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?

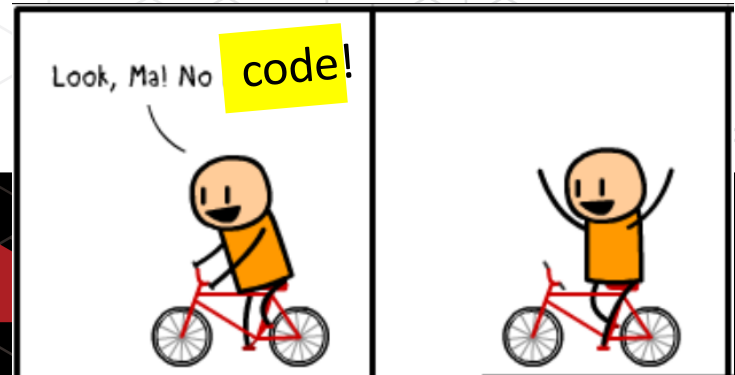
- Ryan Warns
  - Mega-Nerd
  - Likes the Windows kernel and long walks on the beach
  - Reversed malware -> wrote malware -> reversed malware

- 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
- HexRaysPyTools
  - https://github.com/igogo-x86/HexRaysPyTools

Source Code Analysis… without Source Code?



What is the problem?

Q&D Compilation concepts

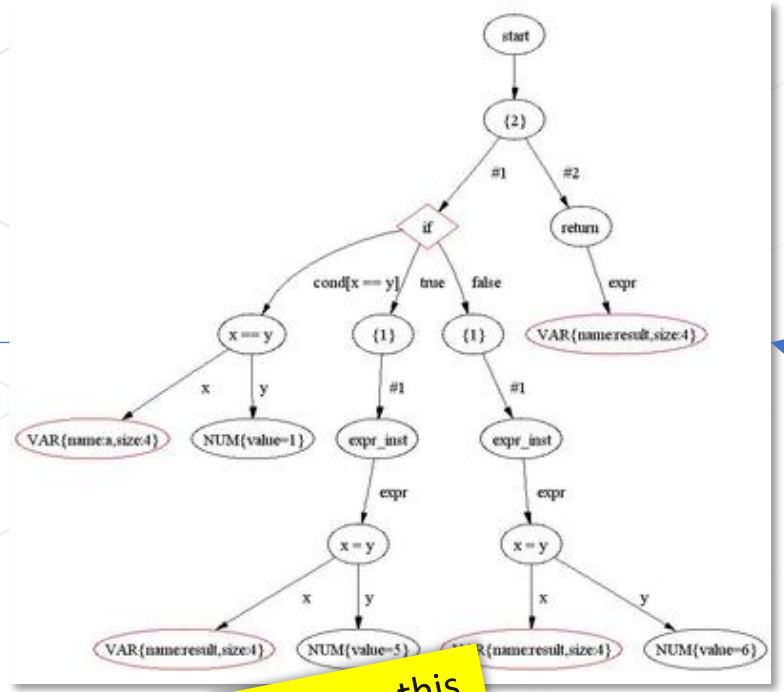
- Compiling source -> binary is a ***lossy*** process

- 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

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



IDA uses this **A LOT**

```
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

IDA Decompiler

```
_DWORD cmd_string[8]; // [rsp+60h] [rbp-1868h] BYREF
_DWORD get_results[34]; // [rsp+80h] [rbp-1848h] BYREF
char v27; // [rsp+108h] [rbp-17C0h]
struct _PROCESS_INFORMATION v28[10]; // [rsp+180h] [rbp-1748h] BYREF
struct _STARTUPINFOA v29[2]; // [rsp+280h] [rbp-1648h] BYREF
char http_contents[256]; // [rsp+380h] [rbp-1548h] BYREF
char _subcommand[1024]; // [rsp+480h] [rbp-1448h] BYREF
char v32[4136]; // [rsp+880h] [rbp-1048h] BYREF

(sub_403600)(lpThreadParameter);
do
{
  while ( 1 )
  {
    while ( 1 )
    {
      while ( 1 )
      {
        while ( 1 )
        {
          do
          {
            Sleep(dwMilliseconds);
            cgp_send_http_msg_wrap_GET(get_results, "45.66.250.110", "80", g_buffer_hello_uri);
            cmd_get_result = strtok(get_results, "&");
          }
          while ( !cmd_get_result );
          strcpy(cmd_string, cmd_get_result);
          subcommand = strtok(0i64, "&");
          if ( subcommand )
            strcpy(_subcommand, subcommand);
          if ( strcmp(cmd_string, "sleep") )
            break;
          delay_timer = atoi(_subcommand);
          if ( delay_timer > 100 )
            dwMilliseconds = delay_timer;
        }
```

```
void FUN_0040060c(void)

{
  uint uVar1;
  int iVar2;
  ulong uVar3;
  int iVar4;
  bool bVar5;
  uint local_18;
  int local_14;

