Crash analysis with BitBlaze

Charlie Miller
Independent Security Evaluators

Noah Johnson
University of California Berkeley

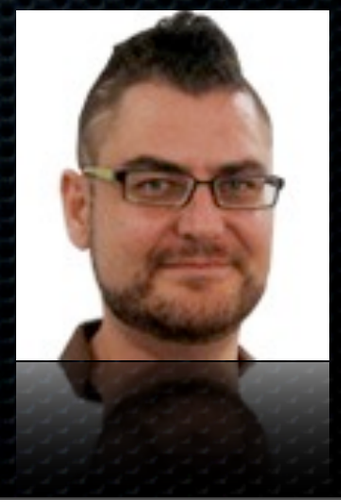
Agenda

- The problem with fuzzing
- The BitBlaze platform
- Adobe Reader case study
- 4 real life examples of its usage
- Conclusions

Backstory

The problem

- In 2009, at Syscan Ben Nagy gives a talk titled “Finding Microsoft Vulnerabilities by Fuzzing Binary Files with Ruby - A New Fuzzing Framework”
- Fuzzes Microsoft Word
- Finds 200,000 crashes in 61 bins
- Too many to examine by hand
  - Turns out 4 are exploitable security vulns
  - Turns to MS for help in triaging the crashes



Now its my problem

- At CanSecWest, I present some fuzzing runs in my talk "Babysitting an Army of Monkeys"
- In Preview I find crashes at 1373 EIPs, 220-280 are unique, 60+ "exploitable"
- In Adobe Reader I find 30-40 unique crashes
- In Open Office, 70 distinct crashes

Finding bugs isn't the problem

- Figuring out what to do next is the problem
- Prioritize the crashes
- Which are actually exploitable?
- Find the underlying vulnerability behind each crash
- Unlike fuzzing this can't be entirely automated
  - We need tools to help speed this up

BitBlaze: Binary Analysis Infrastructure

BitBlaze Research Foci

- Automatically extracting security-related properties from binary code

1. Build a unified binary analysis platform for security
   - Static analysis + Dynamic analysis + Symbolic Analysis
   - Leverages recent advances in program analysis, formal methods, binary instrumentation…
2. Solve security problems via binary analysis
   - More than a dozen different security applications
   - Over 25 research publications

BitBlaze Overview

- Unified binary analysis platform for security
- Enables a wide-range of security applications

Detecting Vulnerabilities

Generating Filters

Dissecting Malware

BitBlaze Binary Analysis Infrastructure

BitBlaze Overview: Challenges

- Important to handle binary-only setting
  - Malware and COTS
- Complexity
  - IA-32 manuals weigh over 11 pounds!!
- Lack higher-level semantics
  - No functions, no variables, no types
- Require whole-system view
  - Interactions with kernel and other processes
- Malicious code may obfuscate
  - Code packing, encryption
  - Dynamically generated code

BitBlaze Overview: Design Goals

- Accuracy
  - Enable precise analysis, e.g., symbolic execution
- Extensibility
  - Build your own tools that use the functionality!
- Fusion of static & dynamic analysis
  - Static analysis
    - Pros: more complete results
    - Cons: pointer aliasing, indirect jumps, code obfuscation
  - Dynamic analysis
    - Pros: easier
    - Cons: limited coverage
  - Solution: combining both

Some Security Applications

- Vulnerability detection and analysis
  - Vulnerabilities in malware: Zbot, MegaD, Gheg, Cutwail
- Model extraction
  - Content-sniffing XSS attacks (Wikipedia/IE7)
- Protocol reverse engineering
  - MegaD C&C protocol revealed!
- Patch-based exploit generation
  - GDI Integer Overflow (MS07-046)
- Detecting privacy breaches
  - Google Desktop
- In-depth malware analysis
  - Universal unpacker

BitBlaze Overview: Infrastructure

Static Analysis Component

Dynamic Analysis Component

Symbolic Exploration Components

BitBlaze Overview: Infrastructure

Static Analysis Component

VINE

Dynamic Analysis Component

Symbolic Exploration Components

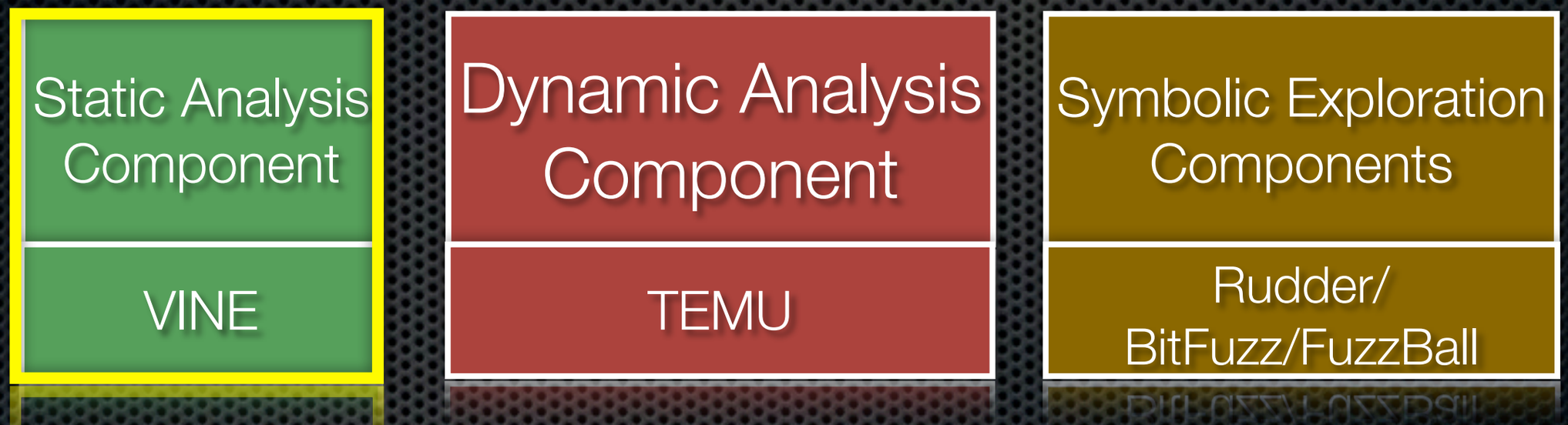
BitBlaze Overview: Infrastructure

Static Analysis Component	Dynamic Analysis Component	Symbolic Exploration Components
VINE	TEMU	

