0x0000

i'm learning a lesson called patience.
can't wait 'til i have it all learned.

- “walk on water”

# fun with symboliks

symbolik analysis in pure python

# 0x0001 — who am i?

- Jesus dude
- husband
- father
- hobby farmer
- biker

# 0x0002 — who am i?

- oh, and i'm atlas 0f d00m
- re
- vr
- hw
- fw
- radio
- cars/meddevs/SmartMeters/embedded
- Vivisect/envi/symboliks
- atlas@r4780y.com

# 0x0100 — symboliks - wtfo?

- part of Vivisect, invisigoth's binary analysis framework
- Symbolic Analysis
  - based on threads of execution
- Symbolic Emulation
  - granular control of symbolic analysis
- pure python

# 0x0200 — intro to Vivisect

- binary analysis framework
- pure python
- vdb debugger
- emulators
- gooey
- symboliks
- extensible
- scalpals
- interactive python
- scripting
- client/server model collaboration
- peer-to-peer model collaboration

# 0x0201 – intro to Vivisect

- binary analysis framework
- pure python
- vdb programmatic debugger
- emulators
- gooey
- symboliks
- extensible
- scalpals
- interactive python
- scripting
- client/server model collaboration
- peer-to-peer model collaboration

# 0x0210 — intro to Vivisect (2)

- analyzing and viewing workspace

```
$ vivbin -B stage3
Failed to find file for 0x0804a1a4 (__bss_start) (and filelocal == True!)
Failed to find file for 0x0804a1a4 (_edata) (and filelocal == True!)
Loaded (0.0296 sec) stage3
ANALYSIS TIME: 0.277778863907
stats: {'functions': 67, 'relocations': 0}
Saving workspace: stage3.viv
$ vivbin stage3.viv
```

# 0x0220 — viv/stage3

do you see the vuln?

FuncGraph2' 0x080497c4

```
.text:0x080497c4
.text:0x080497c4  FUNC: int cdecl binary.childrqst( int arg0, ) [1 XREFS]
.text:0x080497c4
.text:0x080497c4  Stack Variables:
.text:0x080497c4              4: int arg0
.text:0x080497c4            -16: int local16
.text:0x080497c4          -1056: int local1056
.text:0x080497c4          -1060: int local1060
.text:0x080497c4          -1064: int local1064
.text:0x080497c4
.text:0x080497c4  55              push ebp
.text:0x080497c5  89e5            mov ebp,esp
.text:0x080497c7  81ec28040000    sub esp,1064
.text:0x080497cd  8d95e8fbffff    lea edx,dword [ebp - 1048]
.text:0x080497d3  b800040000      mov eax,1024
.text:0x080497d8  83ec04          sub esp,4
.text:0x080497db  50              push eax
.text:0x080497dc  6a00            push 0
.text:0x080497de  52              push edx
.text:0x080497df  e864f4ffff      call memset_08048c48    ;memset_08048c48()
.text:0x080497e4  83c410          add esp,16
.text:0x080497e7  c785e4fbffff0000 mov dword [ebp + local1056],0
.text:0x080497f1  83ec0c          sub esp,12
.text:0x080497f4  ff7508          push dword [ebp + arg0]
.text:0x080497f7  e850feffff      call binary.authenticate    ;binary.authenticate(arg0)
.text:0x080497fc  83c410          add esp,16
.text:0x080497ff  83ec04          sub esp,4
.text:0x08049802  6a03            push 3
.text:0x08049804  680a9c0408      push str_OK_08049c0a
.text:0x08049809  ff7508          push dword [ebp + arg0]
.text:0x0804980c  e827f3ffff      call write_08048b38     ;write_08048b38()
.text:0x08049811  83c410          add esp,16
.text:0x08049814  83ec04          sub esp,4
.text:0x08049817  68ff070000      push 2047
.text:0x0804981c  6800a20408      push binary.input_buffer
.text:0x08049821  ff7508          push dword [ebp + arg0]
.text:0x08049824  e88ff3ffff      call read_08048bb8      ;read_08048bb8()
.text:0x08049829  83c410          add esp,16
.text:0x0804982c  8945f4          mov dword [ebp + local16],eax
.text:0x0804982f  83ec04          sub esp,4
.text:0x08049832  8d85e8fbffff    lea eax,dword [ebp - 1048]
.text:0x08049838  50              push eax
.text:0x08049839  680e9c0408      push str_bacon:%s_08049c0e
.text:0x0804983e  6800a20408      push binary.input_buffer
.text:0x08049843  e8d0f3ffff      call sscanf_08048c18    ;sscanf_08048c18()
```

