Interprocedural Framework for Binary Static Analysis

Hayk Aslanyan

hayk@ispras.ru, ISPRAS

ISPRASOpen 2018, Moscow

Protect your software: Static Analysis

- Static code analysis is one of the common approaches for detecting defects. This approach is a program analyzing method that is performed by examining the code without executing the program.

- Through a complete analysis of syntax, semantics, control and data flow, static code analysis can find errors that are difficult or impossible to find in the testing phase of programs.

Binary files analysis is important

- The source code of the program is not always available, thus the use of source code analyzers becomes impossible.

- Not all compiler optimizations are safe, they can lead to errors in binary code that don’t exist in the source code.

- Analysis of a binary can provide more accurate information than a source-level analysis, because, for many programming languages, certain behaviors are left unspecified by the semantics

Why do we need to analyze the binary files?

```
int callee(int a, int b) {
int local;
if (local == 5) return 1;
else return 2;
}

int main() {
int c = 5;
int d = 7;
int v = callee(c,d);
// What is the value of v here?
return 0; }
```

Standard prolog	Prolog for 1 local

push ebp
mov ebp, esp
sub esp, 4
	push ebp
	mov ebp, esp
	push ecx

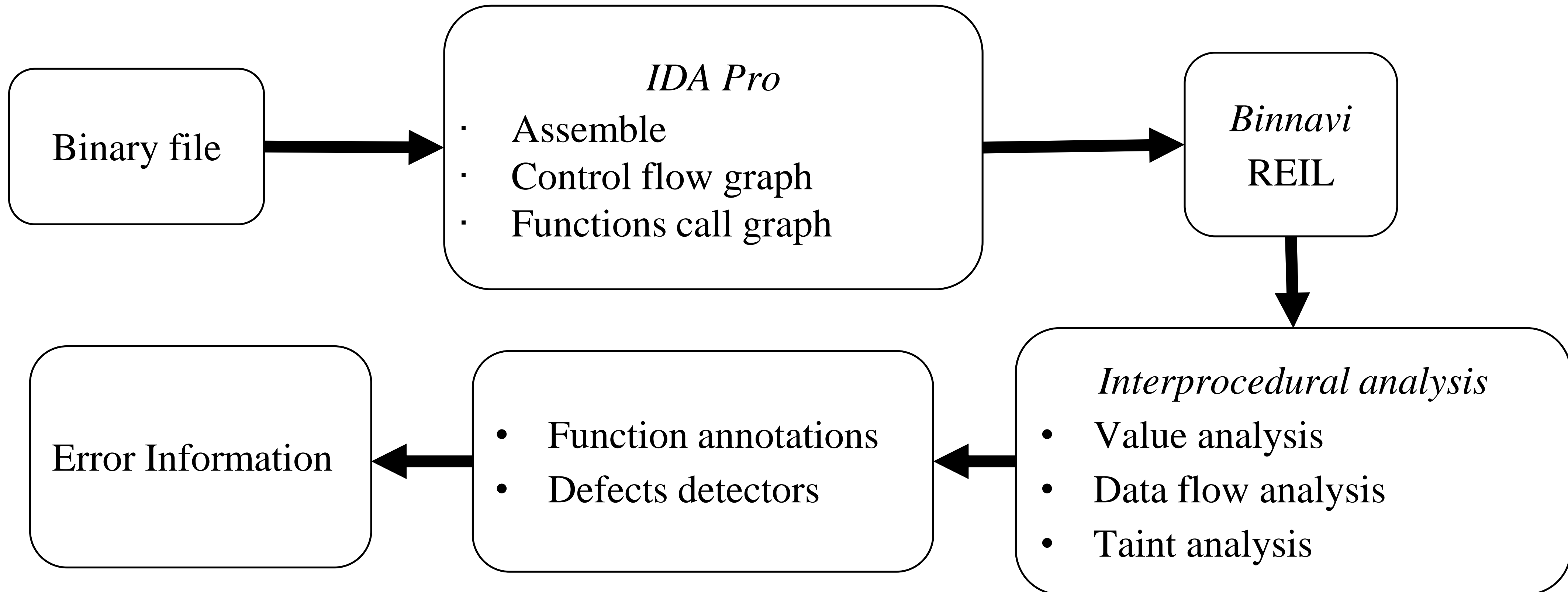
```
mov [ebp+var_8], 5
mov [ebp+var_C], 7
mov eax, [ebp+var_C]
push eax
mov ecx, [ebp+var_8]
push ecx
call _callee
. . .
```

