
How you can find dragons
Static Analysis for beginners
designed by freepik.com

whoamy

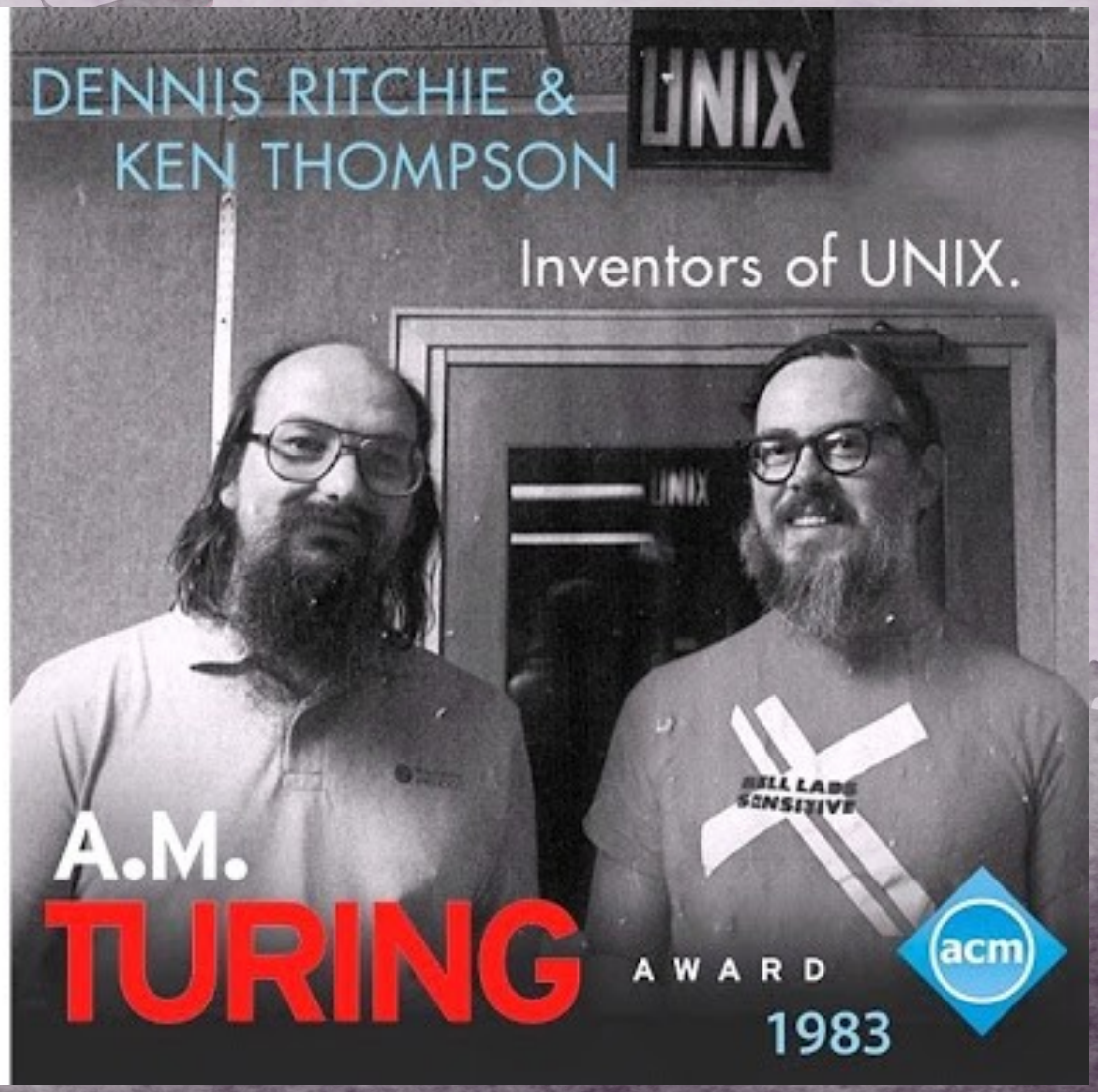
- **Just another Programmer**
- **Security Engineer**
- **Ten years experience**

- **About me:** Github.com/**CoolerVoid**
  Twitter: @Cooler_freenode
  Contact: coolerlair@gmail.com

Etymology

Importance of Bell Labs
Unix

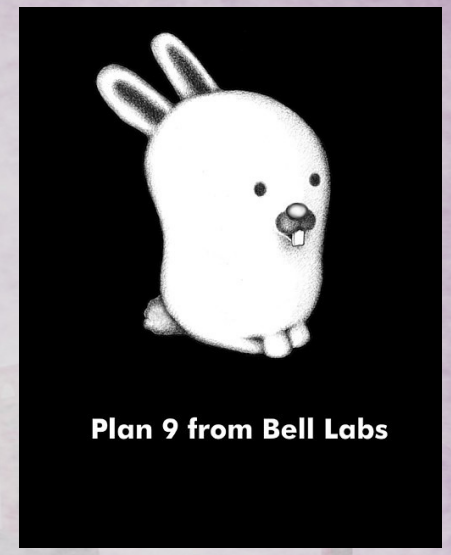
portable operating system with source
multi-user & multi-tasking
text processing tools such as nroff
stdio
c compiler
pcc a portable c compiler
bourne shell
awk, yacc, lex, lint, sed, m4, make
fortran compiler
cal calendar
uucp
grep



Etymology

- theguardian.com/technology/2011/oct/13/dennis-ritchie
- economist.com/obituary/2011/11/05/dennis-ritchie-and-john-mccarthy

