The System of Automatic Searching for Vulnerabilities or *how to use Taint Analysis to find vulnerabilities*

Alex Bazhanyuk (@ABazhanyuk)
Nikita Tarakanov (@NTarakanov)

Who is Alex Bazhanyuk

- Security Researcher
- Organizer of Defcon Ukraine Group
- Working in UC Berkley in BitBtlaze project
- Solves problems of automation of RE

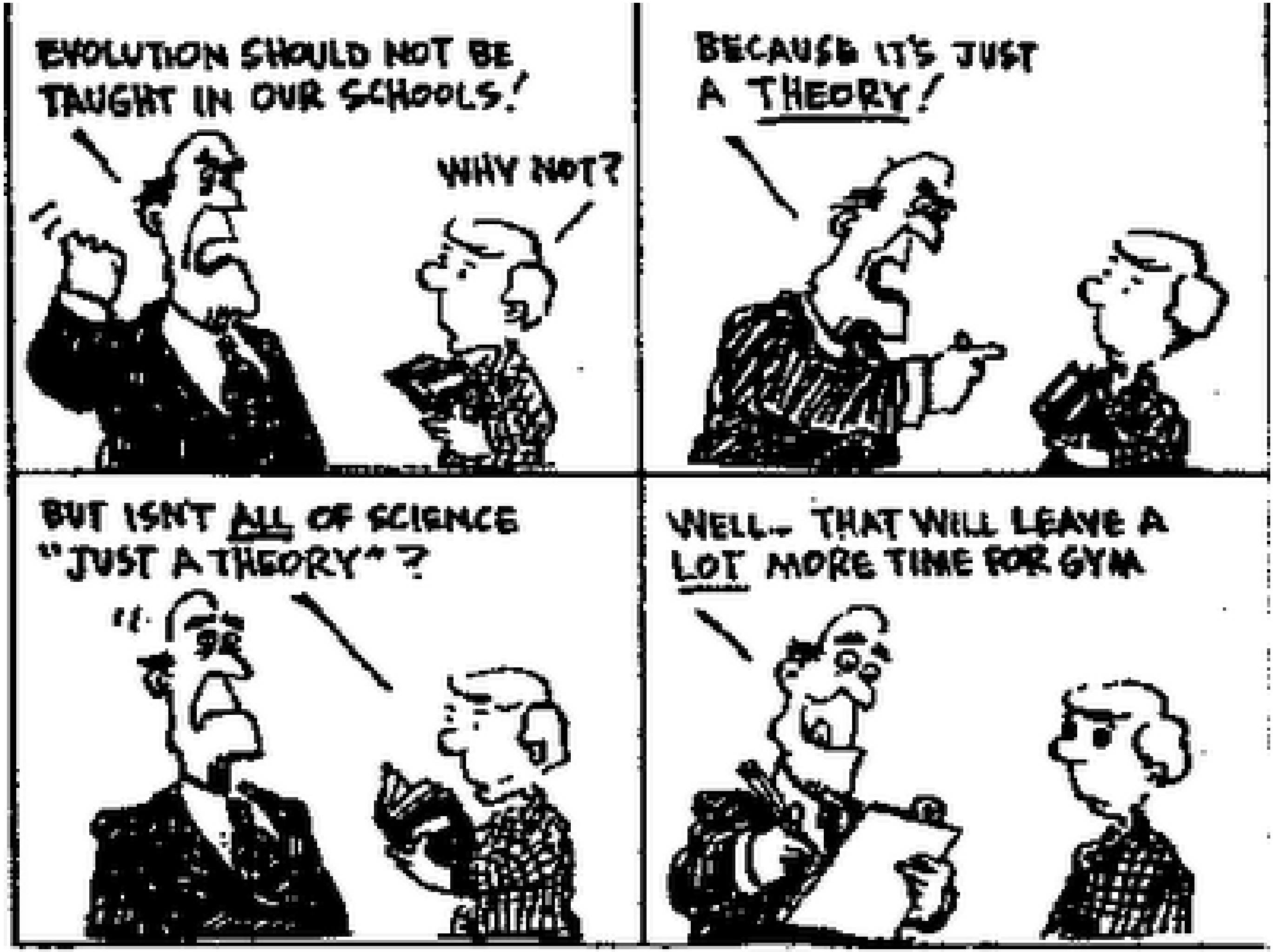
Who is Nikita Tarakanov

- Independent Security Researcher
- Author of some articles in][akep magazine
- Likes to reverse engineer r0 parts
- Discovered a lot of LPE vulnerabilities
- Solves problems of automation of RE

SASV main parts

- IDA Pro plugins
- BitBlaze: Vine+utils, TEMU + plugins

Theory


EVOLUTION SHOULD NOT BE TAUGHT IN OUR SCHOOLS!
WHY NOT?
BECAUSE IT'S JUST A THEORY!
BUT ISN'T ALL OF SCIENCE "JUST A THEORY"?
WELL.. THAT WILL LEAVE A LOT MORE TIME FOR GYM

Tainting

- Taint sources:

Network, Keyboard, Memory, Disk, Function outputs etc.