  write(1,"Welcome to packedup for r2crackmes :)\nFlag << ",0x30);
  read(0,&DAT_00601080,0x2c);
  iVar2 = 0x400614;
  iVar4 = 0xe2;
  uVar3 = 0;
  do {
    uVar1 = (uint)(byte)((char)uVar3 + *(char *)(long)iVar2);
    local_18 = ((uint)uVar3 & 0xffffff00 | uVar1) >> 4 | uVar1 << 0x1
    uVar3 = (ulong)local_18;
    iVar2 = iVar2 + 1;
    iVar4 = iVar4 + -1;
  } while (iVar4 != 0);
  local_14 = 0x2c;
  do {
    bVar5 = (int)local_18 < 0;
    uVar1 = local_18 << 1;
    local_18 = uVar1 | (uint)bVar5;
    if ((uVar1 & 0xff | (uint)bVar5) !=
        (uint)(byte)((&UNK_004007a0)[(long)(local_14 + -1)] ^ (&DAT_(
                    )) {
      write(1,"Try again!\n",0xd);
      goto LAB_004006f6;
    }
    local_14 = local_14 + -1;
  } while (local_14 != 0);
  write(1,"Yep! you got the flag :) \n",0x1c);
LAB_004006f6:
                    /* WARNING: Subroutine does not return */
  exit(0);
}
```

Ain’t nobody got time for [Tab]

Enter FIDL

fireeye / FIDL

Watch

Code | Issues 1 | Pull requests 0 | Actions | Projects 0 | Security 0 | Insights

A sane API for IDA Pro's decompiler. Useful for malware RE and vulnerability research https://fidl.readthe

ida | decompiler | api | vulnerability | research | reversing | malware

29 commits | 1 branch | 0 packages | 3 releases

Branch: master | New pull request

Carl OS Updated offline documentation and Sphinx config

FIDL	Updated offline documentation and Sphinx config
.gitignore	Initial commit
.readthedocs.yml	Updated readthedocs info
LICENSE	Added MIT license
MANIFEST.in	Initial commit
README.md	README shields
README.rst	Initial commit
requirements.txt	RTD fix
setup.py	Bumped version

Ilfak Guilfanov @ilfak

Hey, our API is usable but I get your point.

> **m0n0sapiens** @m0n0sapiens · Nov 25, 2019
> What? A library that wraps IDA decompiler API and makes it usable? *and* documented?
>
> We just released a thing.
>
> Blog: fireeye.com/blog/threat-re...
>
> Github: github.com/fireeye/FIDL
>
> Docs!!!: fidl.readthedocs.io/en/latest/

12:54 AM · Nov 26, 2019 · Twitter Web App

Rolf Rolles @RolfRolles

Replying to @m0n0sapiens and @bdcht

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

11:15 AM · Nov 26, 2019 · Twitter Web App

Ilfak Guilfanov @ilfak · Dec 3, 2019

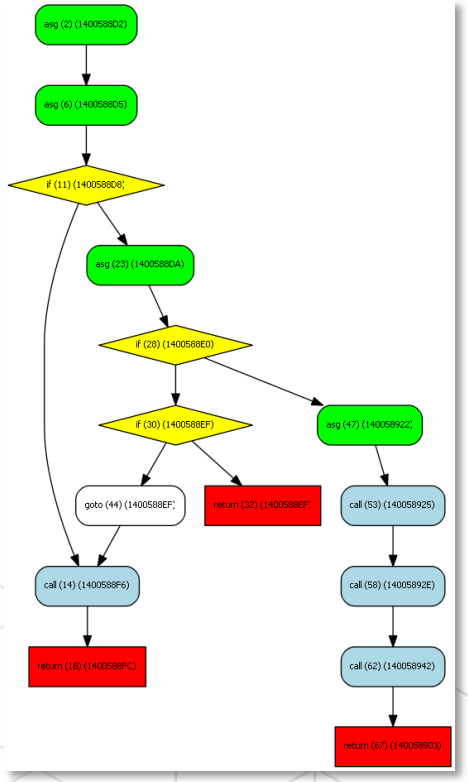
This is wonderful!

> **m0n0sapiens** @m0n0sapiens · Dec 2, 2019
> Thanks to @marc_etienne_ and other fine folks from ESET, #FIDL works in IDA 7.4 and Python 3!
>
> github.com/fireeye/FIDL
>
> Show this thread

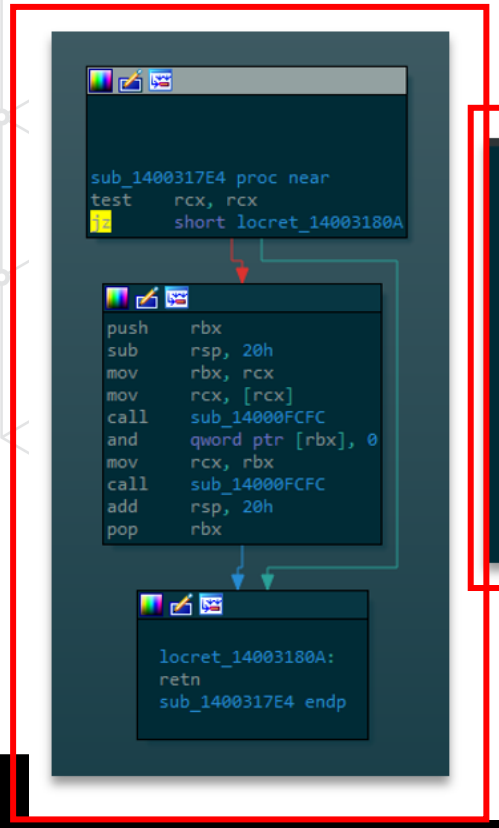
All hail controlFlowinator

- 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.

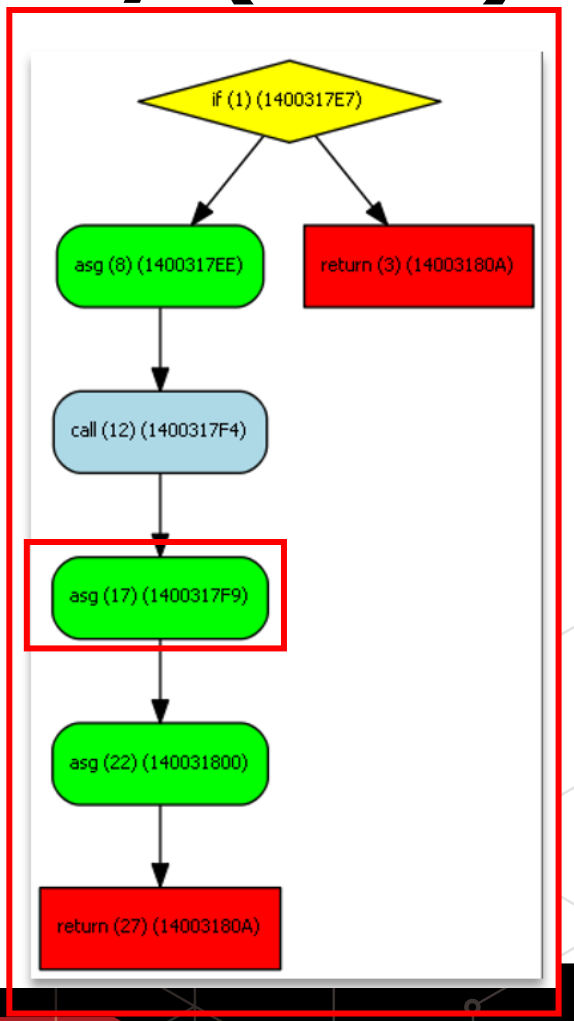
CFG: Control Flow Graph



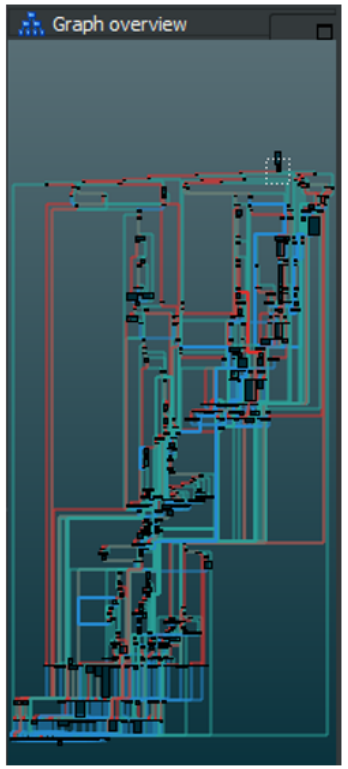
FLARE IDA Decompiler Library (FIDL)

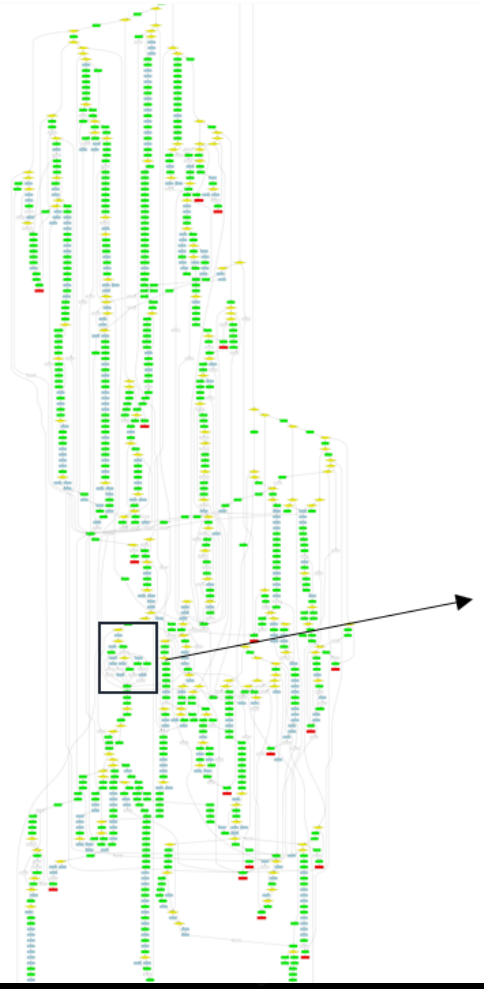


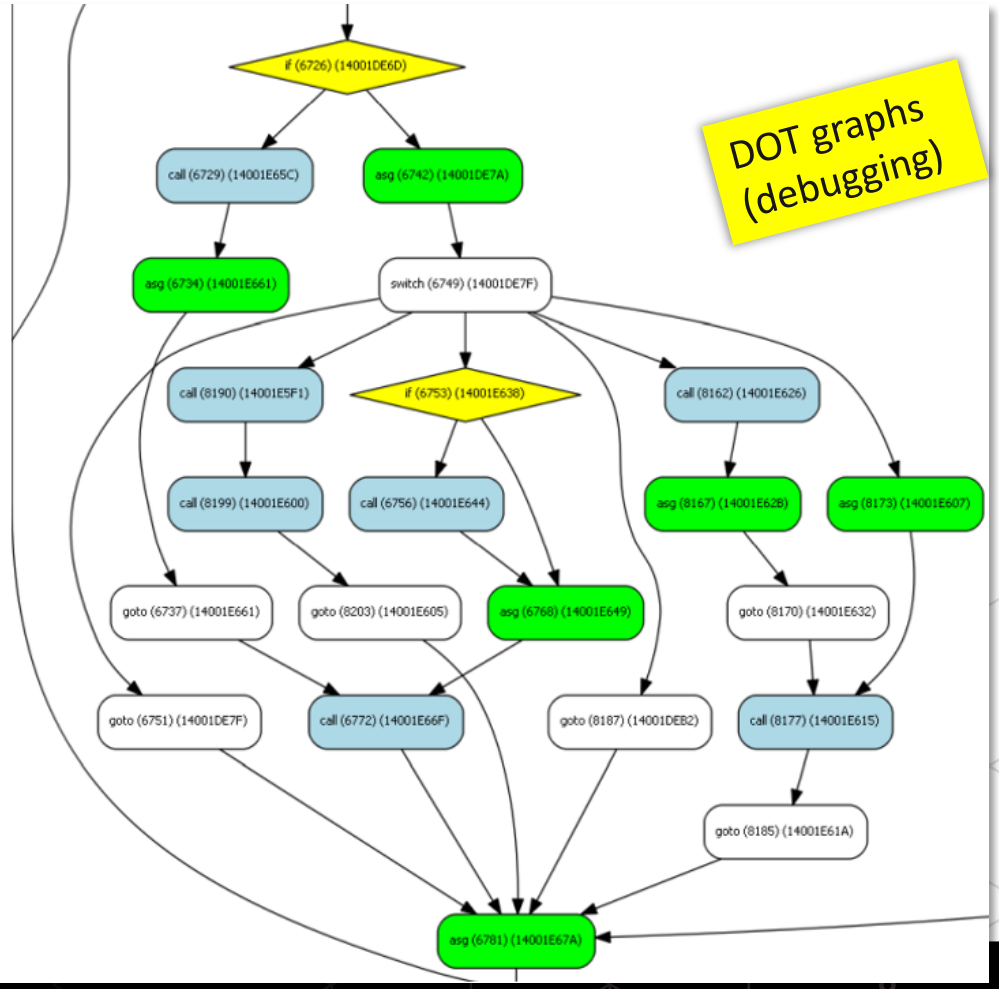