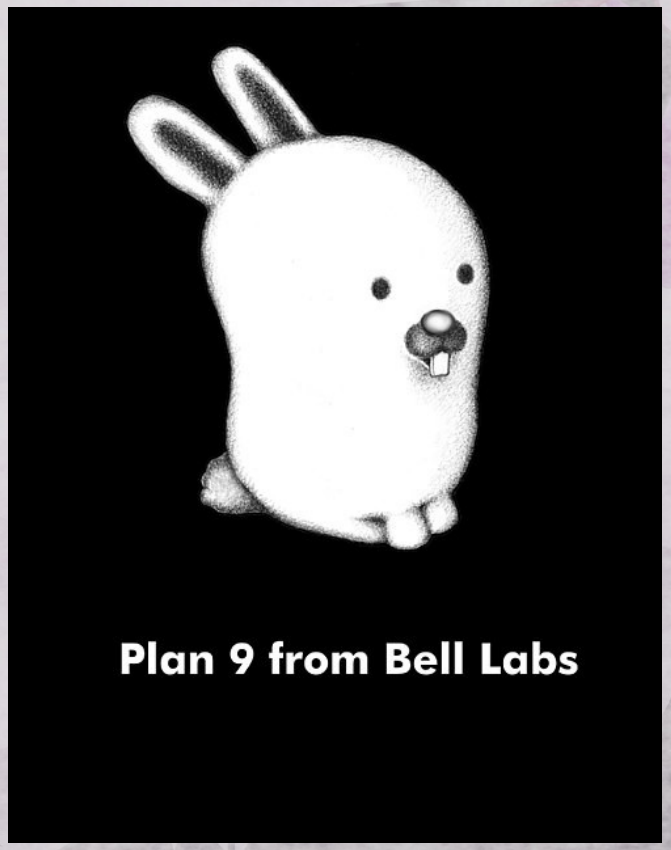
Etymology

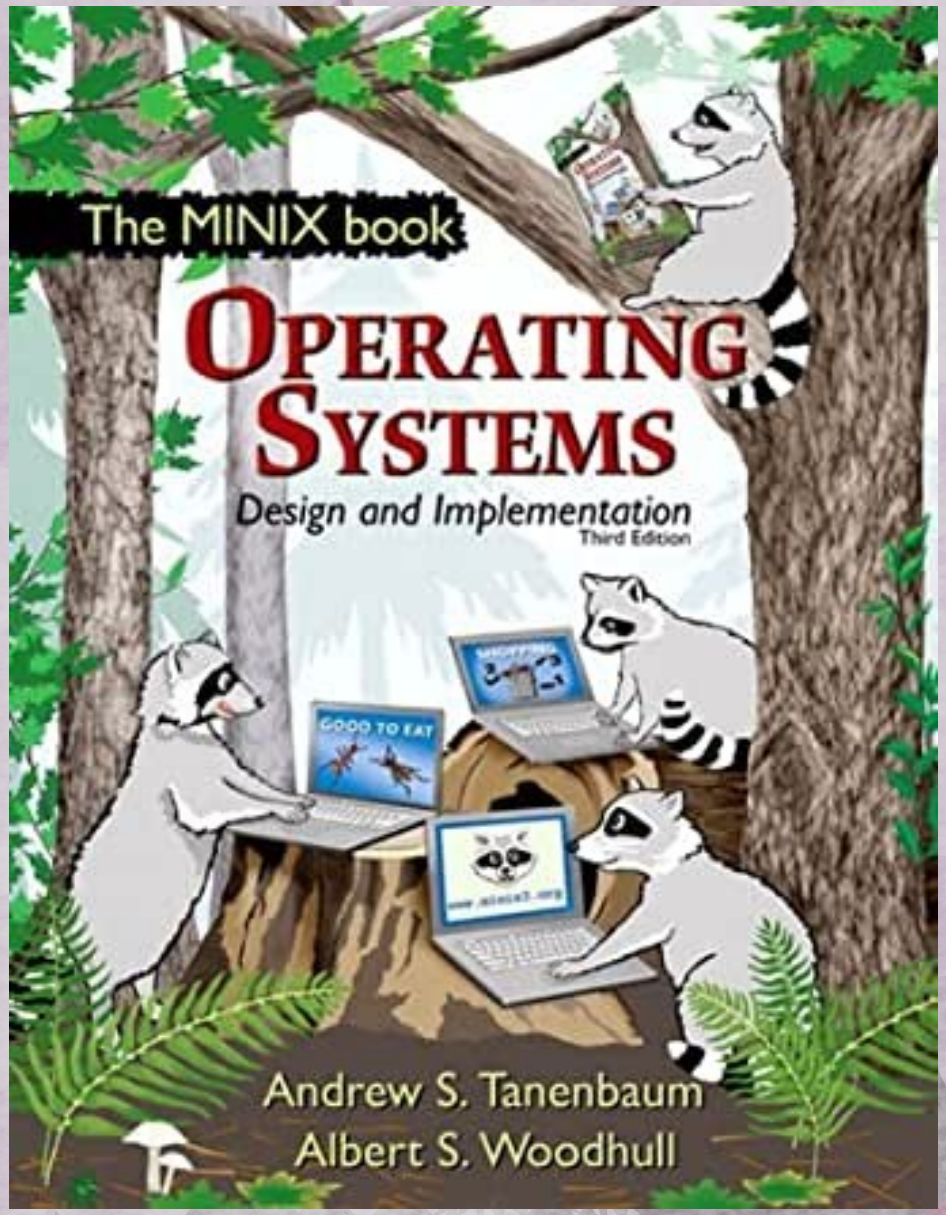


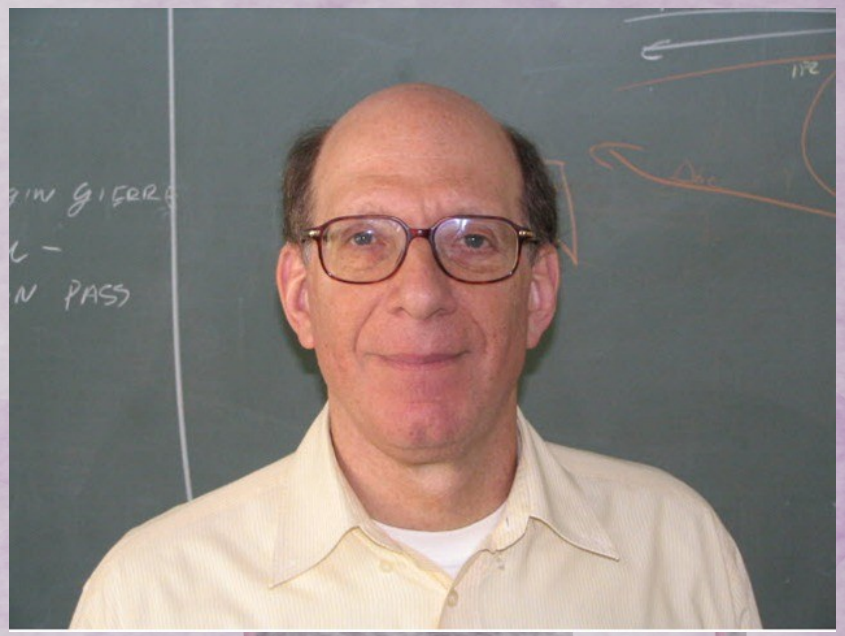
- **Plan9**
- Plan 9 from Bell Labs was originally developed, starting in the late **1980s**.
- members of the Computing Science Research Center at Bell Labs
- the same group that originally developed Unix and the C programming language. The Plan 9 team was initially led by **Rob Pike, Ken Thompson, Dave Presotto and Phil Winterbottom**, with support from **Dennis Ritchie** as head of the Computing Techniques Research Department.
- First release in **1993**