# 0x0230 — viv/stage3 vuln

- look again...

FuncGraph2: 0x080497c4

```
.text:0x080497c4
.text:0x080497c4  FUNC: int cdecl binary.childrqst( int arg0, ) [1 XREFS]
.text:0x080497c4
.text:0x080497c4  Stack Variables:
.text:0x080497c4            4: int arg0
.text:0x080497c4          -16: int local16
.text:0x080497c4        -1056: int local1056
.text:0x080497c4        -1060: int local1060
.text:0x080497c4        -1064: int local1064
.text:0x080497c4
.text:0x080497c4  55              push ebp
.text:0x080497c5  89e5            mov ebp,esp
.text:0x080497c7  81ec28040000    sub esp,1064
.text:0x080497cd  8d95e8fbffff    lea edx,dword [ebp - 1048]
.text:0x080497d3  b800040000      mov eax,1024
.text:0x080497d8  83ec04          sub esp,4
.text:0x080497db  50              push eax
.text:0x080497dc  6a00            push 0
.text:0x080497de  52              push edx
.text:0x080497df  e864f4ffff      call memset_08048c48    ;memset_08048c48()
.text:0x080497e4  83c410          add esp,16
.text:0x080497e7  c785e4fbffff0000 mov dword [ebp + local1056],0
.text:0x080497f1  83ec0c          sub esp,12
.text:0x080497f4  ff7508          push dword [ebp + arg0]
.text:0x080497f7  e850feffff      call binary.authenticate    ;binary.authenticate(arg0)
.text:0x080497fc  83c410          add esp,16
.text:0x080497ff  83ec04          sub esp,4
.text:0x08049802  6a03            push 3
.text:0x08049804  680a9c0408      push str_OK_08049c0a
.text:0x08049809  ff7508          push dword [ebp + arg0]
.text:0x0804980c  e827f3ffff      call write_08048b38     ;write_08048b38()
.text:0x08049811  83c410          add esp,16
.text:0x08049814  83ec04          sub esp,4
.text:0x08049817  68ff070000      push 2047
.text:0x0804981c  6800a20408      push binary.input_buffer
.text:0x08049821  ff7508          push dword [ebp + arg0]
.text:0x08049824  e88ff3ffff      call read_08048bb8      ;read_08048bb8()
.text:0x08049829  83c410          add esp,16
.text:0x0804982c  8945f4          mov dword [ebp + local16],eax
.text:0x0804982f  83ec04          sub esp,4
.text:0x08049832  8d85e8fbffff    lea eax,dword [ebp - 1048]
.text:0x08049838  50              push eax
.text:0x08049839  680e9c0408      push str_bacon:%s_08049c0e
.text:0x0804983e  6800a20408      push binary.input_buffer
.text:0x08049843  e8d0f3ffff      call sscanf_08048c18    ;sscanf_08048c18()
```

# 0x0210 – intro to Vivisect (2)

- pure python

```
$ ipython
In [1]: import vivisect.cli as vivcli
In [2]: vw = vivcli.VivCli()
In [3]: vw.loadFromFile('stage3')
Failed to find file for 0x0804a1a4 (__bss_start) (and filelocal == True!)
Failed to find file for 0x0804a1a4 (_edata) (and filelocal == True!)
Out[3]: 'stage3'
In [4]: vw.analyze()
In [5]: vw.saveWorkspace()
```

