



How you can find dragons

# Static Analysis for beginners

designed by  freepik.com

# whoamy

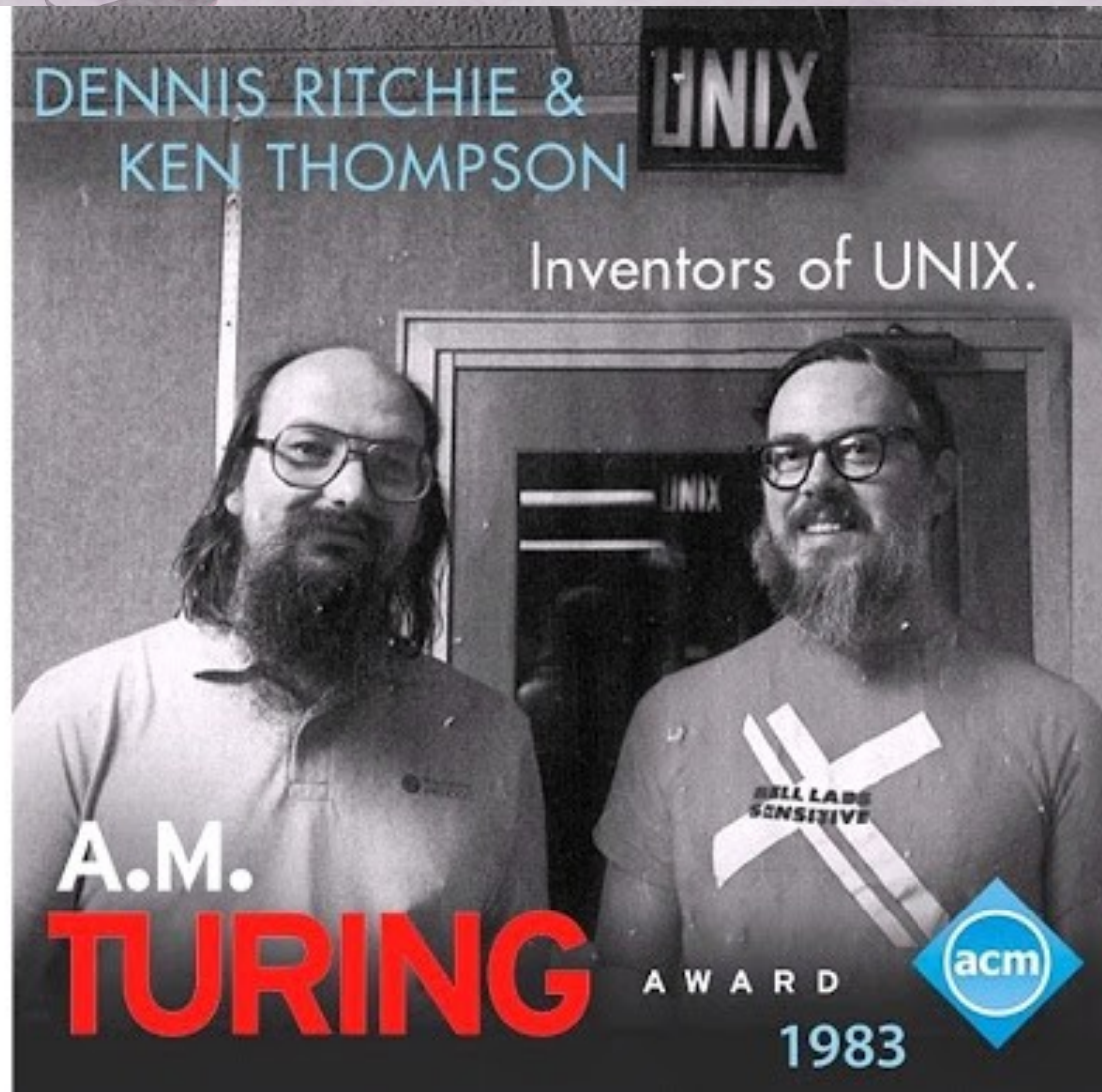
- **Just another Programmer**
  - **Security Engineer**
  - **Ten years experience**
- 
- **About me: [Github.com/CoolerVoid](https://github.com/CoolerVoid)**  
**Twitter: [@Cooler\\_freemove](https://twitter.com/Cooler_freemove)**  
**Contact: [coolerlair@gmail.com](mailto:coolerlair@gmail.com)**



# 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](http://theguardian.com/technology/2011/oct/13/dennis-ritchie)
- [economist.com/obituary/2011/11/05/dennis-ritchie-and-john-mccarthy](http://economist.com/obituary/2011/11/05/dennis-ritchie-and-john-mccarthy)





