Static Analysis:
The Good, The Bad and The Ugly

Marwan Burelle
LSE Summer Week 2014

It's all about theory, security, practice

...

and the rest.

LSE
Security System
Laboratory of Epita

Static Program Analysis ?

Testing the code without running it.

- Mostly undecidable or semi-decidable !
- Specific properties can be tested
- Often hard and complex
- Can’t be both sound and complete

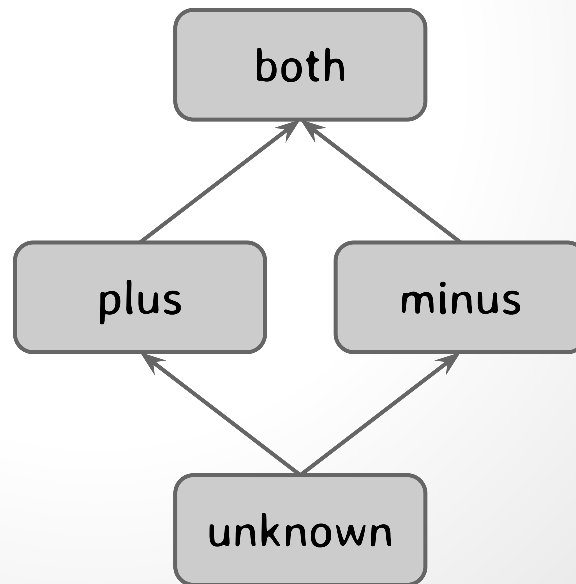
But we need it !

- Detecting corner case errors
- Verifying complex properties
- Get a proven formal verification
- compiler/optimization related stuff

Toy Example

Sign Analysis

- Decide sign of an arithmetical expression
- Use 4-way logic:
  - unknown
  - plus
  - minus
  - both



LSE Security System Laboratory of Epita

Sign Analysis

```
type expr =
    | Int of int
    | Var of string
    | UMinus of expr
    | Add of expr * expr
    | Dif of expr * expr
    | Mul of expr * expr

type sign = UNKNOWN | PLUS | MINUS | BOTH

module Env = Map.Make(String)
```

LSE
Security System
Laboratory of Epita

Sign Analysis

```
let rec sign env = function
    | Int i when i < 0      -> MINUS
    | Int i                 -> PLUS
    | Var x -> Env.find x env
    | UMinus e ->
      begin
        match sign env e with
         | PLUS              -> MINUS
         | MINUS             -> PLUS
         | _                 -> BOTH
      end
```

LSE Security System
Laboratory of Epita

Sign Analysis

```
| Add (e0, e1) ->
begin
  match (sign env e0, sign env e1) with
  | (PLUS, PLUS) -> PLUS
  | (MINUS, MINUS) -> MINUS
  | _ -> BOTH
end
```

LSE
Security
System
Laboratory of Epita

Sign Analysis

```
| Dif (e0, e1) ->
begin
  match (sign env e0, sign env e1) with
    | (PLUS, MINUS) -> PLUS
    | (MINUS, PLUS) -> MINUS
    | _ -> BOTH
end
```

LSE Security System Laboratory of Epita

Sign Analysis

```
| Mul (e0, e1) ->
  begin
    match (sign env e0, sign env e1) with
    | (PLUS, PLUS)
    | (MINUS, MINUS) -> PLUS
    | (PLUS, MINUS)
    | (MINUS, PLUS) -> MINUS
    | _ -> BOTH
  end
```

LSE Security System
Laboratory of Epita

Sound or Complete ?

Analysis verifies a property

Sound Analysis:

identified cases really have the property

LSE
Security
System
Laboratory of Epita

Complete Analysis:

all cases are identified

LSE Security System
Laboratory of Epita

Sound Analysis provides safety

Complete Analysis tracks errors

LSE
Security System
Laboratory of Epita

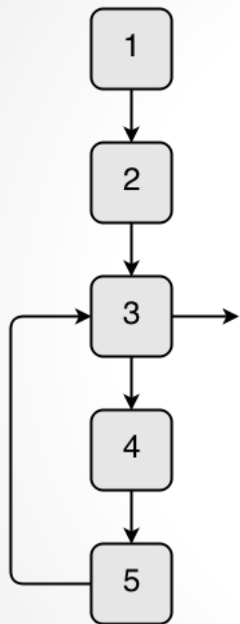
Analysis

- Model Checking
- Data flow Analysis
- Constraint Based Analysis
- Abstract Interpretation
- Type Systems
- *Handcrafted Analysis ;)*
- ...

LSE
Security System
Laboratory of Epita

- Put label on code

```
[x ← a + b]¹
[y ← a * b]²
while [y > a + b]³ do
    [a ← a + 1]⁴
    [x ← a + b]⁵
done
```

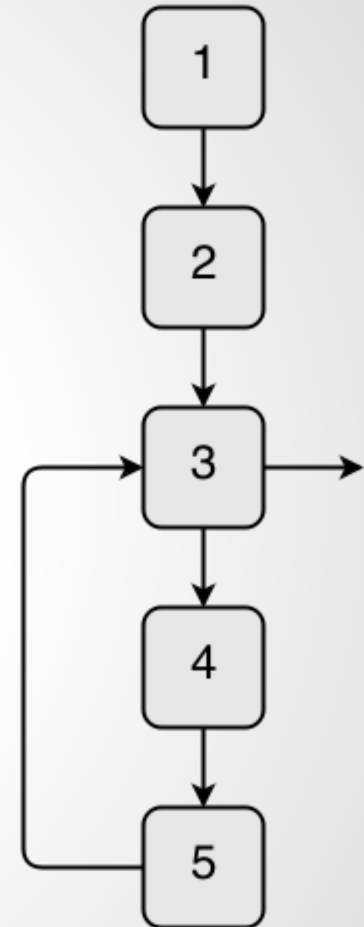


- Build a flow graph

- Build equations and solve them

LSE
Security System
Laboratory of Epita

```
[x ← a + b]¹
[y ← a * b]²
while [y > a + b]³ do
   [a ← a + 1]⁴
   [x ← a + b]⁵
done
```