# 0x0300 – intro to Symboliks

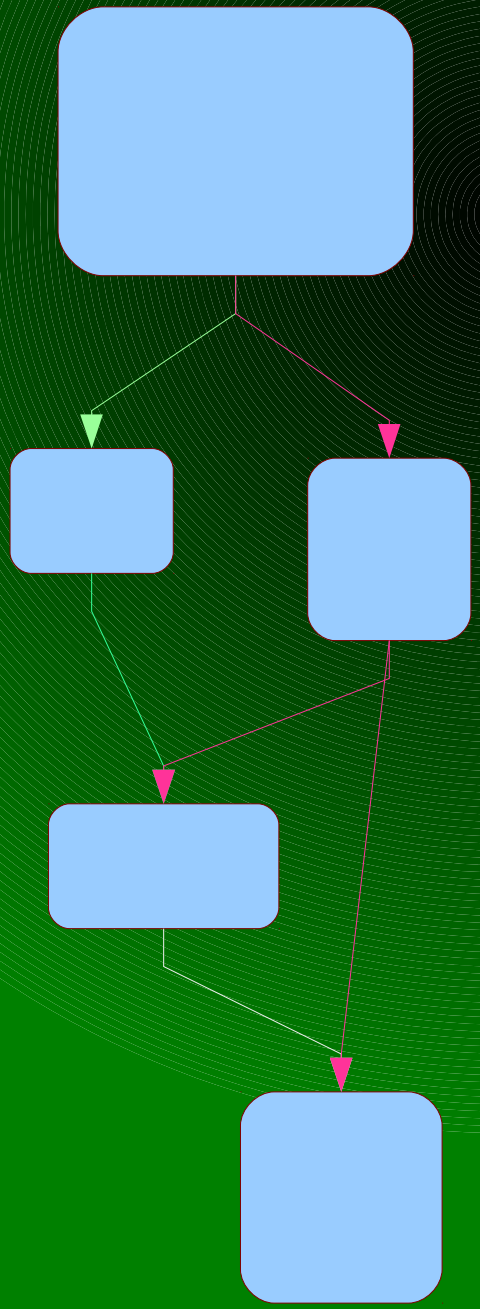
- ENVI disassembler, emulator, symboliks
- drag 'symbolic info' through emulation of each opcode
- at each point, 'symbolic state' in terms of start of trace
- eg:

```
push ebp
mov ebp, esp
    becomes:
esp = 0xbfbfeffc
[ 0xbfbfeffc : 4 ] = ebp
ebp = 0xbfbfeffc
```

# 0x0400 — intro to Graph Theory

- “Your graph just shit on my theory!”
- imagine code blocks as nodes in a **directed graph**
  - connected by directed edges
- using traditional graph theory, paths (threads) of execution can be generated
  - using symboliks, provably impossible paths are culled
- please hold for gratuitous visual ugliness