- Taint propagation: a data flow technique

Memory

Whole-system

Across registers/memory/disk/swapping

Fundamentals of taint analysis

Illustration John Cune

Taint propagation

- If an operation uses the value of some **tainted** object, say X, as assignes value to another, say Y, then object Y becomes **tainted.** Object X taints the object Y
- Taint operator **t**
- **X→ t(Y)**
- Taint operator is transitive

X → t(Y) and Y→ t(Z), then X→ t(Z)

BitBlaze: Binary Analysis Infrastructure

- Automatically extracting security-related properties from
- binary code
- Build a unified binary analysis platform for security

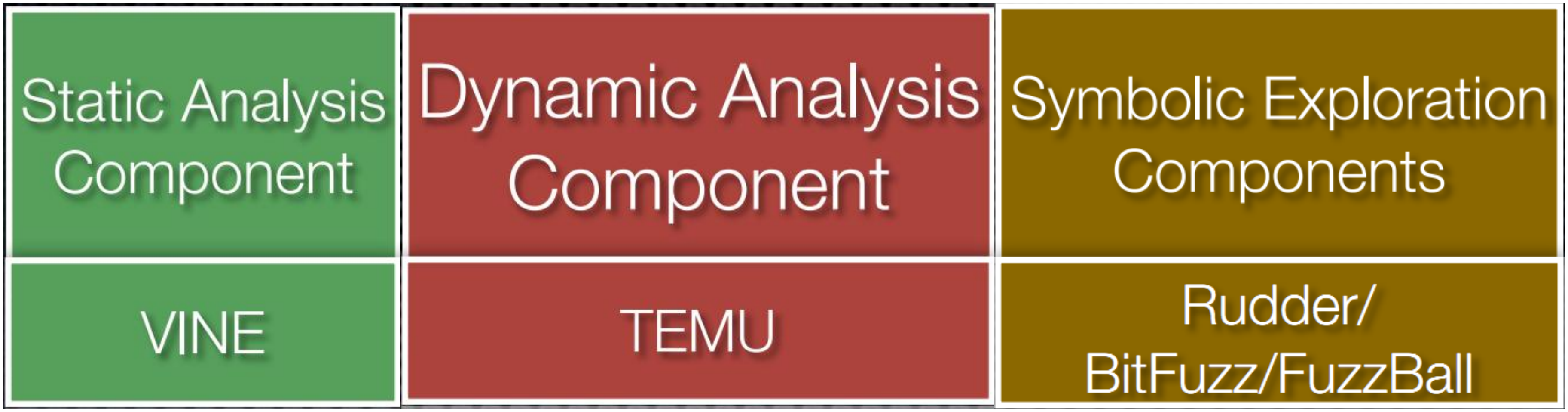
- Static analysis + Dynamic analysis + Symbolic Analysis
- Leverages recent advances in program analysis, formal methods, binary instrumentation…

Solves security problems via binary analysis

- More than a dozen different security applications
- Over 25 research publications

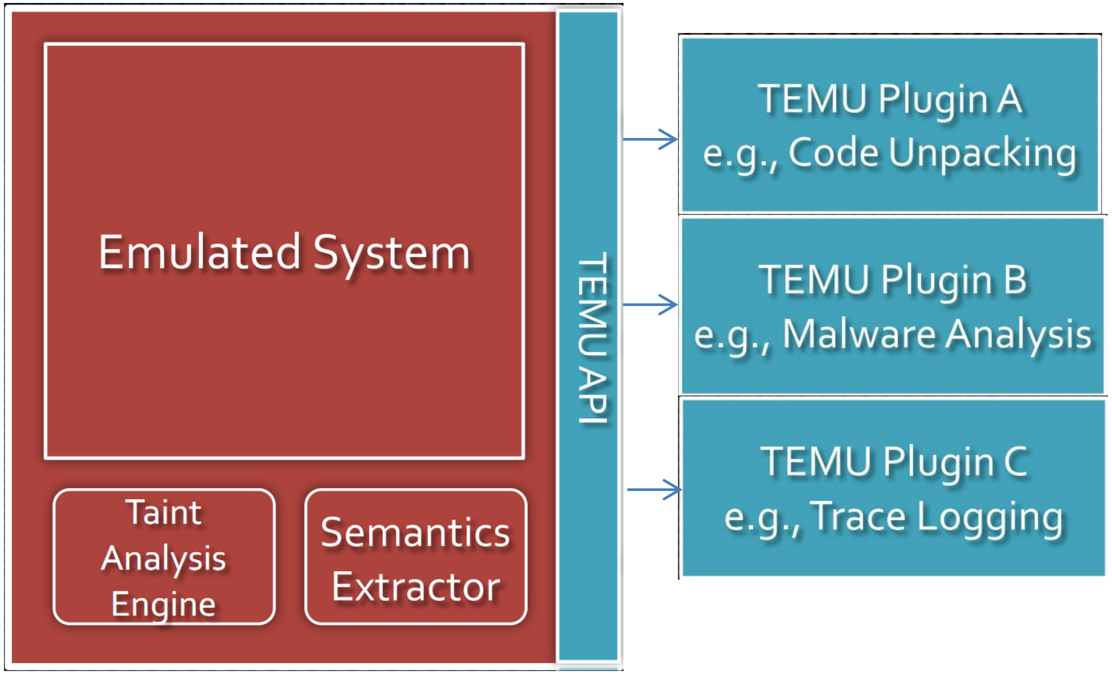
BitBlaze

- http://bitblaze.cs.berkeley.edu/
- TEMU,VINE
- Rudder, Panorama, Renovo



Static Analysis Component	Dynamic Analysis Component	Symbolic Exploration Components
VINE	TEMU	Rudder/ BitFuzz/FuzzBall

