Finding the Needle in the Heap: Combining Static Analysis and Dynamic Symbolic Execution to Trigger Use-After-Free

Josselin Feist†
Laurent Mounier†
Sébastien Bardin‡
Robin David‡
Marie-Laure Potet†

December 2016

† Verimag (France), first.last@imag.fr
‡ CEA LIST (France), first.last@cea.fr

Triggering *Use-After-Free*

Summary of this Presentation

- What is *Use-After-Free*
- Background on static analysis detecting *Use-After-Free* and dynamic symbolic execution (DSE)
- How to combine static analysis with DSE to trigger *Use-After-Free*
- Details for real-world application

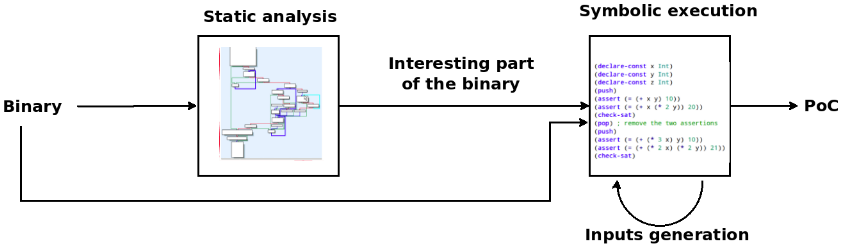
Plan

1. Big Picture
2. Background
   - Use-after-free
   - Static Analysis Detecting UAF
   - Dynamic Symbolic Execution
3. Guided DSE
   - Intuition
   - Weigthed-Slice
   - Oracle
   - Libraries
   - Implementation
4. Conclusion & Perspectives

Big Picture

Our Approach

- Use **static analysis** to extract vulnerable paths as slice of the program
- Create PoC confirming these results with **dynamic symbolic execution**



Techniques Combination

	Static analysis	DSE	Static analysis + DSE
Finding *Use-After-Free*	+	-	+
PoC generation	-	+	+

Advantages

- Allow unsound static analysis
- Complex *Use-After-Free* detection + PoC generation
- Correctness (if *Use-After-Free* found, true positive)

Plan

Plan

1. Big Picture
2. Background
   - Use-after-free
   - Static Analysis Detecting UAF
   - Dynamic Symbolic Execution
3. Guided DSE
   - Intuition
   - Weigthed-Slice
   - Oracle
   - Libraries
   - Implementation
4. Conclusion & Perspectives

Use-After-Free

Dangling Pointer

A dangling pointer is a pointer pointing to a free block, or to a block reallocated to another pointer.

```
login=(char*)malloc(...);
...
free(login); // login is now a dangling pointer
```

Use-After-Free

Dangling Pointer

A dangling pointer is a pointer pointing to a free block, or to a block reallocated to another pointer.

```
login=(char*)malloc(...);
...
free(login); // login is now a dangling pointer
```

Use-After-Free

Use of a dangling pointer

```
login=(char*)malloc(...);
...
free(login); // login is now a dangling pointer
..
printf("%s\n", login)  // use-after-free
```

Use-After-Free

Why it is dangerous?

- Re-allocations → create *indirect* aliases
- Information leakage, control-flow hijacking, ...

```c
login=(char*)malloc(...);
...
free(login); // login is now a dangling pointer
..
passwords=(char*)malloc(...); // may re-allocate memory area used by login
..
printf("%s\n", login)  // prints the passwords !
```

Use-After-Free in the Wild

Particularities

- Difficult to detect (events distant, reasoning with heap, ..)
- No easy "pattern" (like for buffer overflow / string format)
- Lots of *Use-After-Free* in browsers
- Present in other apps (proftpd CVE-2011-4130, privoxy CVE-2015-1031, openssh...)