Etymology

- Compare logo







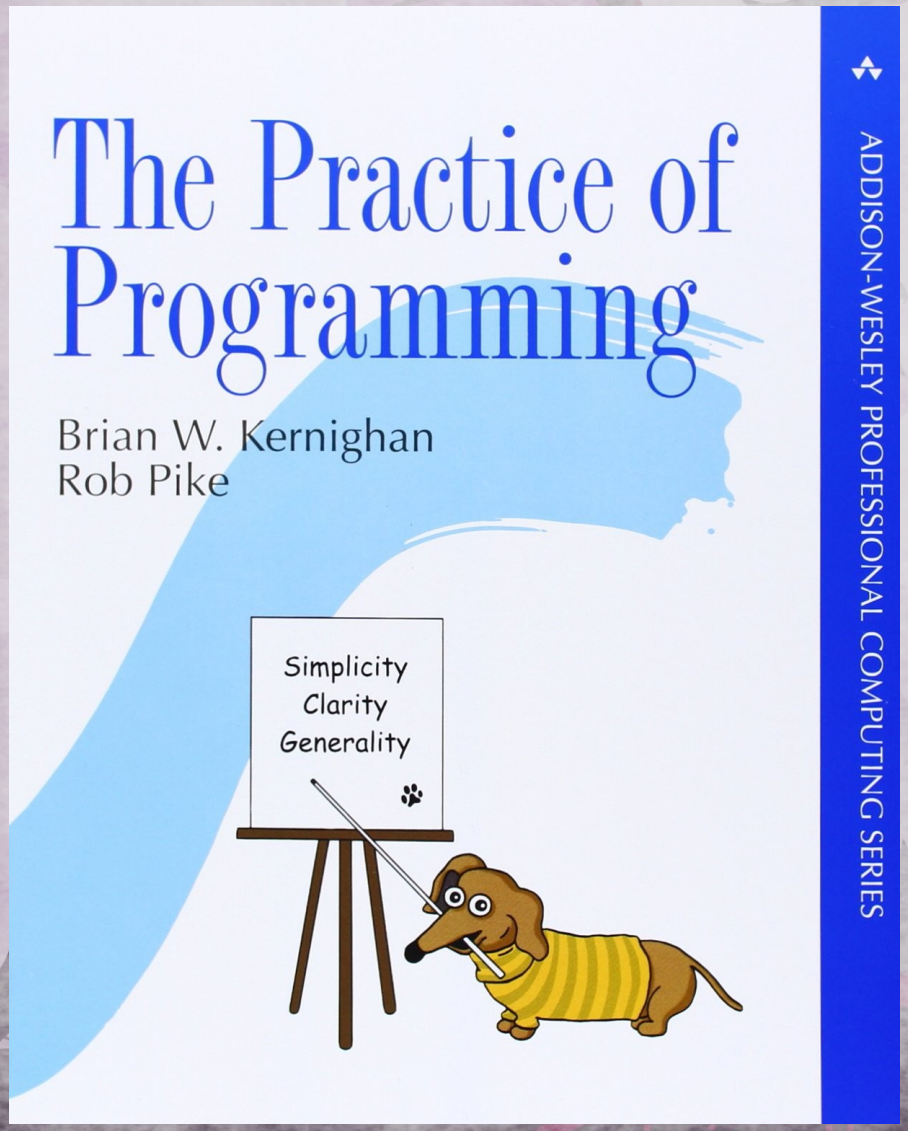
- **Minix 1987**
- **Linux 1991**
- **Plan9 1993**

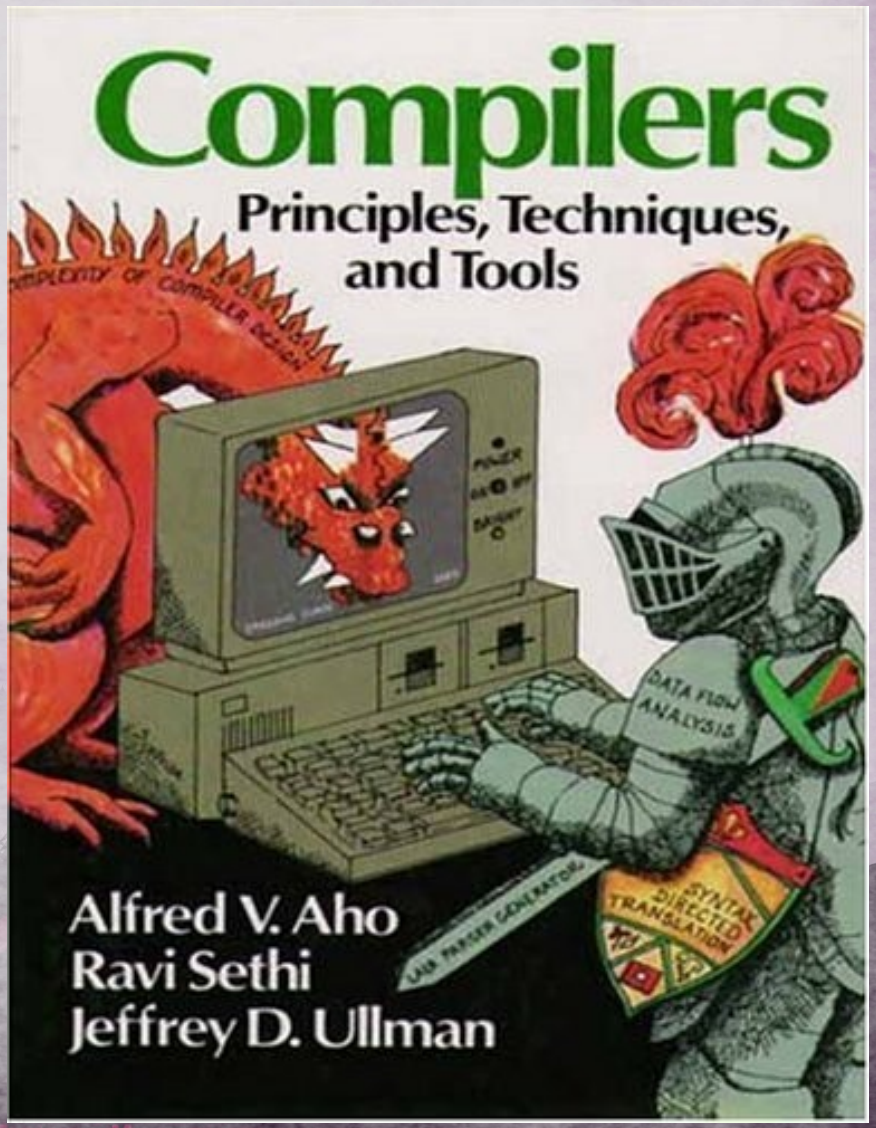
- **Minix 1987**
- **Linux 1991**
- **Plan9 1993**

Linux/Git creation

- **Version control**
- **Manual Codereview**
- **New tools for static analysis in Kernel or drivers to make auto patch etc...**