TEMU

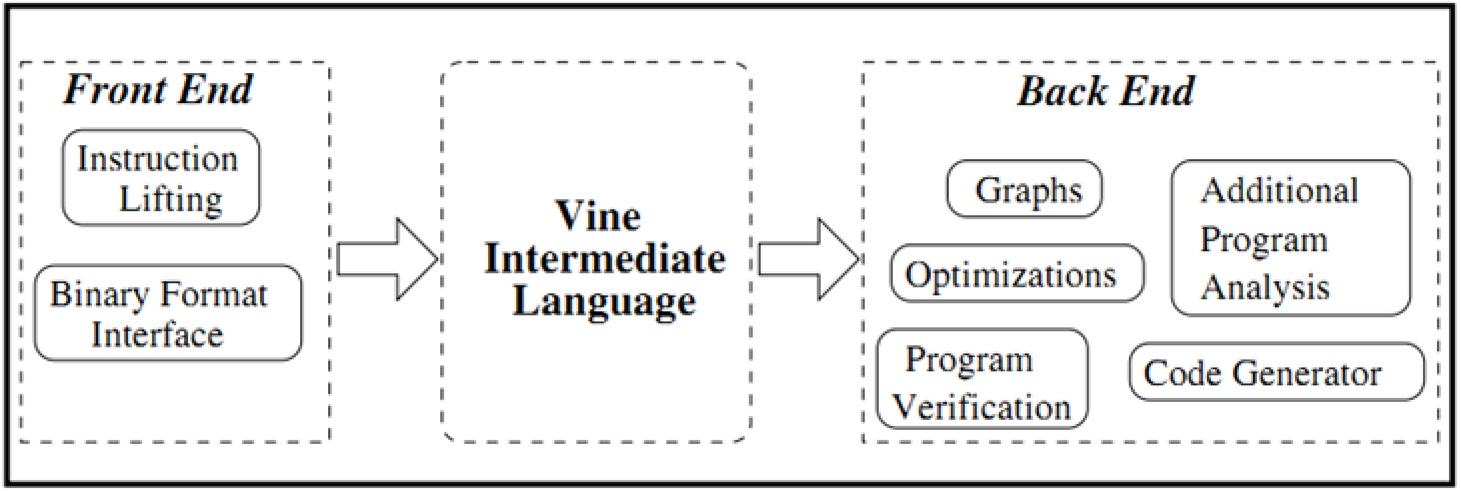

Emulated System
Taint Analysis Engine
Semantics Extractor
TEMU API
TEMU Plugin A
e.g., Code Unpacking
TEMU Plugin B
e.g., Malware Analysis
TEMU Plugin C
e.g., Trace Logging

Limitations of TEMU

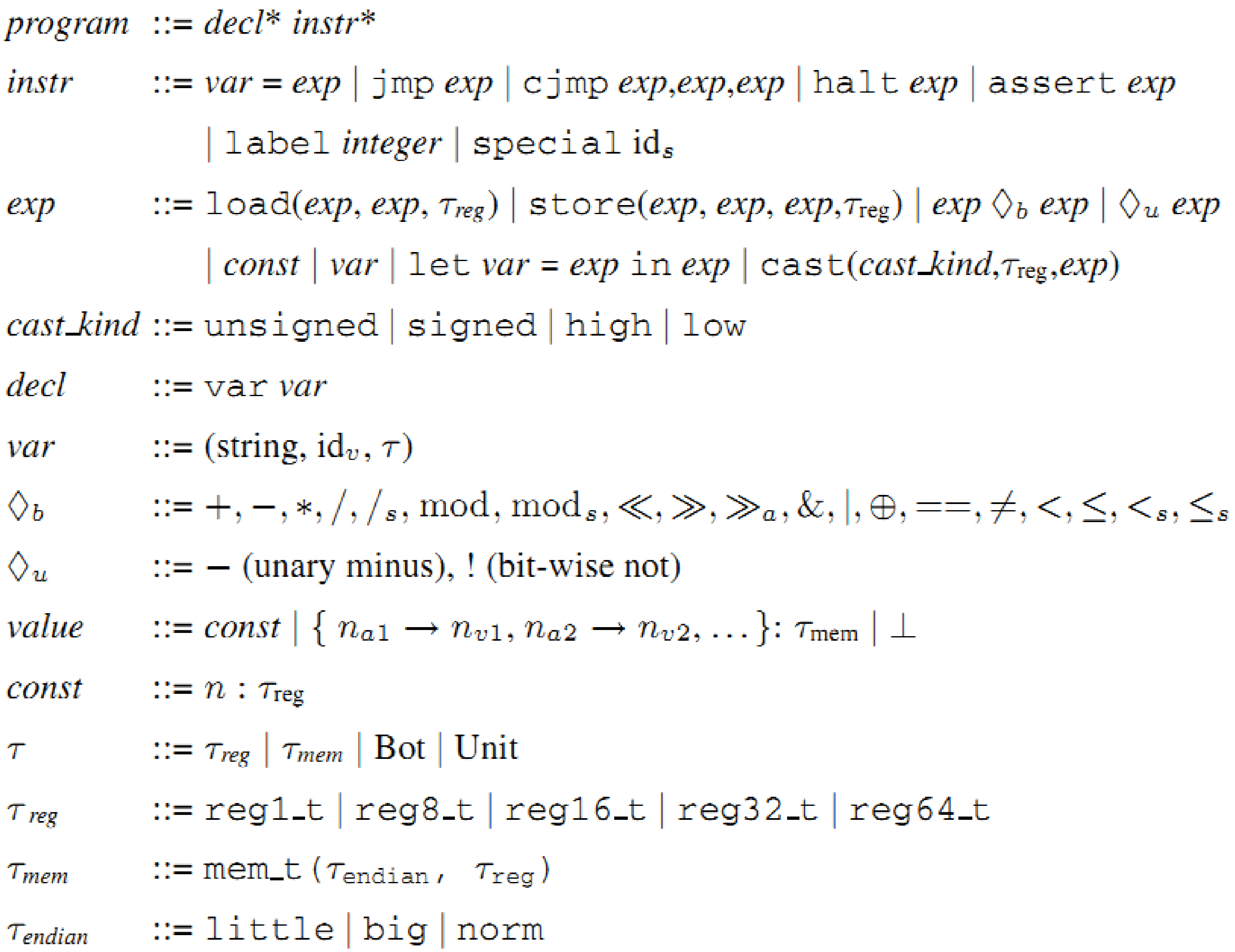
- **Qemu 0.9.1 - TEMU**
- **Qemu 0.10 - TCG(Tiny Code Generator)-TODO**
- **Qemu 0.10 ⬄ Qemu 1.01**

