FIDLing with decompilers

Ryan Warns & Carlos Garcia Prado
Diggers of APIs, FireEye
Agenda

- Source code analysis... without source code?
- Decompilers are hard
- IDA is hard
- FIDL to the rescue!
- Scaling things up!
Who dis?

• Ryan Warns
  • Mega-Nerd
  • Likes the Windows kernel and long walks on the beach
  • Reversed malware -> wrote malware -> reversed malware

• Carlos Garcia Prado (Carl OS)
  • Particle Physics background
  • <3 IDA, Python, Windows stuff
  • Automated > Manual
  • Around 10 years in offensive (security) roles
Other interesting work around decompilation

• Everything from @pat0is
  • [https://github.com/patois](https://github.com/patois)
• HexRaysPyTools
  • [https://github.com/igogo-x86/HexRaysPyTools](https://github.com/igogo-x86/HexRaysPyTools)
Source Code Analysis… without Source Code?
What is the problem?
Q&D Compilation concepts

• Compiling source -> binary is a **lossy** process

• The info we lose is the same information source code analysis requires
  • Structure/object sizes, members
  • Code flow harder to track
  • Relationships between data structures

• We need to do some processing to try to recover this lost information
  • Once we have it our binary analysis is close to our source code analysis
Q&D Decompilation concepts

```c
int func(int a)
{
    int result;
    if ( a == 1 )
    {
        result = 5;
    }
    else
    {
        result = 6;
    }
    return result;
}
```

IDA uses this A LOT

```c
mov eax,[ebx+0x04]
add eax,[ebx+0x08]
sub [ebx+0x0c],eax

m[ebx+12] := m[ebx+12] - (m[ebx+4] + m[ebx+8]);
```
More on (de)compilation

https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/index.html
void FUN_00400000(uint8_t *cmd_string)
{
    // Variables
    uint8_t *cmd_get_result;
    uint8_t cmd_result;
    char subcommand[1024];
    struct _BASE_T *base_t;
    struct _PROCESS_INFORMATION *pi;
    struct _STARTUPINFOA *si;
    char *http_contents;
    char *local_18;
    char *local_30;
    // Function Body
    // ...
Ain’t nobody got time for [Tab]
Enter FIDL

A sane API for IDA Pro's decompiler. Useful for malware RE and vulnerability research. [FIDL ReadTheDocs](https://fidl.readthedocs.io/en/latest/)

The phrasing here is kind of inflammatory. The Hex-Rays API has a learning curve, but the documentation has increased substantially lately (see hexrays.hpp [hex-rays.com/products/decom...)] and I find it plenty usable.

This is wonderful!

Thanks to @marc_etienne_ and other fine folks from ESET, #FIDL works in IDA 7.4 and Python 3!
All hail controlFlowinator

- Data structure representing **one function**
- Mix between CFG and decompilation
- Picture a CFG where every node is a high level code const
  - if
  - assignment
  - function call
  - return
  - etc.
FLARE IDA Decompiler Library (FIDL)
FLARE IDA Decompiler Library (FIDL)
DOT can’t keep up :’( )
Batteries included

• It contains several interesting information by default
  • Function calls
  • Local variables
  • Arguments
  • Return type

```python
import FIDL.decompiler_utils as du
c = du.controlFlowInator(co-hera(), fast=False)
< FIDL.decompiler_utils.controlFlowInator instance at 0x000001D756D21C0>
```

```python
c.1vors
Name: v4
Type name: __Int64
Size: 8
Name: v5
Type name: const __m128i *
Size: 8
Complex type: __m128i
Pointed object: const __m128i
Name: WideCharStr
Type name: __Int16[255]
Size: 512
Array type: __Int16
```
Diggers of APIs

The API is horrible and there is almost no documentation.

How did you guys figure it out?

We did a lot of "dir()" in IDA's output box.