LSE
Security System
Laboratory of Epita

	Kill	Gen
1	∅	{a+b}
2	∅	{a*b}
3	∅	{a+b}
4	{a+b, a*b, a+1}	∅
5	∅	{a+b}

	Entry	Exit
1	∅	{a+b}
2	{a+b}	{a+b, a*b}
3	{a+b}	{a+b}
4	{a+b}	∅
5	∅	{a+b}

LSE
Security System
Laboratory of Epita

For real ?

Traditional code analysis requires:

- some language properties
- well founded semantics
- some execution model

LSE Security System
Laboratory of Epita

C doesn't fit this description !

C has the following drawbacks:

- no formal semantics
- the standard is sometimes *fuzzy*
- there’s still ambiguous syntactic aspects

LSE
Security System
Laboratory of Epita

Are we doomed ?

We can still have:

- ➢ unsound, incomplete but useful analysis
- ➢ guidelines for other methods
- ➢ working analysis on very specific cases

LSE
Security System
Laboratory of Epita

Buffer Overflow

```c
void ugly(char *src) {
  char              buf[8];
  strcpy(buf, src);
}

int main(int argc, char *argv[]) {
  if (argc > 1) {
    ugly(argv[1]);
  }
  return 0;
}
```

- Write outside of buffer boundaries
- Most common mistake
- Over the last 25 years:
  - 14% of security vulnerabilities
  - **23% of top severity vulnerabilities**
- Known for years (1972, 1988 …)

LSE Security System
Laboratory of Epita

What can be done ?

- ➢ track usage of risky functions (strcpy ;)
- ➢ check size constraints on function calls
- ➢ when constraints doesn't hold

  → **raise a warning**

- ➢ use code review/tests to confirm bug

LSE Security System
Laboratory of Epita

Statically Detecting Likely Buffer Overflow Vulnerabilities

David Larochelle and David Evans (Usenix 2001)

- Using LCLint (now splint)
- Annotate libc headers
- Verify constraints on buffer read/write

LSE Security System
Laboratory of Epita

```c
void ugly(char *src) {
  char          buf[8];
  strcpy(buf, src);
}

int main(int argc, char *argv[]) {
  if (argc > 1) {
    ugly(argv[1]);
  }
  return 0;
}
```

splint detects `strcpy(buf, src)`

```
Possible out-of-bounds store: strcpy(buf, src)
[...]
A memory write may write to an address beyond the allocated
buffer.
```

LSE Security System
Laboratory of Epita

```c
void mystrcpy(char *dst, char *src) {
  for (; *src != '\0'; src += 1, dst += 1)
    *dst = *src;
}

static
void ugly(char *src) {
  char          buf[8];
  mystrcpy(buf, src);
}

int main(int argc, char *argv[]) {
  if (argc > 1) {
    ugly(argv[1]);
  }
  return 0;
}
```

Detected !

LSE Security System

Laboratory of Epita

```c
static void ugly2(char *src) {
  char          *buf1 = malloc(8);
  char           buf2[8];
  strncpy(buf1, src, 8);
  strcpy(buf2, buf1);
  free(buf1);
}

int main(int argc, char *argv[]) {
  if (argc > 1)
    ugly2(argv[1]);
  return 0;
}
```

Detected !

LSE Security System
Laboratory of Epita

```c
static void ugly2(char *src) {
  char          *buf1 = malloc(8);
  char           buf2[8];
  strncpy(buf1, src, 8);
  buf1[7] = '\0';
  strcpy(buf2, buf1);
  free(buf1);
}

int main(int argc, char *argv[]) {
  if (argc > 1)
    ugly2(argv[1]);
  return 0;
}
```

False Warning !

LSE Security System
Laboratory of Epita

```c
static void ugly2(char *src) {
  char          *buf1 = malloc(8);
  char           buf2[8];
  strncpy(buf1, src, 7);
  strcpy(buf2, buf1);
  free(buf1);
}

int main(int argc, char *argv[]) {
  if (argc > 1)
    ugly2(argv[1]);
  return 0;
}
```

No warning !

LSE Security System
Laboratory of Epita

Clang Analyzer

LSE Security System
Laboratory of Epita

clang static analyzer:

- ➢ analysis during semantic pass
- ➢ Reusable C++ library
- ➢ you can implement your own checker

LSE
Security System
Laboratory of Epita

- Complete C/C++/ObjC parser
- Full AST traversal
- Some checkers already available
- Still a little bit messy
- Out-of-the-box install doesn’t seem to detect simple buffer overflow

LSE
Security System
Laboratory of Epita

Errors and Vulnerabilities

LSE
Security System
Laboratory of Epita

- Static analysis detects possible code errors
- Code errors may be triggered by attackers
- Code errors may be exploitable

- Eliminating errors is important
- Any error may finally become a vulnerability
- Static analysis can help a lot
- Probably better during dev cycle

LSE
Security System
Laboratory of Epita

- Specific analysis only identifies known flaws
- Too much spurious warning
- Quality is a matter of involvement
  - People don’t review their code, so why analyzing it
  - Beta testing will be done by users
  - As long as it works ...