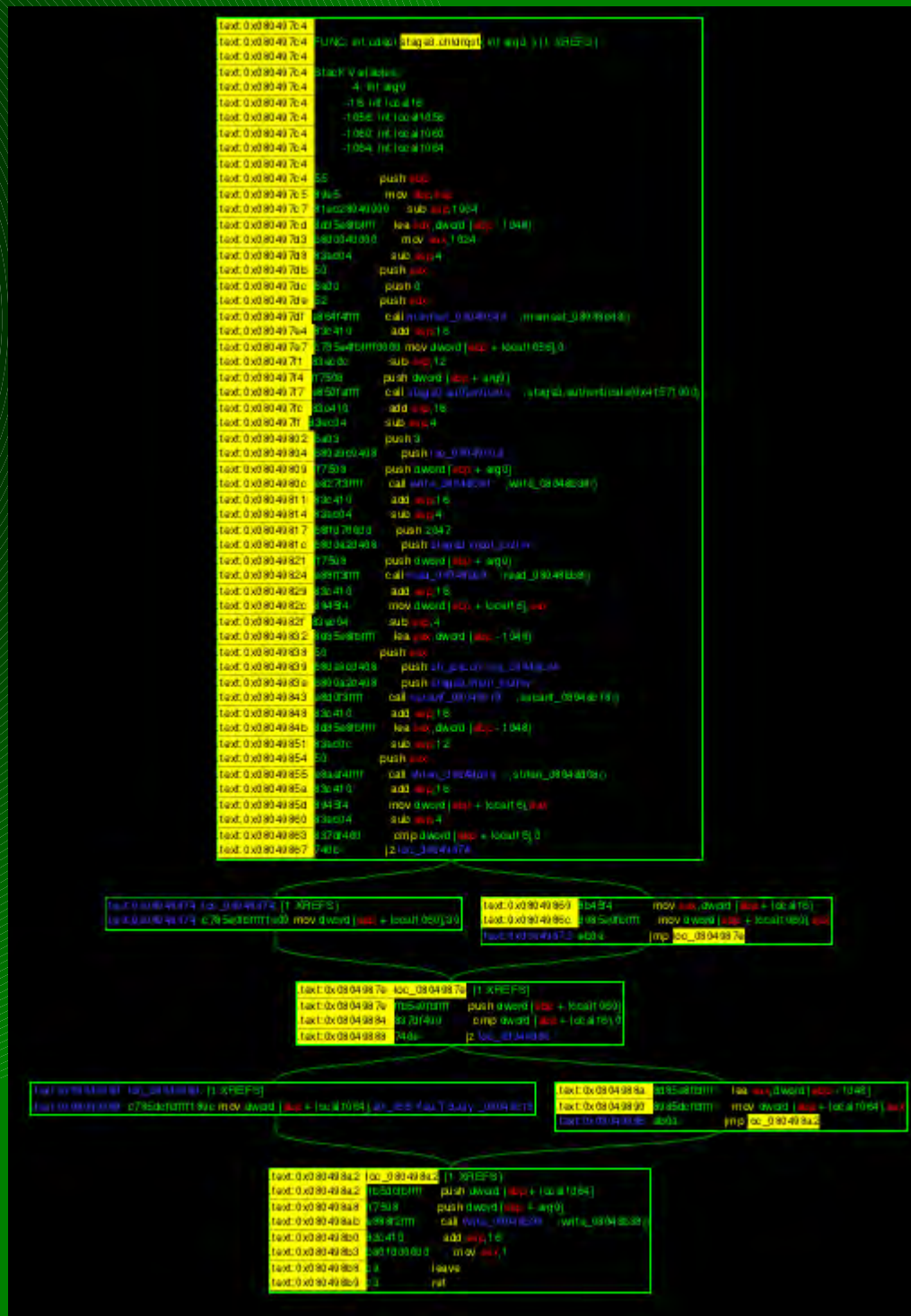
# 0x0410 — Graph Theory primer



- this is a simple function
- look familiar?
- Pathing starts at some point in the graph, and follows edges in the proper direction
- much to this, but simple for now
  - looping and the halting problem

# 0x0420 — Graph Theory

- what you can't see is the childrqst() handler from stage3
- in most cases, Viv's and IDA's graph view represent a code graph... but not always
  - calls aren't followed
  - conditional instructions
    - ARM
  - cmpxchg
  - cmov*



# 0x0500 – basics of symboliks

- symbolik state tracking and expressions
  - edi + 5
  - Mem((esp-4)+0x1500, 4)
- simple symbolik effects
  - ReadMemory( (esp-4)+0x1500, 4 )
  - WriteMemory( (esp-4)+0x1500, 4, Var(ebx, 4) )
  - SetVariable( eax, Const(4, 4) )
- symbolik constraints
  - ConstrainPath( va, nextva, ne( Var('eax'), Const(4, 4), 4))

# 0x0510 — basics of symboliks (pretty)

- verbose (repr):

```
ConstrainPath( 0x08049867, Const(0x08049869,4),
ne(Call(Const(0x08048d08,4),4, argsyms=[]),Const(0x00000000,4)) )

ConstrainPath( 0x08049888, Const(0x0804988a,4),
ne(Call(Const(0x08048d08,4),4, argsyms=[]),Const(0x00000000,4)) )
```