The root of study





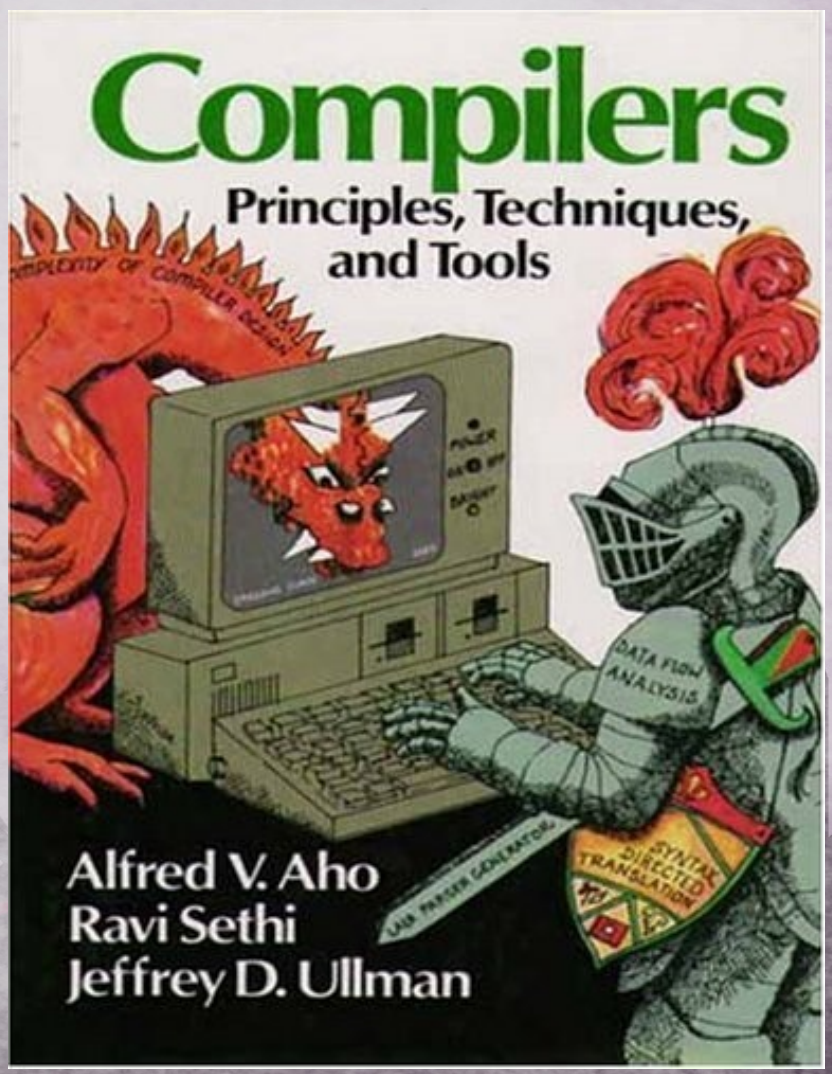
Rob Pike's 5 Rules of Programming

- **Rule 1** - You can't tell where a program is going to spend its time. Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you've proven that's where the bottleneck is.
- **Rule 2** - Measure. Don't tune for speed until you've measured, and even then don't unless one part of the code overwhelms the rest.
- **Rule 3** - Fancy algorithms are slow when n is small, and n is usually small. Fancy algorithms have big constants. Until you know that n is frequently going to be big, don't get fancy. (Even if n does get big, use Rule 2 first.)
- **Rule 4** - Fancy algorithms are buggier than simple ones, and they're much harder to implement. Use simple algorithms as well as simple data structures.
- **Rule 5** - Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.

- Pike's rules 1 and 2 restate Tony Hoare's famous maxim "**Premature optimization is the root of all evil**." Ken Thompson rephrased Pike's rules 3 and 4 as "**When in doubt, use brute force**.". Rules 3 and 4 are instances of the design philosophy KISS. Rule 5 was previously stated by Fred Brooks in The Mythical Man-Month. Rule 5 is often shortened to "write stupid code that uses smart objects".

The root of study

- **The Dragon Book**
- Flex
- Bison
- AST
- Trees
- Tokenizer

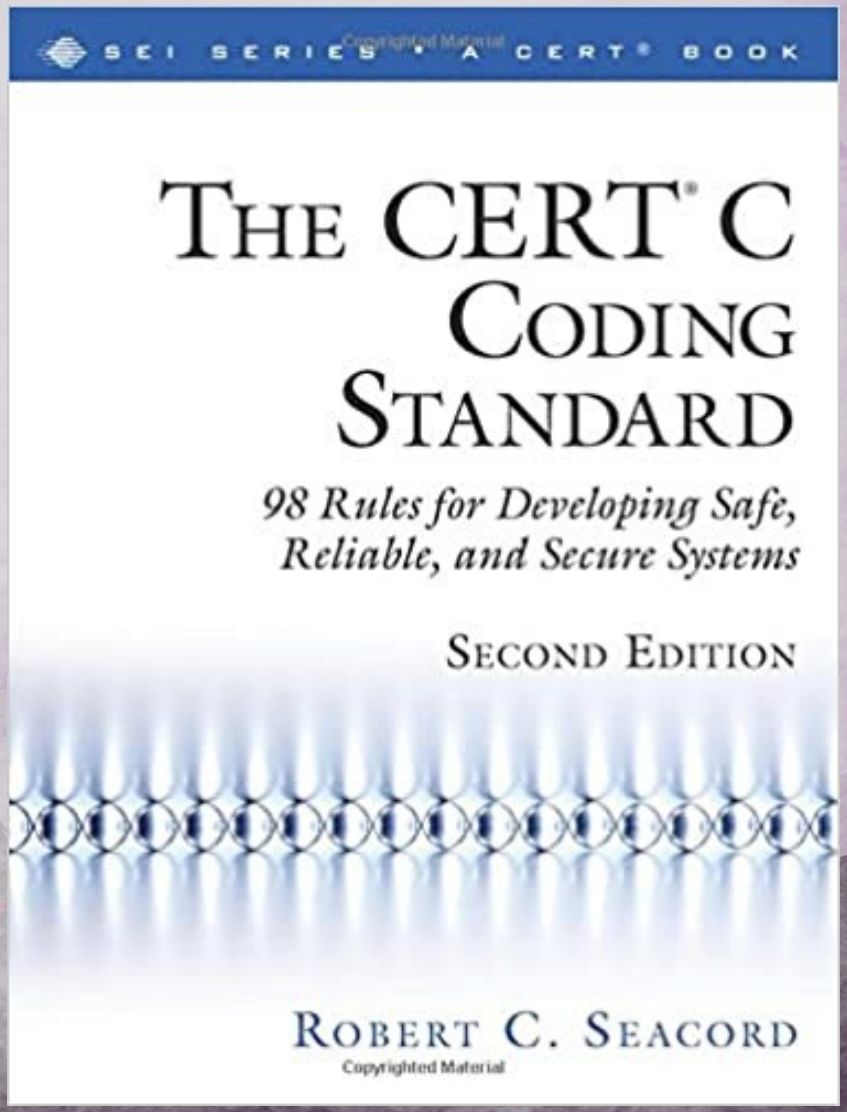


Here be dragons !

- Find Pitfalls
- Fix each point
- Mitigate
- Sometimes its hard…
- Education ???

