0×0000

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

FuncGraph2: 0x080497c4

do you see the vuln?

.text:0x080497c4		
.text:0x060497c4	FUNC: int cdec1	<pre>inary.chldrdat(int arg0,) [1 XREFS]</pre>
.text:0x080497c4		
.text:0x080497c4	Stack Variables:	
.text:0x080497c4	4: int arg0	
text:0x060497c4	-16: int	: locall6
.text:0x08049764	-1056: in	nt local1056
.text:0x080497c4	-1060: in	1t local1060
.text:0x080497c4	-1064: in	nt local1064
.text:0x060497c4		
.text:0x080497c4	55	push ebp
.text:0x080497c5	89e5	mov ebp,esp
.text:0x080497c7	81ec28040000	sub esp,1064
.text:0x060497cd	8d95e8fbffff	lea edx,dword [ebp - 1048]
.cexc:0x080497d3	b800040000	mov eax,1024
.text:0x080497d8	83ec04	sub esp,4
.text:0x080497db	50	push eax
.text:0x060497dc	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	83ecOc	sub esp,12
.text:0x080497f4	ff7508	<pre>push dword [ebp + arg0]</pre>
.text:0x080497±7	e850feffff	<pre>call binary.authenticate ;binary.authenticate(arg0)</pre>
.text:0x060497fc	83c410	add esp,16
.text:0x080497ff	83ec04	sub esp,4
.text:0x08049803	6a03	push 3
.text:0x08049804	680a9c0408	push str_OK_08049c0a
.text:0x08049809	ff7508	push dword [ebp + arg0]
.cexc:0x0804980c	e827f3ffff	call write 08048b38 ;write 08048b38()
.text:0x06049611	83c410	add esp,16
.text:0x08049814	83ec04	sub esp,4
.text:0x08049817	68ff070000	push 2047
.text:0x0804981c	6800a20408	push binary.input_buffer
.text:0x06049631	ff7508	push dword [ebp + arg0]
.cext:0x08049824	e88ff3ffff	call read_08048bb8 ;read_08048bb8()
.text:0x08049829	83c410	add esp,16
.cexc:0x0804982c	8945£4	mov dword [ebp + local16], eax
.text:0x0804983f	83ec04	sub esp,4
.cext:0x08049832	8d85e8fbffff	lea eax,dword [ebp - 1048]
.text:0x06049888	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

FuncGraph2: 0x080497c4

look again...

.text:0x080497c4			
.text:0x080497c4	FUNC: int cdecl]	binary.chldrgst(int arg0,) [1 XREFS]	
.text:0x080497c4			
.text:0x080497c4	Stack Variables:		
.text:0x080497c4	4: int	4: int arg0	
.text:0x060497c4	-16: int	t locali6	
.text:0x080497c4	-1056: ii	nt local1056	
.text:0x080497c4	-1060: in	nt local1060	
.text:0x080497c4	-1064: ii	nt 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]	
.cext: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	83ecOc	sub esp,12	
.text:0x080497f4	ff7508	push dword [ebp + arg0]	
.text:0x080497±7	e850feffff	<pre>call binary.authenticate ;binary.authenticate(arg0)</pre>	
.text:0x060497fc	83c410	add esp,16	
.cexc:0x080497ff	83ec04	sub esp,4	
.text:0x08049803	6a03	push 3	
.text:0x08049804	680a9c0408	push str_OK_08049c0a	
.text:0x08049809	ff7508	push dword [ebp + arg0]	
.cexc: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	
.cext:0x08049831	ff7508	push dword [ebp + arg0]	
.text:0x08049824	e88ff3ffff	call read_08048bb8 ;read_08048bb8()	
.text:0x08049829	83c410	add esp,16	
.text:0x0804982c	894514	mov dword [ebp + local16], eax	
.text:0x0804983f	83ec04	sub esp,4	
.text:0x08049832	8d85e8fbffff	lea eax,dword [ebp - 1048]	
.text:0x08049838	50	push eax	
.text:0x08049839	680e9c0408	push str bacon; is 08049c0e	
.text:0x0804983e	6800a20408	push binary.input_buffer	
.cext: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