- pretty (str)

```
if (0x08048d08() != 0)
if (0x08048d08() != 0)
```

# 0x0520 — basics of symboliks (pretty)

- verbose (repr):

```
 SetVariable(0x080498b3, 'eax', Const(0x00000001,4))
 SetVariable(0x080498b8, 'esp', o_sub(Const(0xbfbff000,4),Const(0x00000004,4),4))
 SetVariable(0x080498b8, 'ebp', Var("ebp", width=4))
 SetVariable(0x080498b8, 'esp',
o_add(o_sub(Const(0xbfbff000,4),Const(0x00000004,4),4),Const(0x00000004,4),4))
 SetVariable(0x080498b9, 'eip',
Mem(o_add(o_sub(Const(0xbfbff000,4),Const(0x00000004,4),4),Const(0x00000004,4),4)
, Const(0x00000004,4)))
 SetVariable(0x080498b9, 'esp',
o_add(o_add(o_sub(Const(0xbfbff000,4),Const(0x00000004,4),4),Const(0x00000004,4),
4),Const(0x00000004,4),4))
```

- pretty (str)

```
 eax = 1
 esp = (0xbfbff000 - 4)
 ebp = ebp
 esp = ((0xbfbff000 - 4) + 4)
 eip = mem[((0xbfbff000 - 4) + 4):4]
 esp = (((0xbfbff000 - 4) + 4) + 4)
```

# 0x0530 — symbolik effects (simple/applied)

- simple effects:

```
esp = (esp - 4)
[ esp : 4 ] = ebp'
ebp = esp'
esp = (esp - 1064)'
edx = (ebp - 1048)'
eax = 1024'
```

- applied effects (run through SymbolikEmulator)

```
esp = (esp - 4)
[ (esp - 4) : 4 ] = ebp
ebp = (esp - 4)
esp = ((esp - 4) - 1064)
edx = ((esp - 4) - 1048)
eax = 1024
```

# 0x0540 – symboliks explained

- disassemble an opcode op = vw.parseOpcode( va )
- translate opcode into “Simple Effects”: xlater.translateOpcode(op)
- run simple effects through emu: apleffs = emu.applyEffects( xlater.getEffects() )
- apleffs now is a list of “Applied Effects”
- emu now has updated state for memory and symbolik variables that have been effected
- emu and apleffs are now both chocked full of data to be analyzed
- basically arch independent (except symbolik variable names)

# 0x0548 – symboliks explained

- python classes
  - with children
    - think RPN: o_add( Var('ebx', 4), Const(15, 4), 4)
    - random 4's are “width” data
- primitives: (subclasses of SymbolikBase)
  - Const
  - Var
  - Mem
  - Call
  - Arg
  - cnot
  - Operator

# 0x0550 — symboliks explained

- Operator (added to symbolik state through python magic)
  - o_add  applied using SymbolikBase.__add__() and .__iadd__()
  - o_sub  ...
  - o_xor
  - o_and
  - o_or
  - o_mul
  - o_div
  - o_mod
  - o_lshift
  - o_rshift
  - o_pow
  - o_sextend

# 0x0560 — symboliks explained

- Effects – subclasses of SymbolikEffect
  - SetVariable
  - ReadMemory
  - WriteMemory
  - CallFunction
  - ConstrainPath
- Constraints – subclasses of Constraint
  - eq
  - ne
  - gt
  - lt
  - ge
  - le
  - UNK
  - NOTUNK

# 0x0600 — deeper into symboliks

- reduce
- solve
- update
- substitution
- reducers

# 0x0610 — deeper symboliks (reduced)

- applied effects (run through SymbolikEmulator)

```
esp = (esp - 4)
[ (esp - 4) : 4 ] = ebp
ebp = (esp - 4)
esp = ((esp - 4) - 1064)
edx = ((esp - 4) - 1048)
eax = 1024
```

- reduced applied effects ( symstate.reduce() )

```
esp = (esp - 4)
[ (esp - 4) : 4 ] = ebp
ebp = (esp - 4)
esp = (esp - 1068)
edx = (esp - 1052)
eax = 1024
```