# Etymology

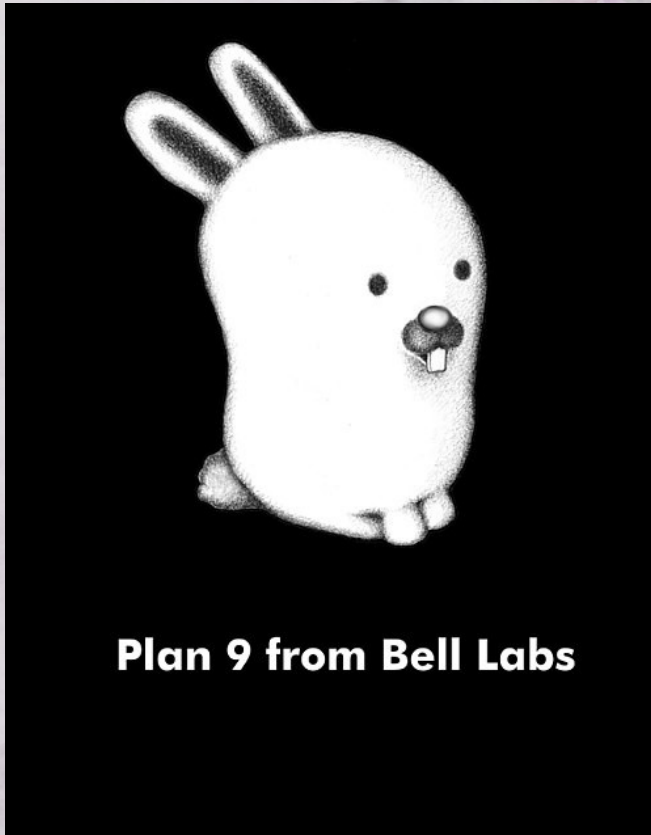


Plan 9 from Bell Labs

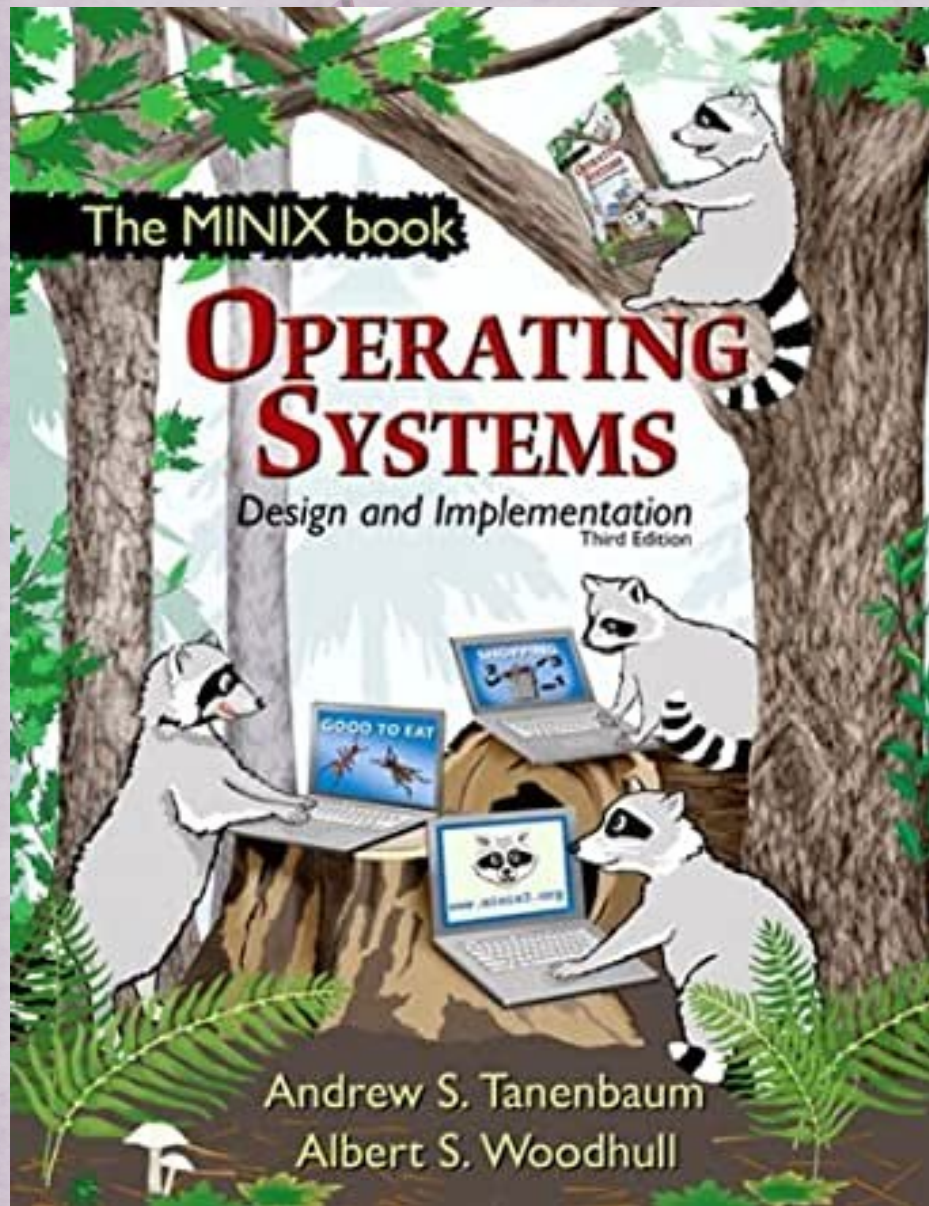
- **Plan9**
- Plan 9 from Bell Labs was originally developed, starting in the late **1980s**.
- members of the Computing Science Research Center at Bell Labs
- the same group that originally developed Unix and the C programming language. The Plan 9 team was initially led by **Rob Pike, Ken Thompson, 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...**