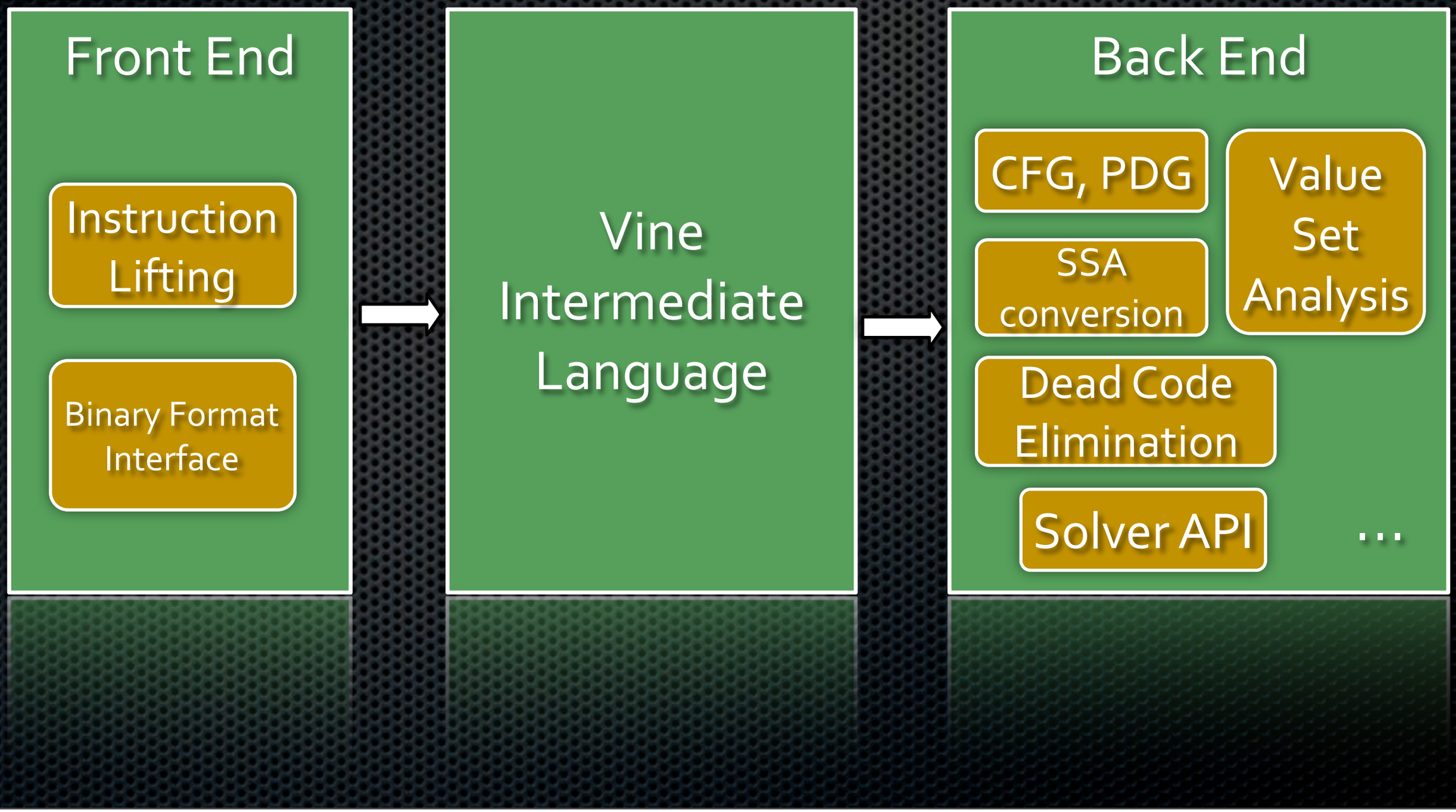
BitBlaze Overview: Infrastructure

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

BitBlaze Overview: Infrastructure



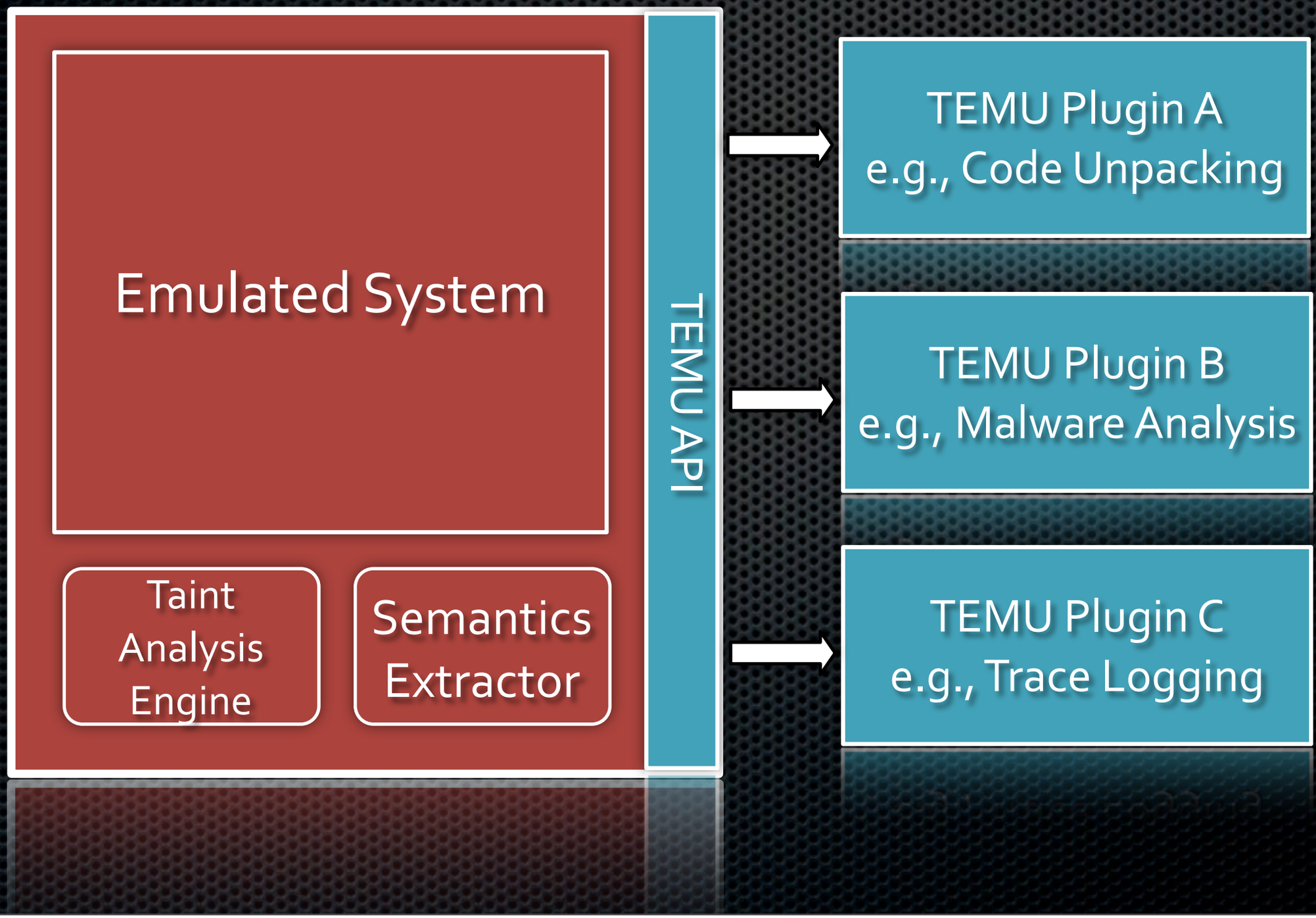
BitBlaze Overview: Vine



BitBlaze Overview: Temu

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

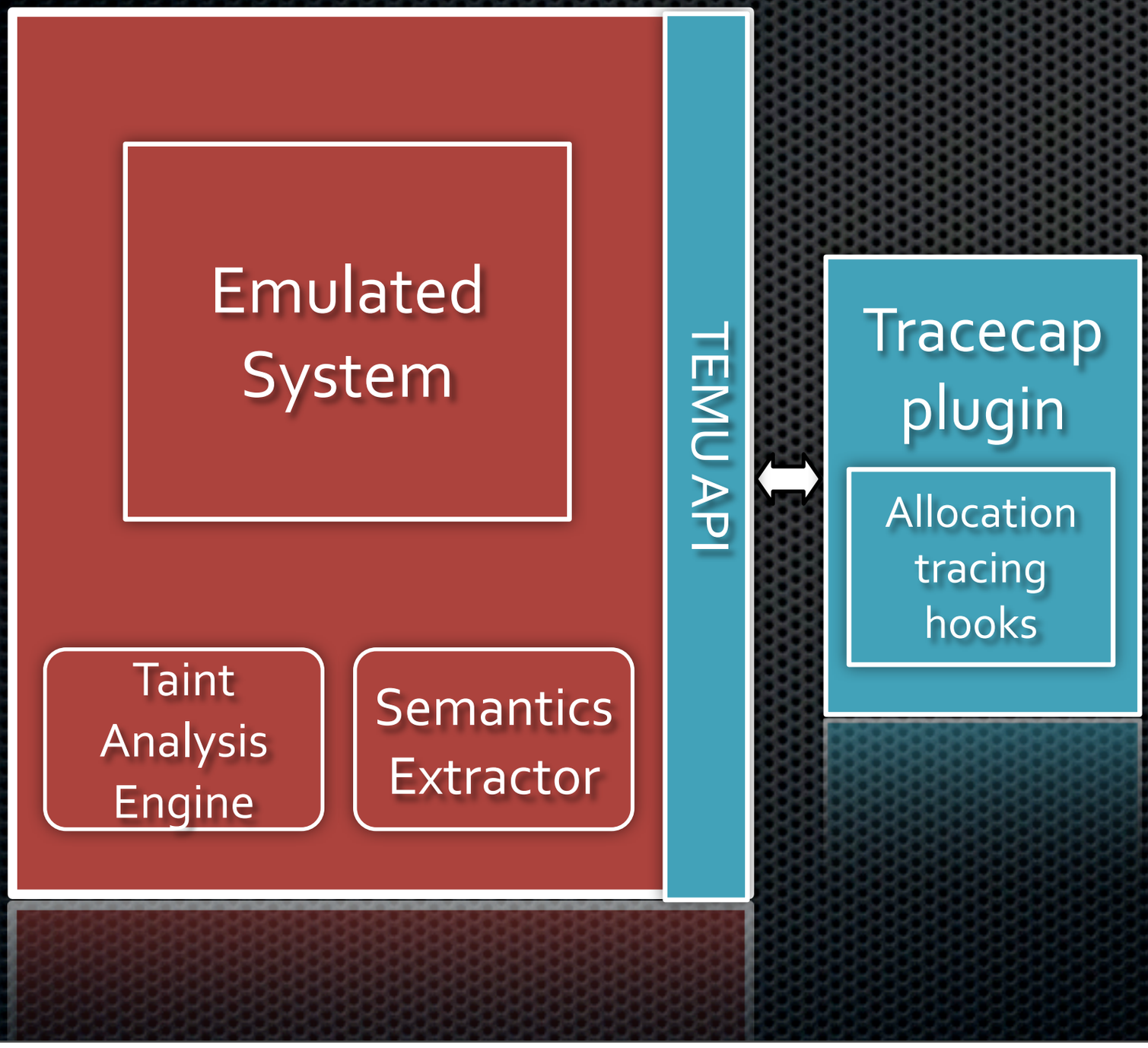
BitBlaze Overview: Temu



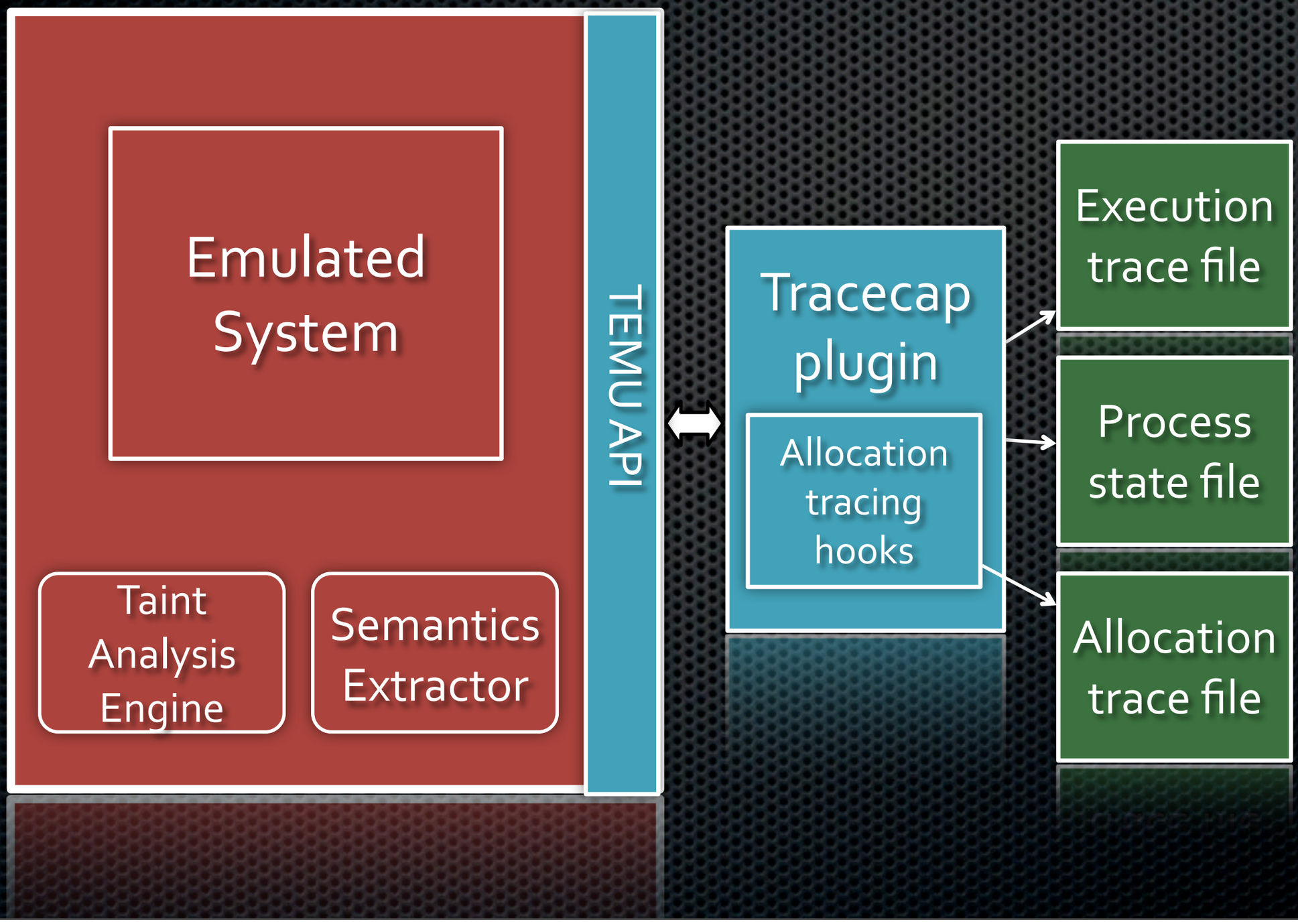
BitBlaze Overview: Taint Analysis

- Mark inputs, follow them in execution
- Taint sources
  - Network, Keyboard, Memory, Disk, Function outputs
- Taint propagation: a data flow technique
  - Shadow memory
  - Whole-system
    - Across register/memory/disk/swapping
- Taint sinks
  - Where to check the taint
  - Application dependent

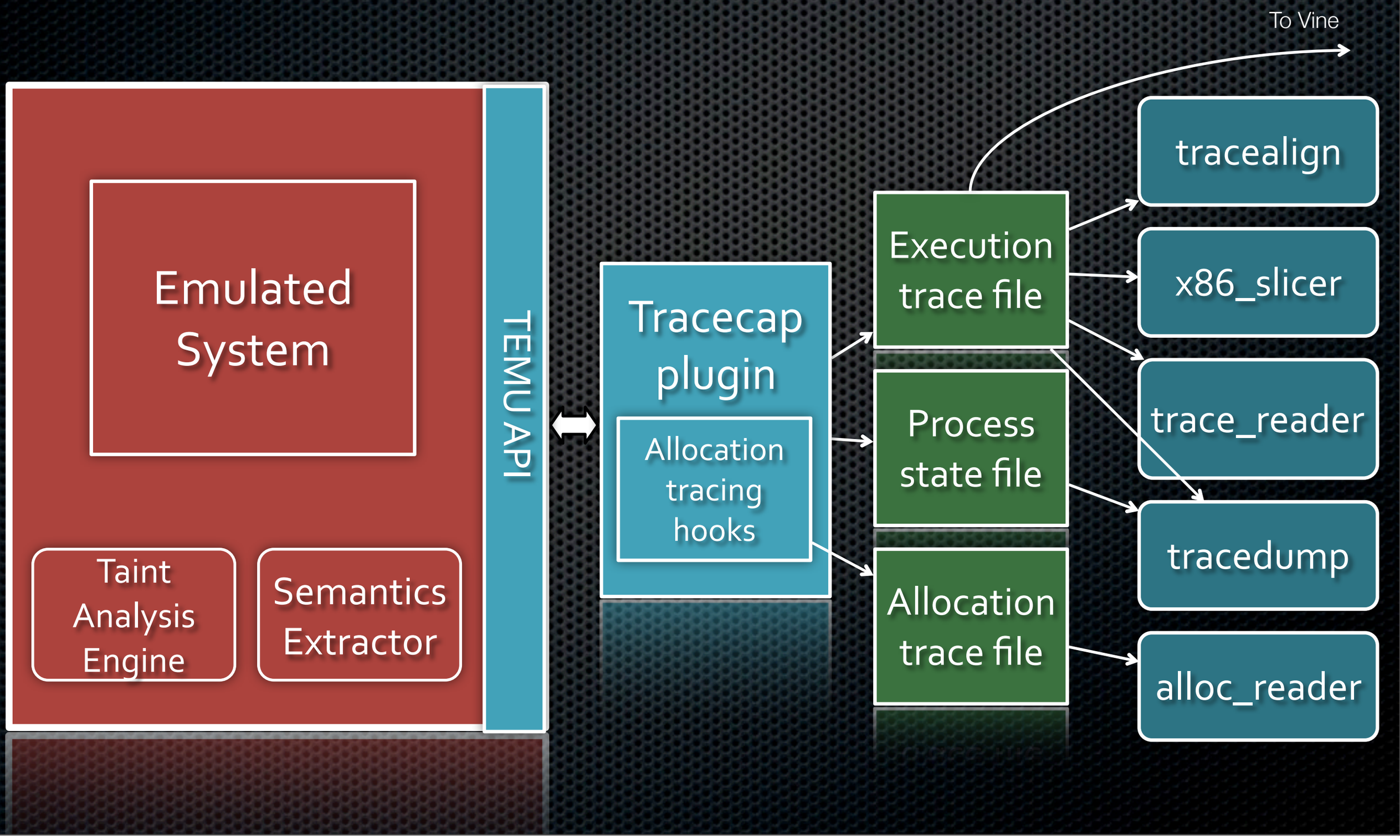
Tools for Crash Analysis



Tools for Crash Analysis



Tools for Crash Analysis



Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

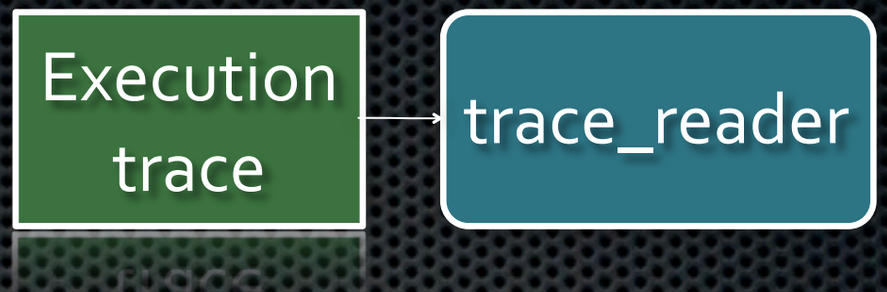
sequential

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

instruction

Trace Reading




```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x00007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

instruction

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP:  NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

four-byte memory operand

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

taint information

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

thread ID

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

taint propogation

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

Condition code flags

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)    T0
```

instruction bytes

Trace Reading


```
$ trace_reader -intel -trace x.trace -first 993850 -v

(00993850)7814507a: rep movs DWORD PTR es: [edi] ,DWORD PTR ds: [esi] M@0x0267ae7c [0xed012800]
[4] (CR) T1 {12 ()()(1111, 5) (1111, 5) } R@ecx[0x000007f3] [4] (RCW)    T0    M@0x0267f0e0
[0x0cc00b2f] [4] (CW)  T1  {15  (1111, 5)  (1111, 5)  (1111, 5)  (1111, 5) }  ESP: NUM_OP:  5
TID:  1756
TP:  TPSrc  EFLAGS:  0x00000202  CC_OP: 0x00000010  DF: 0x00000001  RAW:  0xf3a5  MEMREGS:
R@edi[0x0267f0e0]  [4] (R)      T0     R@esi  [0x0267ae7c]  [4]  (R)   T0
```

Trace Slicing


```
$ x86_slicer -in-trace z.trace -ctr 209160387 -regloc ax

7c90e4f0:       mov    edx, esp        R@esp[0x087ffe90][4](R) T0
R@edx[0x7ffe0300][4](W) T0
5730805b:       push   edx     R@edx[0x0000ff6c][4](R) T1 {3 (1111, 3)
(1111, 3) ()()} M@0x014ad95c[0x014ad754][4](W)  T0
57307f2b:       mov    ax, BYTE PTR [ebp+0x8]    M@0x014ad95c[0x0000ff6c][2](R)  T1
{3 (1111, 3) (1111, 3) ()()}    R@ax[0x0000ffa8][2](W)  T1 {3 (1111, 3)
(1111, 3) ()()}
```

