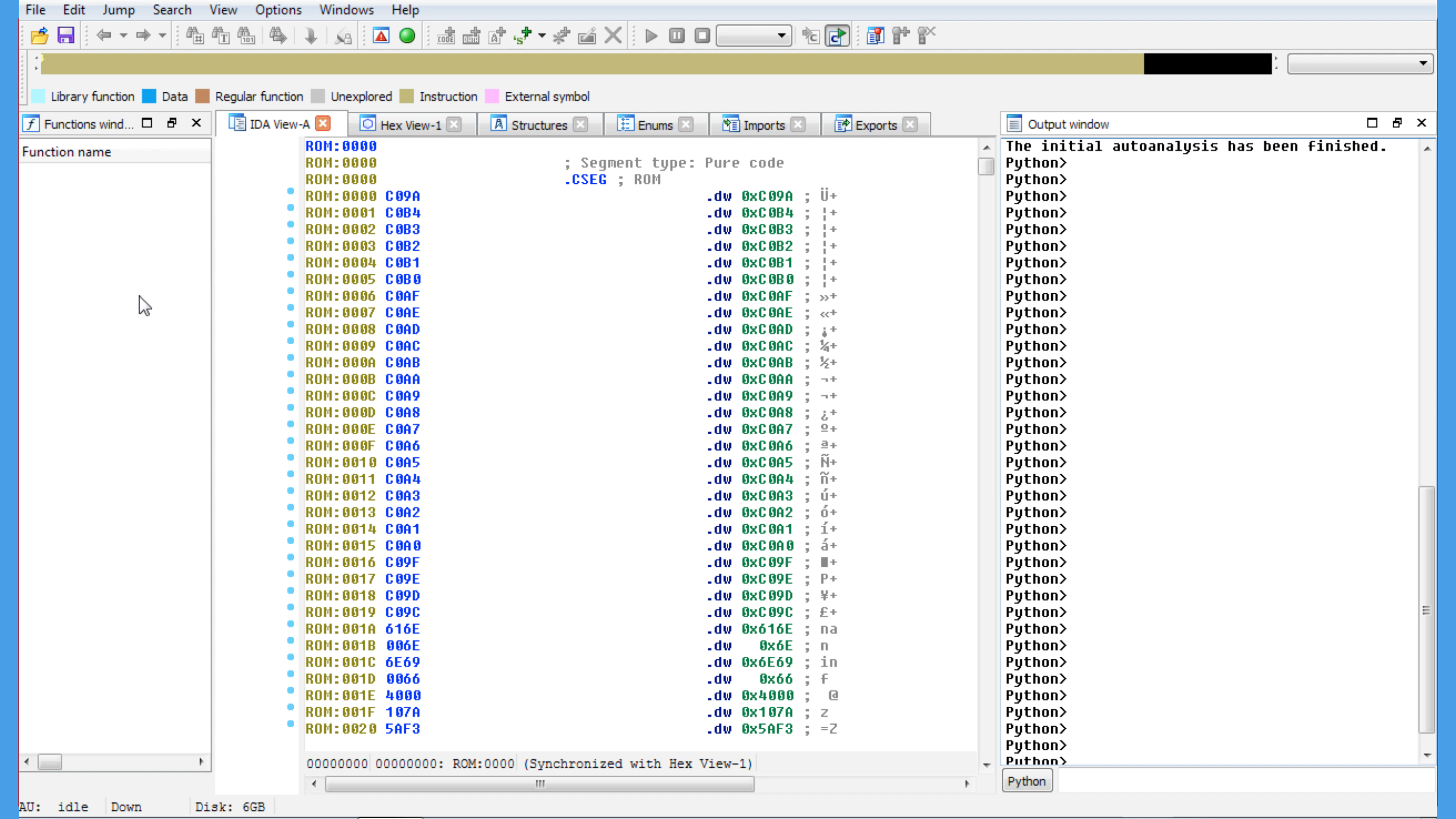
Scripting the Reverse Engineering Process