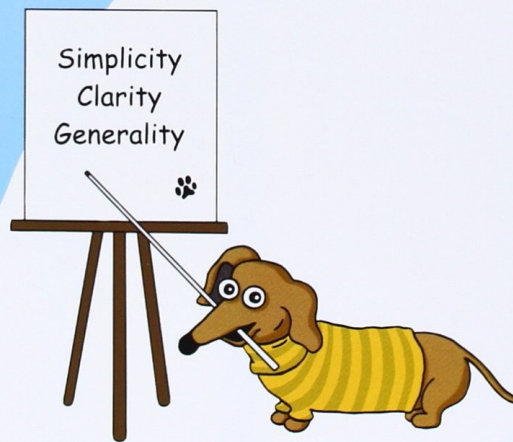




# The root of study

## The Practice of Programming

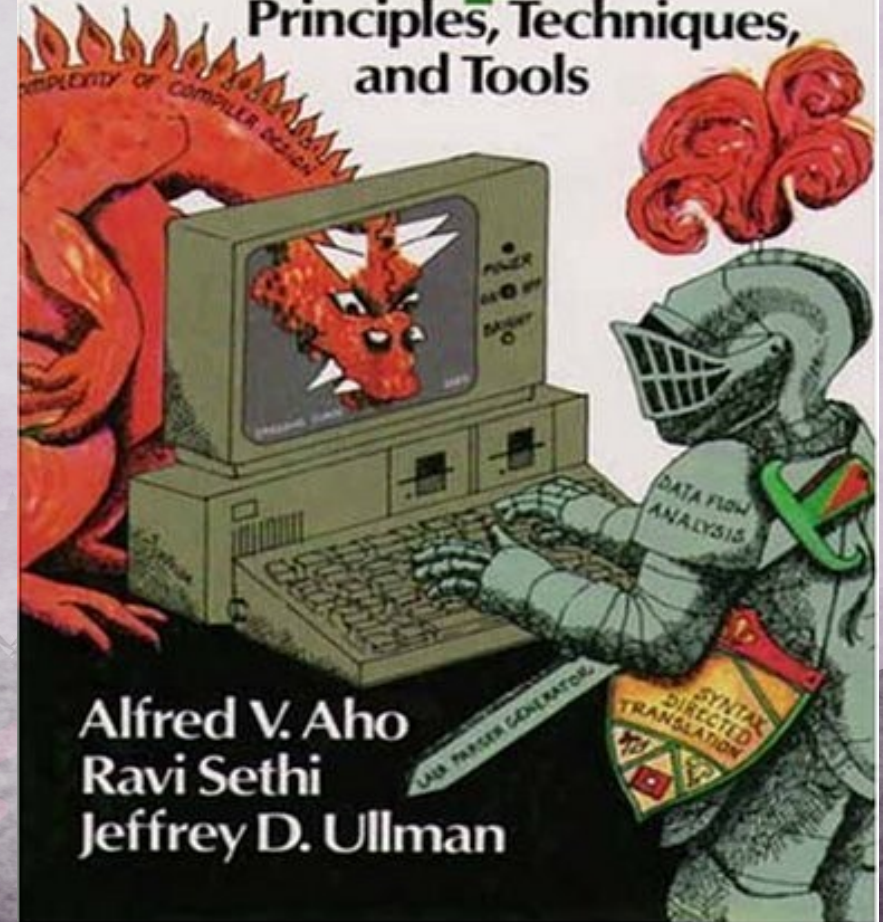
Brian W. Kernighan  
Rob Pike



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

## Compilers

Principles, Techniques,  
and Tools



Alfred V. Aho  
Ravi Sethi  
Jeffrey D. Ullman



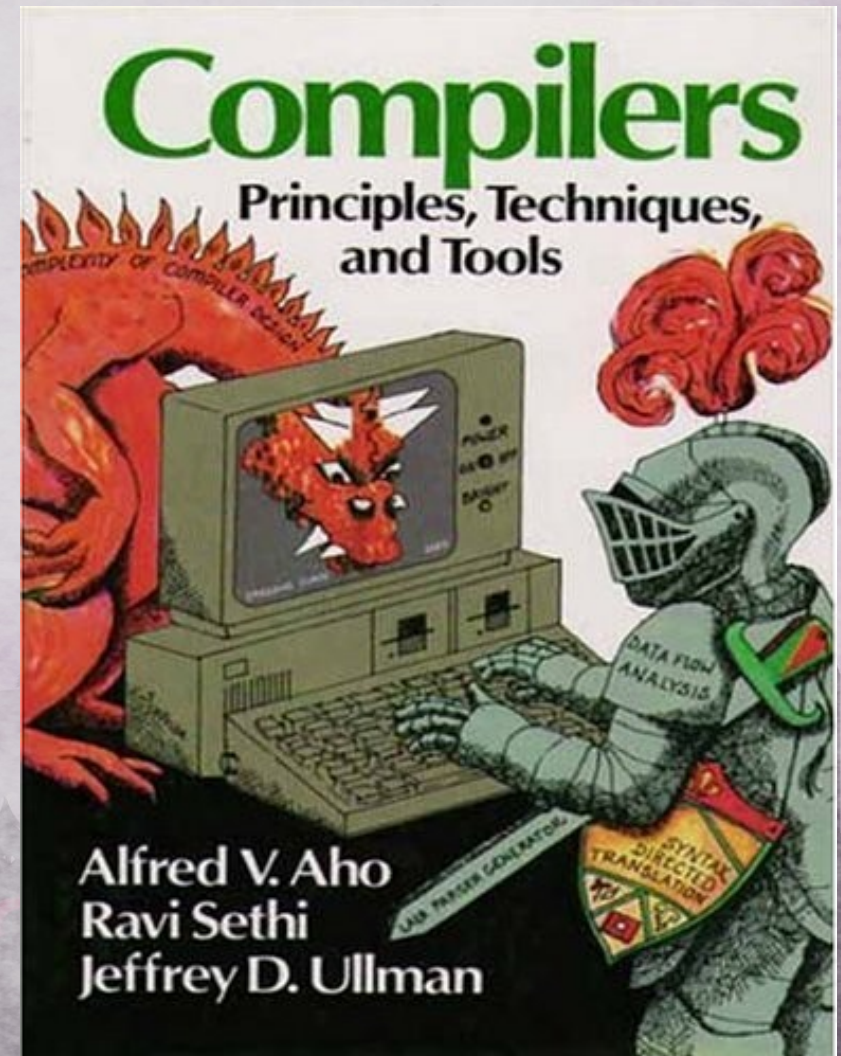
# Rob Pike's 5 Rules of Programming

- **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 ???