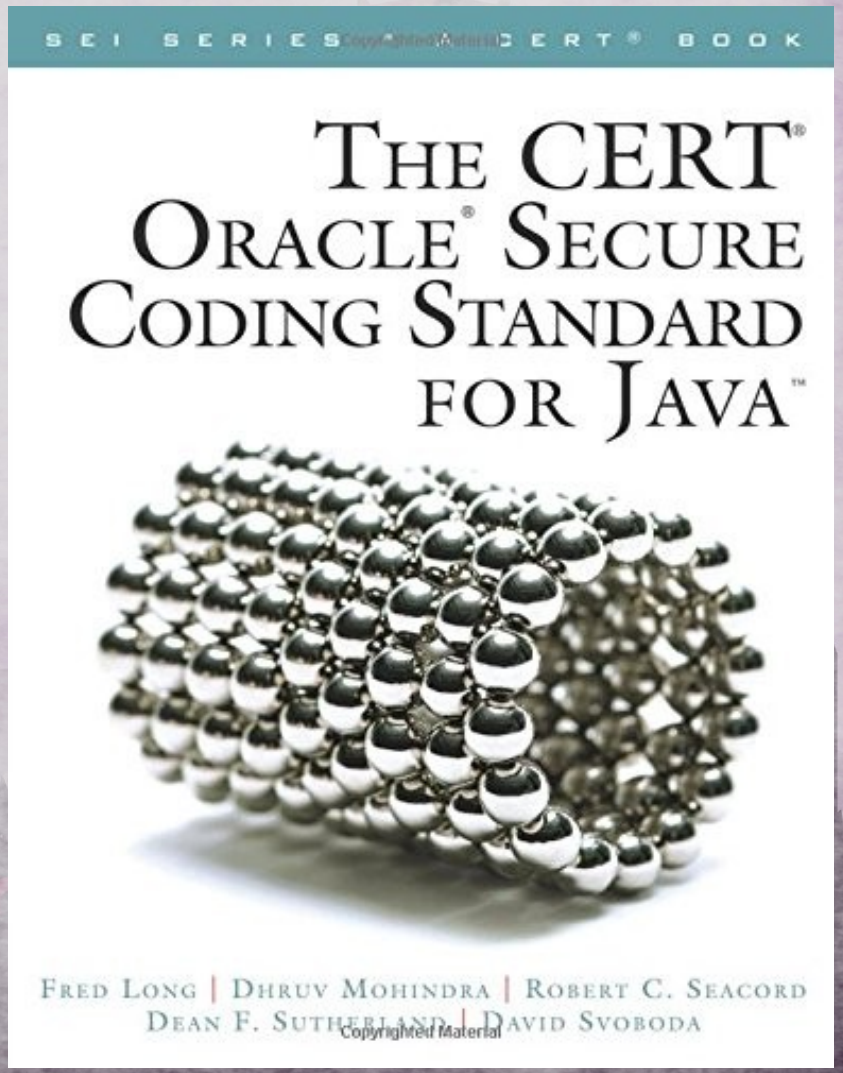
Security focus

- Find Pitfalls
- Fix each point
- Mitigate
- Sometimes its hard…
- **Education ???**
- **Search dragons !**



Security focus

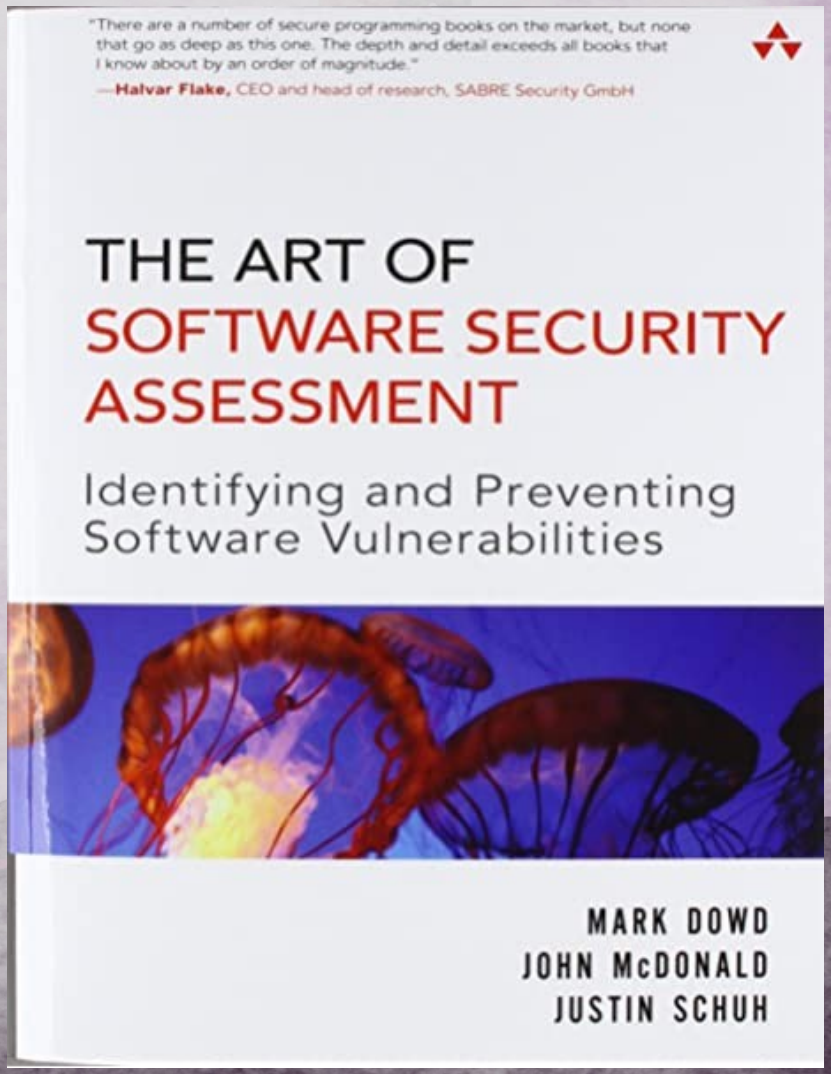
- Find Pitfalls
- Fix each point
- Mitigate
- Sometimes its hard…
- **Education ???**
- **Search dragons !**



Security focus

- Mitre
- OWASP
- **Other resources…**

- **Owasp.org**
- **cve.mitre.org**



File system Pitfalls

- File system problems
- Call Open() but not call close()
- Load config file, but don’t have lock…
- Don’t check permissions to open file
- Don’t check existence of file
- Race condition (TOCTOU)
- Mistake in permissions

Pitfall example 1

- File system problems
- **Call Open() but not call close()**
- Load config file, but don’t have lock…
- Don’t check permissions to open file
- Don’t check existence of file
- Race condition (TOCTOU)
- Mistake in permissions

Pitfall example 1

- This code example is noncompliant because the file opened by the call to fopen() is not closed before function func() returns:

```
#include <stdio.h>

int func(const char *filename) {
  FILE *f = fopen(filename, "r");
  if (NULL == f) {
    return -1;
  }
  /* ... */
  return 0;
}
```

Pitfall example 1

- This code example is right way, because open and close file:

```
#include <stdio.h>

int func(const char *filename) {
  FILE *f = fopen(filename, "r");
  if (NULL == f) {
    return -1;
  }
  /* ... */
  if (fclose(f) == EOF) {
    return -1;
  }
  return 0;
}
```

Detection

- Do you remember **the dragon book** ?
- You can use **DFA**(**D**eterministic **F**inite **A**utomaton) to solve this with rank points.
- You can **tokenize** each word and save in nodes, you can load data structure and walk to collect each rule, the data structure you can use Tree, AST, graph(**this is common but more complex**).
- You can use **Flex+Bison** to generate input extractor and parser…
- You can use **regex(regular expression)**, but don't have a good performance! Its not better path!