PIC18
MIPS
File Edit Jump Search View Options Windows Help
Library function Data Regular function Unexplored Instruction External symbol
Functions window
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
Function name
ROM:6A00 D9 byte 0D9 ; +
ROM:6A01 FF byte 0FF
ROM:6A02 12 byte 12
ROM:6A03 00 byte 0
ROM:6A04 D9 byte 0D9 ; +
ROM:6A05 CF byte 0CF ; -
ROM:6A06 E6 byte 0E6 ; µ
ROM:6A07 FF byte 0FF
ROM:6A08 E1 byte 0E1 ; ß
ROM:6A09 CF byte 0CF ; -
ROM:6A0A D9 byte 0D9 ; +
ROM:6A0B FF byte 0FF
ROM:6A0C 04 byte 4
ROM:6A0D 0E byte 0E
ROM:6A0E E1 byte 0E1 ; ß
ROM:6A0F 26 byte 26 ; &
ROM:6A10 DE byte 0DE ; ¦
ROM:6A11 6A byte 6A ; j
ROM:6A12 DD byte 0DD ; ¦
ROM:6A13 6A byte 6A ; j
ROM:6A14 02 byte 2
ROM:6A15 0E byte 0E
ROM:6A16 DB byte 0DB ; ¦
ROM:6A17 6A byte 6A ; j
ROM:6A18 03 byte 3
ROM:6A19 0E byte 0E
ROM:6A1A DB byte 0DB ; ¦
ROM:6A1B 6A byte 6A ; j
ROM:6A1C DE byte 0DE ; ¦
ROM:6A1D CF byte 0CF ; -
ROM:6A1E 14 byte 14
ROM:6A1F F0 byte 0F0 ; ≡
ROM:6A20 DD byte 0DD ; ¦
ROM:6A21 CF byte 0CF ; -
ROM:6A22 15 byte 15
ROM:6A23 F0 byte 0F0 ; ≡
ROM:6A24 E9 byte 0E9 ; T
ROM:6A25 0E byte 0E
ROM:6A26 14 byte 14
ROM:6A27 5C byte 5C ; \
ROM:6A28 00 byte 0
00006A00 00006A00: ROM:6A00 (Synchronized with Hex View-1)
Output window
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
The initial autoanalysis has been finished.
Python
AU: idle Down Disk: 1608670GB

How IDAPython Helps – triage

- define_data_as_types.py
  - mass assign bytes as instructions, data, offsets
- define_code_functions.py
  - auto-assign "unexplored" bytes as code and attempt to define functions
- make_strings.py
  - searches an address range for series of ASCII characters to define as strings


File Edit Jump Search View Options Windows Help
Library function Data Regular function Unexplored Instruction External symbol
Functions wind...
Function name
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
ROM:0000
; Segment type: Pure code
.CSEG ; ROM
ROM:0000 C09A .dw 0xC09A ; Ü+
ROM:0001 C0B4 .dw 0xC0B4 ; ¦+
ROM:0002 C0B3 .dw 0xC0B3 ; ¦+
ROM:0003 C0B2 .dw 0xC0B2 ; ¦+
ROM:0004 C0B1 .dw 0xC0B1 ; ¦+
ROM:0005 C0B0 .dw 0xC0B0 ; ¦+
ROM:0006 C0AF .dw 0xC0AF ; »+
ROM:0007 C0AE .dw 0xC0AE ; «+
ROM:0008 C0AD .dw 0xC0AD ; ¡+
ROM:0009 C0AC .dw 0xC0AC ; ¼+
ROM:000A C0AB .dw 0xC0AB ; ½+
ROM:000B C0AA .dw 0xC0AA ; ¬+
ROM:000C C0A9 .dw 0xC0A9 ; ¬+
ROM:000D C0A8 .dw 0xC0A8 ; ¿+
ROM:000E C0A7 .dw 0xC0A7 ; º+
ROM:000F C0A6 .dw 0xC0A6 ; ª+
ROM:0010 C0A5 .dw 0xC0A5 ; Ñ+
ROM:0011 C0A4 .dw 0xC0A4 ; ñ+
ROM:0012 C0A3 .dw 0xC0A3 ; ú+
ROM:0013 C0A2 .dw 0xC0A2 ; ó+
ROM:0014 C0A1 .dw 0xC0A1 ; í+
ROM:0015 C0A0 .dw 0xC0A0 ; á+
ROM:0016 C09F .dw 0xC09F ; ▪+
ROM:0017 C09E .dw 0xC09E ; P+
ROM:0018 C09D .dw 0xC09D ; ¥+
ROM:0019 C09C .dw 0xC09C ; £+
ROM:001A 616E .dw 0x616E ; na
ROM:001B 006E .dw 0x6E ; n
ROM:001C 6E69 .dw 0x6E69 ; in
ROM:001D 0066 .dw 0x66 ; f
ROM:001E 4000 .dw 0x4000 ; @
ROM:001F 107A .dw 0x107A ; z
ROM:0020 5AF3 .dw 0x5AF3 ; =Z
00000000 00000000: ROM:0000 (Synchronized with Hex View-1)
Output window
The initial autoanalysis has been finished.
Python>
Python
AU: idle Down Disk: 6GB