Trace Slicing


```
$ x86_slicer -in-trace z.trace -ctr 209160387 -regloc ax

7c90e4f0:       mov    edx, esp        R@esp[0x087ffe90][4](R) T0
R@edx[0x7ffe0300][4](W) T0
5730805b:       push   edx     R@edx[0x0000ff6c][4](R) T1 {3 (1111, 3)
(1111, 3) ()()} M@0x014ad95c[0x014ad754][4](W)  T0
57307f2b:       mov    ax, BYTE PTR [ebp+0x8]    M@0x014ad95c[0x0000ff6c][2](R)  T1
{3 (1111, 3) (1111, 3) ()()}    R@ax[0x0000ffa8][2](W)  T1 {3 (1111, 3)
(1111, 3) ()()}
```

slicing target

Trace Slicing


```
$ x86_slicer -in-trace z.trace -ctr 209160387 -regloc ax

7c90e4f0:       mov    edx, esp        R@esp[0x087ffe90][4](R) T0
R@edx[0x7ffe0300][4](W) T0
5730805b:       push   edx     R@edx[0x0000ff6c][4](R) T1 {3 (1111, 3)
(1111, 3) ()()} M@0x014ad95c[0x014ad754][4](W)  T0
57307f2b:       mov    ax, BYTE PTR [ebp+0x8]    M@0x014ad95c[0x0000ff6c][2](R)  T1
{3 (1111, 3) (1111, 3) ()()}    R@ax[0x0000ffa8][2](W)  T1 {3 (1111, 3)
(1111, 3) ()()}
```

ax loaded from this address

Trace Slicing


```
$ x86_slicer -in-trace z.trace -ctr 209160387 -regloc ax

7c90e4f0:       mov    edx, esp        R@esp[0x087ffe90][4](R) T0
R@edx[0x7ffe0300][4](W) T0
5730805b:       push   edx      R@edx[0x0000ff6c][4](R) T1 {3 (1111, 3)
(1111, 3) ()()} M@0x014ad95c[0x014ad754][4](W)  T0
57307f2b:       mov    ax, BYTE PTR [ebp+0x8]    M@0x014ad95c[0x0000ff6c][2](R)  T1
{3 (1111, 3) (1111, 3) ()()}    R@ax[0x0000ffa8][2](W)  T1 {3 (1111, 3)
(1111, 3) ()()}
```

target value written by this instruction

Trace Slicing

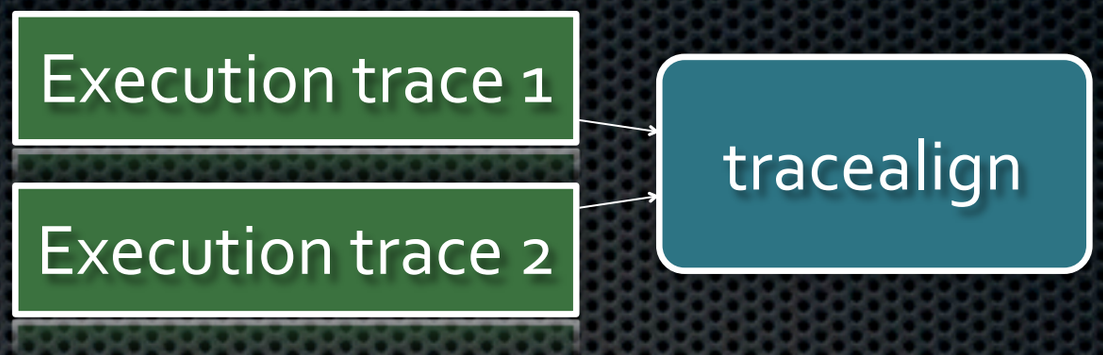

```
$ x86_slicer -in-trace z.trace -ctr 209160387 -regloc ax

7c90e4f0:       mov    edx, esp          R@esp[0x087ffe90][4](R) T0
R@edx[0x7ffe0300][4](W) T0
5730805b:       push   edx      R@edx[0x0000ff6c][4](R) T1 {3 (1111, 3)
(1111, 3) ()()} M@0x014ad95c[0x014ad754][4](W)  T0
57307f2b:       mov    ax, BYTE PTR [ebp+0x8]     M@0x014ad95c[0x0000ff6c][2](R)  T1
{3 (1111, 3) (1111, 3) ()()}    R@ax[0x0000ffa8][2](W)  T1 {3 (1111, 3)
(1111, 3) ()()}
```

edx set by this instruction

Trace Slicing

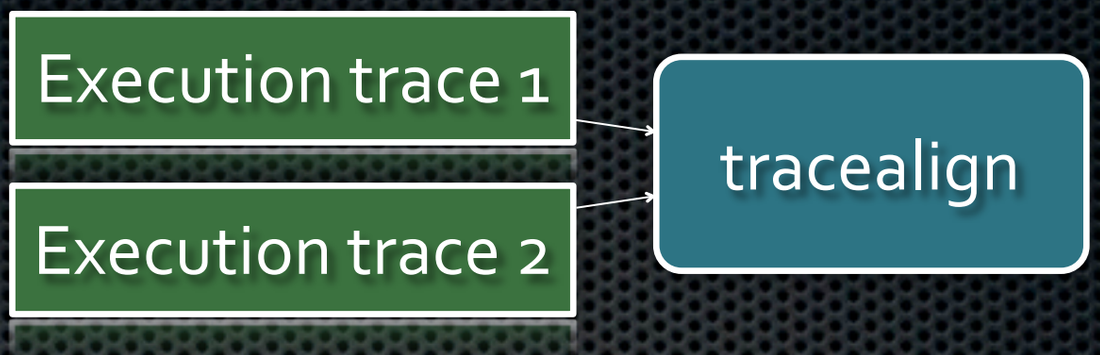

```
$ x86_slicer -in-trace z.trace -ctr 209160387 -regloc ax

7c90e4f0:       mov    edx, esp        R@esp[0x087ffe90][4](R) T0
R@edx[0x7ffe0300][4](W) T0
5730805b:       push   edx     R@edx[0x0000ff6c][4](R) T1 {3 (1111, 3)
(1111, 3) ()()} M@0x014ad95c[0x014ad754][4](W)  T0
57307f2b:       mov    ax, BYTE PTR [ebp+0x8]    M@0x014ad95c[0x0000ff6c][2](R)  T1
{3 (1111, 3) (1111, 3) ()()}    R@ax[0x0000ffa8][2](W)  T1 {3 (1111, 3)
(1111, 3) ()()}
```

Trace Alignment




```
$ align crash.trace noncrash.trace

ALIGNMENT RESULTS:

ALIGNED    @ 0:#00000001-00001362 (00001362 insts)    ~ 1:#00000001-00001362 (00001362 insts)
DISALIGNED @ 0:#00001363-00001683 (00000321 insts) T0 ~ 1:#00001363-00001643 (00000281 insts) T0
ALIGNED    @ 0:#00001684-00002503 (00000820 insts)    ~ 1:#00001644-00002463 (00000820 insts)
DISALIGNED @ 0:#00002504-00003487 (00000984 insts) T1 ~ 1:#00002464-00003344 (00000881 insts) T1
...
```

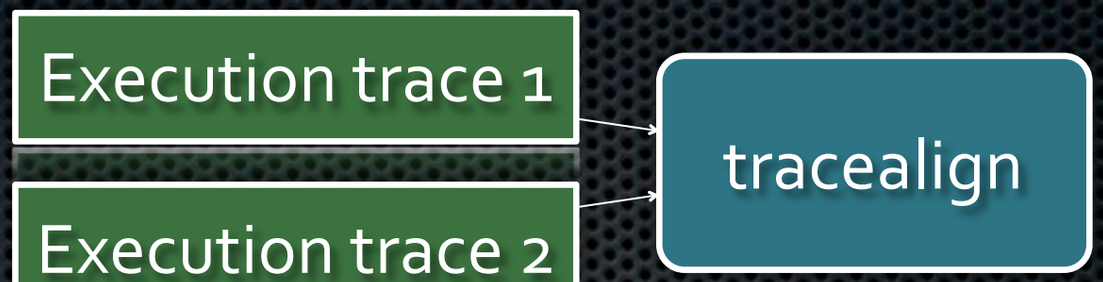
Trace Alignment




```
$ align crash.trace noncrash.trace

ALIGNMENT RESULTS:

ALIGNED    @ 0:#00000001-00001362 (00001362 insts)    ~ 1:#00000001-00001362 (00001362 insts)
DISALIGNED @ 0:#00001363-00001683 (00000321 insts) T0 ~ 1:#00001363-00001643 (00000281 insts) T0
ALIGNED    @ 0:#00001684-00002503 (00000820 insts)    ~ 1:#00001644-00002463 (00000820 insts)
DISALIGNED @ 0:#00002504-00003487 (00000984 insts) T1 ~ 1:#00002464-00003344 (00000881 insts) T1
...
```

aligned region

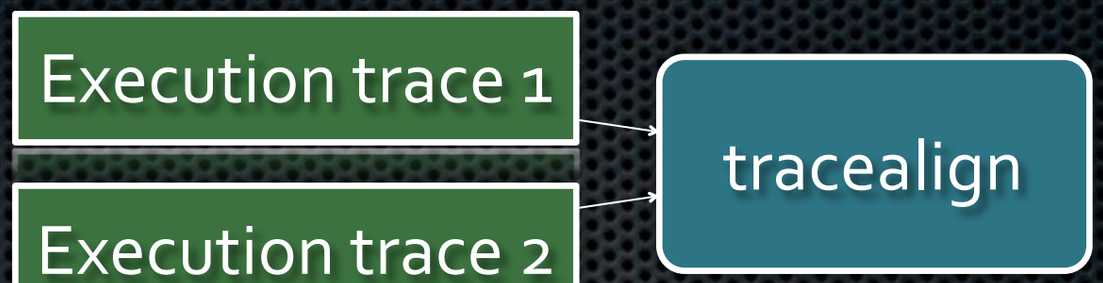
Trace Alignment




```
$ align crash.trace noncrash.trace

ALIGNMENT RESULTS:

ALIGNED    @ 0:#00000001-00001362 (00001362 insts)    ~ 1:#00000001-00001362 (00001362 insts)
DISALIGNED @ 0:#00001363-00001683 (00000321 insts) T0 ~ 1:#00001363-00001643 (00000281 insts) T0
ALIGNED    @ 0:#00001684-00002503 (00000820 insts)    ~ 1:#00001644-00002463 (00000820 insts)
DISALIGNED @ 0:#00002504-00003487 (00000984 insts) T1 ~ 1:#00002464-00003344 (00000881 insts) T1
...
```

divergence point

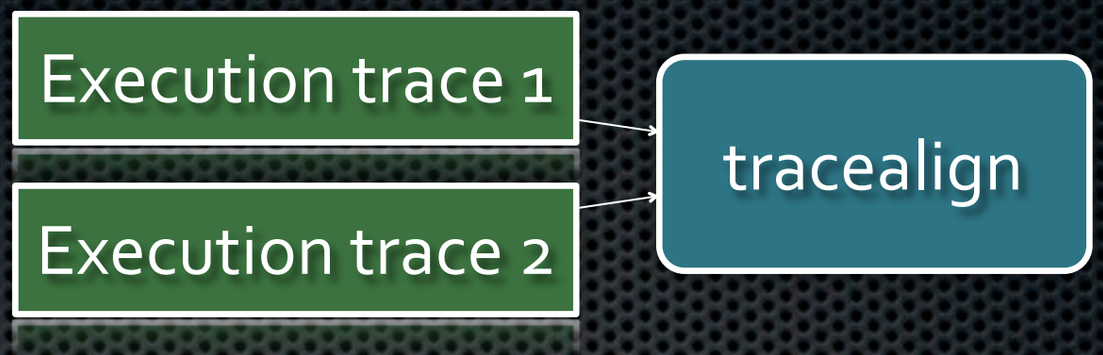
Trace Alignment




```
$ align crash.trace noncrash.trace

ALIGNMENT RESULTS:

ALIGNED    @ 0:#00000001-00001362 (00001362 insts)    ~ 1:#00000001-00001362 (00001362 insts)
DISALIGNED @ 0:#00001363-00001683 (00000321 insts) T0 ~ 1:#00001363-00001643 (00000281 insts) T0
ALIGNED    @ 0:#00001684-00002503 (00000820 insts)    ~ 1:#00001644-00002463 (00000820 insts)
DISALIGNED @ 0:#00002504-00003487 (00000984 insts) T1 ~ 1:#00002464-00003344 (00000881 insts) T1
...
```

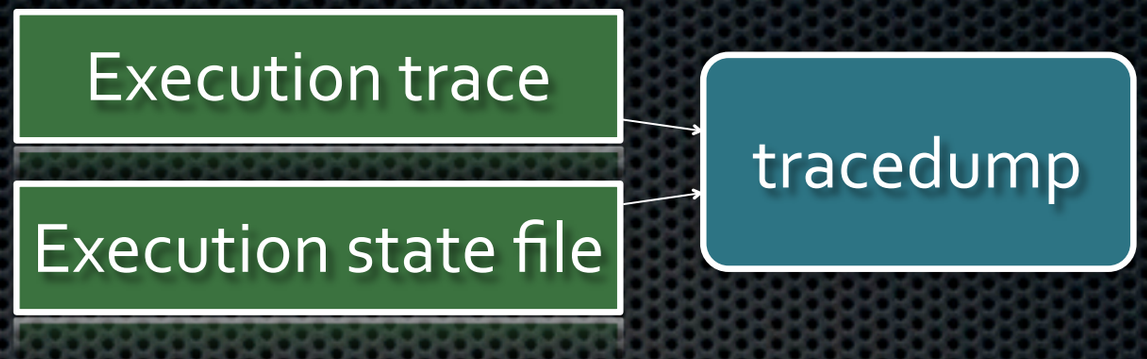
(caused by tainted input)

Trace Alignment



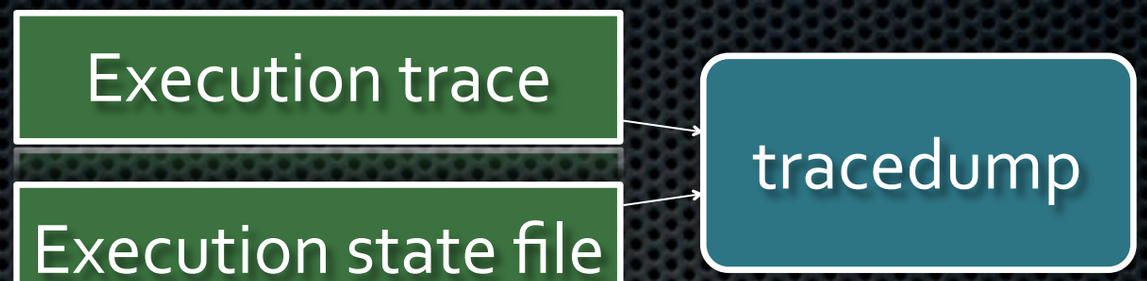

```
$ align crash.trace noncrash.trace

ALIGNMENT RESULTS:

ALIGNED    @ 0:#00000001-00001362 (00001362 insts)    ~ 1:#00000001-00001362 (00001362 insts)
DISALIGNED @ 0:#00001363-00001683 (00000321 insts) T0 ~ 1:#00001363-00001643 (00000281 insts) T0
ALIGNED    @ 0:#00001684-00002503 (00000820 insts)    ~ 1:#00001644-00002463 (00000820 insts)
DISALIGNED @ 0:#00002504-00003487 (00000984 insts) T1 ~ 1:#00002464-00003344 (00000881 insts) T1
...
```

Taint-Enhanced Crash Dumps




```
$ tracedump -trace z.trace -state z.trace.state

Dump at EIP=208063fb,   mov    edx,DWORD PTR [ecx+0x4]
Process: AcroRd32.exe   PID: 364        TID: 368        CTR: 423441674

EAX: 02085ed4   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
     ->
 -12 0207de80   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 -08 000002ec   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 -04 02085eb8   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +00 011dd2d4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +04 020596c4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +08 01f0380c   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +12 00000001   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +16 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)

EBX: 00000010   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
 .....
```