VINE

BORING...
Z


Front End
Instruction Lifting
Binary Format Interface
Vine Intermediate Language
Back End
Graphs
Optimizations
Additional Program Analysis
Program Verification
Code Generator

The Vine Intermediate Language



program ::= *decl** *instr**

instr ::= *var* = *exp* | jmp *exp* | cjmp *exp*,*exp*,*exp* | halt *exp* | assert *exp*
| label *integer* | special id_s

exp ::= load(*exp*, *exp*, τ_{reg}) | store(*exp*, *exp*, *exp*,τ_{reg}) | *exp* $\Diamond_b$ *exp* | $\Diamond_u$ *exp*
| *const* | *var* | let *var* = *exp* in *exp* | cast(*cast_kind*,τ_{reg},*exp*)

cast_kind ::= unsigned | signed | high | low

decl ::= var *var*

var ::= (string, id_v, τ)

$\Diamond_b$ ::= $+, -, *, /, /_s, \mathrm{mod}, \mathrm{mod}_s, \ll, \gg, \gg_a, \&, |, \oplus, ==, \neq, <, \leq, <_s, \leq_s$

$\Diamond_u$ ::= $-$ (unary minus), ! (bit-wise not)

value ::= *const* | $\{\, n_{a1} \rightarrow n_{v1}, n_{a2} \rightarrow n_{v2}, \ldots \}$: τ_{mem} | $\perp$

const ::= $n : \tau_{\mathrm{reg}}$

τ ::= τ_{reg} | τ_{mem} | Bot | Unit

τ_{reg} ::= reg1_t | reg8_t | reg16_t | reg32_t | reg64_t

τ_{mem} ::= mem_t ($\tau_{\mathtt{endian}}$, $\tau_{\mathtt{reg}}$)

τ_{endian} ::= little | big | norm

Example of disasm:

```
fc32dcec:      rep stos %eax,%es:(%edi)       R@eax[0x00000000][4](R)  T0
R@ecx[0x00000002][4](RCW)       T0      M@0xfb7bfff8[0x00000000][4](CW)  T1 {15
(1231, 69624) (1231, 69625) (1231, 69626) (1231, 69627) }

fc32dcec:      rep stos %eax,%es:(%edi)       R@eax[0x00000000][4](R)  T0
R@ecx[0x00000001][4](RCW)       T0      M@0xfb7bfffc[0x00000000][4](CW)  T1 {15
(1231, 69628) (1231, 69629) (1231, 69630) (1231, 69631) }
fc32dcee:      mov    %edx,%ecx       R@edx[0x0000015c][4](R)  T0
R@ecx[0x00000000][4](W)  T0
fc32dcf0:      and    $0×3,%ecx       I@0×00000000[0x00000003][1](R)   T0
R@ecx[0x0000015c][4](RW)        T0
fc32dcf5:      andl   $0×0,-0×4(%ebp)  I@0×00000000[0x00000000][1](R)   T0
M@0xfb5ae738[0x00000002][4](RW)  T0
fc32dcf9:      jmp    0x00000000fc32c726      J@0×00000000[0xffffea2d][4](R)   T0
fc32c726:      cmpl   $0×0,-0×58(%ebp) I@0×00000000[0x00000000][1](R)   T0
M@0xfb5ae6e4[0x00000000][4](R)   T0
```

Taint info

- T0 - means that the statement is not tainted.
- T1 - means that the statement is tainted.
- Here's an example of:
- fc32dcec: rep stos% eax,% es: (% edi) R @ eax [0x00000000] [4] (R) T0 R @ ecx [0x00000001] [4] (RCW) T0 M @ 0xfb7bfffc [0x00000000] [4] (CW) T1 {15 (1231, 628) (1231, 629) (1231, 630) (1231, 631)}
- 4 bits of information tainted and they depend on the offset: 628, 629, 630, 631. 1231 - this number is origin(kind of ID that TEMU plugin sets), and 15 – this number of the source type.

appreplay

- ./vine-1.0/trace_utils/appreplay -trace font.trace -ir-out font.trace.il -assertion-on-var false-use-post-var false

where:

- appreplay - ocaml script that we run;
- -trace - the way to the trace;
- -ir-out - the path to which we write IL code.
- -assertion-on-var false-use-post-var false - flags that show the format of IL code for this to false makes it more readable text.