- Relax here! have other paths to following…

Detection

- Do you remember **the dragon book** ?
- **You can use DFA(Deterministic Finite Automaton) to solve this with rank points.**
- You can **tokenize** each word ans save in nodes, you can load data structure and walk to collect each rule, the data structure you can use Tree, AST, graph(**this is common but more complex**).
- You can use **Flex+Bison** to generate input extractor and parse rules…
- You can use **regex(regular expression)**, but don’t have a good performance! Its not better path!

- Relax here! have other paths to following…

Detection Ex 1

- Its OK my choice is use Re2c to solve the problem!
- Re2c is a free and open-source lexer generator for C, C++ and Go. It compiles regular expressions to **determinisitic finite automata** and encodes the automata in the form of a program in the target language.
- The main advantages of re2c are **speed of the generated code** and a flexible user interface that allows one to adapt the generated lexer to a particular environment and input model.
- Re2c supports fast and lightweight submatch extraction with either POSIX or leftmost greedy semantics.

Detection Ex 1

- github.com/CoolerVoid/**heap_detective**/tree/master/**doc/PoC1**
- **doc/PoC1/rule.c**
- **Rule to generate**
- **Need Re2c tool**
- **Need GCC**

```c
int parse_ion(char** p, char** lex)
{
    char* marker;

    for (;;) {
    *lex = *p;
        /*!re2c
        re2c:define:YYCTYPE  = "char";
        re2c:define:YYCURSOR = *p;
        re2c:define:YYMARKER = marker;
        re2c:yyfill:enable   = 0;
        re2c:yych:conversion = 0;
        re2c:indent:top      = 1;
        "open".*            { return OPEN; }
        "close".*               { return CLOSE; }
        "\x0a"              { return COUNTER; }
        "\x00"               { return END; }
        [^]                  { continue; }
    */
    }
    free(lex);
    free(marker);
}
```

Detection Ex 1

- github.com/CoolerVoid/**heap_detective**/tree/master/**doc/PoC1**
- **doc/PoC1/gen_re2.c**
- **Rule to generate**
- **Need Re2c tool**
- **Need GCC**

```
14  int parse_ion(char** p, char** lex)
15  {
16      char* marker;
17
18      for (;;) {
19      *lex = *p;
20
21  #line 22 "gen_re2.c"
22          {
23                  char yych;
24                  yych = **p;
25                  switch (yych) {
26                  case 0x00:      goto yy2;
27                  case '\n':      goto yy6;
28                  case 'c':       goto yy8;
29                  case 'o':       goto yy9;
30                  default:        goto yy4;
31                  }
32  yy2:                    Other lines...
```

```
45      while(!result )
46          switch (parse_ion(&p, &last))
47          {
48                  case OPEN:
49                  opens++;
50                  test=1;
51                  printf("OPEN function at line %d\n",line_number); /
52                  break;
53
54
55                  case CLOSE:
56                  closes++;
57                  test=0;
58                  printf("CLOSE function at line %d\n",line_number);
59                  break;
60
61                  case COUNTER:
62                  line_number++;
63                  break;
64
65                  case END:
66                  result=1;
67                  break;
68          }
```

- github.com/CoolerVoid/**heap_detective**/tree/master/**doc/PoC1**

Detection Ex 1

```
3   char *Read_file_unsafe(char * NameFile)
4   {
5           FILE * arq=NULL;
6
7           arq = fopen(NameFile, "rx");
8
9           if( arq == NULL )
10          {
11                  exit(1);
12          }
13
14          char *lineBuffer=calloc(1,1);
15          char line[2048];
16  Close file ?
17          while( fgets(line,sizeof line,arq) )
18          {
19                  lineBuffer=realloc(lineBuffer,strlen
20                  strncat(lineBuffer,line,2048);
21          }
22
23          arq=NULL;
```

- github.com/CoolerVoid/**heap_detective**/tree/master/**doc/PoC1/extest.c**

Detection Ex 1

```
cooler@ubuntu:~/static/heap_detective/doc/PoC1$ pwd; tree; date
/home/cooler/static/heap_detective/doc/PoC1
.
├── extest.c
├── gen_re2.c
├── rule.c
├── test
└── test.sh

0 directories, 5 files
Sat Sep 19 00:56:32 -03 2020
cooler@ubuntu:~/static/heap_detective/doc/PoC1$ chmod +x test.sh; ./test.sh | tail -n 6
OPEN function at line 6
OPEN function at line 37
CLOSE function at line 54

 Here be dragons here!
```

- github.com/CoolerVoid/**heap_detective**/tree/master/**doc/PoC1/test.sh**
- **Note need re2c and gcc! (apt-get install re2c gcc)**

Detection Ex 1

The logic its simple !

```
    if(opens!=closes)
        puts("\n Here be dragons here!\n");
    else
        puts("\n Cannot detect pitfalls here!");
```

Detection Ex 1

```
cooler@ubuntu:~/static/heap_detective/doc/PoC1$ pwd; tree; date
/home/cooler/static/heap_detective/doc/PoC1
.
├── extest.c
├── gen_re2.c
├── rule.c
├── test
└── test.sh

0 directories, 5 files
Sat Sep 19 00:56:32 -03 2020
cooler@ubuntu:~/static/heap_detective/doc/PoC1$ chmod +x test.sh; ./test.sh | tail -n 6
OPEN function at line 6
OPEN function at line 37
CLOSE function at line 54

 Here be dragons here!
```

- github.com/CoolerVoid/**heap_detective**/tree/master/**doc/PoC1/test.sh**
- **Note need re2c and gcc! (apt-get install re2c gcc)**

Detection

- Do you remember **the dragon book** ?
- You can use **DFA**(**D**eterministic **F**inite **A**utomaton) to solve this with rank points.
- You can **tokenize** each word ans save in nodes, you can load data structure and walk to collect each rule, the data structure you can use Tree, AST, graph(**this is common but more complex**).
- You can use **Flex+Bison** to generate input extractor and parse rules…
- You can use **regex(regular expression)**, but don’t have a good performance! Its not better path!

- Relax here! have other paths to following…

Detection

- Do you remember **the dragon book** ?
- You can use **DFA**(**D**eterministic **F**inite **A**utomaton) to solve this with rank points.
- **You can tokenize each word ans save in nodes, you can load data structure and walk to collect each rule, the data structure you can use Tree, AST, graph(this is common but more complex).**
- You can use **Flex+Bison** to generate input extractor and parse rules…
- You can use **regex(regular expression)**, but don't have a good performance! Its not better path!