# 0x0620 — reduced deshmooshed. so what!

- applied effects (run through SymbolikEmulator)

```
 [ ((((((((((((((((((((((((((((((esp - 4) - 1064) - 4) - 1064) - 4) - 4) -
4) - 4) + 16) - 12) - 4) + 16) - 4) - 4) - 4) - 4) + 16) - 4) - 4) - 4) -
4) + 16) - 4) - 4) - 4) - 4) + 16) - 12) - 4) : 4 ] = ((((esp - 4) - 1064)
- 4) — 1048)
    simple, right?
```

- reduced applied effects ( symstate.reduce() )

```
[ (esp - 2152) : 4 ] = (esp - 2120)
```

# 0x0630 — solve

- symbolik expressions are either discrete or not
  - symobj.isDiscrete()
- if discrete, symbolik expressions can be solved completely
  - In [50]: o_add(Const(8,4), Const(15,4), 4).solve()
  - Out[50]: 23
- if not discrete, symbolik expressions can be compared…
  - solve() walks through the expression tree and replaces each “unknown” object with some hash of it's repr()

# 0x0640 – solve

- eg: Var._solve()

```
def varsolve(name,width,emu=None):
    '''
    A helper routine which unifies the way symboliks
    "solves" ( aka, generates a repeatable entropic
    value ) for a varible by name.
    '''
    if emu != None:
        name += emu.getRandomSeed()

    md5sum = hashlib.md5(name).hexdigest()
    return long( md5sum[:width*2], 16)
```

# 0x0650 — update

- using certain emulator state and variable values
  - get new updated symbolik state
  - which can often reduce a lot easier to more actionable stuff

# 0x0660 — substitution

- many might consider this the “solve” function, where you can provide ranges and sets of inputs to a symbolik state
- vivisect.symboliks.substitution
  - sset()
  - srange()

# 0x0660 — substitution

- example: (from switchcase analysis)

```
def getRegRange(count, rname, satvals, special_vals, terminator_addr, start=0, interval=1):
    regrange = vs_sub.srange(rname, int(count), imin=start, iinc=interval)
    for reg,val in satvals.items():
        if val == 0: continue

        print vars(regrange)
        print vars(vs_sub.sset(reg, [val]))
        regrange *= vs_sub.sset(reg, [val])
        terminator_addr.append(val)

    for sreg, sval in special_vals.items():
        regrange *= vs_sub.sset(sreg, [sval])
        terminator_addr.append(sval)

    return regrange
```

# 0x0670 — reducers

# 0x0680 — easter egg: archind

- library to make symbolik state more architecture independent
  - useful for comparing functions
  - comparing arch-independent symbolik state
    - inputs
    - outputs
- more at some later date...

# 0x0700 — why do we care about this? nerd

- RE / VR ~= pattern matching
  - but
- RE / VR != pattern matching...
- RE == Identifying Behavior
- VR == Behavior Hunting

- so, we're hunting fat juicy behaviors?
  - EXACTLY

# 0x0710 — case study: rop gadgets

- ROP gadgets are specialized behaviors ending in a transfer of execution
- ROP gadgets often have unintended side effects
- Symboliks can be used to trace effects in order to identify behaviors
  - eg. Register Traversal

# 0x0720 – Register Traversal ROP

```
In [55]: vwdis(vw, findings[1].va, 2)
0x20200002:                 mov eax,ebx
0x20200004:                 ret
```

```
In [60]: print '\n'.join([str(eff) for eff in findings[1].effects[0]])
eax = ebx
eip = mem[esp:4]
esp = (esp + 4)
```

```
    # check through the setting of variables
    for reg1, symobj in variables.items():
        # only care about some of the registers (not flags)
        if reg1 not in self.main_regs:
            continue

        for reg2 in self.main_regs:
            # skip the same reg... duh
            if reg1 == reg2:
                continue

            if self.contains(emu, reg1, reg2):
                print "%x: REG-REG COPY   %s is in %s   (%s=%s)" % (va, reg2, reg1, reg1, symobj)
                flags |= COPY_REG_REG
                if reg1 == self.REG_STACK_PTR:
                    print "  and it's the stack pointer! (PIVOT)"
                    pivot.append( (reg1, reg2) )
                    flags |= PIVOT

                regreg.append( (reg1, reg2) )
                if self.contains(emu, reg2, reg1):
                    print "    and the reverse is true!  (XCHG)"
                    flags |= XCHG
```