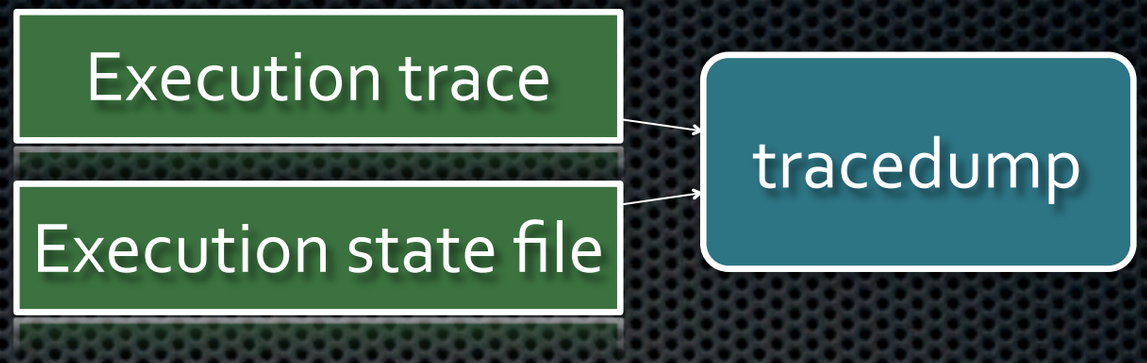
Taint-Enhanced Crash Dumps




```
$ tracedump -trace z.trace -state z.trace.state

Dump at EIP=208063fb,   mov    edx,DWORD PTR [ecx+0x4]
Process: AcroRd32.exe   PID: 364        TID: 368        CTR: 423441674

EAX: 02085ed4   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
     ->
 -12 0207de80   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 -08 000002ec   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 -04 02085eb8   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +00 011dd2d4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +04 020596c4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +08 01f0380c   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +12 00000001   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +16 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)

EBX: 00000010   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
 .....
```

current instruction

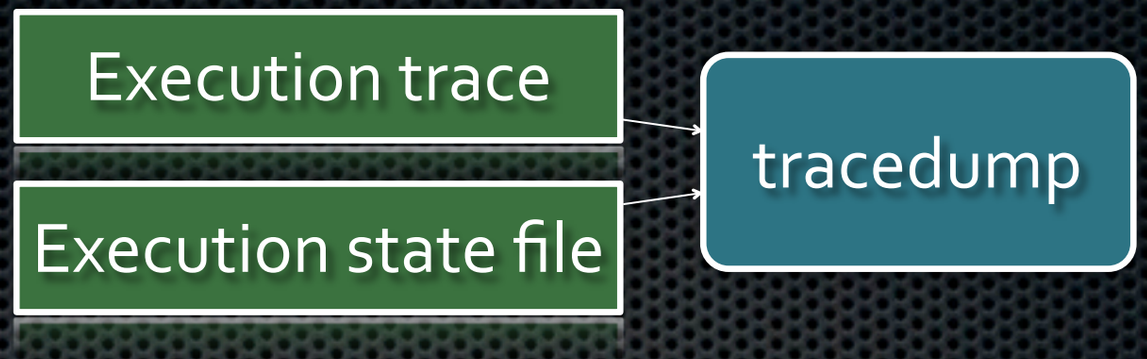
Taint-Enhanced Crash Dumps




```
$ tracedump -trace z.trace -state z.trace.state

Dump at EIP=208063fb,   mov    edx,DWORD PTR [ecx+0x4]
Process: AcroRd32.exe   PID: 364        TID: 368        CTR: 423441674

EAX: 02085ed4   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
     ->
 -12 0207de80   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 -08 000002ec   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 -04 02085eb8   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +00 011dd2d4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +04 020596c4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +08 01f0380c   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +12 00000001   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +16 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)

EBX: 00000010   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
 .....
```

memory near [EAX]

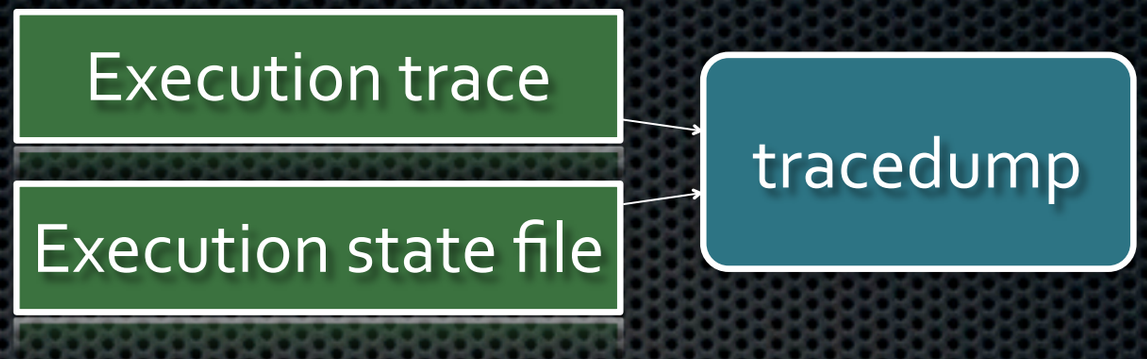
Taint-Enhanced Crash Dumps




```
$ tracedump -trace z.trace -state z.trace.state

Dump at EIP=208063fb,   mov    edx,DWORD PTR [ecx+0x4]
Process: AcroRd32.exe   PID: 364        TID: 368        CTR: 423441674

EAX: 02085ed4   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
     ->
 -12 0207de80   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 -08 000002ec   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 -04 02085eb8   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +00 011dd2d4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +04 020596c4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +08 01f0380c   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +12 00000001   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +16 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)

EBX: 00000010   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
 .....
```

tainted memory

Taint-Enhanced Crash Dumps




```
$ tracedump -trace z.trace -state z.trace.state

Dump at EIP=208063fb,   mov    edx,DWORD PTR [ecx+0x4]
Process: AcroRd32.exe   PID: 364        TID: 368        CTR: 423441674

EAX: 02085ed4   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
     ->
 -12 0207de80   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 -08 000002ec   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 -04 02085eb8   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +00 011dd2d4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +04 020596c4   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +08 01f0380c   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +12 00000001   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +16 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)

EBX: 00000010   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
 .....
```

Allocation Tracing


```
$ alloc_reader -alloc z.trace.alloc -ctr 209430179 -addr 0x07260020

Found 1 buffers
[141406597,end]  0x07260020  (523808)

$ alloc_reader -alloc z.trace.alloc -ctr 209430179


Found 8273 buffers
[00000000,00115090] 0x025d3710 (4)
[00000000,00175927] 0x025d3200 (4)
...
[00010676,00179862] 0x025d36f0 (4)
[00011853,00012729] 0x025d31e0 (4)


$ alloc_reader -alloc z.trace.alloc -dfrees

Double Frees:
 00000000 DFREES [00000000,end] 0x01e009d0 (12)
 ...
 00188196 DFREES [00080542,00188160] 0x025d3160 (4)
```

Allocation Tracing


```
$ alloc_reader -alloc z.trace.alloc -ctr 209430179 -addr 0x07260020

Found 1 buffers
[141406597,end]  0x07260020  (523808)
```

```
$ alloc_reader -alloc z.trace.alloc -ctr 209430179

Found 8273 buffers
[00000000,00115090] 0x025d3710 (4)
[00000000,00175927] 0x025d3200 (4)
...
[00010676,00179862] 0x025d36f0 (4)
[00011853,00012729] 0x025d31e0 (4)
```

```
$ alloc_reader -alloc z.trace.alloc -dfrees

Double Frees:
 00000000 DFREES [00000000,end] 0x01e009d0 (12)
 ...
 00188196 DFREES [00080542,00188160] 0x025d3160 (4)
```

allocation containing desired address

Allocation Tracing


```
$ alloc_reader -alloc z.trace.alloc -ctr 209430179 -addr 0x07260020

Found 1 buffers
[141406597,end]  0x07260020  (523808)

$ alloc_reader -alloc z.trace.alloc -ctr 209430179

Found 8273 buffers
[00000000,00115090] 0x025d3710 (4)
[00000000,00175927] 0x025d3200 (4)
...
[00010676,00179862] 0x025d36f0 (4)
[00011853,00012729] 0x025d31e0 (4)

$ alloc_reader -alloc z.trace.alloc -dfrees

Double Frees:
 00000000 DFREES [00000000,end] 0x01e009d0 (12)
 ...
 00188196 DFREES [00080542,00188160] 0x025d3160 (4)
```

live buffers at given instruction count

Allocation Tracing

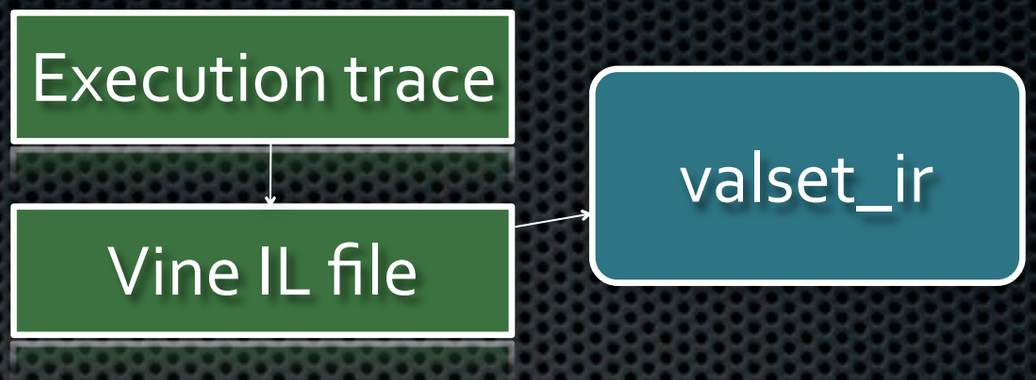

```
$ alloc_reader -alloc z.trace.alloc -ctr 209430179 -addr 0x07260020

Found 1 buffers
[141406597,end]  0x07260020  (523808)

$ alloc_reader -alloc z.trace.alloc -ctr 209430179

Found 8273 buffers
[00000000,00115090] 0x025d3710 (4)
[00000000,00175927] 0x025d3200 (4)
...
[00010676,00179862] 0x025d36f0 (4)
[00011853,00012729] 0x025d31e0 (4)

$ alloc_reader -alloc z.trace.alloc -dfrees

Double Frees:
 00000000 DFREES [00000000,end] 0x01e009d0 (12)
 ...
 00188196 DFREES [00080542,00188160] 0x025d3160 (4)
```

detected double frees

Allocation Tracing

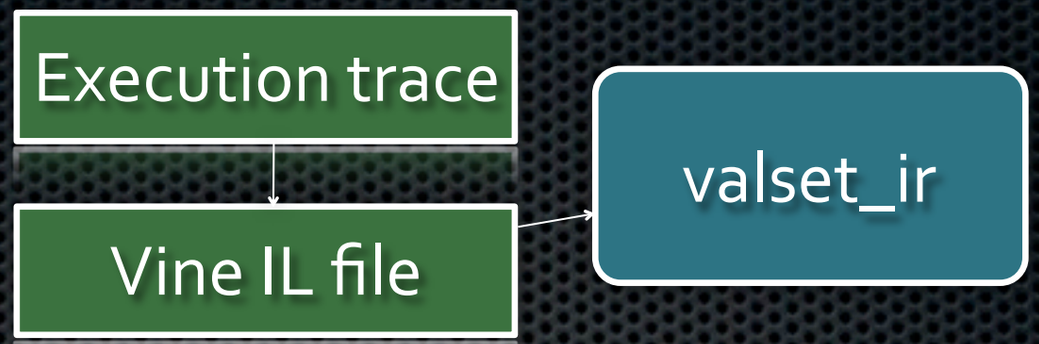

```
$ alloc_reader -alloc z.trace.alloc -ctr 209430179 -addr 0x07260020

Found 1 buffers
[141406597,end]  0x07260020  (523808)

$ alloc_reader -alloc z.trace.alloc -ctr 209430179


Found 8273 buffers
[00000000,00115090] 0x025d3710 (4)
[00000000,00175927] 0x025d3200 (4)
...
[00010676,00179862] 0x025d36f0 (4)
[00011853,00012729] 0x025d31e0 (4)


$ alloc_reader -alloc z.trace.alloc -dfrees

Double Frees:
 00000000 DFREES [00000000,end] 0x01e009d0 (12)
 ...
 00188196 DFREES [00080542,00188160] 0x025d3160 (4)
```

Measuring influence and value sets




```
$ valset_ir -ir-in x-sliced.il -tmp-name R_EDI_3 -
get-val-bounds true -sample-pts 2000

Summary:
Lowest and highest possible values: 0x72f0000 to 0x7370018
Influence bounds after establishing value bounds: 3.321928 to 18.999981
Influence bounds after asking for counterexamples: 6.209453 to 18.999981
Probable influence: 15.954538 (242 hits of 2000 samples in population
524207)
```