Detecting *Use-After-Free*

Dynamic Analysis

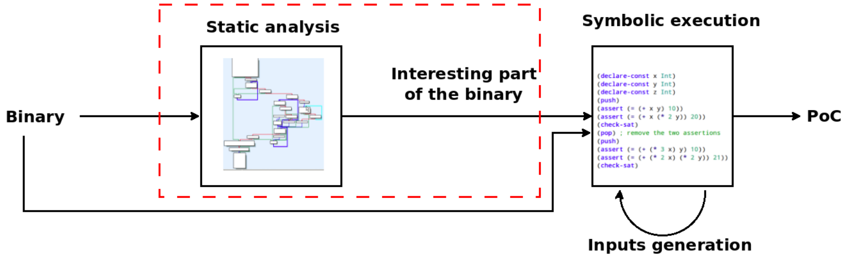
- Good dynamic detection (AddressSanitizer) / mitigation (MemGC in IE,...)
- Limitations:
  - Need to find the good path
  - Overhead during execution

Static Analysis

- Source code: codesonar, coverity , ..
- Binary code: codesonar x86, veracode, ..
- Limitations:
  - Number of false positives
  - No PoC

Plan

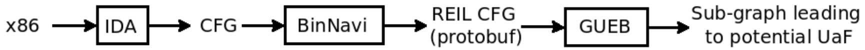
Static Analysis



Static analyzer: GUEB

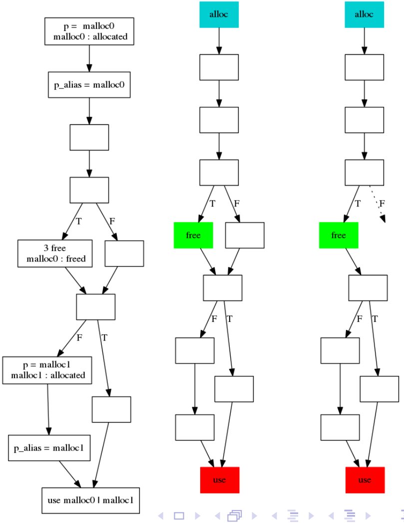
Graph of Use-after-free Extraction from Binary

- Based on value analysis
- Open source: `https://github.com/montyly/gueb`
- Ocaml



GUEB: Example

```
1  p=malloc(sizeof(int));
2  p_alias=p; // p and p_alias points
3          // to the same addr
4  read(f,buf,255); // buf is tainted
5
6  if(strncmp(buf,"BAD\n",4)==0){
7   free(p);
8   // exit() is missing
9  }
10 else{
11  .. // some computation
12 }
13
14 if(strncmp(&buf[4],"is a uaf\n",9)
        ==0){
15   p=malloc(sizeof(int));
16 }
17 else{
18   p=malloc(sizeof(int));
19   p_alias=p;
20 }
21
22 *p=42 ; // not a uaf
23 *p_alias=43 ; // uaf if 6 and 14 =
        true
```



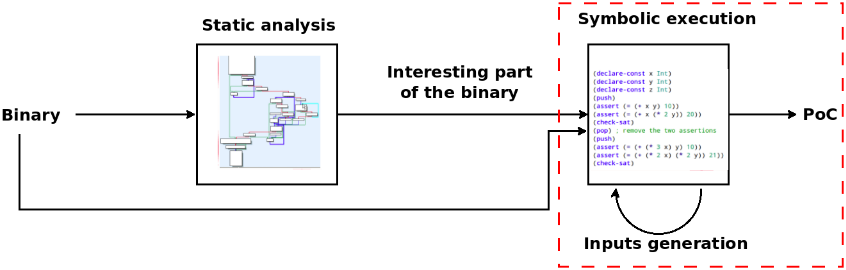
GUEB: Results

Results

- Several **new** *Use-After-Free* found (Jasper-JPEG-200 (CVE-2015-5221), openjpeg (CVE-2015-8871), gnome-nettool, accel-ppp, giflib (CVE-2016-3177), alsbat)
- Some results with good accuracy: gnome-nettool, 5 reports, just 4 false positives
- Some more difficult: jasper, 300+ reports
- Good results, but still need an expert to remove false positives
- Could we automatize the sorting of results?

Plan

Dynamic Symbolic Execution



Dynamic Symbolic Execution

- Also called Whitebox fuzzing / Concolic Execution
- Program exploration technique
- Goal: generating inputs

DSE: Example

```c
void f(int a){
 a = a+1;
 if(a == 0x42){
  printf("Win!\n");
 }
}
```

Using Classical Fuzzing

- Hard to trigger randomly

DSE: Example

```
1 void f(int a){
2  a = a+1;
3  if(a == 0x42){
4   printf("Win!\n");
5  }
6 }
```

Using Dynamic Symbolic Execution

- Choose seed (e.g.:a=0) → Generate trace 1-2-3-5-6
- Make `a` symbolic and generate `path predicate`