- 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*

Last: 0 x0 80 49 7c.4	
text: 0.x0.80.49 7c/4	FUNC: includion stag as childrent, vi) impli (1). VBEFUI(
. taxt: 0.x030.4976.4	
Lext: 0 x0 80 49 70.4	Stack Viel adjust
taxt 0x080497c4	at is not been at the
taxt.0x0.80.497c.4	-1.058 Int 100 al 0.58
Last: 0 x0 80 40 7c 4	-1 (060) (mit 1 ec al 10 e0
Last; 0 x0 80-49 76-4	-1.054 THETGE at 10164
Last: 0 x0.80 49 76/4	
1 and 0 x0 80 40 70 4	no puintes
Text: 0 x0 80 49 7c 7	1 acc290 40000 gub and 1 054
taxt: 0.x0.80.49 7e.d	dad Seatchini - Jana Judy, dwiada (1000)
Last. 0x0.8049 703	5870340000 mov mv 1024
Laxt: 0 x0 80 40 70 9	436014 aub.mit.4
Last, 0 x0 80 40 70 E	ou puin a
Last: 0.x0.80.49 78 e	22 public don
Lext: 0x030-497d7	alishtem edil manual () 104054 () mumaal () 8048643 (
text; 0:x030497e4	steht 0 add wight 6
taxt 0x080497a7	73 Switchmod 00, mov alword [estit + locult 03/e] 0
1 and 0 and 80 40 71	STAD SUB-SUB-SUB-SUB-SUB-SUB-SUB-SUB-SUB-SUB-
text 0x08040777	end and all all all and an and a second
Last: 0.x0.80.49 7tc	Citel 10 add inter 15
Lax: 0 x0 80.49 7ft	3xc)4 size mp.4
text.0x08049802	Sura punha
Text: 0x030493049304	580 2000 408 public to 0 0 0 0 0 0 0
1ext. 0 x0 80 40 80 4	A227300 dat with path eggs
Last. 0x0 80-49 81 1	steat 0 add must s
taxt; 0;e0 80-40 81-4	Kasob4 aub mg 4
.text.0x08049817	seru 70 00 0 push 2047
LaxE 0 x0 80 49 81 c	590 dat20 40 8 plastic plan at vices (plat in
1 avt. 0 x0 80 40 824	seminary of man in particle used in a self-series
Last: 0 x0 80 49 82%	32.410 301 -0.15
text 0x0804982c	(14/94) mov dward (10) + local (c) an
Last. 0x08049821	Bige04 subject.4
Last, 0:00 80 49 83 2	sassistem kaa yoo, awad (ang - 104 ii)
Love 0 40 40 40 40 40 40 40 40 40 40 40 40 4	20 pears and the second s
1 got, 0 x0 80 40 83 g	selfado 408 event component comp
1497,0303049343	ead or service of the method o
text 0x08049848	830,410 a.01 = p.16
taxt.0x03049346	Sassarum Na No, avau (d 1048)
1axt 0x08040854	FA END AND TO THE FACTOR OF TH
text: 0x08049855	elsa ar 4000 you wat an
taxt 0x0804985a	the store the second seco
.text; 0x03049350	14434 inter a word alight + local (6, 0 ali
Text, 0 (d) 30 49 850	STERIA SUB IN A
Text: 0 x0 80 40 867	Zalo 12 loc 3 control 10 control o
terrommentra na omentra II XA	EFS) text:0x03040350 bb434 mov ins.dwad [ibst + loc at (6)
La KALURA FOR A CONSIGNATION MAN	dward) war + least 050(30) .taxt.0x0e049e6c, grad Softerm
	tuvE10x010x4367.2 e684
1.0	COX 03 04 98 Ye HCC_08 04 98 Ye 1 XHE PS
too too	C 0x 03 04 92 84 93 73 (4) (1) (1) (2) (2) (2) (2) (2) (2) (2) (2) (2) (2
tai	1: 0x 03 04 03 83 74 04- 12 Vol., 11 04 10 01
THE REPORT OF MERINAL PLANEFIST	.text:0x0804988a_848566111111111110100011111111111111111111
1.1.00 c785denamit sile mev awaid [alg +	lacatrine() an use Kau Taugy Unionalized in taxt 0x08049890 ana5demmini mew awaid (amain lacatrine
	Tage of the the tage and the tage of the tage
1 mile 2 mile	analass (av. danalass () versen)
fault 0 x0	8041842 [10500000] pist dword (100 + 100 a 1064]
Land Oxf	a040aua (7518 guái dward (ing. 4 arg0)
Last: 0 x0	a) 4) aub 109 a 2111 and 100 a
test did	3141 (300 5 30 4 1 0 3 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1438.036	
Last_0 ad	a) 4) ac)

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(0x0000000,4)))

