

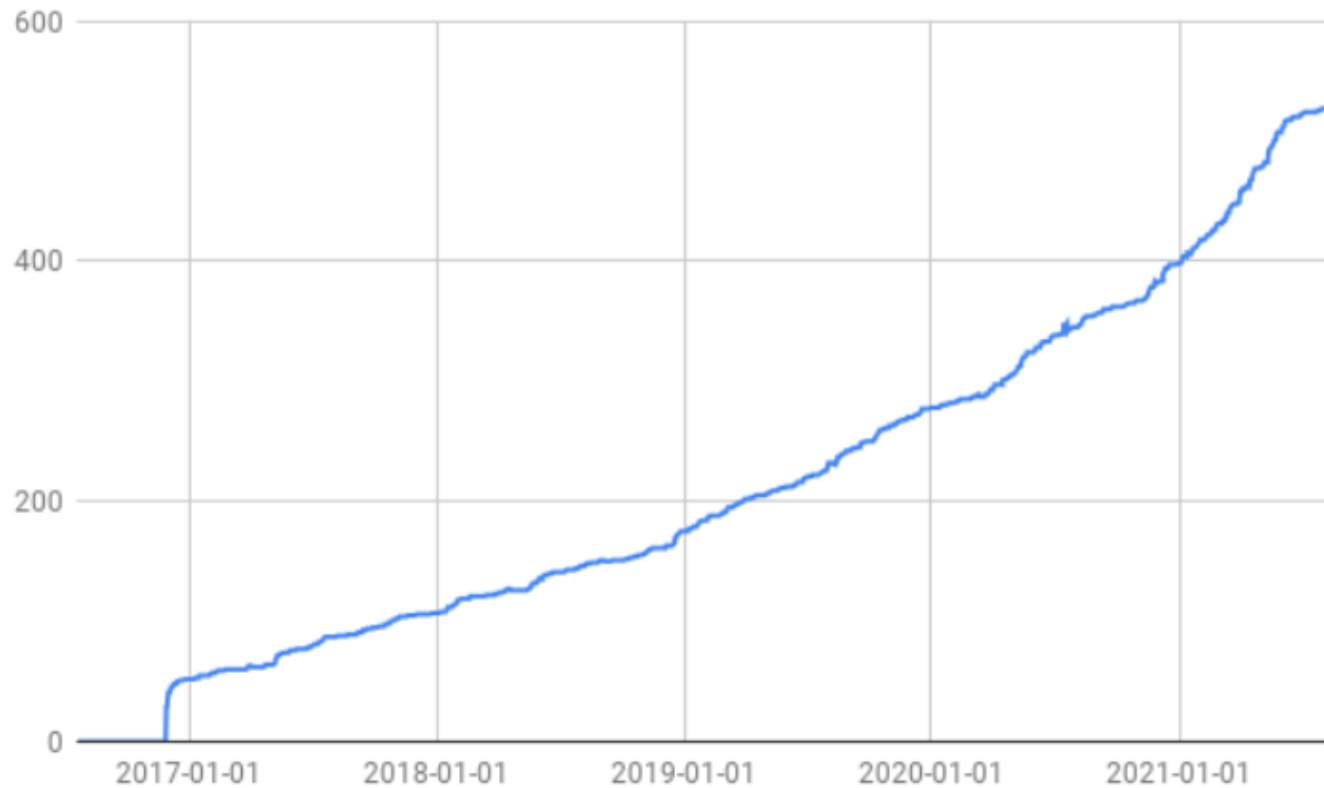
Taint Checker: Tracking the taint propagation on Binary Programs

Sangjun Park



Finding Software bugs Is Important

- Software bugs increase every years.



Bugs type

- Memory corruption
ex) bof / UAF / Double free bugs
- Non-memory corruption
ex) Logical bugs, command injection

How to detect the bugs
non-memory corruption?

Existing Methods

- Symbolic Execution
QSym, arbirter

- Taint analysis
SaTC, Taint pipe, code sonar, Condysta

- Concolic Execution
Symsan

Taint analysis

- Track information flow from input.



Taint analysis

- Source: origin of data.
- Sink: dangerous function, exploitable targets.

```
void ping(char *target) ← source
{
    char command[256];
    sprintf(command, "ping %s", target);
    system(command); ← sink
}
```

Taint analysis

- Taint Propagation
 - Mark source as taint.
 - Pass the taint if value is copied to another variable/buffer.
 - Check if argument to sink holds the taint.



Taint analysis

- Taint Propagation
 - Mark source as taint.
 - Pass the taint if value is copied to another variable/buffer.
 - Check if argument to sink holds the taint.