DSE: Example

```
1 void f(int a)
2  a = a+1;
3  if(a == 0x42){
4   printf("Win!\n");
5  }
6 }
```

Code	Concrete Values	Path Predicate
`1 : void f(int a){`	$a = 0$	a_0
`2 : a = a+1;`		
`3 : if(a == 0x42){`		

DSE: Example

```
void f(int a){
 a = a+1;
 if(a == 0x42){
  printf("Win!\n");
 }
}
```

Code	Concrete Values	Path Predicate
`1 : void f(int a){`	$a = 0$	a_0
`2 : a = a+1;`	$a = 1$	$a_1 = a_0 + 1$
`3 : if(a == 0x42){`		

DSE: Example

```
1 void f(int a){
2  a = a+1;
3  if(a == 0x42)
4   printf("Win!\n");
5  }
6 }
```

Code	Concrete Values	Path Predicate
`1 : void f(int a){`	$a = 0$	a_0
`2 : a = a+1;`	$a = 1$	$a_1 = a_0 + 1$
`3 : if(a == 0x42){`	$a = 1$	$a_1 = a_0 + 1 \wedge a_1 \neq 0x42$

DSE: Example

```c
1 void f(int a){
2  a = a+1;
3  if(a == 0x42)
4   printf("Win!\n");
5  }
6 }
```

- Path predicate: invert conditional branch $\rightarrow$ explore new path
- Use SMT Solver (Z3, Boolector,..) to solve it
- Here, SAT, $a_0 = 0x41$
- Generate new trace with this new input

Code	Concrete Values	Path Predicate
`1 : void f(int a){`	$a = 0$	a_0
`2 : a = a+1;`	$a = 1$	$a_1 = a_0 + 1$
`3 : if(a == 0x42){`	$a = 1$	$a_1 = a_0 + 1 \wedge a_1 \neq 0x42$

DSE

- Large interest these past years
- Academic & Industrial interest
- SAGE, KLEE, S2E, Mayhem, Angr, Triton..
- Young topic, still a lot of limitations (and tuning needed)
- $\rightarrow$ Path explosion problem
- One key feature $\rightarrow$ How to prioritize paths exploration?

Instead of coverage $\rightarrow$ triggering specific path(s)

Plan

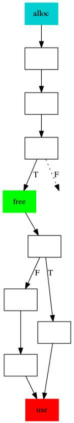
Plan

Guided DSE

Using Results from Static Analysis

- Explore only a sub-part of the program using GUEB

```
1  p=malloc(sizeof(int));
2  p_alias=p; // p and p_alias points
3         // to the same addr
4  read(f,buf,255); // buf is tainted
5
6  if(strncmp(buf,"BAD\n",4)==0){
7   free(p);
8   // exit() is missing
9  }
10 else{
11  .. // some computation
12 }
13
14 if(strncmp(&buf[4],"is a uaf\n",9)==0){
15   p=malloc(sizeof(int));
16 }
17 else{
18   p=malloc(sizeof(int));
19   p_alias=p;
20 }
21
22 *p=42 ; // not a uaf
23 *p_alias=43 ; // uaf if 6 and 14 = true
```



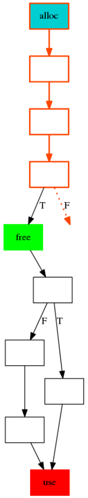
Guided DSE: Example

Example

- First input: filled with 'A'
- All condition of the trace except one go out of the subgraph
- Formula: $input[0..3] \neq "BAD\backslash n"$

```
read(f,buf,255); // buf is tainted

if(strncmp(buf,"BAD\n",4) == 0){ free(p);}
 else{ ..} // some computation
```



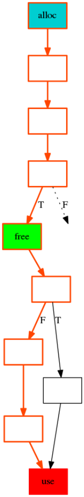
Guided DSE: Example

Example

- Second input: `"BAD\n"`
- No UaF
- Formula: $input[0..3] = "BAD\backslash n" \land input[4..13] \neq "is\ a\ uaf\backslash n"$

```
read(f,buf,255); // buf is tainted

if(strncmp(buf,"BAD\n",4) == 0){ free(p);}
if(strncmp(&buf[4],"is a uaf\n",9) == 0){
else{ ...}
```