"Processor Agnostic" Structure

- Write scripts to minimalize processor-specific attributes
- Regular expressions for architecture-specific syntax/information
- Get processor via IDA API using (Thanks to Tamir Bahar @tmr232!)

```
idaapi.get_inf_structure().procName
```

define_code_functions.py

```
################### USER DEFINED VALUES ###################
# Enter a regular expression for how this architecture usually
# begins and ends functions. If the architecture does not
# dictate how to start or end a function use r".*" to allow
# for any instruction.
#
processor_name = idaapi.get_inf_structure().procName

if processor_name == '8051':      # 8051 Architecture Prologue and Epilogue
        smart_prolog = re.compile(r".*")
        smart_epilog = re.compile(r"reti{0,1}")
elif processor_name == 'PIC18Cxx':# PIC18 Architecture Prologue and Epilogue
        smart_prolog = re.compile(r".*")
        smart_epilog = re.compile(r"return  0")
elif processor_name == 'm32r':    # Mitsubishi M32R Architecutre Prologue and Epilogue
        smart_prolog = re.compile(r"push +lr")
        smart_epilog = re.compile(r"jmp +lr.*")
elif processor_name == 'TMS32028':# Texas Instruments TMS320C28x
        smart_prolog = re.compile(r".*")
        smart_epilog = re.compile(r"lretr")
elif processor_name == 'AVR':     # AVR
        smart_prolog = re.compile(r"push +r")
        smart_epilog = re.compile(r"reti{0,1}")
else:
        print "[define_code_functions.py] UNSUPPORTED PROCESSOR. Processor = %s is
unsupported. Exiting." % processor_name
        raise NotImplementedError('Unsupported Processor Type.')
```

How IDAPython Helps – analysis

- find_mem_accesses.py
  - identifies all memory accesses for architectures such as 8051 which use a variable to access memory (DPTR)
- data_offset_calc.py
  - find the memory address accesses and
    - 1) create a data cross-reference to the memory address
    - 2) write the value at the memory address as a comment at the instructions
    - 3) create a file with all of the accesses memory address and the instructions accessing them

```
ld      R1, @(0x4114, fp)
add3    R10, fp, 0x4147
```

```
ld      R1, @[0x80C114]
add3    R10, fp, 0x4147 ; @[0x80C147]
```

File Edit Jump Search View Options Windows Help

Library function | Data | Regular function | Unexplored | Instruction | External symbol

Functions window

Function name

- sub_2F38
- sub_2F80
- sub_3054
- sub_3098
- sub_30B0
- sub_3308
- sub_3548
- sub_3678
- sub_3710
- sub_3754
- sub_3774
- sub_37C8
- sub_3804
- sub_38F8
- sub_396C
- sub_3D34
- sub_3D54
- sub_3D64
- sub_3DB0
- sub_3FCC
- sub_4008
- sub_409C
- sub_4118
- sub_4268
- sub_4310
- sub_431C
- sub_4350
- sub_44A0
- sub_4590
- nullsub_1
- nullsub_2
- sub_4694
- sub_4780
- sub_4B78
- sub_4C8C