Example of IL code:

- Begins with the declaration of variables:
- INPUT - it's free memory cells, those that are tested in the very beginning (back in TEMU), input into the program from an external source.

```
var cond_000017_0x4010ce_00_162:reg1_t;

var cond_000013_0x4010c3_00_161:reg1_t;
var cond_000012_0x4010c0_00_160:reg1_t;
var cond_000007_0x4010b6_00_159:reg1_t;
var INPUT_10000_0000_62:reg8_t;
var INPUT_10000_0001_63:reg8_t;
var INPUT_10000_0002_64:reg8_t;
var INPUT_10000_0003_65:reg8_t;
var mem_arr_57:reg8_t[4294967296];   – memory as an array
var mem_35:mem32l_t;
```

```
R_EAX_5:reg32_t =
0×73657930:reg32_t;
{
var idx_144:reg32_t;
var val_143:reg8_t;
idx_144:reg32_t =
0x12fef0:reg32_t;
val_143:reg8_t =
INPUT_10000_0000_62:reg
8_t;
mem_arr_57[idx_144:reg32
_t + 0:reg32_t]:reg8_t =
cast((val_143:reg8_t &
0xff:reg8_t) >>
0:reg8_t)L:reg8_t;
```

```
T_32t2_60:reg32_t = R_ESP_1:reg32_t;
T_32t1_59:reg32_t = T_32t2_60:reg32_t
      + 0x1c8:reg32_t;
T_32t3_61:reg32_t =((
cast(mem_arr_57[T_32t1_59:reg32_t +
      0:reg32_t]:reg8_t)U:reg32_t
<< 0:reg32_t
|
cast(mem_arr_57[T_32t1_59:reg32_t +
      1:reg32_t]:reg8_t)U:reg32_t
<< 8:reg32_t)
|
cast(mem_arr_57[T_32t1_59:reg32_t +
      2:reg32_t]:reg8_t)U:reg32_t
<< 0×10:reg32_t)
|
cast(mem_arr_57[T_32t1_59:reg32_t +
      3:reg32_t]:reg8_t)U:reg32_t
<< 0×18:reg32_t
;
R_EAX_5:reg32_t = T_32t3_61:reg32_t;
}
```

What is STP and what it does?

- STP - constraint solver for bit-vector expressions.
- separate project independent of the BitBlaze
- To produce STP code from IL code:
- ./vine-1.0/utils/wputil trace.il -stpout stp.code
- where the input is IL code, and the output is STP code

STP program example

```
mem_arr_57_8 : ARRAY BITVECTOR(64) OF BITVECTOR(8);
INPUT_10000_0000_62_4 : BITVECTOR(8);
ASSERT( 0bin1 =
(LET R_EAX_5_232 =
0hex73657930
IN
(LET idx_144_233 =
0hex0012fef0
IN
(LET val_143_234 =
INPUT_10000_0000_62_4
IN
(LET mem_arr_57_393 =
(mem_arr_57_8 WITH [(0bin00000000000000000000000000000000 @ BVPLUS(32,
idx_144_233,0hex00000000))] := (val_143_234;0hexff)[7:0])
…….
IN
(cond_000017_0x4010ce_00_162_392;0bin1))))))));
```

Is this expression false?

QUERY (FALSE);

And give a counter example:

COUNTEREXAMPLE;

STP output example

- ./stp stp.code
- Example of STP output:

```
ASSERT( INPUT_10000_0001_63_5 = 0×00 );
ASSERT( INPUT_10000_0002_64_6 = 0×00 );
ASSERT( INPUT_10000_0000_62_4 = 0×61 );
ASSERT( INPUT_10000_0003_65_7 = 0×00 );
Invalid.
```

SASV Components:

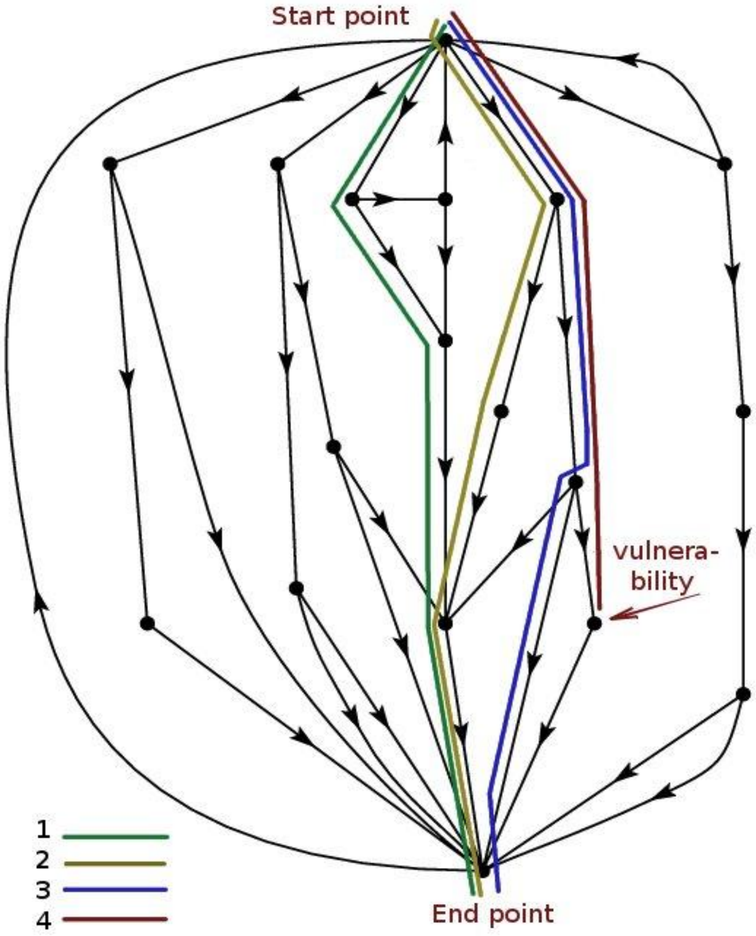
- **Temu** (tracecap: start/stop tracing. Various additions to tracecap(hooks etc.))
- **Vine** (appreplay, wputil)