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body style= $0
    <div class="wrapper">...</div>
    <div class="footer">...</div>
    <div class="banner">...</div>
    <script async src="//www.google-analytics.com/
    analytics.js"></script>
    <script src="/public/js/vendor.js"></script>
    <script src="/public/js/bundle.js"></script>
    <!-- Here be dragons. Brace yourselves. -->
    <!--
```

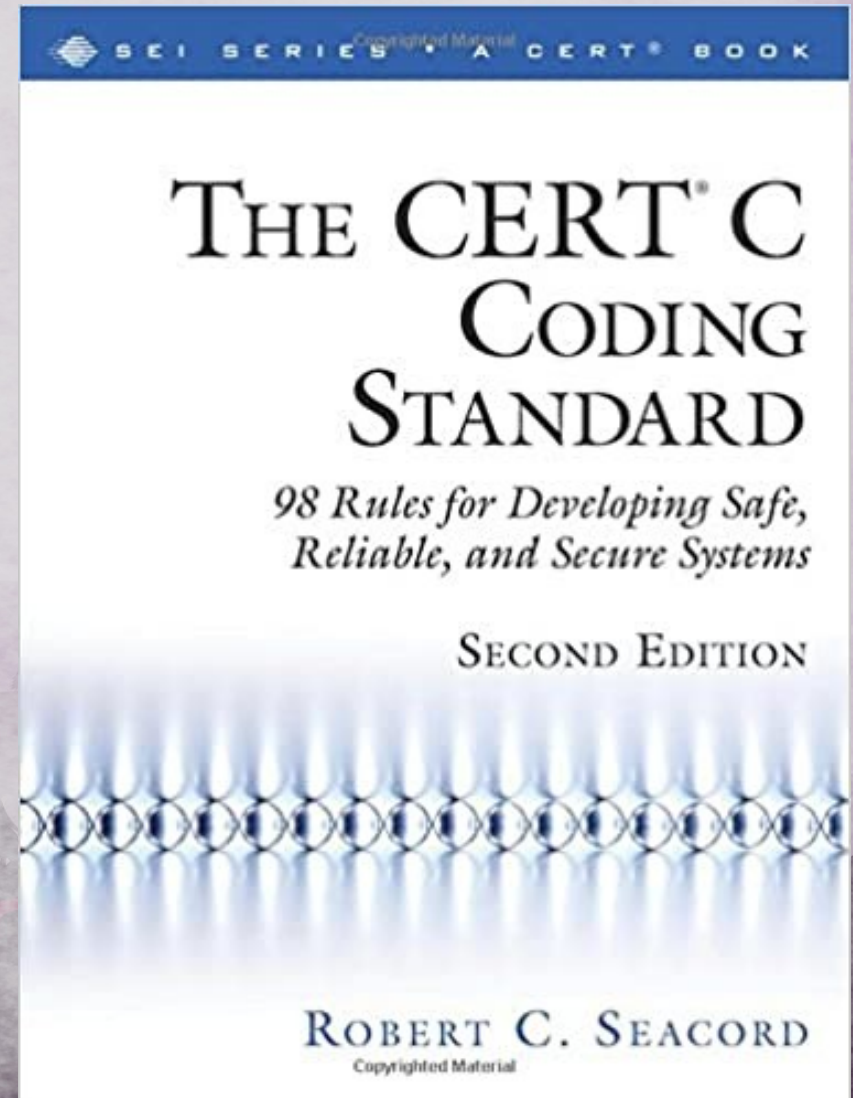