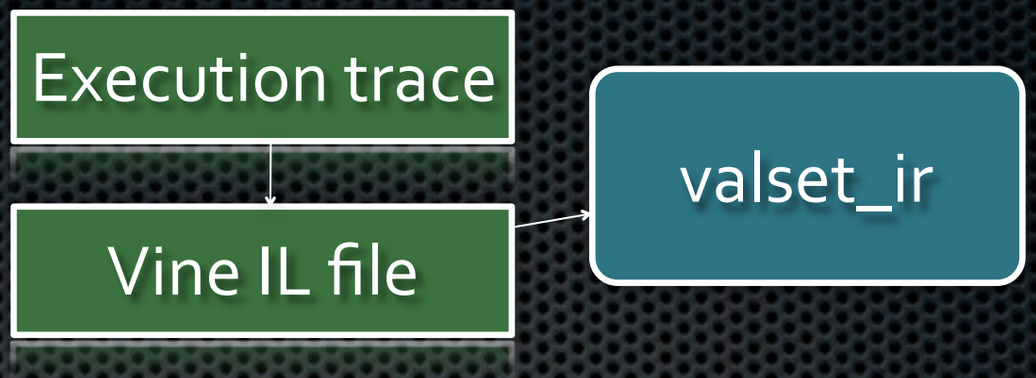
Measuring influence and value sets




```
$ valset_ir -ir-in x-sliced.il -tmp-name R_EDI_3 -
get-val-bounds true -sample-pts 2000

Summary:
Lowest and highest possible values: 0x72f0000 to 0x7370018
Influence bounds after establishing value bounds: 3.321928 to 18.999981
Influence bounds after asking for counterexamples: 6.209453 to 18.999981
Probable influence: 15.954538 (242 hits of 2000 samples in population
524207)
```

value set of EAX

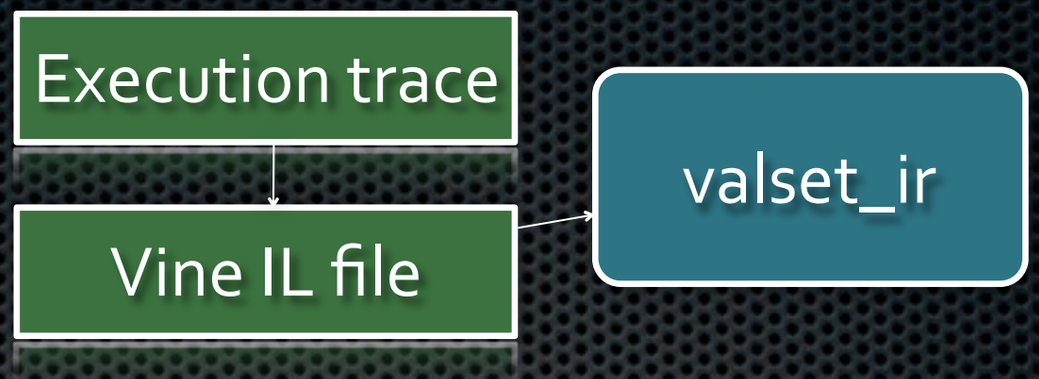
Measuring influence and value sets




```
$ valset_ir -ir-in x-sliced.il -tmp-name R_EDI_3 -
get-val-bounds true -sample-pts 2000

Summary:
Lowest and highest possible values: 0x72f0000 to 0x7370018
Influence bounds after establishing value bounds: 3.321928 to 18.999981
Influence bounds after asking for counterexamples: 6.209453 to 18.999981
Probable influence: 15.954538 (242 hits of 2000 samples in population
524207)
```

quantitative influence

Measuring influence and value sets




```
$ valset_ir -ir-in x-sliced.il -tmp-name R_EDI_3 -
get-val-bounds true -sample-pts 2000

Summary:
Lowest and highest possible values: 0x72f0000 to 0x7370018
Influence bounds after establishing value bounds: 3.321928 to 18.999981
Influence bounds after asking for counterexamples: 6.209453 to 18.999981
Probable influence: 15.954538 (242 hits of 2000 samples in population
524207)
```

More information about these tools is available at: http://bitblaze.cs.berkeley.edu

INVESTIGATOR'S RECEIPT: Tear along perforated

Case Number:

Evidence Bag Sealed by:

Date Sealed:

Description of Enclosed Evidence:

Glue Line

Glue Line

CONTROL NO: M 10015141

EVIDENCE

(TO BE OPENED BY AUTHORIZED PERSONNEL ONLY)

NOTE

A) Do not use this bag for any evidence that has wet/damp body fluids on it.
B) To seal bag, peel off blue release liner, then seal bag by pressing down on red glue line.

Case Number: Adobe Reader

Description of Enclosed Evidence:

Submitting Agency:

Telephone Number:

Evidence Recovered By: (PRINT NAME)

Victim's Full Name:

Suspect's Full Name:

Evidence Bag Sealed By: (PRINT NAME)

(SIGNATURE)

Date Sealed: Time Sealed: AM PM

CHAIN OF CUSTODY

FROM	TO	DATE
Charlie	Noah	2/26/10

FOR CRIME LAB PERSONNEL ONLY

CONDITION OF EVIDENCE BAG UPON RECEIPT AT LAB:

☐ SEALED ☐ OTHER (DESCRIBE)

CRIME LAB CASE NO:

NOTES:

↑ CHECK BORDER PATTERN FOR EVIDENCE OF SEAM TAMPERING ↑

↑ CHECK BORDER PATTERN FOR EVIDENCE OF SEAM TAMPERING ↑

Wednesday, July 28, 2010

Dumb fuzzing of Reader

- 3 weeks, 1515 seed files, 3 million test cases
- Crashes at 100 different EIP's
- Approx 33 unique crashes
- 4 Exploitable, 8 Probably Exploitable, 17 Unknown, 4 Probably Not Exploitable

!exploitable vs BitBlaze

- !exploitable assumes everything is tainted and only analyzes the basic block the crash occurs in
- BitBlaze has access to full taint information and can slice back to the time the test case was input
- However, BitBlaze is waaaaay slower
  - Trace can take many hours
  - Slices can take from 10 sec to 90 min

BitBlaze strengths/weakness

- Very good at eliminating Null-ptr derefs
  - 6 of the 17 Unknowns quickly found to be not exploitable
- Gives crash reports with taint information
- Not so good at invalid reads
  - Cannot tell what would happen next

```
Dump at EIP=20a2fd72,     mov    (%eax),%eax
Process: AcroRd32.exe     PID: 1908    TID: 1192    CTR: 132684549

Backtrace:
 (132684549)  unknown                0x20a2fd72  sub_20A2FD6E(?) +4
 (132684547)  unknown                0x20889825  sub_2088980F(?) +22
 (132684534)  unknown                0x2088b1d5  sub_2088B1BC(?) +25
 (132684519)  unknown                0x2088c3cf  sub_2088C3C8(?) +7
 (132684515)  unknown                0x208f4535  sub_208F4525(?) +16
 (132684506)  unknown                0x208f5a5d  sub_208F5A42(?) +27
 (132684478)  unknown                0x208f7087  sub_208F6FD4(?) +179
 (132668130)  unknown                0x208ca772  sub_208CA547(?) +555
 (96842557)  unknown                0x208774bb  sub_208773B3(?) +264
 (96833560)  AcroRd32.dll           0x009f3480  sub_9F3414(?) +108
...

EAX: 00000000
     -> N/A
EBX: 00000001
     -> N/A
ECX: 0268e354
     ->
 -12 00000000   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 -08 02085ed4   ( 1111,43991)   ( 1111,43991)   ( 1111,43991)   ( 1111,43991)
 -04 00000001   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 +00 7439574f   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 +04 02057238   ( 1111,43991)   ( 1111,43991)   ( 1111,43991)   ( 1111,43991)
 +08 00000003   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 +12 32526963   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 +16 76455349   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)

EDX: 01e3c0c0   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
     ->
 -12 00000000   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 -08 00e9008d   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 -04 00e90031   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 +00 001620c8   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 +04 ffffffff   ( 1111,43991)   ( 1111,43991)   ( 1111,43991)   ( 1111,43991)
 +08 00000000   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 +12 00000000   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
 +16 00000000   (NO TAINT)      (NO TAINT)      (NO TAINT)      (NO TAINT)
...
```

```
Dump at EIP=08036f88,   mov    0x8(%edx),%ah
Process: AcroRd32.exe   PID: 316   TID: 324   CTR: 358264925

EAX: 00000000
     -> N/A
EBX: 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
ECX: 00000827   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
EDX: 0aaf90ef   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
ESI: 0aaf90fb
     -> N/A
EDI: 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     -> N/A
EBP: 0012d158   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
     ->
 -12 0012d798   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 -08 08181bc8   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 -04 00000002   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +00 0012d158   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +04 0005760b   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +08 02823aa4   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +12 0012d824   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +16 02823bd4   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)

ESP: 0012ce04
     ->
 -12 397875ab   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 -08 ac8b0000   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 -04 20140000   ( 1111,43991)  ( 1111,43991)  ( 1111,43991)  ( 1111,43991)
 +00 0012ce0c   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +04 0012ce28   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +08 c0000005   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +12 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
 +16 00000000   (NO TAINT)     (NO TAINT)     (NO TAINT)     (NO TAINT)
```

An unknown crash

```
208063f8 c20400          ret     4
208063fb 8b5104          mov     edx,dword ptr [ecx+4] ds:0023:00000014=????????
208063fe 85d2            test    edx,edx
20806400 7503            jne     AcroForm!PlugInMain+0x4203 (20806405)
```

Command

```
ModLoad: 77120000 771ab000   C:\WINDOWS\system32\OLEAUT32.dll
(16c.3e4): Access violation - code c0000005 (!!! second chance !!!)
eax=024e9a24 ebx=00000010 ecx=00000010 edx=00000000 esi=00000000 edi=000000d4
eip=208063fb esp=0012e434 ebp=0012e4c4 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** WARNING: Unable to verify checksum for C:\Program Files\Adobe\Reader 9.0\Reader\plug_ins\Acro
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Adob
AcroForm!PlugInMain+0x41f9:
208063fb 8b5104          mov     edx,dword ptr [ecx+4] ds:0023:00000014=????????
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Adob
*** WARNING: Unable to verify checksum for C:\Program Files\Adobe\Reader 9.0\Reader\plug_ins\Anno
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Adob
0:000> !exploitable
Exploitability Classification: UNKNOWN
Recommended Bug Title: Data from Faulting Address controls Branch Selection starting at AcroForm!

The data from the faulting address is later used to determine whether or not a branch is taken.
```

Looks like a null-ptr dereference.

But what if the 0x10 came from a wild read, uninitialized variable, or from our input?

BitBlaze to the rescue

```
$ trace_reader -trace z.trace -header | grep Number
Number of instructions: 272998
$ trace_reader_intel -v -trace z.trace -first 272998
208063fb:    mov    edx,DWORD PTR [ecx+0x4]     M@0x00000014[0x00000000][4](R)
    T0  R@edx[0x00000000][4](W)   T0 ESP:  NUM OP: 4 TID: 1712 TP: TPNone EFLAGS:
0x00000083 CC_OP: 0x00000008 DF: 0x00000001 RAW: 0x8b5104 MEMREGS:  R@ecx[0x00000010]
[4](R)T0
```

Note: `ecx` is not tainted

Slice on ecx

```
$ x86_slicer -in-trace z.trace -ctr 272998 -regloc ecx > /dev/null
$ trace_reader_intel -trace ECX_0.trace
208f46df: xor    esi,esi   R@esi[0xc0000000][4](R) T0  R@esi[0xc0000000][4]
(RW)   T0
208f4bbf: mov    eax,esi   R@esi[0x00000000][4](R) T0  R@eax[0x00000b12][4](W) T0
208f5c6b: mov    ebx,eax   R@eax[0x00000000][4](R) T0  R@ebx[0x025cdf3c][4](W) T0
208f5c91: mov    eax,ebx   R@ebx[0x00000000][4](R) T0  R@eax[0x00000000][4](W) T0
208f63f8: mov    esi,eax   R@eax[0x00000000][4](R) T0  R@esi[0xc0000000][4](W) T0
208f648f: mov    ecx,esi   R@esi[0x00000000][4](R) T0  R@ecx[0x00000000][4](W) T0
2088c9a2: mov    esi,ecx   R@ecx[0x00000000][4](R) T0  R@esi[0x00000000][4](W) T0
2088c9a4: lea    ebx,[esi+0x10]  A@0x00000010[0x00000000][4](R)    T0  R@ebx
[0xc0000000][4](W)  T0
2088c9a7: mov    ecx,ebx   R@ebx[0x00000010][4](R) T0  R@ecx[0x00000000][4](W) T0
```

ecx did not come from memory.

This is not exploitable!

(Note: push/pop pairs removed for clarity)

Good vs evil file

```
$ trace_reader_intel -trace z.trace -first 272998 -last 272998
208063fb: mov    edx,DWORD PTR [ecx+0x4]    M@0x00000014[0x00000000][4](R)
   T0  R@edx[0x00000000][4](W) T0

$ aligned.pl y_z.aligned.txt T1:272998
<T0:271645> ~ T1:272998
(T0:00271451-00271645 ~ T1:00272804-00272998)

$ trace_reader_intel -trace y.trace -first 271645 -last 271645
208063fb: mov    edx,DWORD PTR [ecx+0x4]    M@0x0202c5fc[0x025d91f8][4](R)
   T0  R@edx[0x00000000][4](W) T0
```

The good value get set

```
$ x86_slicer -in-trace y.trace -ctr 271645 -regloc ecx > /dev/null
$ trace_reader_intel -trace ECX_0.trace
94433a:   mov    eax,DWORD PTR [ecx+0x4]   M@0x0039b57c[0x0202c5e8][4](R)   T0
   R@eax[0x00000014][4](W) T0
944341:   mov    esi,eax   R@eax[0x0202c5e8][4](R) T0  R@esi[0x0039b2d0][4](W) T0
944407:   mov    eax,esi   R@esi[0x0202c5e8][4](R) T0  R@eax[0x00000000][4](W) T0
20803c44: mov    esi,eax   R@eax[0x0202c5e8][4](R) T0  R@esi[0x00000000][4](W) T0
20803c59: mov    eax,esi   R@esi[0x0202c5e8][4](R) T0  R@eax[0x0202c5e8][4](W) T0
20804cee: mov    esi,eax   R@eax[0x0202c5e8][4](R) T0  R@esi[0x00000000][4](W) T0
20804d01: mov    eax,esi   R@esi[0x0202c5e8][4](R) T0  R@eax[0x0202c5e8][4](W) T0
208f4a48: mov    ecx,eax   R@eax[0x0202c5e8][4](R) T0  R@ecx[0x00000001][4](W) T0
208f35f7: mov    esi,ecx   R@ecx[0x0202c5e8][4](R) T0  R@esi[0x00000000][4](W) T0
208f3610: mov    eax,esi   R@esi[0x0202c5e8][4](R) T0  R@eax[0x0202c5e8][4](W) T0
208f4ad9: mov    esi,eax   R@eax[0x0202c5e8][4](R) T0  R@esi[0x00000000][4](W) T0
208f4bbf: mov    eax,esi   R@esi[0x0202c5e8][4](R) T0  R@eax[0x20d6db64][4](W) T0
208f5c6b: mov    ebx,eax   R@eax[0x0202c5e8][4](R) T0  R@ebx[0x025cdf0c][4](W) T0
208f5c91: mov    eax,ebx   R@ebx[0x0202c5e8][4](R) T0  R@eax[0x00000000][4](W) T0
208f63f8: mov    esi,eax   R@eax[0x0202c5e8][4](R) T0  R@esi[0xc0000000][4](W) T0
208f648f: mov    ecx,esi   R@esi[0x0202c5e8][4](R) T0  R@ecx[0x00000000][4](W) T0
2088c9a2: mov    esi,ecx   R@ecx[0x0202c5e8][4](R) T0  R@esi[0x0202c5e8][4](W) T0
2088c9a4: lea    ebx,[esi+0x10]   A@0x0202c5f8[0x00000000][4](R)   T0  R@ebx
[0xc0000000][4](W)  T0
2088c9a7: mov    ecx,ebx   R@ebx[0x0202c5f8][4](R) T0  R@ecx[0x0202c5e8][4](W) T0
```