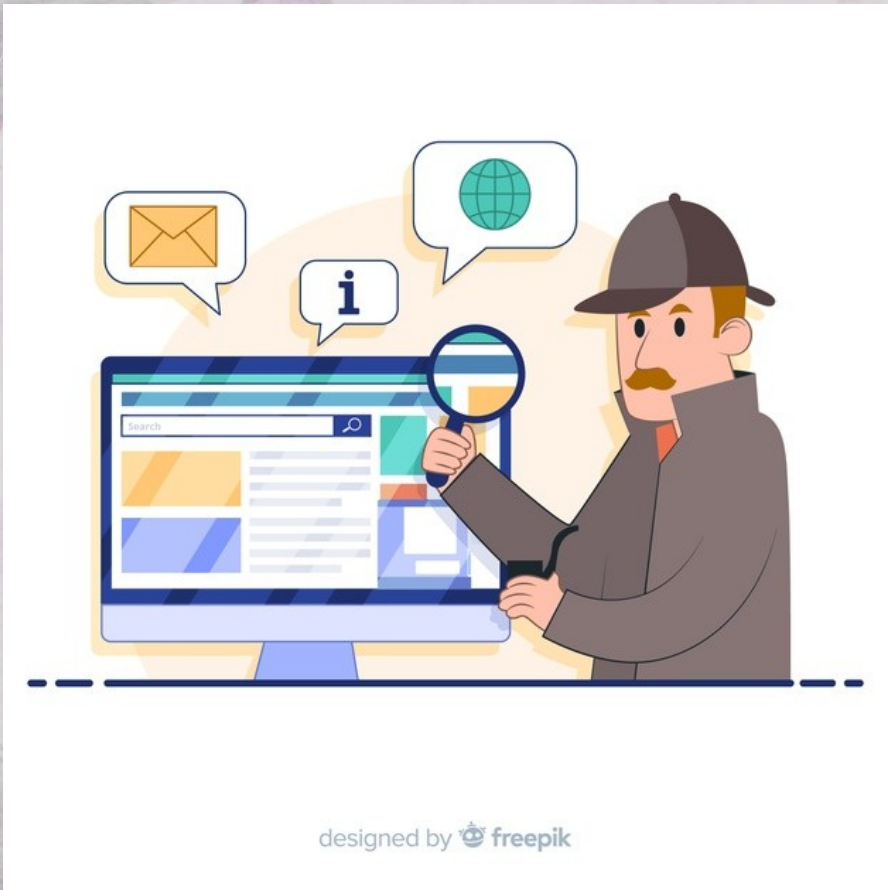
- Relax here! have other paths to following…

Heap detective

- All languages uses heap memory
- In C its commom when you use functions like **malloc()**, **calloc()**, **realloc()**, **strdup()** etc…
- In C++ its common when you use “**new**”.
- Heap use can have a lot pitfalls if you not follow good practices.
- Memory leak, double free, use after free, wild pointer, heap overflow, crash(DoS) other pitfalls…
- Some languages like Java have garbage collector to clean the heap memory to manage this, but if programmer don’t know good practices the problem with memory leak or crash can be found.

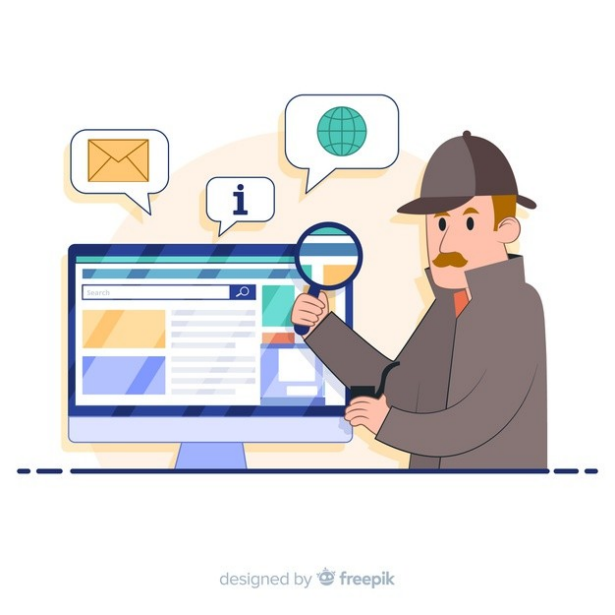
Heap detective

- How you can map heap memory use ?



Heap detective

- How you can map heap memory usage in static analysis ?
- Use my Tool heap detective
- https://github.com/**CoolerVoid/heap_detective**



Heap detective

- How you can map heap memory use ?
- List the functions that use heap memory
- List functions to liberate heap

```
8           ch_ptr = malloc(100);
9           int i=0;
10
11          while (i < 99)
12          {
13                  ch_ptr = 'A';
14                  free(ch_ptr);
15                  printf("%s\n", ch_ptr);
16                  i++;
17          }
```

Overview

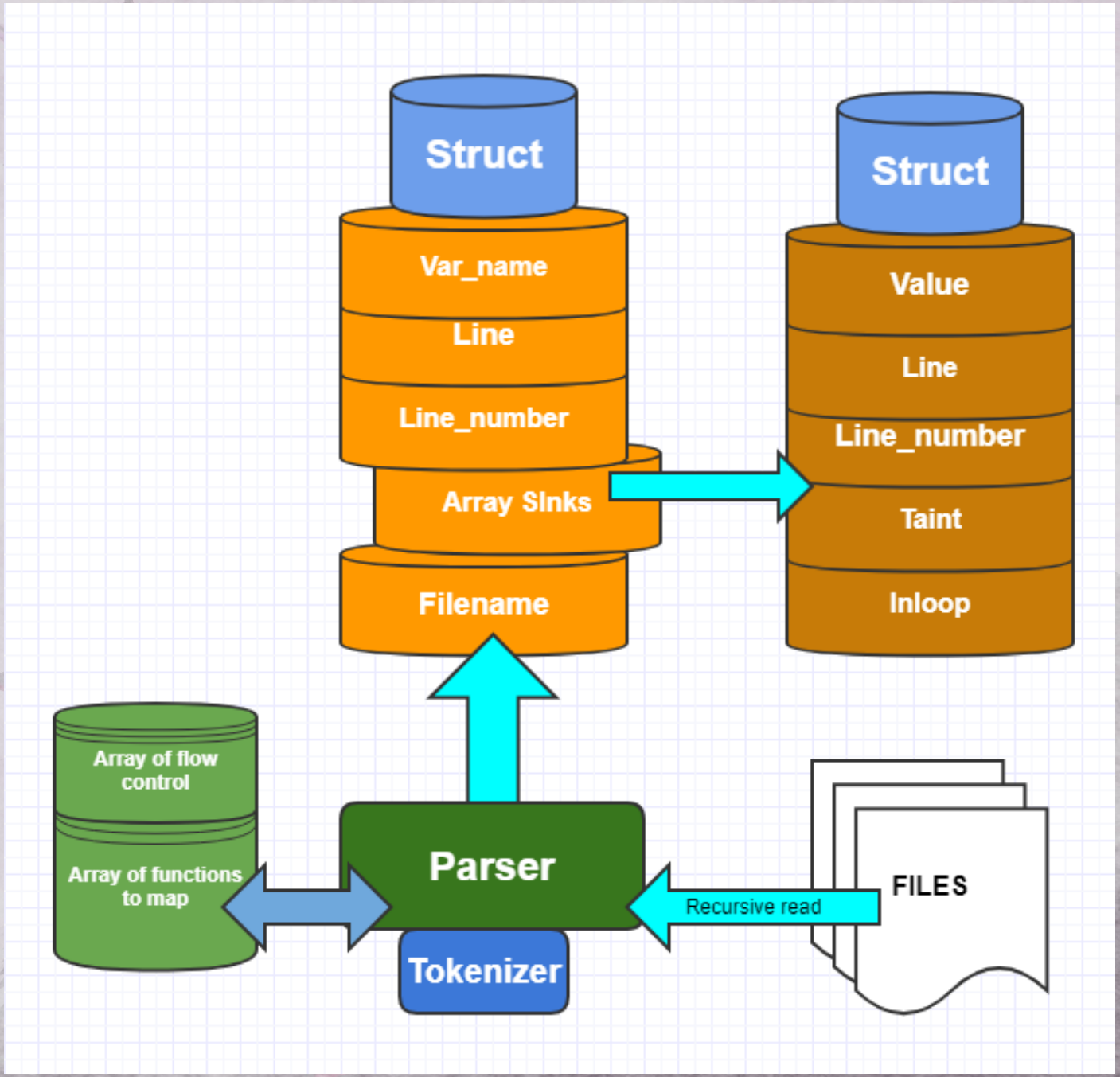
```
heap_detective/
├── bin
│   └── heap_detective
├── doc
│   └── PoC1
│       ├── extest.c
│       ├── gen_re2.c
│       ├── rule.c
│       ├── test
│       └── test.sh
├── license
├── Makefile
├── README.md
├── samplers
│   ├── example1.c
│   ├── example2.c
│   ├── example3.c
│   ├── example4.c
│   ├── example5.c
│   └── example6.c
└── src
    ├── Detective.cpp
    ├── Detective.h
    ├── heap_detective.cpp
    ├── Tokenizer.cpp
    └── Tokenizer.h
```