```
1 __int64 __fastcall sub_1400317E4(void **a1)
2 {
3   void **v1; // rbx
4   __int64 result; // rax
5
6   if ( !a1 )
7     return result;
8   v1 = a1;
9   sub_14000FCFC(*a1);
10  *v1 = 0i64;
11  result = sub_14000FCFC(v1);
12  return result;
13 }
```



FLARE IDA Decompiler Library (FIDL)

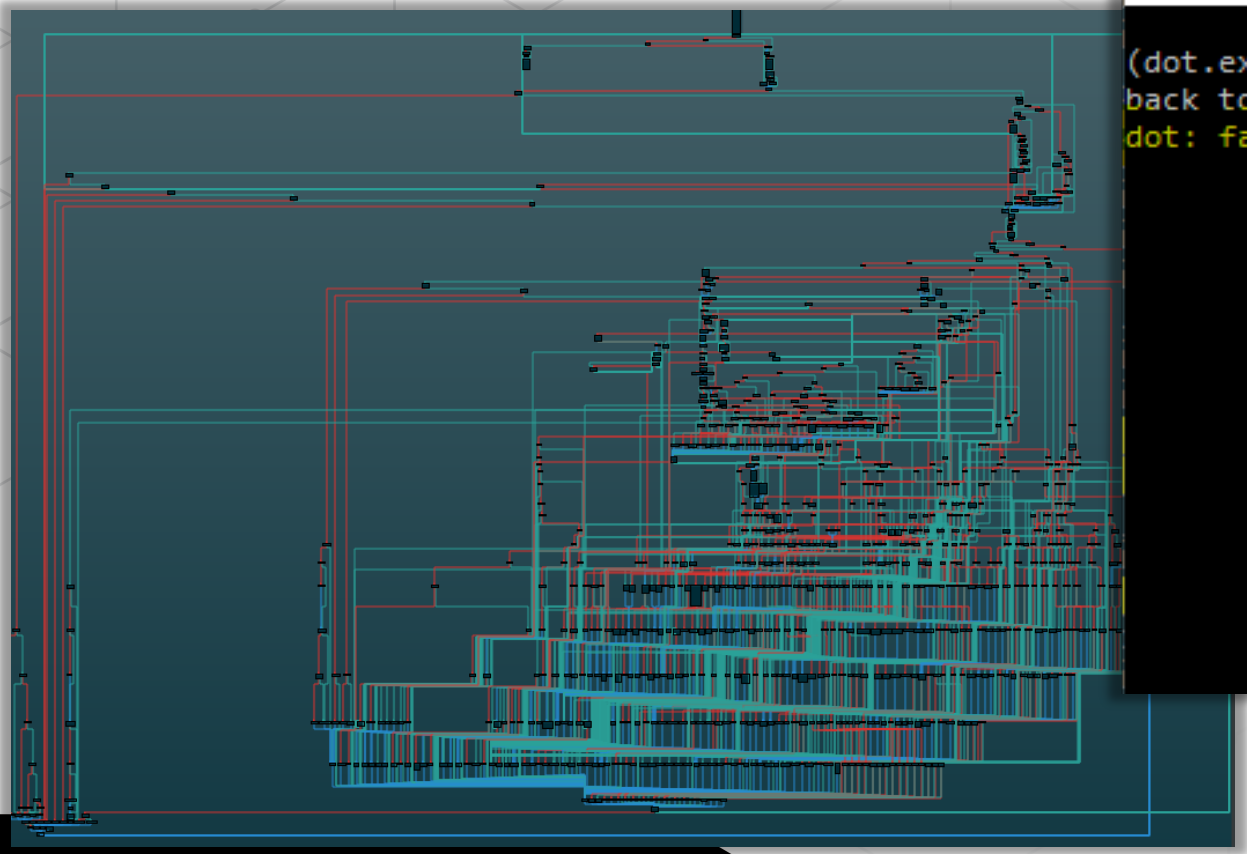






DOT graphs (debugging)

DOT can't keep up :')



C:\WINDOWS\system32\cmd.exe

```
(dot.exe:16284): Pango-WARNING **: couldn't load font "Sans serif Not-
back to "Sans Not-Rotated 8", expect ugly output.
dot: failure to create cairo surface: out of memory
```

Choose Just-In-Time Debugger

An unhandled win32 exception occurred in dot.exe [16284].

Available Debuggers:

New instance of Visual Studio Community 2017