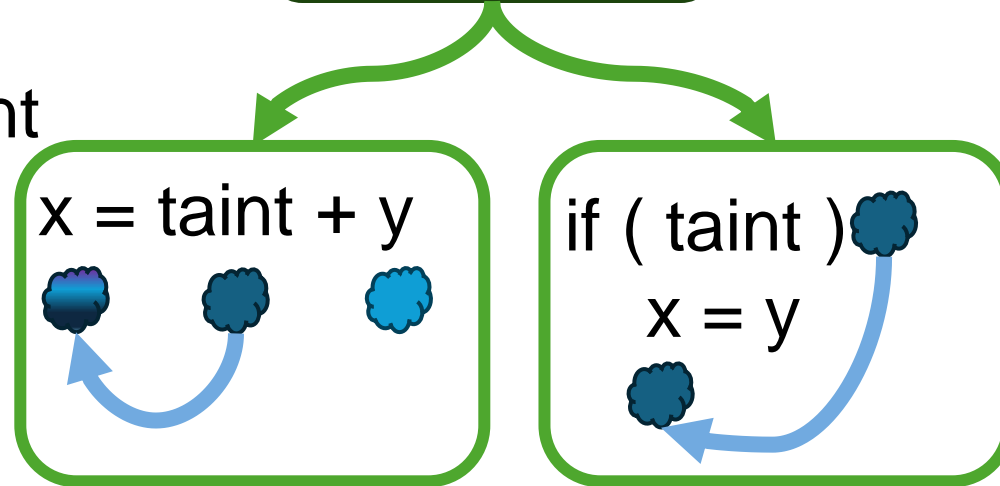
```
C test.c > ping(char *)  
1 void ping(char *target)  
2 {  
3     char command[256];  
4     sprintf(command, "ping %s", target);  
5     system(command);  
6 }
```



Taint analysis



- program argument
- keyboard
- network
- files



explicit
data flows

implicit
data flows

How to implement taint propagation?

Propagation Implementation

- Need to make a rule for each function or all data transmitted.
ex) explicit data flow / implicit data flow
- Use angr for binary analysis.
 - Can simulate the program.
 - Support ARM / MIPS.
 - Symbolic execution support.
 - Actively maintained.
 - Run program to track taints.



Propagation Implementation

- angr simulate program
- When we reach a copying function like sprintf
 - ➔ Pass the taint from the source string to the destination string
- When we reach the sink function
 - ➔ Check if the sink argument is tainted

```
void ping(char *target)
{
    char command[256];
    sprintf(command, "ping %s", target);
    system(command);
}
```

```
target: BitVec(1024*8)
command : BitVec(256*8)
```

```
target <- taint
command <- taint
```

when we reach the sink function, check if command is tainted.

Challenge 1

- Need to make a rule for each function or all data transmitted.

Best case

```
char *source;  
char *propagate_1;  
propagate_1 = source;  
sprintf(propagate_1, "%s", source);  
strcpy(propagate_1, source);  
system(propagate_1)
```

Challenge 1

- Need to make a rule for each function or all data transmitted.

Best case scenario

```
char *source;  
char *propagate_1;  
  
propagate_1 = source;  
  
sprintf(propagate_1, "%s", source);
```

Worst case scenario

```
char *source;  
char *propagate_1;  
  
user_defined_function(propagate_1, source);
```

If you set the propagation rule incorrectly for user_defined_function, you will not be able to find the bug.


Challenge 2

- Need to make a rule for each function or all data transmitted.

Best case scenario

```
char *source;  
char *propagate_1;  
  
propagate_1 = source;  
  
sprintf(propagate_1,"%s",source);  
  
strcpy(propagate_1,source);  
  
system(propagate_1)
```

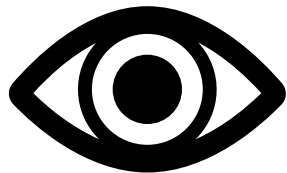
Worst case scenario2

```
char *x = "/bin/sh";  
char *y = "/bin/ls";  
int source;  
  
  
if( source )  
    x = y  
system(x);
```


How to find the limitation of
propagation rule?

Observation

- we can evaluate the propagation rules through visualizing.
- Can find limitation of current propagation rules.