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(0x0000004,4),4))
SetVariable(0x080498b8, 'ebp', Var("ebp", width=4))
SetVariable(0x080498b8, 'esp',
o_add(o_sub(Const(0xbfbff000,4),Const(0x0000004,4),4),Const(0x0000004,4),4))
SetVariable(0x080498b9, 'eip',
Mem(o_add(o_sub(Const(0xbfbff000,4),Const(0x0000004,4),4),Const(0x0000004,4),4),
, Const(0x0000004,4)))
SetVariable(0x080498b9, 'esp',
o_add(o_add(o_sub(Const(0xbfbff000,4),Const(0x0000004,4),4),Const(0x0000004,4),4),
, Const(0x0000004,4)))
```

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":

O

run simple effects through emu: apleffs = emu.applyEffects(xlater.getEffects())

xlater.translateOpcode(op)

- 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 ...
 - 0_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)

reduced applied effects (symstate.reduce())

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

$0 \times 0630 - solve$

- 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]: <mark>23</mark>
- 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()

$0 \times 0640 - solve$

• eg: Var._solve()

def varsolve(name,width,emu=None):

1 1 1

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)

$0 \times 0650 - update$

- 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
```



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

	[55]: vwdis(vw,	<pre>findings[1].va, 2)</pre>
0x2	0200002:	mov eax, ebx
0x2	0200004:	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 priables
 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 req1 == req2:
             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 req1 == 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):
                            and the reverse is true! (XCHG)"
                 print "
                 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', OL) (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', OL)
                                                          (mem[ecx:4])
.2020001b REG ANY eax is a initialized from ('ecx', OL)
                                                          (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 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

FuncGraph2: 0x080497c4

look again...

.text:0x080497c4		
.text:0x060497c4	FUNC: int cdec1 1	binary.chldrgst(int arg0,) [1 XREFS]
.text:0x080497c4		
.text:0x080497c4	Stack Variables:	
.text:0x080497c4	4: int arg0	
.text:0x060497c4	-16: int locall6	
.text:0x080497c4	-1056: ii	nt local1056
.text:0x080497c4	-1060: int local1060	
.text:0x080497c4	-1064: ii	nt local1064
.text:0x060497c4		
.text:0x080497c4	55	push ebp
.text:0x080497c5	89e5	mov ebp,esp
.text:0x080497c7	81ec28040000	sub esp,1064
.text:0x060497cd	8d95e8fbffff	lea edx,dword [ebg - 1048]
.text:0x080497d3	b800040000	mov eax, 1024
.text:0x080497d8	83ec04	sub esp,4
.text:0x080497db	50	push eax
.text:0x060497dc	6a00	push 0
.text:0x080497de	52	push edx
.text:0x080497df	e864f4ffff	call memset_06048c48 ;memset_06048c48()
.text:0x080497e4	83c410	add esp,16
.text:0x080497e7	c785e4fbffff0000	mov dword [ebp + local1056],0
.text:0x080497f1	83ecOc	sub esp,12
.text:0x080497f4	ff7508	<pre>push dword [ebp + arg0]</pre>
.text:0x080497±7	e850feffff	<pre>call binary.authenticate ;binary.authenticate(arg0)</pre>
.text:0x060497fc	83c410	add esp,16
.cexc:0x080497ff	83ec04	sub esp,4
.text:0x08049803	6a03	push 3
.text:0x08049804	680a9c0408	push str_OK_08049c0a
.text:0x08049809	ff7508	push dword [ebp + arg0]
.cexc:0x0804980c	e827f3ffff	call write 08048b38 ;write 08048b38()
.text:0x06049611	83c410	add esp,16
.text:0x08049814	83ec04	sub esp.4
.text:0x06049817	£8ff070000	push 2047
.cexc:0x0804981c	6800a20408	push binary.input_buffer
stext:0x06049631	ff7508	push dword [ebp + arg0]
.cext:0x08049824	e88ff3ffff	call read_08048bb8 ;read_08048bb8()
.cext:0x08049829	83c410	add esp,16
.cexc:0x0804982c	8945£4	mov dword [ebp + local16], eax
.text:0x0804983f	83ec04	sub esp,4
.text:0x08049832	8d85e8fbffff	lea eax, dword [ebp - 1048]
.text:0x08049838	50	push eax
.cexc:0x08049839	680e9c0408	push str bacon:%s 08049c0e
.text:0x0804983e	6800a20408	push binary.input_buffer
.cext: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)
- Oxbfbfebe4 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 08048bb80
0x08049829: esp = 0xbfbfebd4
0x0804982c: [ 0xbfbfeff0 : 4 ] = read 08048bb8()
0x0804982f: eflags gt = None
0x0804982f: eflags_lt = None
0x0804982f: eflags_sf = None
0x0804982f: eflags_eg = None
0 \times 0804982f: esp = 0 \times bfbfebd0
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