- **STP**

- **IDA plugins:**
- - *DangerousFunctions* – finds calls to malloc,strcpy,memcpy etc.
- - *IndirectCalls* – indirect jumps, indirect calls.
- - *ida2sql* (zynamics) –idb in the mysql db. (http://blog.zynamics.com/2010/06/29/ida2sql-exporting-ida-databases-to-mysql/)

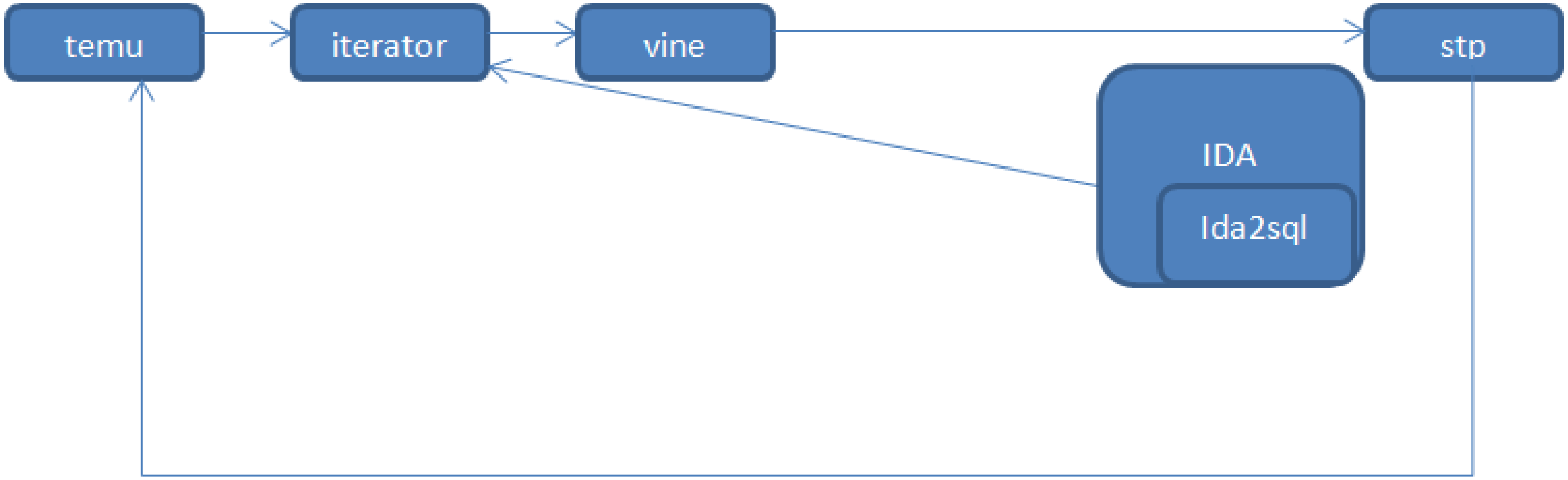
- **Iterators** – wrapper for temu, vine, stp.
- **Various publishers –** for DeviceIoControl etc.

How does SASV work?



SASV

- Scheme:

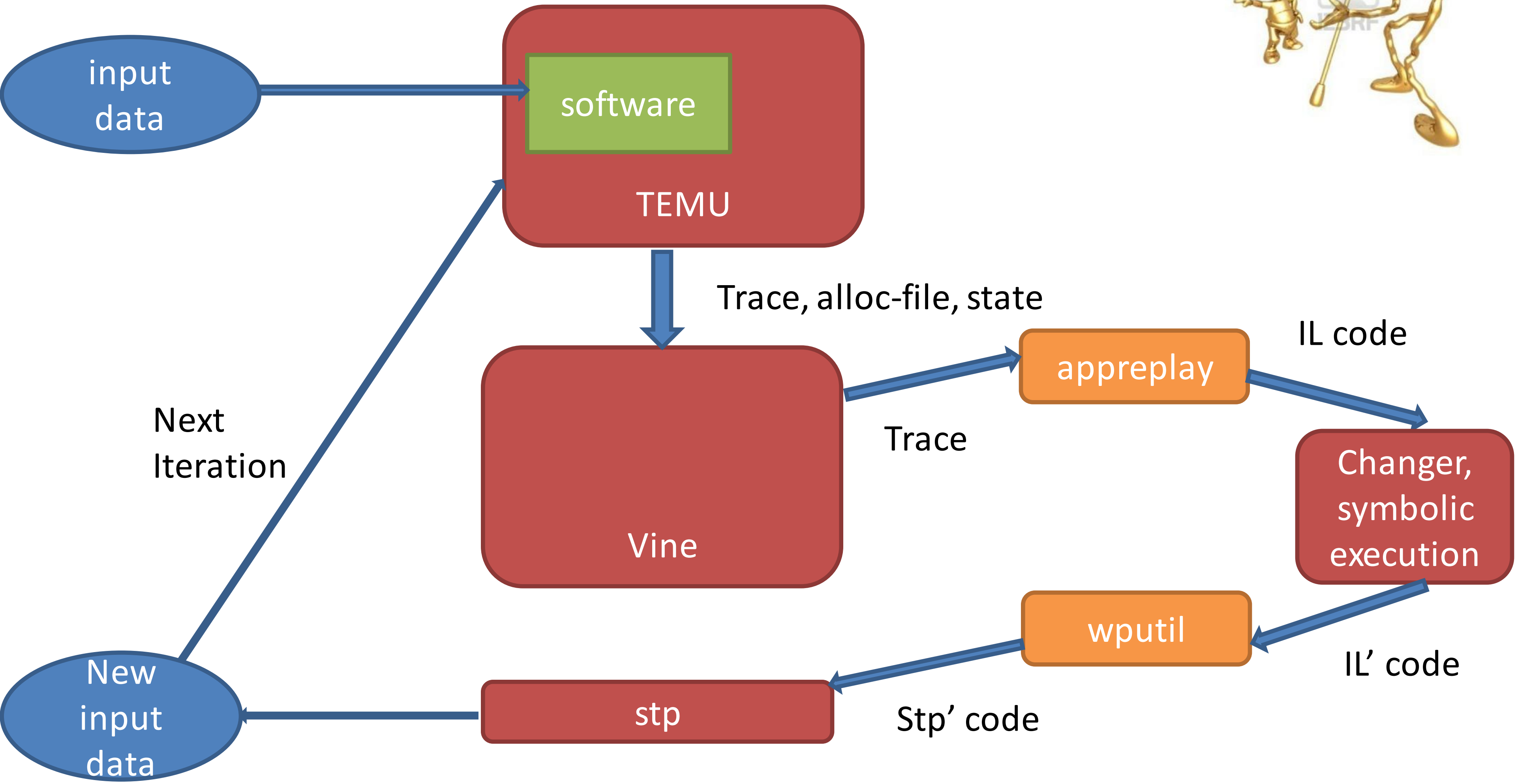


- Min Goal: max coverage of the dangerous code
- Max Goal: max coverage of the all code

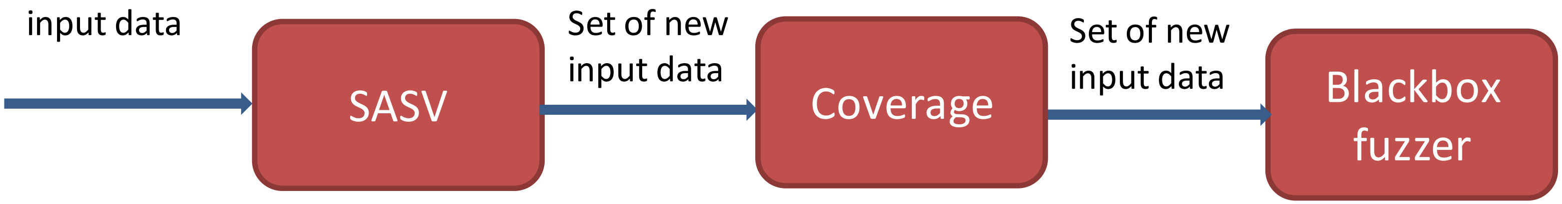
SASV basic algorithm

1. Work of IDA plugins -> dangerous places
2. Publisher(s) -> invoke targeted code
3. TEMU -> trace
4. Trace -> appreplay -> IL
5. IL -> change path algo -> IL’
6. IL’ -> wputil -> STP_prorgam’
7. STP_prorgam’ -> STP -> data for **n+1** iteration
8. Goto #2

Diagram for new path in graph



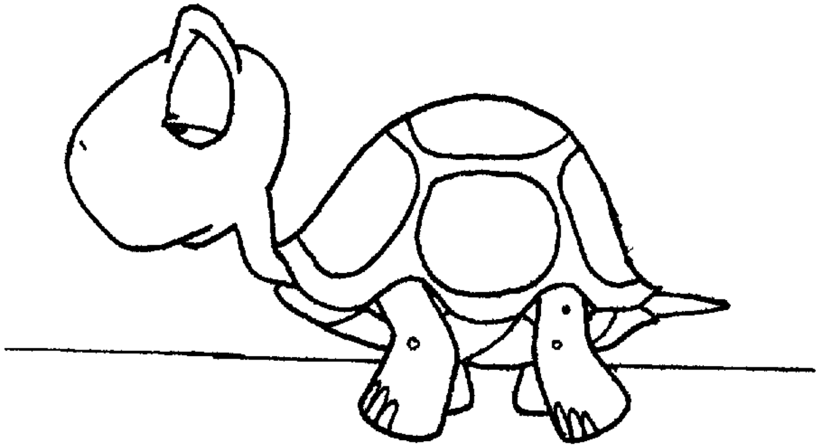
Combo system: Dumb+Smart



Disadvantages

- Definition of the vulnerability is difficult task.
- Performance – speed of tracing in TEMU is

AWFUL



TO DO

DATE

FINISH BY

SIGN.

No.	TASKS	DONE
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		

ERRANDS

DONE

No.	CORRESPONDENCE	DONE
01		
02		
03		
04		
05		
06		
07		
08		
09		
10		

NOTES

□ ALL DONE

"Make a list—you'll feel better."