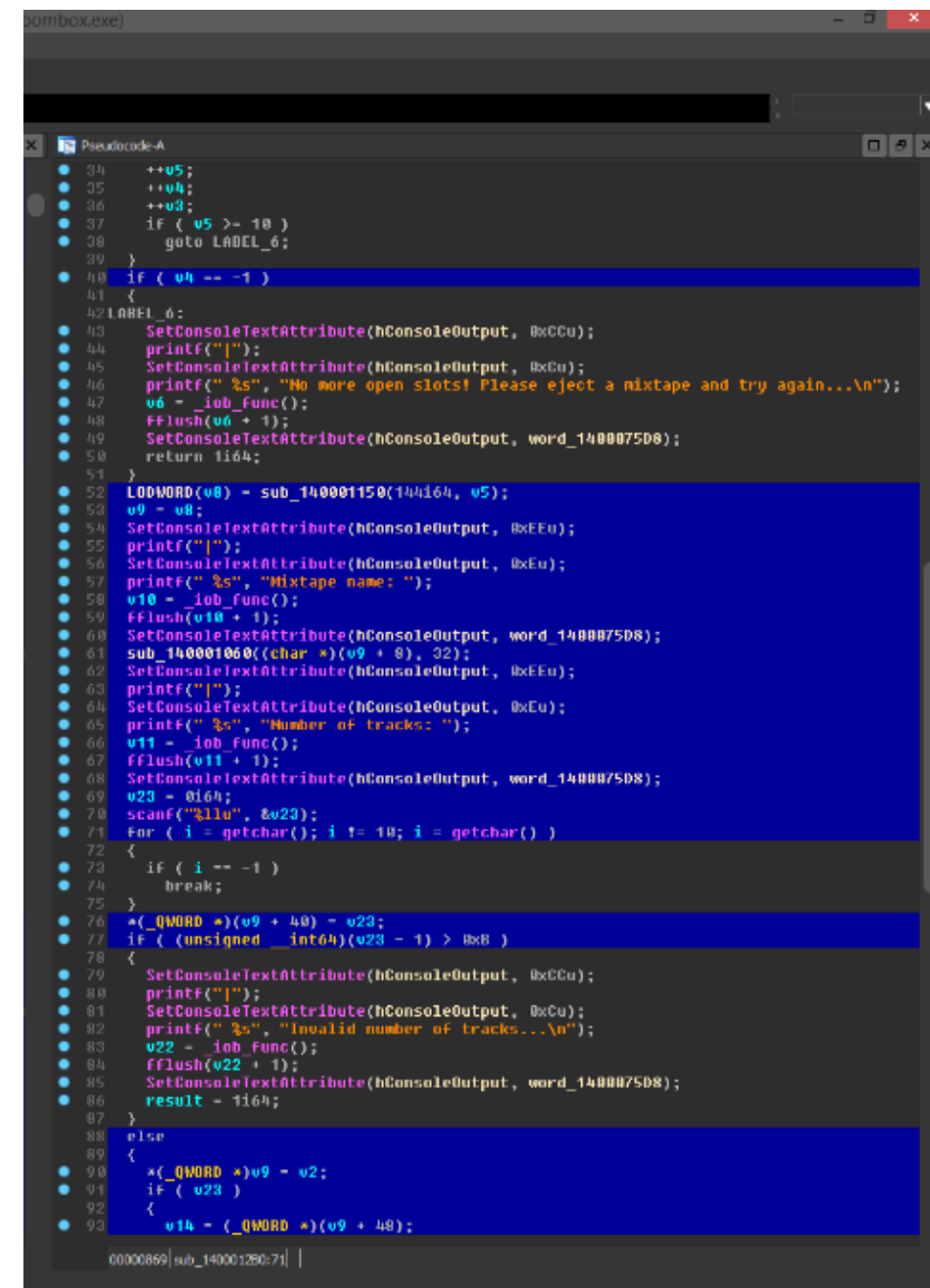
Observation

- key idea
 - Matches the process in which taint propagation occurs and the decompiler source code.

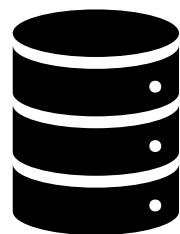


Observation

- Coverage Viewer
- IDA Light House



Observation



Observation

- Remark where is source and sink
- Remark propagation

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    const char *source; // [rsp+0h] [rbp-120h]
    char propagation[264]; // [rsp+10h] [rbp-110h] BYREF
    unsigned __int64 v6; // [rsp+118h] [rbp-8h]

    v6 = __readfsqword(0x28u);
    source = (const char *)input(argc, argv, envp);
    if ( source )
        printf("AAAAAAAAAA");
    else
        printf("BBBBBBBBBB");
    sprintf(propagation, "%s", source);
    if ( !malloc(0x100uLL) )
        exit(1);
    system(propagation);
    return 0;
}
```

Expectation

- Find the limitation of taint propagation implement
- Find ways to improve to create better taint analysis tools.

Thank You

Questions?

Email: sangjuns@kaist.ac.kr