# 0x0730 — more to think about

```
findings = analyzer.analyzeWorkspaceROP()
20200000: REG-REG COPY   esp is in eax   (eax=esp)
    and the reverse is true!  (XCHG)
20200000: REG-REG COPY   eax is in esp   (esp=(eax + 4))
  and it's the stack pointer! (PIVOT)
    and the reverse is true!  (XCHG)
.20200002: REG-REG COPY   ebx is in eax   (eax=ebx)
.2020000d: REG<-IMM eax is set to 0x47145  (0x00047145)
.2020000f: REG<-IMM eax is set to 0x47145  (0x00047145)
.20200016  REG_ANY eax is a initialized from ('ecx', 0L)   (mem[ecx:4])
20200016: REG-REG COPY   eax is in ebx   (ebx=(((((ebx & 255) << 137) | ((ebx & 255) >> (8 - 137))) |
.20200018 WRITE WHAT WHERE!  [ecx] = eax
.20200019  REG_ANY eax is a initialized from ('ecx', 0L)   (mem[ecx:4])
.2020001b  REG_ANY eax is a initialized from ('ecx', 0L)   (mem[ecx:4])
.20200023: REG<-IMM eax is set to 0x0  (0)
```

# 0x0740 — case study: switch case analysis

- how do we tell the computer to do what we do in our magical portable computer^H^H^H^H^H^H^H^Hbrain
  - start at JMP REG
  - backup just enough to figure out the index register and any base register (which points to start of module)
  - now, backup to the start of function
    - trace through to the JMP REG
    - look through effects for constraints/o_sub to index register
      - bounding the valid indexes for this switchcase component
    - identify the symbolik state of REG (from JMP REG)
    - use substitution to ratchet through valid indexes to see where each index jmps to
    - wrack and stack

# 0x0750 — case study: 0-day

- wide wide wide wide array of options
  - much opportunity for the enterprising young soul
- two primary appoaches to symbolic bug hunting:
  - targeted
    - more efficient
    - more coding for more edge cases
  - directed bruting
    - less efficient
    - easier to code the checks
- how might we identify the vuln from stage3?

# 0x0760 — case study: viv/stage3 vuln

- look again...

FuncGraph2' 0x080497c4

```
.text:0x080497c4
.text:0x080497c4  FUNC: int cdecl binary.childrqst( int arg0, ) [1 XREFS]
.text:0x080497c4
.text:0x080497c4  Stack Variables:
.text:0x080497c4            4: int arg0
.text:0x080497c4          -16: int local16
.text:0x080497c4        -1056: int local1056
.text:0x080497c4        -1060: int local1060
.text:0x080497c4        -1064: int local1064
.text:0x080497c4
.text:0x080497c4  55              push ebp
.text:0x080497c5  89e5            mov ebp,esp
.text:0x080497c7  81ec28040000    sub esp,1064
.text:0x080497cd  8d95e8fbffff    lea edx,dword [ebp - 1048]
.text:0x080497d3  b800040000      mov eax,1024
.text:0x080497d8  83ec04          sub esp,4
.text:0x080497db  50              push eax
.text:0x080497dc  6a00            push 0
.text:0x080497de  52              push edx
.text:0x080497df  e864f4ffff      call memset_08048c48    ;memset_08048c48()
.text:0x080497e4  83c410          add esp,16
.text:0x080497e7  c785e4fbffff0000 mov dword [ebp + local1056],0
.text:0x080497f1  83ec0c          sub esp,12
.text:0x080497f4  ff7508          push dword [ebp + arg0]
.text:0x080497f7  e850feffff      call binary.authenticate    ;binary.authenticate(arg0)
.text:0x080497fc  83c410          add esp,16
.text:0x080497ff  83ec04          sub esp,4
.text:0x08049802  6a03            push 3
.text:0x08049804  680a9c0408      push str_OK_08049c0a
.text:0x08049809  ff7508          push dword [ebp + arg0]
.text:0x0804980c  e827f3ffff      call write_08048b38     ;write_08048b38()
.text:0x08049811  83c410          add esp,16
.text:0x08049814  83ec04          sub esp,4
.text:0x08049817  68ff070000      push 2047
.text:0x0804981c  6800a20408      push binary.input_buffer
.text:0x08049821  ff7508          push dword [ebp + arg0]
.text:0x08049824  e88ff3ffff      call read_08048bb8      ;read_08048bb8()
.text:0x08049829  83c410          add esp,16
.text:0x0804982c  8945f4          mov dword [ebp + local16],eax
.text:0x0804982f  83ec04          sub esp,4
.text:0x08049832  8d85e8fbffff    lea eax,dword [ebp - 1048]
.text:0x08049838  50              push eax
.text:0x08049839  680e9c0408      push str_bacon:%s_08049c0e
.text:0x0804983e  6800a20408      push binary.input_buffer
.text:0x08049843  e8d0f3ffff      call sscanf_08048c18    ;sscanf_08048c18()
```