```
Python>c
<__main__.controlFlowinator instance at 0x0000022B6628B148>
```

Python `c = controlFlowinator(ea=here(), fast=False)`

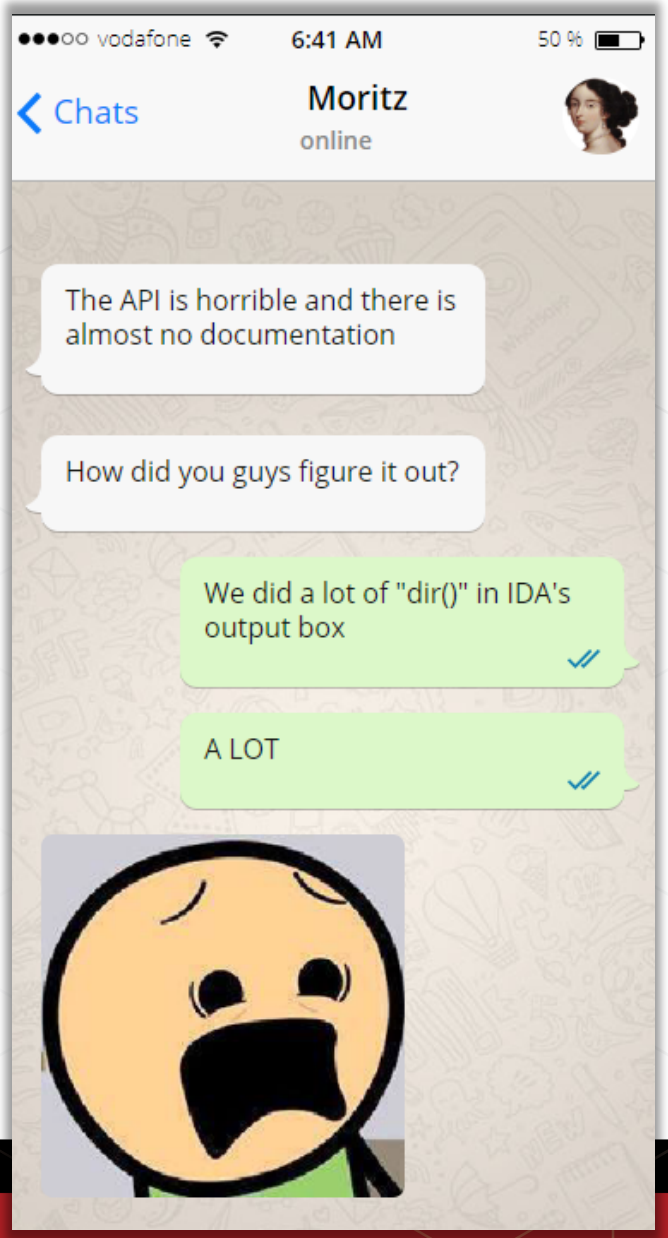
Batteries included

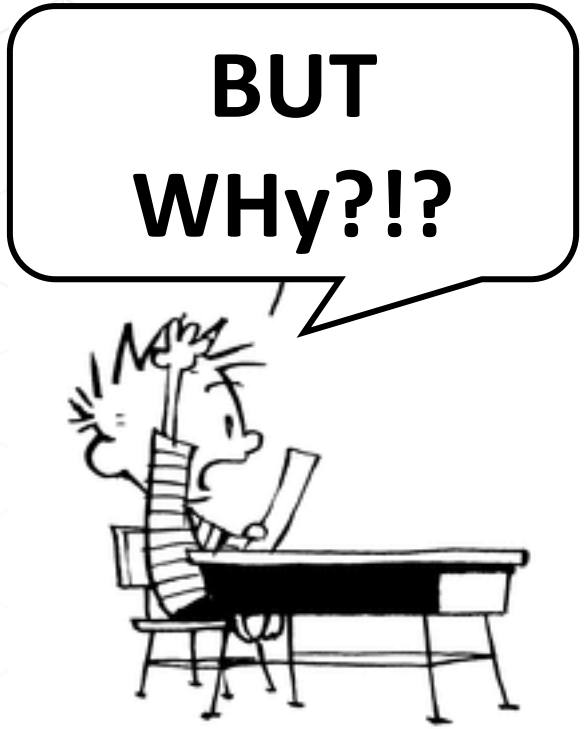
- It contains several interesting information by default
  - Function calls
  - Local variables
  - Arguments
  - Return type

```
Python>import FIDL.decompiler_utils as du
Python>c = du.controlFlowinator(ea=here(), fast=False)
Python>c
<FIDL.decompiler_utils.controlFlowinator instance at 0x000001D756DB21C8>
```

```
Python>c.lvars
Name: v4
  Type name: __int64
  Size: 8
Name: v5
  Type name: const __m128i *
  Size: 8
Complex type: __m128i
Pointed object: const __m128i
<snip...>
Name: WideCharStr
  Type name: __int16[256]
  Size: 512
Array type: __int16
```

Diggers of APIs




BUT
WHy?!?

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?

```
SetWindowTextA(v3, v7);
sub_14000FCFC(v7);
SetDlgItemTextA(
  v3,
  1002,
  "PuTTY is copyright 1997-2017 Simon Tatham.\r\n"
  "\r\n"
  "Portions copyright Robert de Bath, Joris van Rantwijk,
  "Nicolas Barry, Justin Bradford, Ben Harris, Malcolm Smi
  "ite, and CORE SDI S.A.\r\n"
  "\r\n"
  "Permission is hereby granted, free of charge, to any pe
  "tation files (the \"Software\"), to deal in the Softwar
  "s to use, copy, modify, merge, publish, distribute, sub
  "rsons to whom the Software is furnished to do so, subje
  "\r\n"
  "The above copyright notice and this permission notice s
  " Software.\r\n"
  "\r\n"
  "THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF
  "HE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICU
  "IGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  "E, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTW
return 1i64;
```

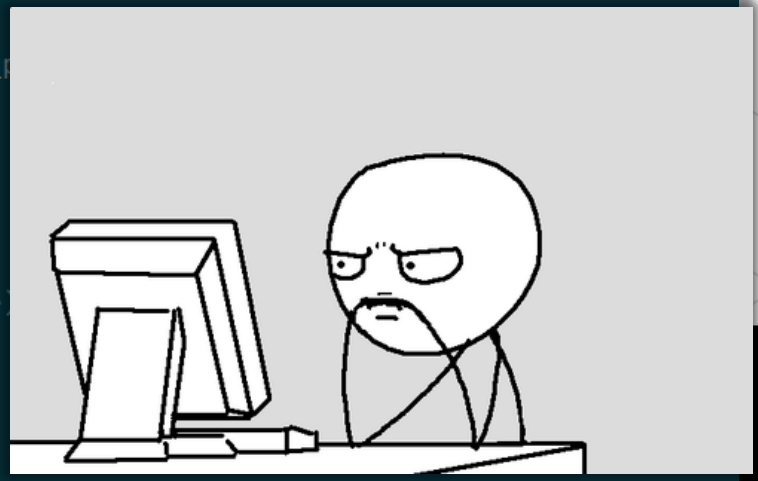
IDA's idea of easy ;)

Output window

```
Hex-rays version 7.2.0.181105 has been detected
Decompida v.0.5 "nerdilicious" loaded...

Python>c = controlFlowinator(ea=here(), fast=False)
Python>for co in c.calls:
   if co.name == 'SetDlgItemTextA':
      for arg in co.args.values():
         if arg.type == 'string':
            print arg.val
Python>
PuTTY is copyright 1997-2017 Si

Portions copyright Robert de Bath, Joris van Rantwijk, Delian
Delchev, Andreas Schultz, Jeroen Massar, Wez Furlong, Nicolas
Barry, Justin Bradford, Ben Harris, Malcolm Smith, Ahmad
Khalifa, Markus Kuhn, Colin Watson, Christopher Staite, and
CORE SDI S.A.