Unexpected behavior due to compiler optimization

Formulation of the problem

- Develop a framework for static analysis of binary code that is independent from the architecture, scalable and easily extensible
- Develop methods of data flow analysis, value analysis, taint analysis for the binary code
- Develop methods for finding defects of use-after-free, double free, format string, buffer overflow and command injection

Architecture

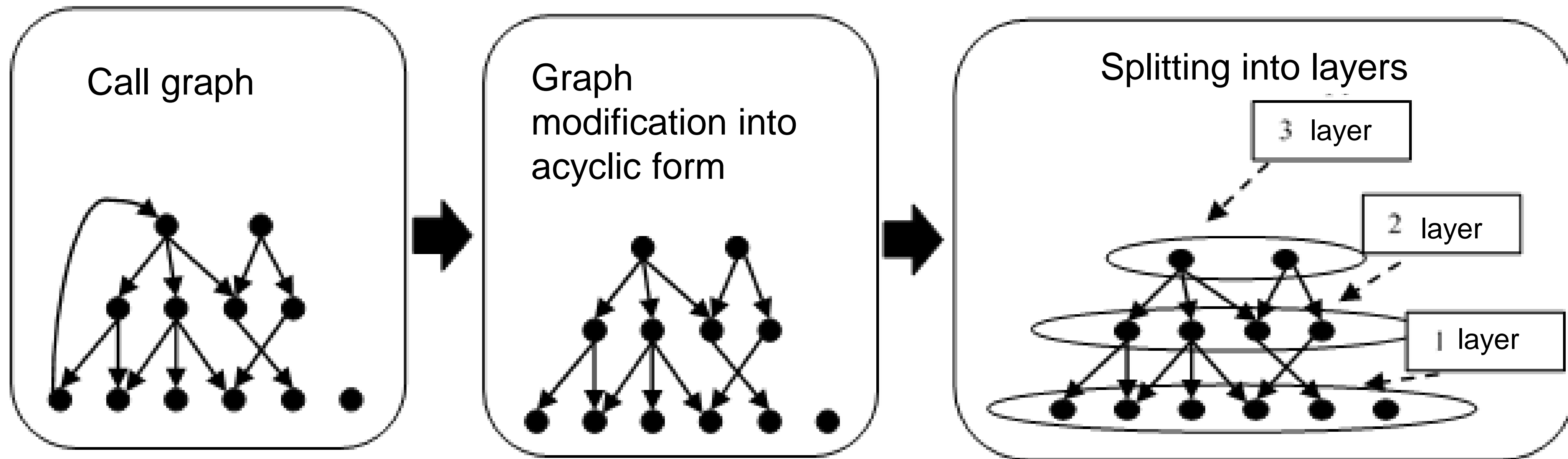


REIL representation

- Platform independent
- 17 simple instructions (and, add, ldm, stm…)
- It has no side effects

Architecture - Interprocedural Analysis

- Splitting a call graph into groups
- Each group is analyzed in parallel



Intraprocedural analyzes

- Value analysis
- Analysis of reaching definitions
- Constructing DEF-USE and USE-DEF chains
- Dead code elimination
- Constant propagation transformation
- Taint analysis
- Dynamic memory analysis (tracing memory allocation and deallocation)

Value analysis

At each program point compute all possible values that the given register or memory address can have:

- Memory simulation in the stack
- Memory simulation in the heap
- Static memory and global variables