```
-->
  <!-- Google Analytics -->
  <script>...</script>
  <!-- HotJar -->
  <script>...</script>
  <!-- Facebook Pixel Code -->
  <script>...</script>
  <noscript>...</noscript>
  <!-- Intercom -->
  <script>...</script>
  <script>...</script>
```



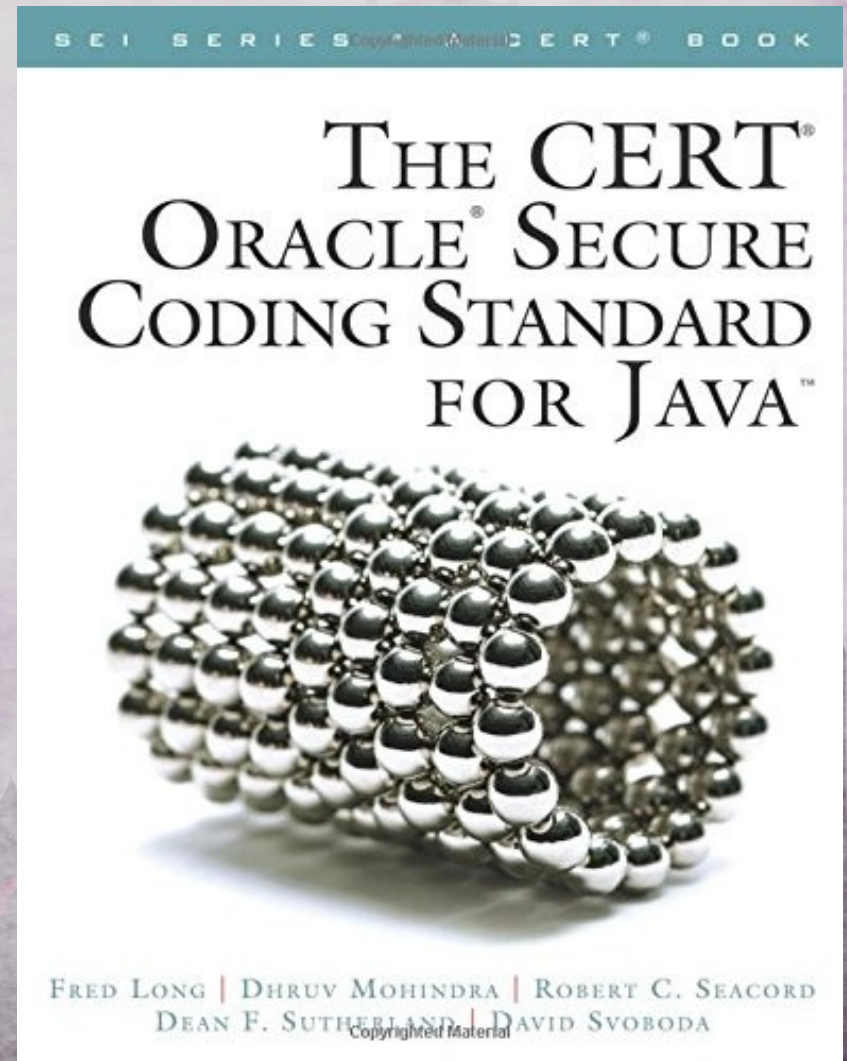
# Security focus

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



# Security focus

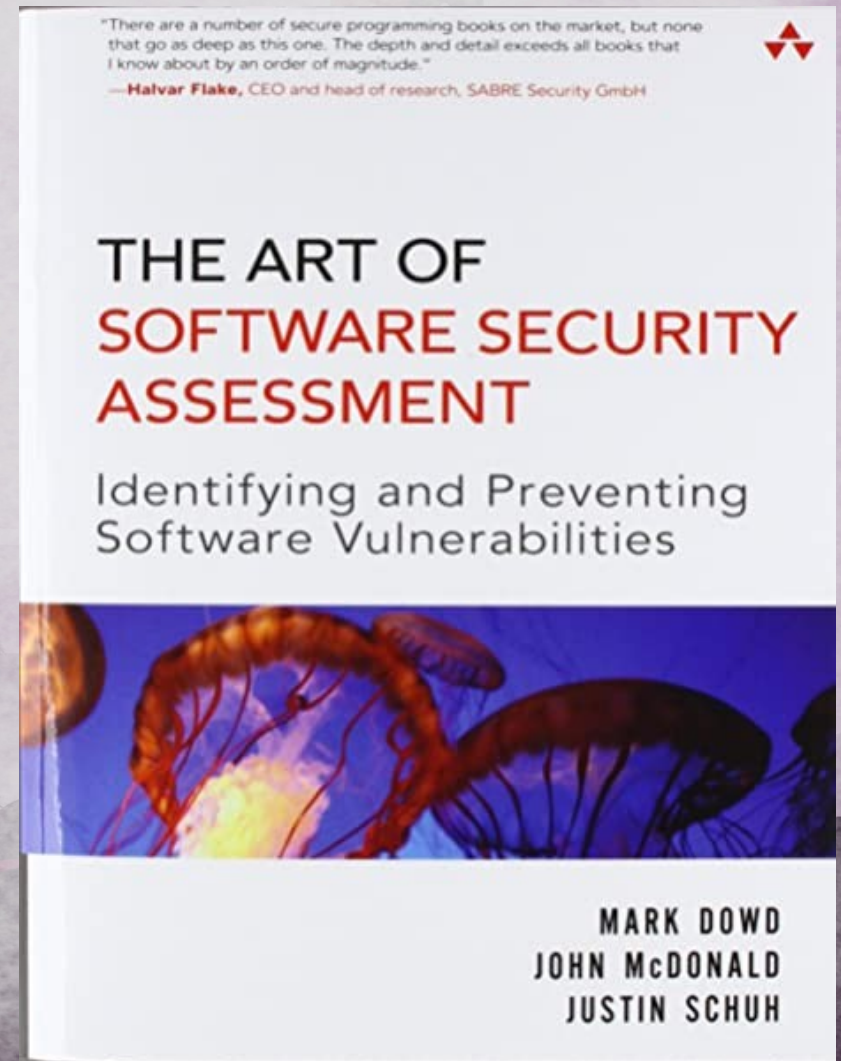
- Find Pitfalls
- Fix each point
- Mitigate
- Sometimes its hard...
- **Education ???**
- **Search dragons !**





# 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 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 parse rules...
- You can use **regex(regular expression)**, but don't have a good performance! It's 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 **deterministic 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](https://github.com/CoolerVoid/heap_detective/tree/master/doc/PoC1)
- **doc/PoC1/rule.c**
- **Rule to generate**
- **Need Re2c tool**
- **Need GCC**