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
```

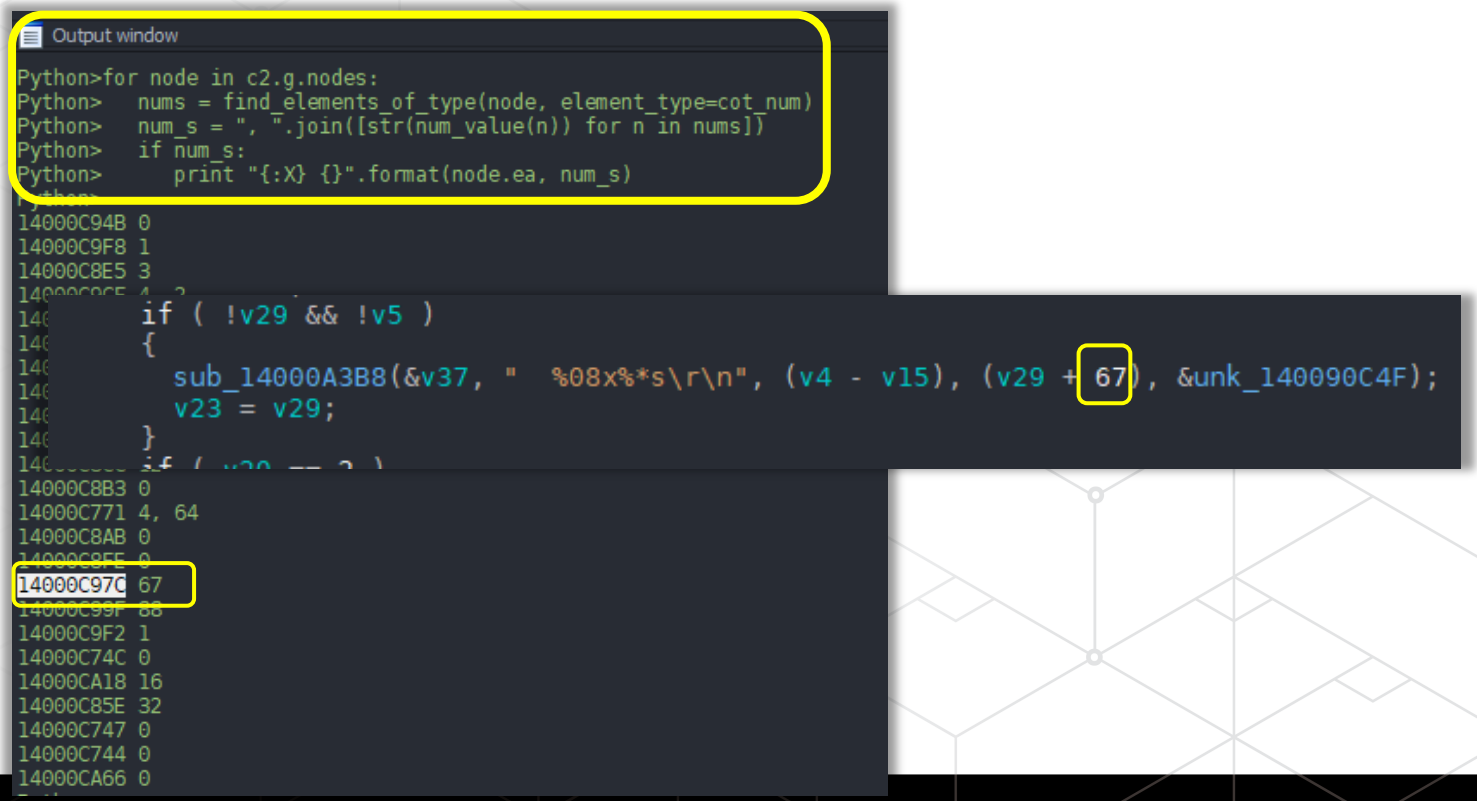
Named tuples!!!

```python
import idc
import idaapi

import ida_hexrays


class visitor(idaapi.ctree_visitor_t):
    def __init__(self, cfunc, result):
        idaapi.ctree_visitor_t.__init__(self, idaapi.CV_FAST)
        self.cfunc = cfunc
        self.result = result

    def visit_expr(self, i):
        if i.op == idaapi.cot_call:
            # look for calls to GetProcAddress
            if idc.Name(i.x.obj_ea) == "SetDlgItemTextA":

                for idx in xrange(len(i.a)):
                    # ASCSTR_C == 0
                    # Check to see if the second argument is a C string
                    if idc.GetStringType(i.a[idx].obj_ea) == 0:
                        da_string = idc.GetString(i.a[idx].obj_ea, -1, 0)
                        self.result.add(da_string)

        return 0


def main():
    if not idaapi.init_hexrays_
        return False

    result = set([])

    cfunc = decompile(here())
    v = visitor(cfunc, result)
    v.apply_to(cfunc.body, None

    print result


if __name__ == "__main__":
    main()
```



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

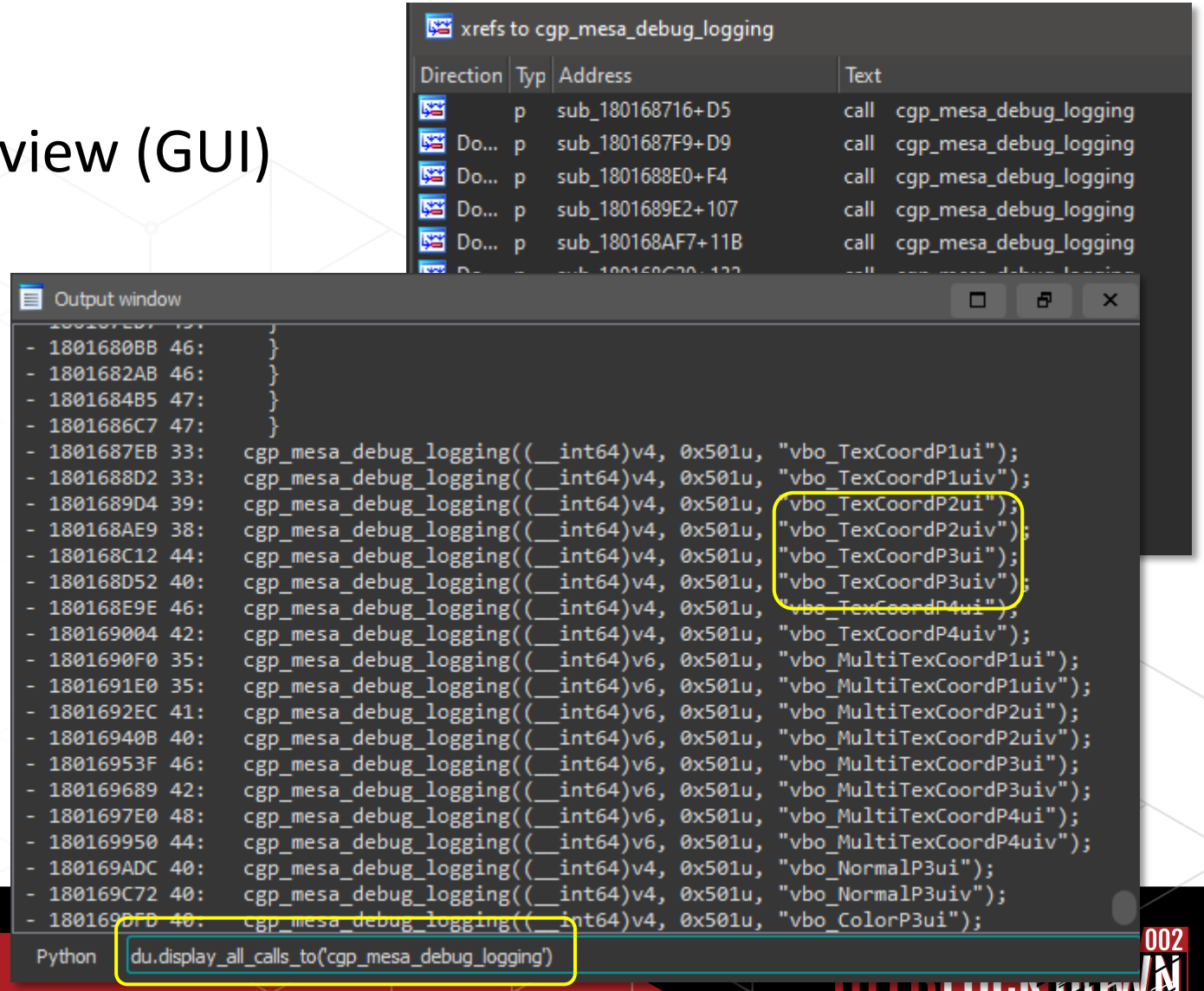


Much moar