Intraprocedural analyzes

- Function annotations
  - The function dereferences the argument
  - The function returns tainted argument (gets, …)
  - The function deletes the memory pointed by its argument (free, delete)
  - The function is format function (printf, fprintf,…)
  - Buffer overflow function (strcpy, memcpy)

Intraprocedural analyzes

- Function annotations
  - System function annotations
  - The function deletes the memory pointed by its argument (free, delete)

```
My_free(int* p) {
        free(p);
}
```

  - My_free gets annotation - The function deletes the memory pointed by its argument

Intraprocedural analyzes

- Defects detectors
  - Use-after-free
  - Double free
  - Format string
  - Buffer overflow
  - Command injection

Defect detectors

```
void f () {
  ...
  gets(p);
  prinf(p);
}
```

40073C	push	rbp
40073D	mov	rbp, rsp
400740	sub	rsp, 70h
40074E	lea	rax, [rbp+format]
400752	mov	rdi, rax
400755	**call**	**_gets**
40075A	lea	rax, [rbp+format]
40075E	mov	rdi, rax
400761	mov	eax, 0
400766	**call**	**_printf**
40076B	mov	eax, 0
400770	leave	
400771	retn	

Trace

400755 400766

Results (Use-After-Free, Double-Free)

Project	Architecture	Size	Analysis time	Number of found UAF and DF	Percentage of correct handling
accel_pppd 1.10.0	x86	232 KB	3m	4	100%
gnome-nettool 3.8.1	x86	336 KB	1m 40s	1	100%
slpd 1.2.1	x86	128 KB	50s	1	100%
libssh 0.5.2	x86	632 KB	3m	14	57%
jasper 1.900.1	x86	980 KB	11m 41s	1	100%
libtiff 4.0.3	x86	1 KB	2m 58s	3	67%
accel_pppd 1.10.0	x64	244 KB	4m 1s	1	100%
gnome-nettool 3.8.1	x64	436 KB	1m 50s	3	67%
libssh 0.5.2	x64	324 KB	3m 50s	13	53%
slpd 1.2.1	x64	128 KB	3m 1s	1	100%
pbs_server 2.4.8	x64	1.6 KB	11m 48s	1	100%

Results(comparison with GUEB)

Project	Working time of GUEB	Found UAF and DF with GUEB	Percentage of right handling of GUEB	Found UAF and DF	Percentage of right handling
gnome-nettool 3.8.1	16 s	4	25%	1	100%
gifcolor 5.1.2	21 s	15	6%	1	100%
jasper 1.900.1	4m 23s	255	1.2%	3	100%
accel-pppd 1.10.0	5m 5s	35	11.4%	8	50%

Results (Buffer overflow, Format String , Code injection)

Project	Architecture	Size	Analysis time	Number of found defects	Percentage of correct handling
dba 2.4.1	x86	312 KB	1m 40s	12	50%
httpd 0.5.0	x86	6.4 MB	6m 51s	22	90.9%
iwconfig 26	x86	44 KB	24 s	3	100%
mkfs 1.1.12	x86	56 KB	25 s	9	100%
pswdb 2.4.1	x86	300 KB	55 s	9	33%
hsolinkcontrol 1.0.118	x86	28 KB	2 s	22	100%
alsa_in 1.1.3	x86	28 KB	8 s	2	100%
htget 0.1	x64	28 KB	11 s	12	100%
mkfs 1.1.12	x64	56 KB	19 s	7	100%
libtorque 2.0.0	x64	892 KB	57 s	12	100%
alsa_out 1.1.3	x64	28 KB	10 s	2	100%
pbs_server 2.4.8	x64	320 KB	3m 20s	4	75%

Results(comparison with Loongchecker)

Project	Size	LoongChecker	Percentage of correct handling of LoongChecker	Number of found defects	Percentage of correct handling
Serenity.exe	19.6 MB	8	12.5%	2	50%
FoxPlayer.exe	33 MB	27	4%	2	100%

Thanks for attention