Heap detective

```
// This is array of functions names that use HEAP
// brk(), sbrk() ? at the future
        vector<string> heap_in = {"kmalloc","malloc","xmalloc","kmalloc","realloc","krealloc","xrealloc","calloc","kcalloc",
        vector<string> heap_out = {"free","xfree","FREE","XFREE","delete ","kfree"};
        vector<string> loop_in = {"for","while","do"};
        vector<string> cond = {"if","else","elseif","switch"};
        vector<string> files_path; // list of paths to open each file...
        vector<startpoint> array;
        vector<sink> sinks;
```

Flow control

```
if(!heap_in[count_functions].empty())
        if ( (line.find(heap_in[count_functions],0)!=string::npos) && (line.find(array[pos2].var_name)!=stri
        {
                sink makestruct2 = {array[pos2].var_name,line,line_counter,true,loop_counter>=1?true:false};

                if(array[pos3].sinks.size() != SIZE_MAX)
                {
                        array[pos2].sinks.push_back(makestruct2);
                        heap_test=true;
                } else {
                        HEAP_DETECTIVE_DEBUG("ERROR: Out of limit in array.sinks vector");
                        exit(0);
                }
```



Heap detective

```
...::: Heap static route :::...
File path: samplers/example6.c
Var name: X
line: 7:        X = malloc(100);
Sinks:
        line: 7:        X = malloc(100);
        Taint: True
        In Loop: false
        line: 9:        X = malloc(101);
        Taint: True
        In Loop: false
        line: 13:               X = 'A';
        Taint: false
        In Loop: True
        line: 16:       X=malloc(1023);
        Taint: True
        In Loop: false
        line: 17:       free(X);
        Taint: false
        In Loop: false

...::: Heap static route :::...
File path: samplers/example6.c
Var name: X
line: 9:        X = malloc(101);
Sinks:
        line: 13:               X = 'A';
        Taint: false
        In Loop: True
```

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main(void)
5  {
6          char *X = NULL;
7          X = malloc(100);
8          int i;
9          X = malloc(101);
10
11         for (i = 0; i < 99; i++)
12         {
13                 X = 'A';
14                 printf("%s\n", ch_ptr);
15         }
16         X=malloc(1023);
17         free(X);
18
19
20         return 0;
21
22 }
~
"samplers/example6.c" 22L, 255C
```

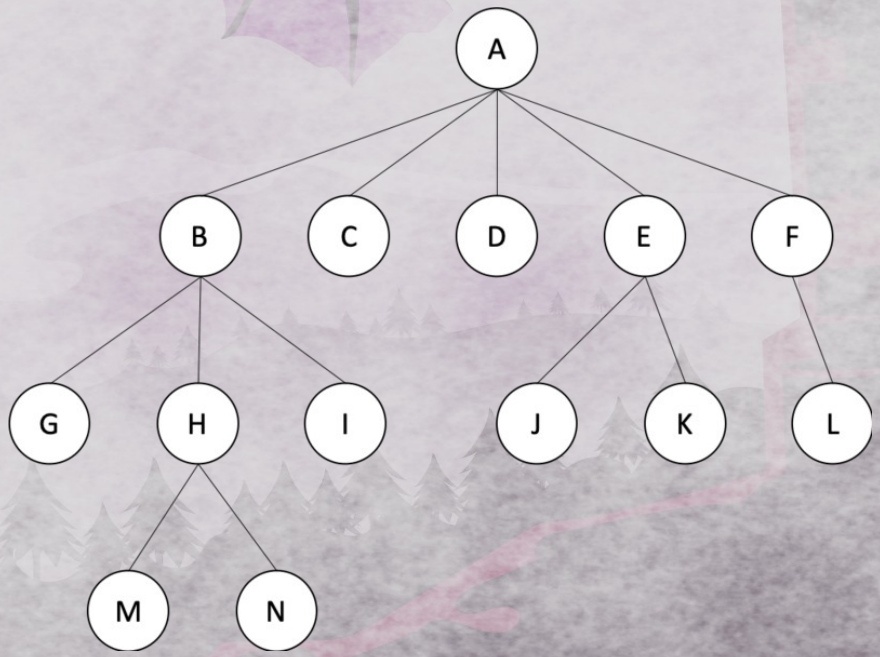
Heap detective

```
        In Loop: True

 ...::: Heap static route :::...
File path: samplers/example4.c
Var name: ch_ptr
line: 8:        ch_ptr = malloc(100);
Sinks:
        line: 8:        ch_ptr = malloc(100);
        Taint: True
        In Loop: false
        line: 13:               ch_ptr = 'A';
        Taint: false
        In Loop: True
        line: 14:               free(ch_ptr);
        Taint: True
        In Loop: True
        line: 15:               printf("%s\n", ch_ptr);
        Taint: false
        In Loop: True

 ...::: Heap static route :::...
```

```c
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
        char *ch_ptr = NULL;

        ch_ptr = malloc(100);
        int i=0;

        while (i < 99)
        {
                ch_ptr = 'A';
                free(ch_ptr);
                printf("%s\n", ch_ptr);
                i++;
        }

        return 0;

}
```

Cool stuff

- My Tree library in C
- Ice n-ary tree based on glib tree functions
- https://github.com/**CoolerVoid/icenarytree**



Other projects

- Code walk, code search, regex with rule based…
- github.com/**CoolerVoid/codewarrior**
- github.com/**CoolerVoid/codecat**

- **https://semgrep.dev (very cool)**

Questions ?

Thank you!

Credits: CoolerVoid

Contact: coolerlair@gmail.com