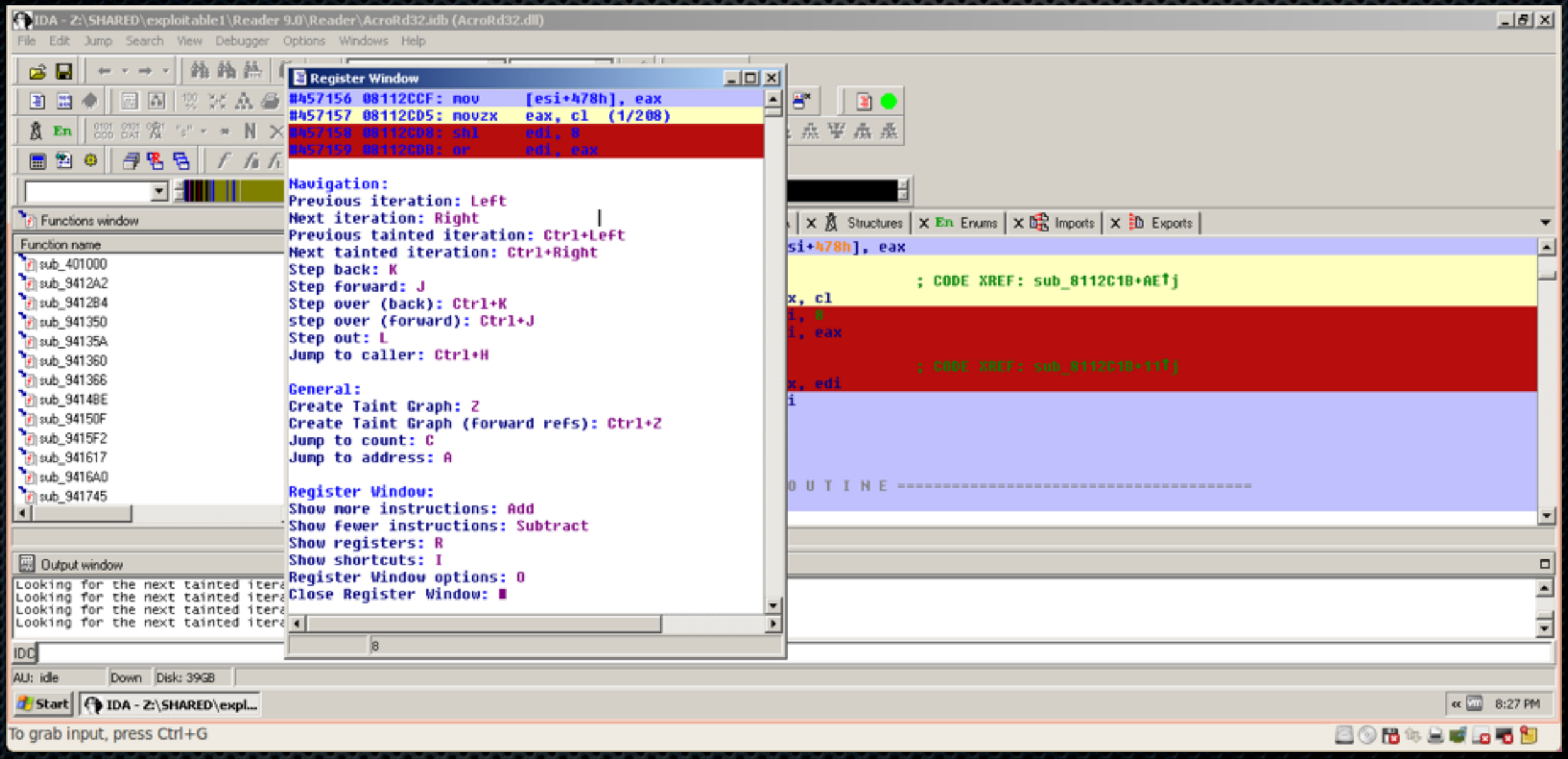
Filled in at 0x208f35f7 (instruction counter 266161)

Alignment

```
$ tail -6 y_z.aligned.txt
ALIGNED    @ T0:#00234897-00265964 (00031068 insts) ~ T1:#00241546-00272613 (00031068 insts)
DISALIGNED @ T0:#00265965-00271264 (00005300 insts) ~ T1:#00272614-00272619 (00000006 insts)
ALIGNED    @ T0:#00271265-00271448 (00000184 insts) ~ T1:#00272620-00272803 (00000184 insts)
DISALIGNED @ T0:#00271449-00271450 (00000002 insts) ~
ALIGNED    @ T0:#00271451-00271645 (00000195 insts) ~ T1:#00272804-00272998 (00000195 insts)
DISALIGNED @ T0:#00271646-00354492 (00082847 insts) ~
```

The correct value is set in this second to last unaligned region

Demo (Visualization tool)



“exploitable” Reader crash

```
ModLoad: 77920000 77a13000   C:\WINDOWS\system32\SETUPAPI.dll
ModLoad: 02400000 026c5000   C:\WINDOWS\system32\xpsp2res.dll
ModLoad: 22100000 225aa000   C:\Program Files\Adobe\Reader 9.0\Reader\plug_ins\Annots.api
ModLoad: 76fd0000 7704f000   C:\WINDOWS\system32\CLBCATQ.DLL
ModLoad: 77050000 77115000   C:\WINDOWS\system32\COMRes.dll
ModLoad: 77120000 771ab000   C:\WINDOWS\system32\OLEAUT32.dll
(e90.458): Access violation - code c0000005 (!!! second chance !!!)
eax=00000000 ebx=02b69e40 ecx=036bf6cc edx=e1140d83 esi=06003eaf edi=00000000
eip=e1140d83 esp=036bf68c ebp=02b69e00 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
e1140d83 ??              ???
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Adob
0:006> !exploitable
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - Data Execution Prevention Violation starting at Unknown Symb

User mode DEP access violations are exploitable.
```

What is the problem?

```
$ trace_reader -trace z.trace -header | grep Number
Number of instructions: 1816519

$ trace_reader_intel -trace z.trace -count -first 1816519
(01816519)603e598:  call   edx   R@edx[0xe1140d83][4](R)T0  M@0x02ecf68c
[0x02699024][4](W)  T0

$ aligned.pl y_z.aligned.txt T1:01816518
<T0:2616698> ~ T1:01816518
(T0:02615515-02616698 ~ T1:01815335-01816518)

$ trace_reader_intel -trace y.trace -count -first 2616698 -last 2616698
(02616698)603e598:  call   edx   R@edx[0x08046e5d][4](R)T0  M@0x02e5f68c
[0x0269900c][4](W)  T1 {15 (1111, 5) (1111, 5) (1111, 5) (1111, 5) }
```

`edx` is not tainted, but is clearly not valid in the bad execution trace

Slice edx

```
$ x86_slicer -in-trace z.trace -ctr 1816519 -regloc edx > /dev/null

$ trace_reader_intel -trace EDX_0.trace
603e58f:  mov    edx,DWORD PTR [esi+0x48]  M@0x06003ef7[0xe1140d83]
[4](R) T0  R@edx[0x0269904c][4](W) T0
```

edx came from *0x6003ef7 but slice stops there

Value at that address is either set before we traced or is uninitialized

Slice `esi` from there

```
$ x86_slicer -in-trace z.trace -ctr 01816515 -regloc esi > /dev/null
$ trace_reader_intel -trace ESI_0.trace
6003eaa:  call   0x0000000006066d7e  J@0x00000000[0x00062ed4][4](R)   T0  M@0x02ecf738
[0x0652e260][4](W)  T0
6066d88:  pop    ecx   M@0x02ecf738[0x06003eaf][4](R)   T0  R@ecx[0x02ecf7e0][4](W) T0
6066d90:  push   ecx   R@ecx[0x06003eaf][4](R) T0  M@0x02ecf75c[0x02ecf7ec][4](W)   T0
6064bab:  mov    eax,DWORD PTR [edi+0x4]   M@0x02ecf75c[0x06003eaf][4](R)   T0  R@eax
[0x02ecf7b4][4](W)  T0
6064bae:  mov    DWORD PTR [esi+0x4],eax   R@eax[0x06003eaf][4](R) T0  M@0x02ecf7b8
[0x00000000][4](W)  T0
603e583:  mov    esi,DWORD PTR [esi+0x4]   M@0x02ecf7b8[0x06003eaf][4](R)   T0  R@esi
[0x02ecf7b4][4](W)  T0
```

`esi` comes from a value pushed onto the stack during a function call, i.e. is an uninitialized stack value.

“exploitable” OpenOffice

Disassembly

Offset: @$scopeip — Previous — Next

```
78544626 fc              cld
78544627 8975f8          mov     dword ptr [ebp-8],esi
7854462a 8b750c          mov     esi,dword ptr [ebp+0Ch]
7854462d 8b7d08          mov     edi,dword ptr [ebp+8]
78544630 8b4d10          mov     ecx,dword ptr [ebp+10h]
78544633 c1e907          shr     ecx,7
78544636 eb06            jmp     MSVCR90!_wcwild+0xe2 (7854463e)
78544638 8d9b00000000    lea     ebx,[ebx]
7854463e 660f6f06        movdqa  xmm0,xmmword ptr [esi]
78544642 660f6f4e10      movdqa  xmm1,xmmword ptr [esi+10h]
78544647 660f6f5620      movdqa  xmm2,xmmword ptr [esi+20h]
7854464c 660f6f5e30      movdqa  xmm3,xmmword ptr [esi+30h]
78544651 660f7f07        movdqa  xmmword ptr [edi],xmm0 ds:0023:195301e0=????????????
78544655 660f7f4f10      movdqa  xmmword ptr [edi+10h],xmm1
7854465a 660f7f5720      movdqa  xmmword ptr [edi+20h],xmm2
7854465f 660f7f5f30      movdqa  xmmword ptr [edi+30h],xmm3
78544664 660f6f6640      movdqa  xmm4,xmmword ptr [esi+40h]
78544669 660f6f6e50      movdqa  xmm5,xmmword ptr [esi+50h]
7854466e 660f6f7660      movdqa  xmm6,xmmword ptr [esi+60h]
78544673 660f6f7e70      movdqa  xmm7,xmmword ptr [esi+70h]
78544678 660f7f6740      movdqa  xmmword ptr [edi+40h],xmm4
7854467d 660f7f6f50      movdqa  xmmword ptr [edi+50h],xmm5
78544682 660f7f7760      movdqa  xmmword ptr [edi+60h],xmm6
78544687 660f7f7f70      movdqa  xmmword ptr [edi+70h],xmm7
```

```
(f34.b20): C++ EH exception - code e06d7363 (first chance)
(f34.b20): C++ EH exception - code e06d7363 (first chance)
(f34.b20): C++ EH exception - code e06d7363 (first chance)
(f34.b20): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=195301e0 ebx=193b01e0 ecx=00000ffc edx=00000000 esi=193b01e0 edi=195301e0
eip=78544651 esp=014ad698 ebp=014ad6a0 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010216
MSVCR90!_wcwild+0xf5:
78544651 660f7f07        movdqa  xmmword ptr [edi],xmm0 ds:0023:195301e0=????????????
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
0:000> !exploitable
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Progr
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Progr
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Progr
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Progr
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Progr
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Progr
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Progr
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Progr
*** ERROR: Module load completed but symbols could not be loaded for soffice.exe
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - User Mode Write AV starting at MSVCR90!_wcwild+
User mode write access violations that are not near NULL are exploitable.
```

This is probably OpenOffice

The crash

```
$ trace_reader -trace z.trace -header | grep Number
Number of instructions: 209430179
$ trace_reader_intel -v -trace z.trace -first 209430179
7855ab1a: rep movs DWORD PTR es:[edi],DWORD PTR ds:[esi] M@0x072602c0[0x00001bcf][4]
(CR)   T0 R@ecx[0x0001fee0][4](RCW)  T1 {15 (1111, 3) (1111, 3) (1111, 3) (1111,
3) }   M@0x07360000[0x00000000][4](CW)   T0 ESP:  NUM_OP: 5 TID: 1380 TP: TPSrc
EFLAGS: 0x00000083 CC_OP: 0x00000010 DF: 0x00000001 RAW: 0xf3a5 MEMREGS:  R@edi
[0x07360000][4](R)  T1 {15 (1111, 3) (1111, 3) (1111, 3) (1111, 3) }  R@esi
 0x072602c0][4](R)  T0
```

Crash in memcpy. Length is tainted.
Destination address is tainted.

Valset analysis on destination

```
$ x86_slicer -in-trace z.trace -ctr 209430179 -regloc edi > /dev/null
$ cp EDI_0.trace z-sliced.trace
$ dynslicer -concall -nomemconstraints -ir z-sliced.trace
$ valset ir -ir-in z-sliced.ir.concall -tmp-name R_EDI_3 -get-val-bounds true -sample-pts 2000
...
Summary:
Lowest and highest possible values: 0x72e0020 to 0x7360018
Influence bounds after establishing value bounds: 3.321928 to 18.999981
Influence bounds after asking for counterexamples: 6.209453 to 18.999981
Probable influence: 16.012888 (252 hits of 2000 samples in population 524207)
```

Attacker has influence on approximately 2 bytes of the destination address

“Backtrace”

```
$ trace_reader_intel -trace z.trace -last 209430179 -count | grep -v ")78" | tail

(209429976)57308071:   mov    eax,DWORD PTR [ebp+0xc]    M@0x014ad988[0x0000ffc4][4](R)    T1
{3 (1111, 3) (1111, 3) ()()}  R@eax[0x072eff6c][4](W) T1 {3 (1111, 3) (1111, 3) ()()}

(209429977)57308074:   shl    eax,0x3   I@0x00000000[0x00000003][1](R)    T0  R@eax
[0x0000ffc4][4](RW) T1 {3 (1111, 3) (1111, 3) ()()}

(209429978)57308077:   push   eax    R@eax[0x0007fe20][4](R) T1 {15 (1111, 3) (1111, 3)
(1111, 3) (1111, 3) }  M@0x014ad960[0x00000001][4](W)    T0

(209429979)57308078:   mov    eax,DWORD PTR [esi] M@0x08a34718[0x072e0020][4](R)    T0  R@eax
[0x0007fe20][4](W)  T1 {15 (1111, 3) (1111, 3) (1111, 3) (1111, 3) }

(209429980)5730807a:   push   DWORD PTR [edi] M@0x08a348c8[0x07260020][4](R)    T0
   M@0x014ad95c[0x0000ff6c][4](W)    T1 {3 (1111, 3) (1111, 3) ()()}

(209429981)5730807c:   movzx  ebx,bx R@bx[0x0000ffa8][2](R)  T1 {3 (1111, 3) (1111, 3) ()
()} R@ebx[0x0000ffa8][4](W) T1 {3 (1111, 3) (1111, 3) ()()}

(209429982)5730807f:   lea    eax,[eax+ebx*8] A@0x0735fd60[0x00000000][4](R)    T0  R@eax
[0x072e0020][4](W)  T0

(209429983)57308082:   push   eax    R@eax[0x0735fd60][4](R) T1 {15 (1111, 3) (1111, 3)
(1111, 3) (1111, 3) }  M@0x014ad958[0x57308066][4](W)    T0

(209429984)57308083:   call   0x0000000057349d30  J@0x00000000[0x00041cad][4](R)    T0
   M@0x014ad954[0x014ad97c][4](W)    T0
(209429985)57349d30:   jmp    DWORD PTR ds:0x573511d4   M@0x573511d4[0x7855aac0][4](R)    T0
```