```
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

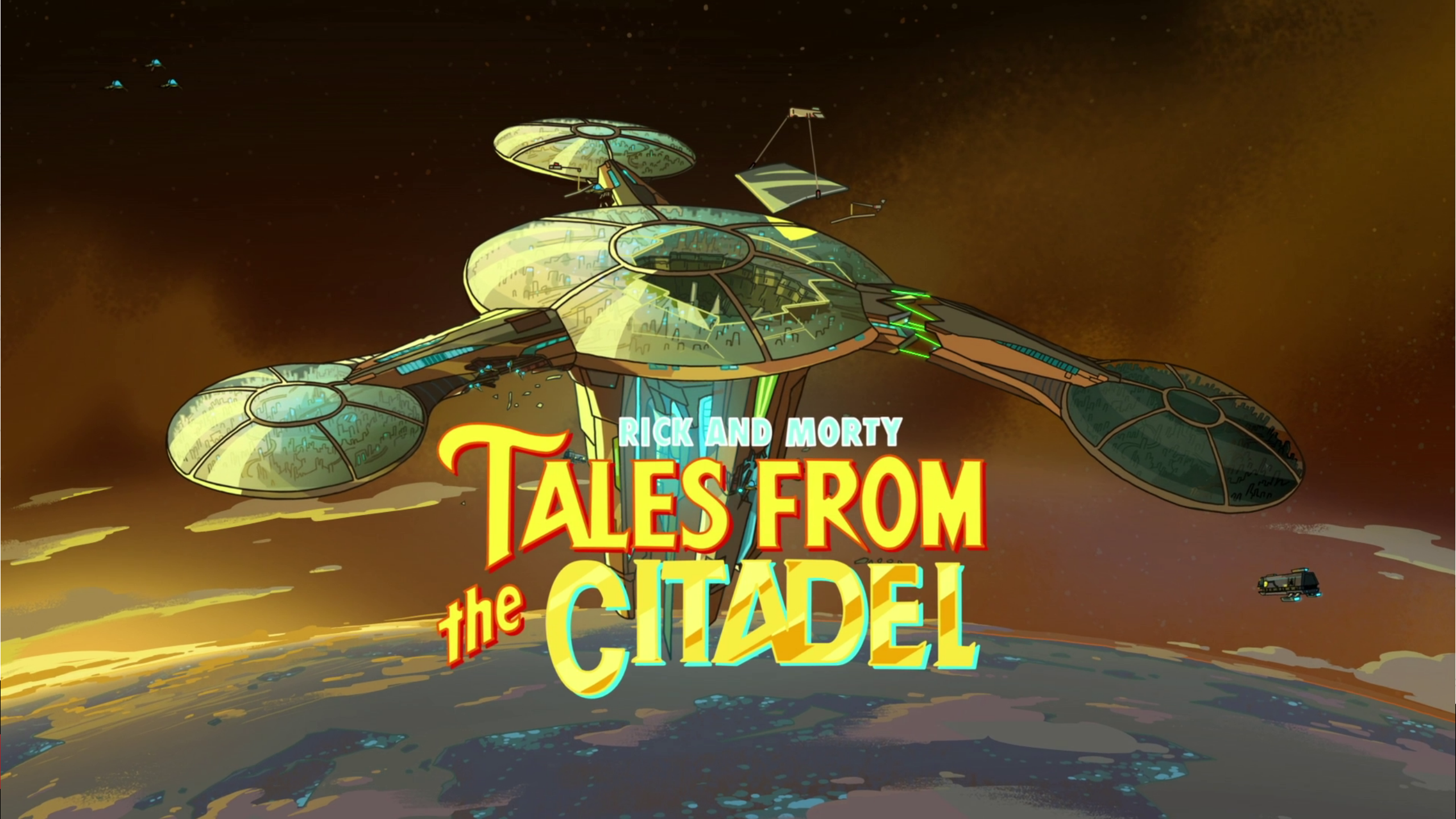

Scale it up!

Scaling up

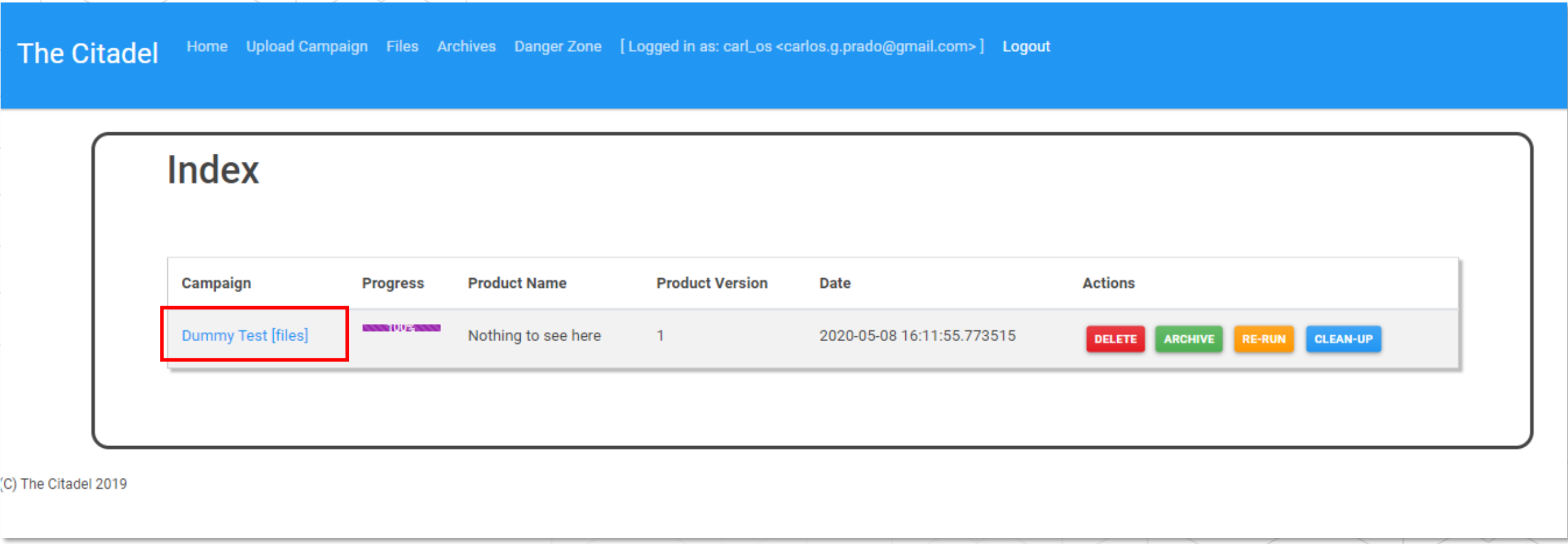
- Initially, our workflow
  - open IDA
  - load binary
  - execute FIDL script
  - close IDA
  - repeat