```
12 int parse_ion(char** p, char** lex)
13 {
14     char* marker;
15
16     for (;;) {
17         *lex = *p;
18         /*!re2c
19         re2c:define:YYCTYPE = "char";
20         re2c:define:YYCURSOR = *p;
21         re2c:define:YYMARKER = marker;
22         re2c:yyfill:enable = 0;
23         re2c:yych:conversion = 0;
24         re2c:indent:top = 1;
25         "open".* { return OPEN; }
26         "close".* { return CLOSE; }
27         "\x0a" { return COUNTER; }
28         "\x00" { return END; }
29         [^] { continue; }
30     */
31 }
32 free(lex);
33 free(marker);
34 }
```



# Detection Ex 1

- [github.com/CoolerVoid/heap\\_detective/tree/master/doc/PoC1](https://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 yychar;
24              yychar = **p;
25              switch (yychar) {
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](https://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](https://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](https://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 !**

```
70  if(opens!=closes)
71      puts("\n Here be dragons here!\n");
72  else
73      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](https://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 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 parse rules...
- You can use **regex(regular expression)**, but don't have a good performance! It's 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 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 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...

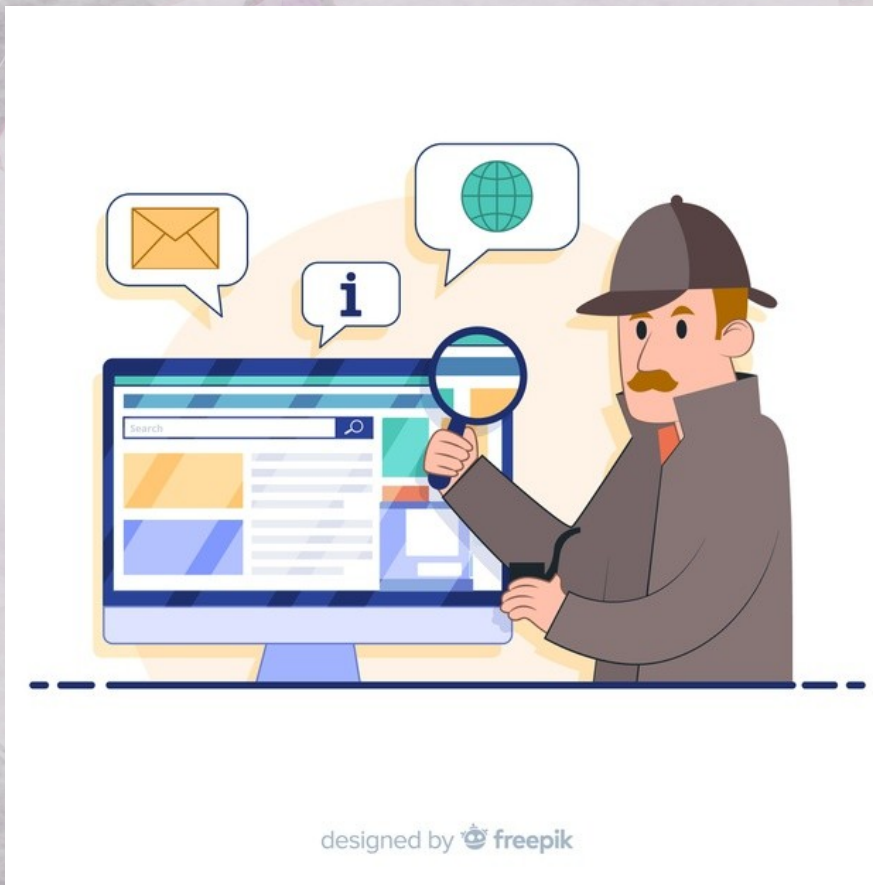


# Heap detective

- 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 ?