# case study: 0-day

- call to read(arg0, input_buffer, 2047)
  - limits our input to 2047
  - input_buffer is big enuf
- call to sscanf(input_buffer, “bacon:%s\x00”, 0xbfbfebcc)
- 0xbfbfebe4 is 1052 bytes from the top of the stack (RET)
- 1052 – 2047 = -995

```
0x08049817: esp = 0xbfbfebcc
0x08049817: [ 0xbfbfebcc : 4 ] = 2047
0x0804981c: esp = 0xbfbfebc8
0x0804981c: [ 0xbfbfebc8 : 4 ] = stage3.input_buffer
0x08049821: [ 0xbfbff004 : 4 ]
0x08049821: esp = 0xbfbfebc4
0x08049821: [ 0xbfbfebc4 : 4 ] = arg0
0x08049824: read_08048bb8()
0x08049829: esp = 0xbfbfebd4
0x0804982c: [ 0xbfbfeff0 : 4 ] = read_08048bb8()
0x0804982f: eflags_gt = None
0x0804982f: eflags_lt = None
0x0804982f: eflags_sf = None
0x0804982f: eflags_eq = None
0x0804982f: esp = 0xbfbfebd0
0x08049832: eax = 0xbfbfebe4
0x08049838: esp = 0xbfbfebcc
0x08049838: [ 0xbfbfebcc : 4 ] = 0xbfbfebe4
0x08049839: esp = 0xbfbfebc8
0x08049839: [ 0xbfbfebc8 : 4 ] = "bacon:%s\x00"
0x0804983e: esp = 0xbfbfebc4
0x0804983e: [ 0xbfbfebc4 : 4 ] = stage3.input_buffer
0x08049843: sscanf_08048c18()
```

# case study: 0-day

- to take this approach, the following information is important:
  - buffer tracking
  - buffer and input/control limitations
  - functions which help bound these intelligently
- at the end of the day, we're trying to teach the computer to do what we do intuitively
- other approaches use more brutish efforts
- both are good, combined is better

# 0x-001 – for your playtime...

- import vivisect.cli as vivcli
- vw = vivcli.VivCli()
- vw.loadFromFile(“some_poor_bin.exe”)
- vw.verbose=1 ; vw.analyze()

or...

- vw.loadWorkspace(“some_poor_bin.exe.viv”)
- import vivisect.symboliks.analysis as vs_anal
- sctx = vs_anal.getSymbolikAnalysisContext(vw)
- graph = sctx.getSymbolikGraph(func_va)
- spaths = sctx.getSymbolikPaths(func_va)
- symemu, symeffs = spaths.next()
- symeffs  # play around with this.  inspect!  learn!  play! WIN!

# resources

- https://github.com/vivisect/vivisect
- https://github.com/atlas0fd00m/vivisect
  atlas' fork, often includes extras not yet merged