Scaling up

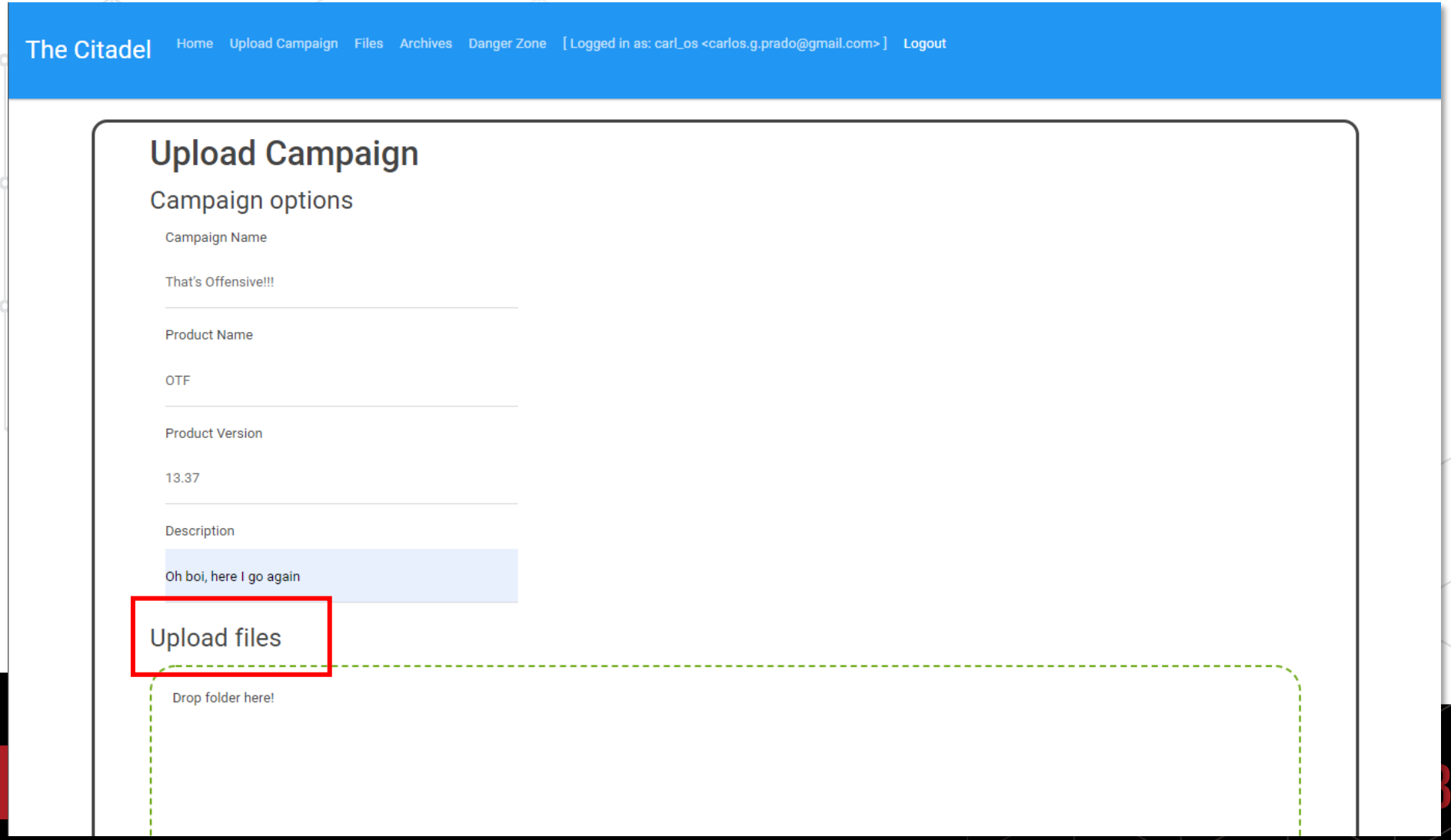
- The Citadel
  - Web interface
  - Drag & Drop
  - Distributed processing
  - IDA Headless execution
  - Dockerz!


RICK AND MORTY
TALES FROM
the CITADEL

TheCitadel::Home



TheCitadel::Upload



TheCitadel::Upload

All files uploaded!

Analysis Workflow

drag me Attack surface Analysis

drag me Decompiler-based bug hunting tools

drag me Assembly-based bug hunting tools

drag me Perform a dummy test. Debugging purposes.