Get rid of that damned QEMU!

- Move taint propagation to Hypervisor!
- Damn good idea!
- But not implemented yet ☹

AV

Vulnerabilities in drivers

- Overflows: stack, pool, integer
- Pointer overwrite
- Null pointer dereference
- Race condition
- Various logical vulnerabilities

Attack vectors(r3->r0)

- **IOCTL**
- SSDT hooks(Native & Shadow)
- various notification routines

DeviceIoControl

- Parameters:
  - hDevice
  - *dwIoControlCode*
  - *lpInBuffer*
  - *nInBufferSize*
  - *lpOutBuffer*
  - *nOutBufferSize*
  - *lpBytesReturned*
  - *lpOverlapped*

Concept

IOCTL:

Data to taint:

- *dwIoControlCode* - to get list of supported ioctl codes
- *lpInBuffer - pointer(METHOD_NEITHER) and data (METHOD_BUFFERED)*
- *nInBufferSize - size ranges*
- *lpOutBuffer - pointer(METHOD_NEITHER) and data (METHOD_BUFFERED)*
- *nOutBufferSize- size ranges*

Tracing only targeted driver code

- 0dayz Time!

TrendMicro tmtdi.sys #1

- Ioctl code 0x220044 (METHOD_BUFFERED)
- No range check for size
- Just check for correct address – NPD check (MmIsAddressValid)
- Pool corruption in cycle
- No control of overflowing data ☹

TrendMicro tmtdi.sys #1

```
• .text:0001D881          mov     edi, [ebx+0Ch]
• .text:0001D884          push    edi          ; our buffer
• .text:0001D885          call    esi ; MmIsAddressValid
• .text:0001D887          test    al, al
• .text:0001D889          jz      loc_1DDAB
• .text:0001D88F          push    [ebp+output_buff_size]
• .text:0001D892          push    edi
• .text:0001D893          push    offset rules_list
• .text:0001D898          call    ioctl_0x220044_vuln
• [..]
```

TrendMicro tmtdi.sys #1

```
•  .text:000156EA        mov    ebx, [ebp+our_buffer_size_controlled]
•  .text:000156ED        mov    [ebp+NewIrql], al
•  .text:000156F0        mov    eax, dword_22CA0
•  .text:000156F5        mov    edx, offset dword_22CA0
•  .text:000156FA        cmp    eax, edx
•  .text:000156FC        jz     short loc_15748
•  [..]
•  .text:00015700        mov    ecx, [eax+0Ch]
•  .text:00015703        mov    [ebx], ecx
•  .text:00015705        mov    ecx, [eax+10h]
•  .text:00015708        mov    [ebx+4], ecx
•  .text:0001570B        mov    ecx, [eax+14h]
•  .text:0001570E        mov    [ebx+8], ecx ← write outside of the pool chunk
•  .text:00015711        mov    ecx, [eax+18h]
•  .text:00015714        mov    [ebx+0Ch], ecx
```

TrendMicro tmtdi.sys #2

- Ioctl code 0x220030
- Range check for inbuff_size >= 0x2AA
- Range check for outbuff_size >= 0x4D0
- Allocs pool memory for const size 0x4D0
- And…
- Zeroing it with outbuff_size length! LOL

TrendMicro tmtdi.sys #2

- .text:0001D704 cmp [ebp+inbuff_size], 2AAh
- .text:0001D70B jb loc_1DDAB
- .text:0001D711 mov esi, **4D0h**
- .text:0001D716 cmp [ebp+output_buff_size], esi
- .text:0001D719 jb loc_1DDAB
- .text:0001D71F push 746D74h ; Tag
- .text:0001D724 push **esi** ; NumberOfBytes
- .text:0001D725 push 0 ; PoolType
- .text:0001D727 call ds:ExAllocatePoolWithTag
- [..]

TrendMicro tmtdi.sys #2

```
• .text:0001D74B          push    edi ; pool_mem_const_size
• .text:0001D74C          lea     eax, [ebp+output_buff_size]
• .text:0001D74F          push    eax           ; output_buff_size
• .text:0001D750          push    [ebp+NewIrql]   ; inbuff
• .text:0001D753          push    220030h        ; ioctl_code
• .text:0001D758          call    ioctl_several_ioctl_codes
• [..]
• .text:00014918          mov     esi, [ebp+output_buff_size]
• [..]
• .text:00014977          push    dword ptr [esi] ;
• .text:00014979          push    0             ;
• .text:0001497B          push    [ebp+pool_mem_const_size] ;
• .text:0001497E          call    memset
```

Pitfalls of tainting r0

- Taint info lost

- Check of system environment variables

- System defense mechanism(s) (win32k.sys

  WATCHDOG BugCheck)

Pitfalls of tainting r0(IOCTL)

- KeGetPreviousMode
- IoGetCurrentProcess
- Even hooking NtDeviceIoControlFile!

How some AVs kill LPE 0dayz

- Check for previous mode:
- .text:0001DC32 cmp byte ptr [ebx+**20**h], 0
- .text:0001DC36 jnz loc_1DDAB
- .text:0001DC3C mov eax, [edi]

- The vuln is here, dword_22934 is function ptr
- .text:0001DC3E mov dword_22934, eax

Thanks, ☺

- Questions?

http://twitter.com/#!/ABazhanyuk
http://twitter.com/#!/NTarakanov