Guided DSE: Example

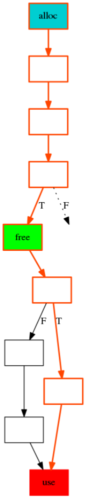
Example

- Third input: "BAD\nis a uaf\n"
- UaF Found !

```
read(f,buf,255); // buf is tainted

if(strncmp(buf,"BAD\n",4) == 0){ free(p);}
if(strncmp(&buf[4],"is a uaf\n",9) == 0)

*p_alias = 42 ;
```



Plan

Weigthed-Slice Guided DSE

WS-Guided

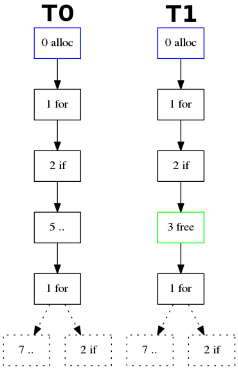
- Slice leads the exploration
- If a node is outside the slice $\rightarrow$ do not explore
- If a node is inside the slice $\rightarrow$ use score to prioritize
- Score: distance metric (i.e.: shortest path)
- Compute score as pre-preprocessing of the exploration
- Slice is *weighted* by the score

WS-Guided DSE

WS-Guided

- 3 events (alloc/free/use) → 3 different scores
- For a node, select score according past events on the trace

```
p=malloc(..);
for(..){
 if(cond1)
  free(p);
 else
  ..
}
*p=42;
```



Plan

1. Big Picture
2. Background
   - Use-after-free
   - Static Analysis Detecting UAF
   - Dynamic Symbolic Execution
3. Guided DSE
   - Intuition
   - Weigthed-Slice
   - Oracle
   - Libraries
   - Implementation
4. Conclusion & Perspectives

Oracle

Use-After-Free in a trace?

- *Use-After-Free* detection on a trace is not so easy
- Trouble with aliases

```c
int *p=malloc();
p_alias=p;
free(p);
if(cond){
 p=malloc()
}
else{
 p=malloc();
 p_alias=p;
}
*p=0; // never uaf
*p_alias=0; // uaf if cond
```

Two paths (according *cond*), yet, *p* is always equal to *p_alias*.
Only *Use-After-Free* in *p_alias* if (*cond*)

Oracle

Use-After-Free in a trace?

(i) $t = (\ldots, n_{alloc}(size_{alloc}), \ldots, n_{free}(a_f), \ldots, n_{use}(a_u))$

(ii) a_f is a reaching definition of the block returned by n_{alloc}

(iii) a_u is a reaching definition of an address in the block returned by n_{alloc}

SMT-based solution

- Could use data dependencies analysis but:
  - Traces incomplet, need to have stubs for data dependencies
- Data dependencies kept *implicitely* with path predicate

Oracle

SMT-based solution

- New path predicate φ, *malloc* returns S_{alloc}, inputs conc
- $\Phi' = (a_f \neq S_{alloc}) \vee (a_u \notin [S_{alloc}, S_{alloc} + size_{alloc} - 1])$
- Oracle: $\varphi \wedge \Phi'$ is UNSAT $\rightarrow$ *Use-After-Free*

Explanation

- Instead of $(\forall X \rightarrow SAT)$ we use $(\exists \overline{X} \rightarrow UNSAT)$
- $a_f \neq S_{alloc}$: the pointer given as the parameter for `free` is not the one allocated at n_{alloc} (negation of property (ii))
- $a_u \notin [S_{alloc}, S_{alloc} + size_{alloc} - 1]$: the pointer used is not a reaching definition of the pointer allocated at n_{alloc} (negation of property (iii))

Oracle

```c
int *p=malloc(4);
p_alias=p;
free(p);
if(cond){
 p=malloc()
}
else{
 p=malloc();
 p_alias=p;
}
*p=0; // never uaf
*p_alias=0; // uaf if cond
```

First path

- Path predicate:
  $\varphi = (p_0 = S_{alloc} \wedge p\_alias_0 = p_0 \wedge p_1 = 0x8040000)$
- $\Phi' = (p_0 \neq S_{alloc} \vee p\_alias_0 \notin [S_{alloc}, S_{alloc} + 3])$
- $\varphi \wedge \Phi'$ UNSAT: there is *Use-After-Free*

Oracle

```
int *p=malloc(4);
p_alias=p;
free(p);
if(cond){
 p=malloc()
}
else{
 p=malloc();
 p_alias=p;
}
*p=0; // never uaf
*p_alias=0; // uaf if cond
```