drag me User defined 1

drag me User defined 2

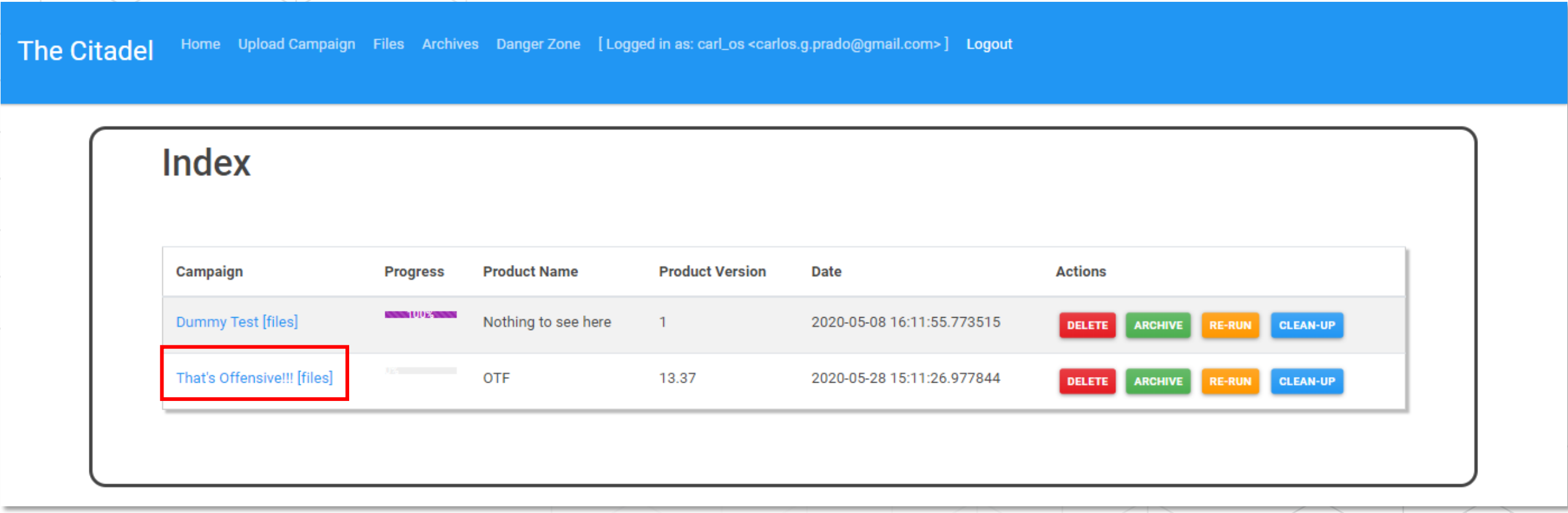
drag me User defined 2

drag me Attack surface Analysis

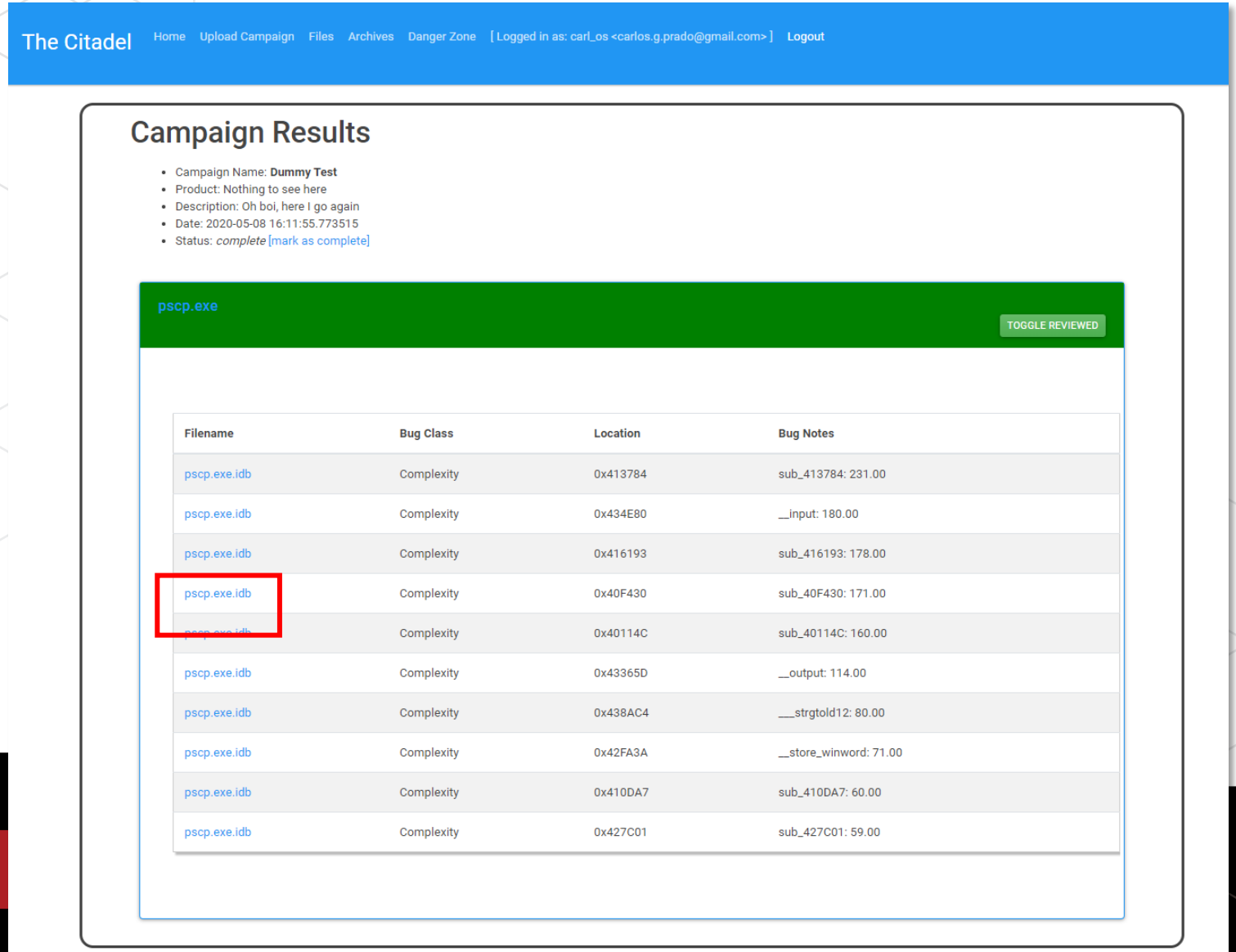
drag me Decompiler-based bug hunting tools

Create Campaign!

TheCitadel::NewCampaign



TheCitadel::Results

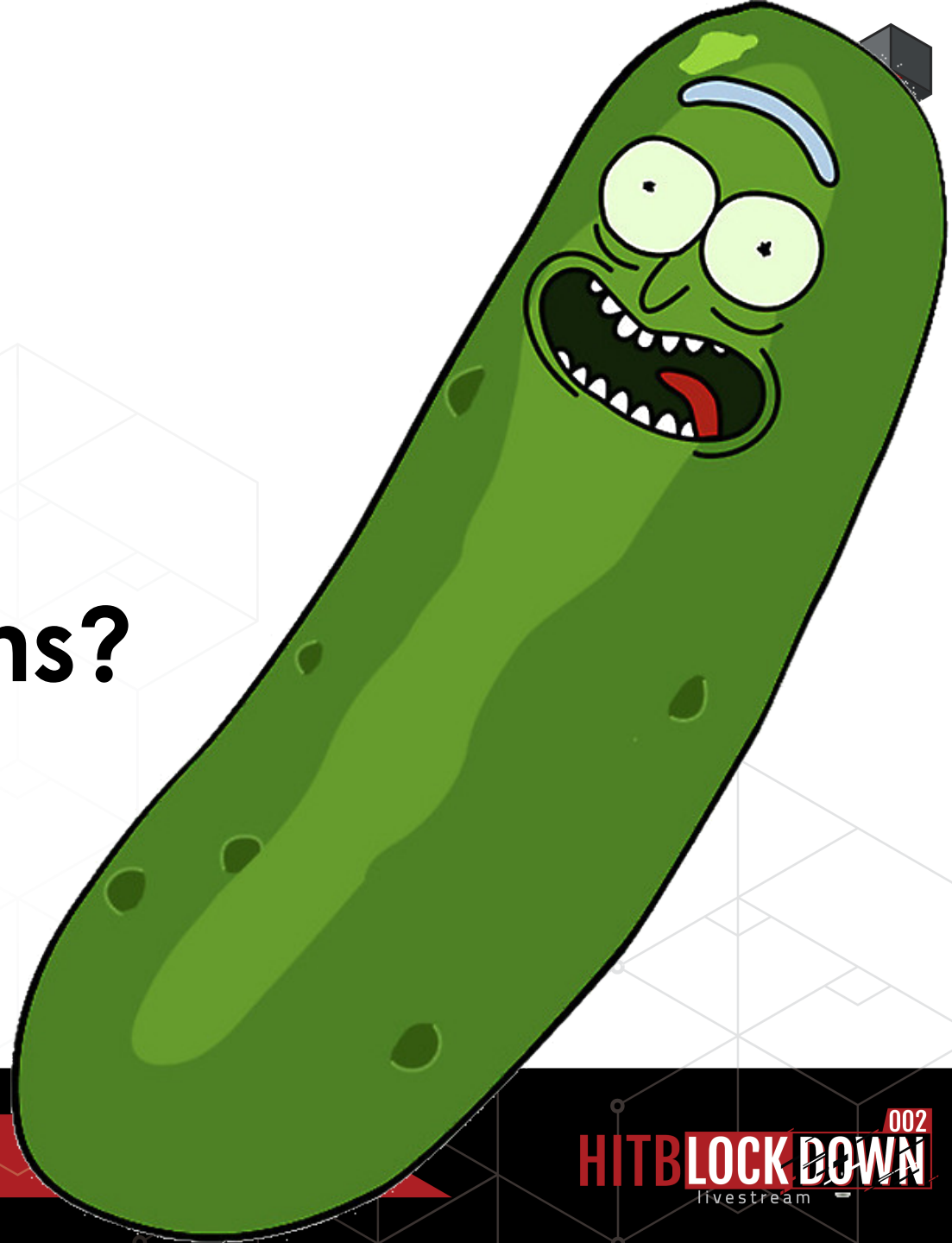


Where do I get it?

- https://github.com/fireeye/FIDL
- https://fidl.readthedocs.io/en/latest/
  - Online documentation! *(automatically updated)*

Decompilation is the future. I've seen it with my eyes…

Questions?



Thank You!

HITB LOCKDOWN 002
livestream

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