The memcpy source

```
memcpy(0x0735fd60, 0x07260020, 0x0007fe20);
```

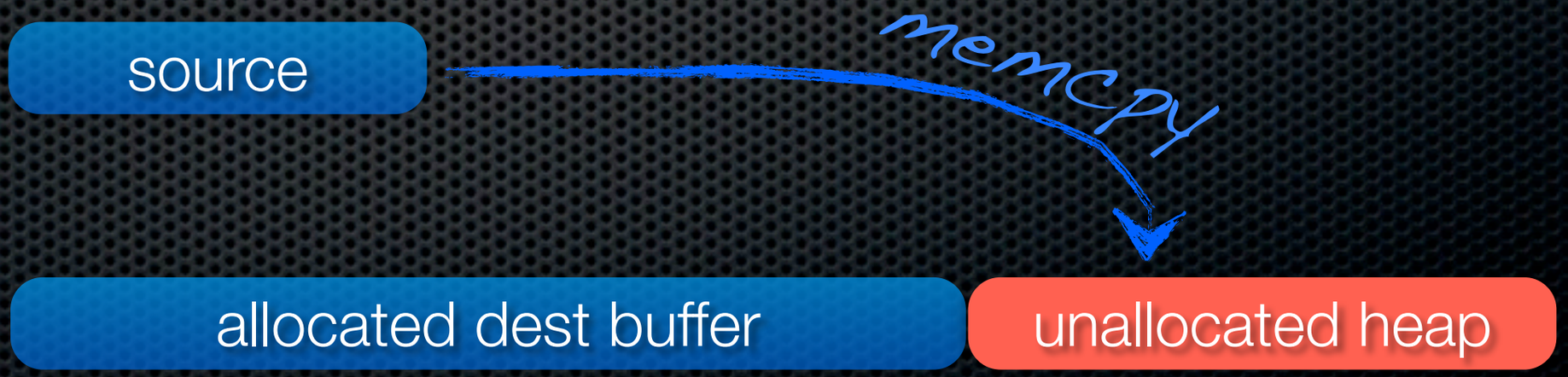
```
$ alloc_reader -alloc z.trace.alloc -ctr 209430179 -addr 0x07260020 -closest 1
Found 1 buffers
[141406597,end] 0x07260020 (523808)
```

Memcpy source is 0x7260020
Allocated with size 523808 (0x7fe20)
Allocated at instruction counter 141406597

The memcpy destination

```
$ alloc_reader -alloc z.trace.alloc -ctr 209429984 -addr 0x0735fd60 -closest 1
Found 0 buffers. Closest are:
  [209160625,end] 0x072e0020 (523104) Dist: 481
```

Hmm...dest is already past the end of a buffer



Destination origin

```
$ trace_reader_intel -trace z.trace -first 209429982 -last 209429982 -count -v
(209429982)5730807f:    lea     eax,[eax+ebx*8]  A@0x0735fd60[0x00000000][4](R)    T0
    R@eax[0x072e0020][4](W) T0 ESP:  NUM_OP: 5 TID: 1380 TP: TPSrc EFLAGS: 0x00000202
CC_OP: 0x00000024 DF: 0x00000001 RAW: 0x8d04d8 MEMREGS:  R@eax[0x072e0020][4](R) T0
    R@ebx[0x0000ffa8][4](R) T1 {3 (1111, 3) (1111, 3) ()()}
```

Destination is 0x72e0020 (which has size 0x7fb60)
plus 8*0xffa8

Either the allocation size or offset is bad (or both)

Allocation size origin

```
$ trace_reader_intel -trace z.trace -last 209160625 -count | grep -v ")7c" | grep -v ")
78" | tail -6

(209160387)57307f43:    movzx   edi,ax R@ax[0x0000ff6c][2](R)  T1 {3 (1111, 3) (1111, 3)
()()}  R@edi[0x0000ffa8][4](W) T1 {3 (1111, 3) (1111, 3) ()()}

(209160388)57307f46:    mov     ebx,edi   R@edi[0x0000ff6c][4](R) T1 {3 (1111, 3) (1111,
3) ()()}  R@ebx[0x0000ffa8][4](W) T1 {3 (1111, 3) (1111, 3) ()()}

(209160389)57307f48:    shl     ebx,0x3   I@0x00000000[0x00000003][1](R)    T0  R@ebx
[0x0000ff6c][4](RW) T1 {3 (1111, 3) (1111, 3) ()()}

(209160390)57307f4b:    push    ebx    R@ebx[0x0007fb60][4](R) T1 {15 (1111, 3) (1111, 3)
(1111, 3) (1111, 3) }  M@0x014ad940[0x0000ffd0][4](W)    T0

(209160391)57307f4c:    call    0x0000000057349d2a  J@0x00000000[0x00041dde][4](R)
   T0  M@0x014ad93c[0x089a6000][4](W)    T0

(209160392)57349d2a:    jmp     DWORD PTR ds:0x573511d8    M@0x573511d8[0x78583bb3][4]
(R) T0
```

Allocation size is 8*ax

Slice ax

```
$ x86_slicer -in-trace z.trace -ctr 209160387 -regloc ax > /dev/null
$ trace_reader_intel -trace EAX_0.trace  | tail -6

5730803a: mov    ecx,DWORD PTR [ebp+0xc]   M@0x014ad988[0x0000ffc4][4](R)    T1 {3 (1111,
3) (1111, 3) ()()}  R@ecx[0x08a34718][4](W) T0

57308043: movzx  eax,WORD PTR [esi+0x8] M@0x08a34720[0x0000ffa8][2](R)    T1 {3 (1111, 3)
(1111, 3) ()()}  R@eax[0x0000ffc4][4](W) T1 {3 (1111, 3) (1111, 3) ()()}

57308047: lea    edx,[eax+ecx*1]  A@0x0001ff6c[0x00000000][4](R)    T0  R@edx[0x00000000]
[4](W) T1 {3 (1111, 3) (1111, 3) ()()}

5730804e: movzx  edx,dx R@dx[0x0000ff6c][2](R)  T1 {3 (1111, 3) (1111, 3) ()()}  R@edx
[0x0001ff6c][4](W)  T1 {15 (1111, 3) (1111, 3) (1111, 3) (1111, 3) }

5730805b: push   edx    R@edx[0x0000ff6c][4](R) T1 {3 (1111, 3) (1111, 3) ()()}
   M@0x014ad95c[0x014ad754][4](W)    T0

57307f2b: mov    ax,WORD PTR [ebp+0x8]  M@0x014ad95c[0x0000ff6c][2](R)    T1 {3 (1111, 3)
(1111, 3) ()()}  R@ax[0x0000ffa8][2](W)  T1 {3 (1111, 3) (1111, 3) ()()}
```

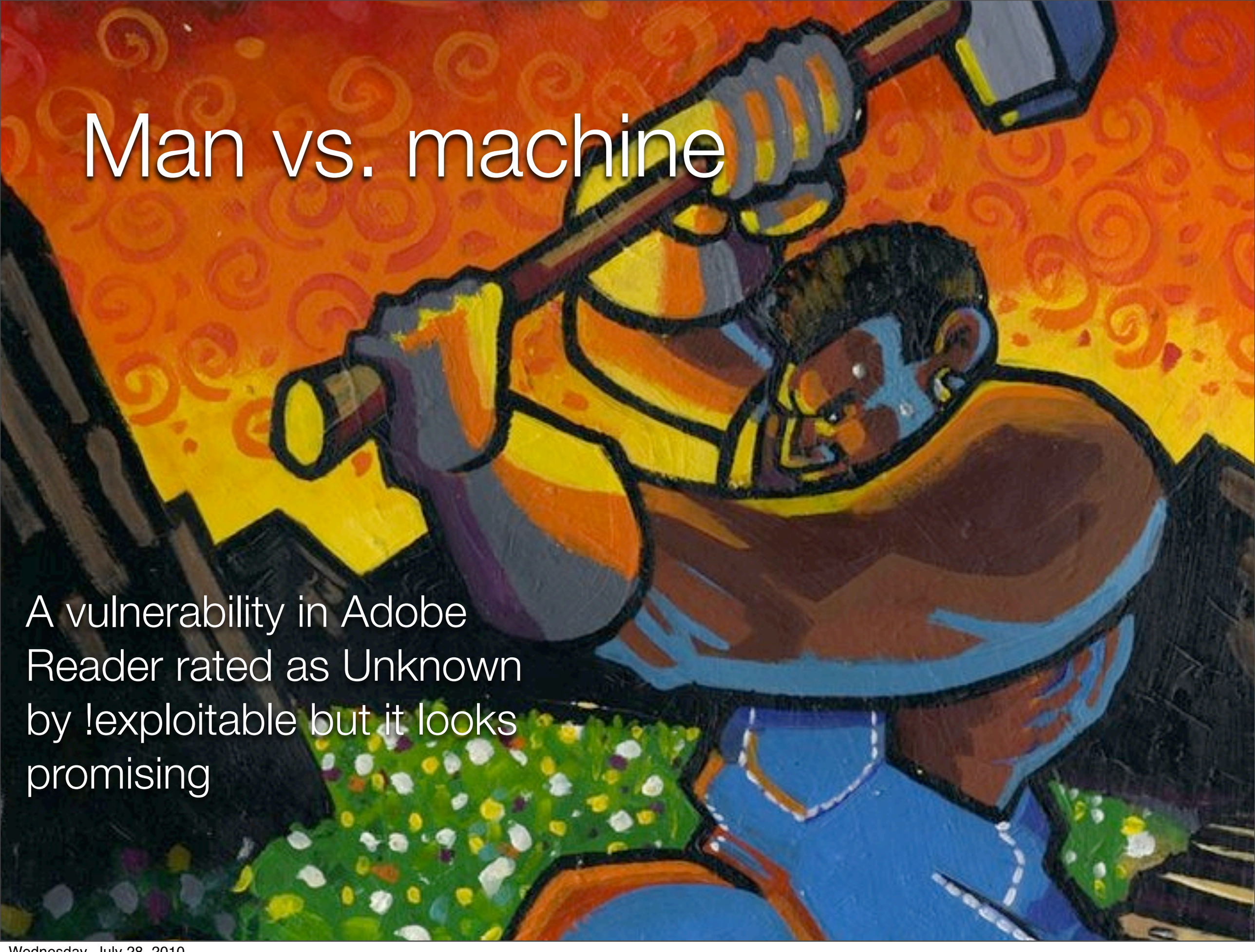
ax comes from addition of two (tainted) words 0xffc4 and 0xffa8. This is an (short) integer overflow

Now the offset

```
$ x86_slicer -in-trace z.trace -ctr 209429982 -regloc ebx > /dev/null
$ trace_reader_intel -trace EBX_0.trace | tail -5

57308043: movzx  eax,WORD PTR [esi+0x8] M@0x08a34720[0x0000ffa8][2](R)    T1 {3
(1111, 3) (1111, 3) ()()}  R@eax[0x0000ffc4][4](W) T1 {3 (1111, 3) (1111, 3) ()()}

5730805e: movzx  ebx,ax R@ax[0x0000ffa8][2](R)  T1 {3 (1111, 3) (1111, 3) ()()}
   R@ebx[0x0000ffa8][4](W) T1 {3 (1111, 3) (1111, 3) ()()}

57307f3c: push   ebx    R@ebx[0x0000ffa8][4](R) T1 {3 (1111, 3) (1111, 3) ()()}
   M@0x014ad948[0x08a34718][4](W)    T0

5730802a: pop    ebx    M@0x014ad948[0x0000ffa8][4](R)    T1 {3 (1111, 3) (1111,
3) ()()}  R@ebx[0x089b5fd8][4](W) T0

5730807c: movzx  ebx,bx R@bx[0x0000ffa8][2](R)  T1 {3 (1111, 3) (1111, 3) ()()}
   R@ebx[0x0000ffa8][4](W) T1 {3 (1111, 3) (1111, 3) ()()}
```

The offset is one of the summands used for the allocation

Man vs. machine

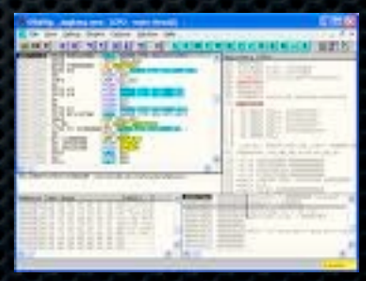


A vulnerability in Adobe Reader rated as Unknown by !exploitable but it looks promising