Second path

- Path predicate: $\varphi = (p_0 = S_{alloc} \wedge p\_alias_0 = p_0 \wedge p_1 = 0x8040000) \wedge p\_alias_1 = p_1)$
- $\Phi' = (p_1 \neq S_{alloc} \vee p\_alias_1 \notin [S_{alloc}, S_{alloc} + 3])$
- $\varphi \wedge \Phi'$ SAT (e.g.: $S_{alloc} = 0$), no *Use-After-Free*

Plan

Libraries

Handling libraries

- When apply to real-world, libraries are widely present
- Do not want to explore all of them
- We use two solutions:
  - Stubs → model effects on path predicate without tracing instructions (e.g.: if *realloc* returns new pointer, it performs copy of data)
  - Library Driven Heuristics (LDH) → known-behavior used to improve guiding

Library Driven Heuristics (LDH): Example

```
p=malloc(size) ;
if(p == NULL)
{
 // path to trigger
}
```

Example

- To trigger the path → *malloc* needs to returns 0

malloc heuristic

- Principle: on allocation functions, try as parameter a large number

Library Driven Heuristics (LDH): Example

```
read(f,tmp,255);
for(i=0;i<255;i++){
  if(tmp[i]=='\0') break;
  buf[i] = tmp[i];
)
buf[i]='\0';
if(strcmp(buf,"this is really bad") == 0)
  ..
```

Example

- Comparison on string whose length depends of loop iteration number
- Here, every time loop iterates: $buf[i] \neq$ '\0'
- To solve *strcmp* condition, need a trace unrolling 18 times the loop

Library Driven Heuristics (LDH): Example

```
read(f,tmp,255);
for(i=0;i<255;i++){
  if(tmp[i]=='\0') break;
  buf[i] = tmp[i];
)
buf[i]='\0';
if(strcmp(buf,"this is really bad") == 0)
  ..
```

strcmp heuristic

- Principle: on this pattern, use size of constant strings passed to `strcmp` to find the desired iteration.

Plan

Guided DSE

BINSEC/SE

- Open source plaftorm to perform static analysis and DSE [a]
  - Guiding module based on Ocaml functors
  - Concretization / Symbolization Policies
  - Other nice features (opaque predicates, ..)
- Ocaml
- `http://binsec.gforge.inria.fr/tools`

[a] Code of this talk available in the next release (this month)

Guided DSE

Demo Binsec (Jasper-JPEG-200 CVE-2015-5221)

Jasper

Details on DSE Exploration

- 20 mins
- 9 test cases generated
- Last test case triggering the *Use-After-Free*
- More details in the paper (comparison with fuzzing)

Plan

Conclusion

Guided DSE

- First *end-to-end* approach targeting *Use-After-Free*
- Operates on binaries
- Working on real CVE
- Everything is open source
- Combining program analysis techniques → way to go

Limitation

- More complex software (browser, multi-thread)..
- Scalability issue ?

Perspectives

Guided DSE

- Improve guiding
- Other score (data-flow?)
- Need more benchmarks (network, etc...)
- Combine with fuzzing
- Other applications (BinDiff, 1day generation, ..)

XP

```
MIF
component
```

Figure: PoC of CVE-2015-5221 (test.plain)

```
jasper --input test.plain --input-format mif --output out --output-format mif
```

XP

Name	Time	MIF line	UAF found	# Paths
DSE (in BINSEC/SE)				
WS-Guided+*LDH*	$20m$	$3min$	**Yes**	9
WS-Guided	$6h$	$3min$	*No*	44
DFS(*slice*)	$6h$	$3min$	*No*	68
DFS	$6h$	$3min$	*No*	354
standard fuzzers (arbitrary seed)				
AFL	$7h$	$< 1min$	*No*	$174^{\dagger}$
Radamsa	$7h$	$> 1h$	*No*	$\sim 1000000^{\ddagger}$
standard fuzzers (MIF seed)				
AFL (*MIF input*)	$< 1min$	$< 1min$	**Yes**	< 10
Radamsa (*MIF input*)	$< 1min$	$< 1min$	**Yes**	< 10

† *AFL generates more input, 174 is the number of unique paths.*

‡ *For radamsa it is not trivial to count the number of unique path.*