Line 84 of 2679

IDA View-A | Hex View-1 | Structures | Enums | Imports | Exports

```
ROM_:000039BC 60 FF                   ldi8    R0, #-1
ROM_:000039BE 50 10                   srli    R0, #0x10
ROM_:000039C0 A1 CD 41 14             ld      R1, @(0x4114, fp)
ROM_:000039C4 A2 CD 41 18             ld      R2, @(0x4118, fp)
ROM_:000039C8 01 22                   sub     R1, R2
ROM_:000039CA 12 80                   mv      R2, R0
ROM_:000039CC 01 52                   cmpu    R1, R2
ROM_:000039CE 7D 03                   bnc.s   loc_39D8
ROM_:000039D0 10 81                   mv      R0, R1
ROM_:000039D2 7F 02                   bra.s   loc_39D8
ROM_:000039D4                 ; ---------------------------------------------------------------
ROM_:000039D4
ROM_:000039D4                 loc_39D4:                               ; CODE XREF: sub_396
ROM_:000039D4 60 00 F0 00             ldi8    R0, #0 || nop
ROM_:000039D8
ROM_:000039D8                 loc_39D8:                               ; CODE XREF: sub_396
ROM_:000039D8                                                         ; sub_396C+66↑j
ROM_:000039D8 81 C8 FF FF             and3    R1, R8, #-1
ROM_:000039DC 80 C0 FF FF             and3    R0, R0, #-1
ROM_:000039E0 00 51                   cmpu    R0, R1
ROM_:000039E2 7D 06                   bnc.s   loc_39F8
ROM_:000039E4
ROM_:000039E4                 loc_39E4:                               ; CODE XREF: sub_396
ROM_:000039E4 84 AF 00 2B             add3    R4, sp, #0x2B
ROM_:000039E8 60 05                   ldi8    R0, #5
ROM_:000039EA 20 04                   stb     R0, @R4
ROM_:000039EC 61 FF F0 00             ldi8    R1, #-1 || nop
ROM_:000039F0 A1 4D 41 18             st      R1, @(0x4118, fp)
ROM_:000039F4 FF 00 00 BC             bra.l   loc_3CE4
ROM_:000039F8                 ; ---------------------------------------------------------------
ROM_:000039F8
ROM_:000039F8                 loc_39F8:                               ; CODE XREF: sub_396
ROM_:000039F8 A0 CD 41 10             ld      R0, @(0x4110, fp)
ROM_:000039FC A1 CD 41 18             ld      R1, @(0x4118, fp)
ROM_:00003A00 00 A1 F0 00             add     R0, R1 || nop
ROM_:00003A04 A0 4F 00 24             st      R0, @(0x24, sp)
ROM_:00003A08 8A AD 41 71             add3    R10, fp, #0x4171
ROM_:00003A0C 19 88 F0 00             mv      R9, R8 || nop
ROM_:00003A10 A8 2F 00 22             sth     R8, @(0x22, sp)
ROM_:00003A14 A1 BD 41 22             lduh    R1, @(0x4122, fp)
ROM_:00003A18 62 01 F0 00             ldi8    R2, #1 || nop
```

000039D8 000039D8: sub_396C:loc_39D8 (Synchronized with Hex View-1)

Output window

```
Python>
Start Addr: 81920 End Addr: 733680
[make_code_functions] Running script on 0x14000 to 0xb31f0
[define_data_as_types.py] STARTING. start_addr: 0xB31F0, end_addr: 0xBFFFF, data_size: 4
[define_data_as_types.py] Undefined all data between 0xB31F0 and 0xBFFFF
[define_data_as_types.py] Defining all data as size 4
[define_data_as_types.py] FINISHED.
```

Python

AU: idle Down Disk: 6GB

data_offset_calc.py

```
operand = GetOpnd(curr_addr, 1)
----------------------
if (offset):
    if '-' in operand :
      new_opnd = offset_var_value - int(offset[0], 16)
    else:
      new_opnd = offset_var_value + int(offset[0], 16)
    OpAlt(curr_addr, 1, new_opnd_display % new_opnd)
    result = add_dref(curr_addr, new_opnd, dr_T)
----------------------
MakeComm(curr_addr, '0x%08x' % new_opnd)
----------------------------
curr_addr = NextHead(curr_addr)
```

index of operand to get

change how the operand is displayed

create a data cross-reference

dr_T: text
dr_R: read
dr_W: write
dr_O: offset

```
ld      R1, @(0x4114, fp)
add3    R10, fp, 0x4147
```

```
ld
add3                    ; @[0x80C147]
```

How IDAPython Helps – annotate

- lable_funcs_with_no_xrefs.py
  - check for functions with no cross-references to them and annotate their function name with a “noXrefs” prefix
- identify_port_use_locations.py
  - searches all code for pin/port operations based on the defined regex for the architecture and lists all references in a text file and optionally labels each function
- identify_operand_locations.py
  - searches user-defined address range for operand matching regular expression

IDAPython Functions Used

AskAddr
AskFile
AskLong
AskYN
GetDisasm
GetFunctionAttr
GetFunctionName
GetOperandValue
GetOpnd
get_inf_structure*
MakeByte
MakeCode
MakeComm
MakeDword
MakeFunction
MakeName
MakeStr
MakeUnkn
MakeWord
Warning
OpAlt
add_dref*
NextFunction
NextHead
PrevHead
FindUnexplored
XrefsTo*
isCode(GetFlags())
Byte
Word

all can be found in the idc module except (*)

What's Next?

- ida python embedded toolkit: https://github.com/maddiestone/IDAPythonEmbeddedToolkit
- other script ideas
  - architecture independent CAN or serial identifiers
  - integrate and automate more of the triage processes
  - segment creation
  - automate device selection for scripts
  - other manners to display information
  - more robust examples and docs

Thank You!
Questions?

Maddie Stone
@maddiestone