The match-up

BitMaster

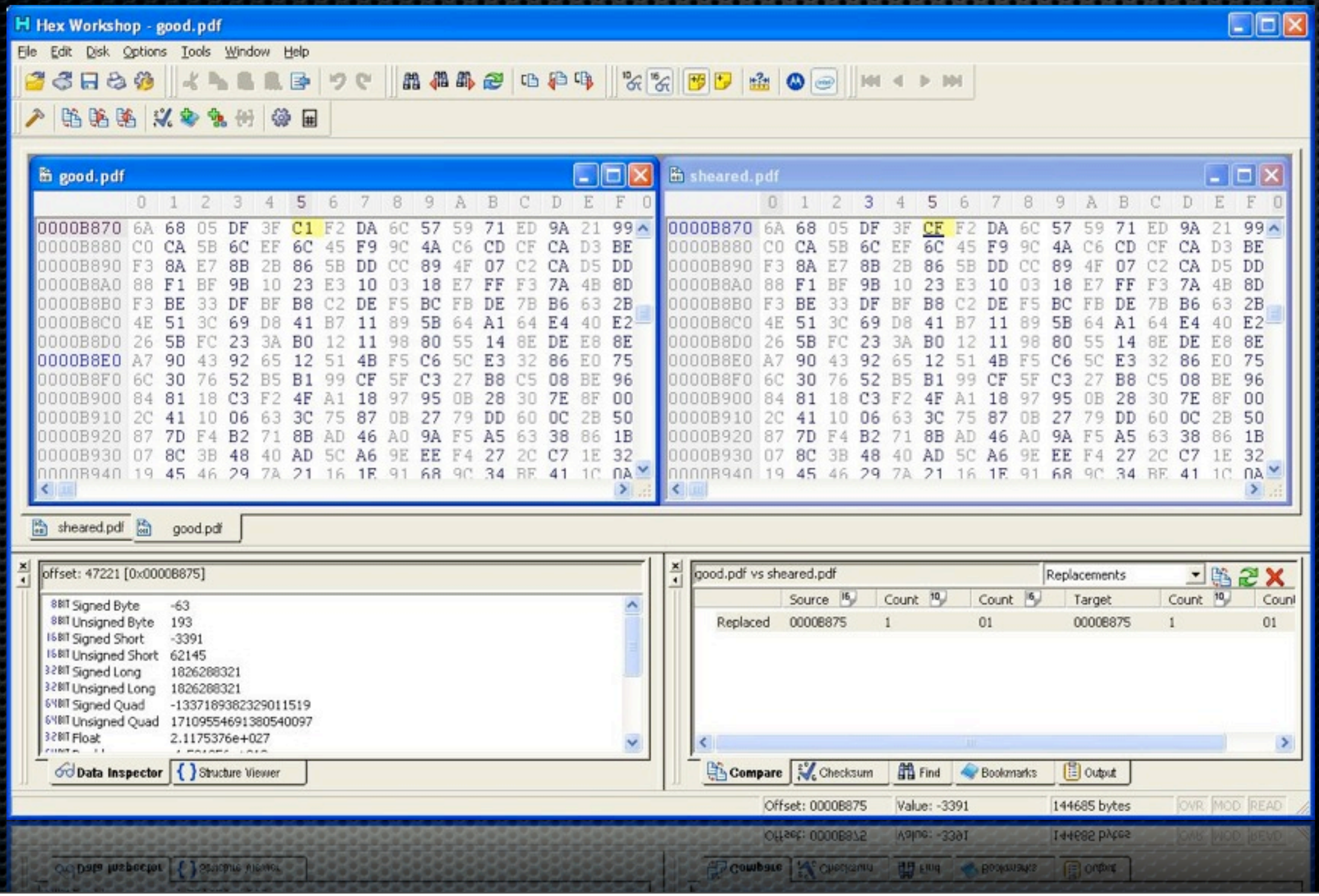
vs



The crash

```
08074fc6 8b09            mov     ecx,dword ptr [ecx]
08074fc8 66397104        cmp     word ptr [ecx+4],si          ds:0023:07471c70=????
08074fcc 7cf2            jl      CoolType!CTInit+0x2e9a3 (08074fc0)
08074fce 8b4804          mov     ecx,dword ptr [eax+4]
08074fd1 eb08            jmp     CoolType!CTInit+0x2e9be (08074fdb)
08074fd3 8bf8            mov     edi,eax
08074fd5 2bfa            sub     edi,edx
08074fd7 0139            add     dword ptr [ecx],edi
08074fd9 8b09            mov     ecx,dword ptr [ecx]
08074fdb 66397104        cmp     word ptr [ecx+4],si
ModLoad: 10000000 10029000   C:\Program Files\Adobe\Reader 9.0\Reader\BIBUtils.dll
(f8c.f34): Access violation - code c0000005 (!!! second chance !!!)
eax=03a38e2c ebx=03a38e2c ecx=07471c6c edx=00000000 esi=00007fff edi=03a38e58
eip=08074fc8 esp=022fedd0 ebp=022fedf4 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Adob
CoolType!CTInit+0x2e9ab:
08074fc8 66397104        cmp     word ptr [ecx+4],si          ds:0023:07471c70=????
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
0:002> !exploitable
Exploitability Classification: UNKNOWN
Recommended Bug Title: Data from Faulting Address controls Branch Selection starting at CoolType!

The data from the faulting address is later used to determine whether or not a branch is taken.
```

The good and bad file



Go!

- Bitmaster taints 0xcf, gets traces for good and bad file, differential taint analysis, slices on `ecx`, etc.
- Dion fires up IDA, gets file crashing in a debugger, etc

The Bitmaster knows

```
$ trace_reader -trace z.trace -header | grep Number
Number of instructions: 243897288
$ trace_reader_intel -trace z.trace -first 243897288 -v
8074fc8:  cmp    WORD PTR [ecx+0x4],si  R@si[0x00007fff][2](R)  T0  M@0x06fb1c70
[0x00000000][2](R)  T0 ESP:  NUM_OP: 4 TID: 1660 TP: TPNone EFLAGS: 0x00000083 CC_OP:
0x00000008 DF: 0x00000001 RAW: 0x66397104 MEMREGS:   R@ecx[0x06fb1c6c][4](R) T0

$ x86_slicer -in-trace z.trace -ctr 243897288 -regloc ecx > /dev/null
$ trace_reader_intel -trace ECX_0.trace | tail -2
8074fad:  add    DWORD PTR [eax],ecx R@ecx[0x037d8e2c][4](R) T0  M@0x037d8e2c[0x037d8e40]
[4](RW)   T0
8074faf:  mov    ecx,DWORD PTR [eax] M@0x037d8e2c[0x06fb1c6c][4](R)   T0  R@ecx
[0x037d8e2c][4](W)  T0
```

The bad value of ecx is the result of adding two heap pointers together.

This address is reachable by heap spray.

Looks exploitable to me

The score

BitMaster

VS

1

0

0xcf is worse than 0xc1

- The crash occurred in CoolType.dll
- Lets see when tainted data enters this DLL

```
$ trace_reader -trace z.trace -header | grep CoolType
    Module: CoolType.dll @ 0x08000000 Size: 2486272

$ trace_reader -trace z.trace -taintedonly -count | grep ")80" | head
(01519331)808b015:  push   ecx   R@ecx[0x01e7546c][4](R) T0  M@0x0012e950[0xe49b0000][4]
(W) T1 {8 ()()()(1111, 4) }
(01738319)8048170:  push   DWORD PTR [ebp-0x4] M@0x0012d99c[0x08083305][4](R)    T0
   M@0x0012d97c[0xe49b0000][4](W)    T1 {8 ()()()(1111, 4) }
(01754161)8015367:  movzx  eax,BYTE PTR [esi+0xb] M@0x0012d4ef[0x000000e4][1](R)    T1 {1
(1111, 4) ()()()}   R@eax[0x0000009b][4](W) T0
...
```

The good counterpart of e4

```
(01754161)8015367:  movzx  eax,BYTE PTR [esi+0xb] M@0x0012d4ef[0x000000e4][1]
(R) T1 {1 (1111, 4) ()()()} R@eax[0x0000009b][4](W) T0
```

```
$ ./aligned.pl y_z.aligned.txt T1:01754161
<T0:1753039> ~ T1:01754161
(T0:01750116-01753248 ~ T1:01751238-01754370)
```

```
(01753039)8015367:  movzx  eax,BYTE PTR [esi+0xb] M@0x0012d4ef[0x000000d4][1]
(R) T1 {1 (1111, 4) ()()()} R@eax[0x0000009b][4](W) T1 {1 (1111, 4) ()()()}
```

How did e4/d4 come from cf/c1?

Right before the DLL

```
$ trace_reader_intel -trace z.trace -first 01753161 -last 01754161 -taintedonly -v -count

(01753389)7814507a: rep movs DWORD PTR es:[edi],DWORD PTR ds:[esi] M@0x02e21a78[0xe49b0000]
[4](CR)   T1 {8 ()()()(1111, 4) }R@ecx[0x00000057][4](RCW)  T0  M@0x0293860c[0xe49b0000][4]
(CW)   T1 {8 ()()()(1111, 4) } ESP:  NUM_OP: 5 TID: 1660 TP: TPSrc EFLAGS: 0x00000202 CC_OP:
0x00000010 DF: 0x00000001 RAW: 0xf3a5 MEMREGS:    R@edi[0x0293860c][4](R) T0  R@esi
[0x02e21a78][4](R)  T0

(01754027)7814507a: rep movs DWORD PTR es:[edi],DWORD PTR ds:[esi] M@0x0293860c[0xe49b0000]
[4](CR)   T1 {8 ()()()(1111, 4) }R@ecx[0x0000000e][4](RCW)  T0  M@0x0012d4ec[0xffc95f62][4]
(CW)   T0 ESP:  NUM_OP: 5 TID: 1660 TP: TPSrc EFLAGS: 0x00000202 CC_OP: 0x00000010 DF:
0x00000001 RAW: 0xf3a5 MEMREGS:  R@edi[0x0012d4ec][4](R) T0  R@esi[0x0293860c][4](R) T0

(01754161)8015367:  movzx  eax,BYTE PTR [esi+0xb] M@0x0012d4ef[0x000000e4][1](R)   T1 {1
(1111, 4) ()()()}   R@eax[0x0000009b][4](W) T0 ESP:  NUM_OP: 4 TID: 1660 TP: TPSrc EFLAGS:
0x00000203 CC_OP: 0x00000008 DF: 0x00000001 RAW: 0x0fb6460b MEMREGS:  R@esi[0x0012d4e4][4]
(R) T0
```

Nearby bytes

```
$ state_reader -in-state z.trace.state -in-state-range 0x02e21a60:0x02e21a8f  -out-
raw z.state.dump.raw
$ hexdump -C z.state.dump.raw
00000000  6c 6f 63 61 01 99 84 e2  00 00 79 e8 00 00 1a 2c  |loca......y....,|
00000010  6d 61 78 70 0d 84 01 93  00 00 9b e4 00 00 00 20  |maxp........... |
00000020  6e 61 6d 65 8c 9d 3d 42  00 00 95 f8 00 00 05 dc  |name..=B........|

$ state_reader -in-state z.trace.state -in-state-range 0x02e21a60:0x02e21a8f -out-
text z.state.dump.txt
$ grep -v "Taint: none" z.state.dump.txt
0x02e21a7b (0xe4) Taint: (4444, 1111, 4)
```

Only one tainted byte, surrounded by things from a TrueTypeFont. This is the uncompressed stream

Meanwhile...

1. Decompress good PDF
2. Examine good decompressed font
3. Look for strings from good decompressed font in heap of Reader when bad file crashes it
4. Extract decompressed bad font from memory
5. Compare good/bad decompressed font
6. These fonts differ in e4 (bad) d4 (good)

The score

BitMaster

vs

1

1

Round 3

That byte is in the maxp directory entry
Offset from beginning of sfnt is 0x9b**e4**/0x9b**d4**
Results in different maxp tables being used
Instead of maxSizeOfInstruction = 0xffff (good)
you get maxComponentPoints = 0xffff (bad)
Create new PDF with uncompressed font with 0xffff as maxComponentPoints in maxp table
Same crash, easier to analyze

The score

BitMaster

VS

1

2

Thanks, human

- Take simplified PDF, taint the 0xffff maxComponentPoints word, get BitBlaze traces

0xffff goes to CoolIType

```
$ trace_reader_intel -trace x.trace -taintedonly -count | grep ")80" | head -20
(79500371)80112f3:  mov    dh,BYTE PTR [eax+0xa]  M@0x0267ae86[0x000000ff][1](R)    T1
{1 (1111, 6) ()()()}   R@dh[0x00000000][1](W)  T0
(79500372)80112f6:  mov    dl,BYTE PTR [eax+0xb]  M@0x0267ae87[0x000000ff][1](R)    T1
{1 (1111, 7) ()()()}   R@dl[0x00000000][1](W)  T0
(79500373)80112f9:  mov    WORD PTR [ecx+0xa],dx  R@dx[0x0000ffff][2](R)  T1 {3 (1111,
7) (1111, 6) ()()}  M@0x01e7592e[0x00000000][2](W)    T0
(79500374)80112fd:  xor    edx,edx   R@edx[0x0000ffff][4](R)T1 {3 (1111, 7) (1111, 6)
()()}  R@edx[0x0000ffff][4](RW)   T1 {3 (1111, 7) (1111, 6) ()()}
(79501343)80053e7:  movzx  edx,WORD PTR [ebx+0xa] M@0x01e7592e[0x0000ffff][2](R)   T1
{3 (1111, 7) (1111, 6) ()()}  R@edx[0x000027fc][4](W)T0
(79501344)80053eb:  cmp    ax,dx  R@dx[0x0000ffff][2](R)  T1 {3 (1111, 7) (1111, 6) ()
()}R@ax[0x000000f2][2](R)  T0
(79501347)80053f3:  movzx  eax,dxR@dx[0x0000ffff][2](R)  T1 {3 (1111, 7) (1111, 6) ()
()}R@eax[0x000000f2][4](W)T0
(79501350)80053fa:  lea    edx,[ebp+0xc]    A@0x0012d618[0x00000000][4](R)    T0  R@edx
[0x0000ffff][4](W)  T1 {3 (1111, 7) (1111, 6) ()()}
(79501355)8005403:  add    eax,0x8   I@0x00000000[0x00000008][1](R)    T0  R@eax
[0x0000ffff][4](RW) T1 {3 (1111, 7) (1111, 6) ()()}
(79501356)8005406:  push   eax    R@eax[0x00010007][4](R)T1 {15 (1111, 7) (1111, 7)
(1111, 7) (1111, 7) }  M@0x0012d5ec[0x01e759c0][4](W)    T0
(79501360)8004745:  mov    eax,DWORD PTR [ebp+0x14]  M@0x0012d5f8[0x0012d618][4]
(R)T0  R@eax[0x00010007][4](W)T1 {15 (1111, 7) (1111, 7) (1111, 7) (1111, 7) }
(79501365)8004751:  movzx  esi,WORD PTR [ebp+0x8] M@0x0012d5ec[0x00000007][2](R)    T1
{3 (1111, 7) (1111, 7) ()()}  R@esi[0x01e75968][4](W)T0
...
```

The score

BitMaster

2

vs

2

The rest of the story

- 0xffff has 8 added to it and then is used as a word
- This word is multiplied by 4 to get a buffer size (0x1c)
- 11 buffers are allocated of this size
- The routines that operate on these buffers assume they are quite large ((0xffff+8)*4 = 0x4001c)
- The data in these buffers stomp all over each other

Sudden death overtime

BitMaster

vs

2

2

Sudden death overtime

BitMaster

vs

2

2

Writes a POC exploit

Sudden death overtime

BitMaster

vs

2

2

Writes a POC exploit

Sudden death overtime

BitMaster

vs

2

2 3

Writes a POC exploit

Sudden death overtime

BitMaster

vs

2

2 3

Writes a POC exploit

Tradeoffs

- BitBlaze is not fast
  - Traces can take hours to collect (think overnight runs)
  - Slices can take a few minutes (normally) to a couple of hours (rarely)
- Memory constaints
  - Sometimes BitBlaze cannot handle runs with full taint
- Disk space hog
  - Traces are sometimes many GB in size
- Not necessarily faster in wall clock time, but faster in brain cycles (modulo latency)

Conclusions

- BitBlaze can perform many tasks which might help you in security research
- Crash analysis is hard, but some of BitBlaze's tools can help reduce the time and frustration needed to analyze crashes
- Whitepaper and code is available, check it out for yourself




bitblaze.cs.berkeley.edu




Links

- Get the code
  - http://bitblaze.cs.berkeley.edu/release/blackhat-2010.html
- Get the slides and whitepaper for this talk
  - http://securityevaluators.com/fiiles/papers/CrashAnalysisSlides.pdf
  - http://securityevaluators.com/files/papers/CrashAnalysis.pdf

Questions?

- cmiller@securityevaluators.com
  - Some usage questions
  - Questions on examples
- bitblaze@gmail.com
  - Everything else