A LOT.
I just want to do a simple thing

• Find a specific *function call* within a function
• Search for any *argument* that is a string
• Get that ASCII string

• Sounds easy right?
IDA's idea of easy ;)
Much moar

- Extracting information from a decompiled function
  - get_function_vars
  - get_function_args
  - get_all_vars_in_node
  - blowup_expression
  - find_elements_of_type
  - get_cond_from_statement
  - assigns_to_var
  - does_constrain
  - get_interesting_calls
def main():
    STRINGS_ = decrypt_strings()
    calls = du.find_all_calls_to(f_name='t_string_decryption')
    for co in calls:
        if co.args and co.args[0].type == 'number':
            temp = STRINGS_[co.args[0].val:]
            temp = temp.split('\x00',1)[0]
            du.create_comment(co.c,co.ea,'%x %s' % (co.args[0].val,temp))
Much moar

• Interacting with the decompiler view (GUI)
  • display_node
  • display_path
  • display_line_at
  • display_all_calls_to
Much moar

• Much of the basic instrumentation is built-in
  • specifically the tedious stuff

• FIDL implements **generic functionality**
  • building blocks

• You can focus on higher-level logic
  • heuristics for vulnerability research, malware analysis, etc.
  • the fun stuff 😊
A more complete example

https://fidl.readthedocs.io/en/latest/tutorial.html#a-more-complete-example
Scale it up!
Scaling up

- Initially, our workflow
  - open IDA
  - load binary
  - execute FIDL script
  - close IDA
  - repeat
Scaling up

• The Citadel
  • Web interface
  • Drag & Drop
  • Distributed processing
  • IDA Headless execution
  • Dockerz!
RICK AND MORTY

TALES FROM THE CITADEL
### Index

<table>
<thead>
<tr>
<th>Campaign</th>
<th>Progress</th>
<th>Product Name</th>
<th>Product Version</th>
<th>Date</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy Test [files]</td>
<td></td>
<td>Nothing to see here</td>
<td>1</td>
<td>2020-05-08 16:11:55.773515</td>
<td>[DELETE][ARCHIVE][RE-RUN][CLEAN-UP]</td>
</tr>
</tbody>
</table>
The Citadel::Upload

Upload Campaign

Campaign options

Campaign Name
That's Offensive!!

Product Name

OTP

Product Version
13.37

Description
Oh boy, here I go again

Upload files

Drop folder here!
The Citadel::Upload

All files uploaded!

Analysis Workflow

- Attack surface Analysis
- Decompiler-based bug hunting tools
- Assembly-based bug hunting tools
- Perform a dummy test. Debugging purposes.
- User defined 1
- User defined 2
- User defined 2

Create Campaign!
The Citadel - New Campaign

Index

<table>
<thead>
<tr>
<th>Campaign</th>
<th>Progress</th>
<th>Product Name</th>
<th>Product Version</th>
<th>Date</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy Test [files]</td>
<td></td>
<td>Nothing to see here</td>
<td>1</td>
<td>2020-05-08 16:11:55.773515</td>
<td></td>
</tr>
</tbody>
</table>
The Citadel:::Results

Campaign Results

- Campaign Name: Dummy Test
- Product: Nothing to see here
- Description: Oh boy, here I go again
- Code: 2020-05-03 14:11:06.77551
- Status: complete [nearly complete]

<table>
<thead>
<tr>
<th>Filename</th>
<th>Bug Class</th>
<th>Location</th>
<th>Bug Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcp.exe</td>
<td>Complexity</td>
<td>0x413784</td>
<td>sub_413784: 231.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x416580</td>
<td>__input: 180.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x416993</td>
<td>sub_416993: 179.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x416f30</td>
<td>sub_416f30: 171.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x417c14</td>
<td>sub_417c14: 160.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x41855d</td>
<td>__output: 114.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x4186c4</td>
<td>__stringf12: 88.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x42a03a</td>
<td>__string: 71.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x410da7</td>
<td>sub_410da7: 50.00</td>
</tr>
<tr>
<td>pcp.exe db</td>
<td>Complexity</td>
<td>0x427c01</td>
<td>sub_427c01: 59.00</td>
</tr>
</tbody>
</table>
Where do I get it?

- [https://github.com/fireeye/FIDL](https://github.com/fireeye/FIDL)
  - Online documentation! *(automatically updated)*
Decompilation is the future. I've seen it with my eyes…
Questions?
Thank You!

HIT BLOCK DOWN

livelivestream

Ryan Warns, Carlos Garcia Prado <{ryan.warns, carlos.garcia}@fireeye.com>