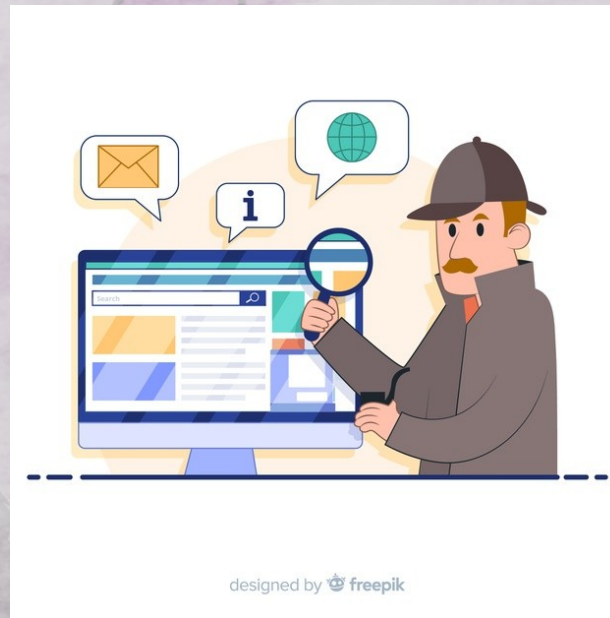
designed by freepik

designed by freepik.com



# Heap detective

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

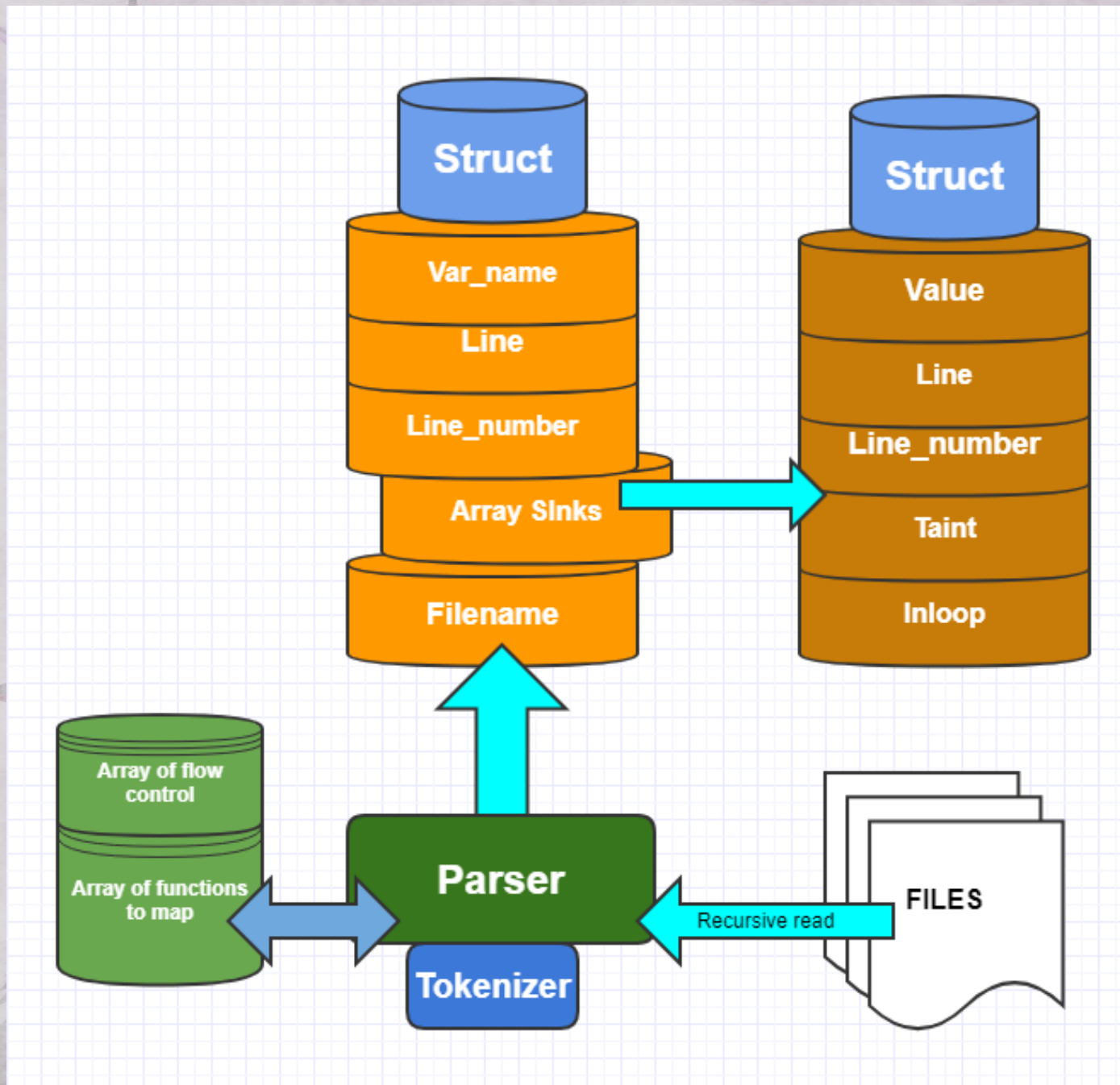
```
50 // This is array of functions names that use HEAP
51 // brk(), sbrk() ? at the future
52     vector<string> heap_in = {"kmalloc", "malloc", "xmalloc", "kalloc", "realloc", "krealloc", "xrealloc", "calloc", "kcalloc",
53     vector<string> heap_out = {"free", "xfree", "FREE", "XFREE", "delete ",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

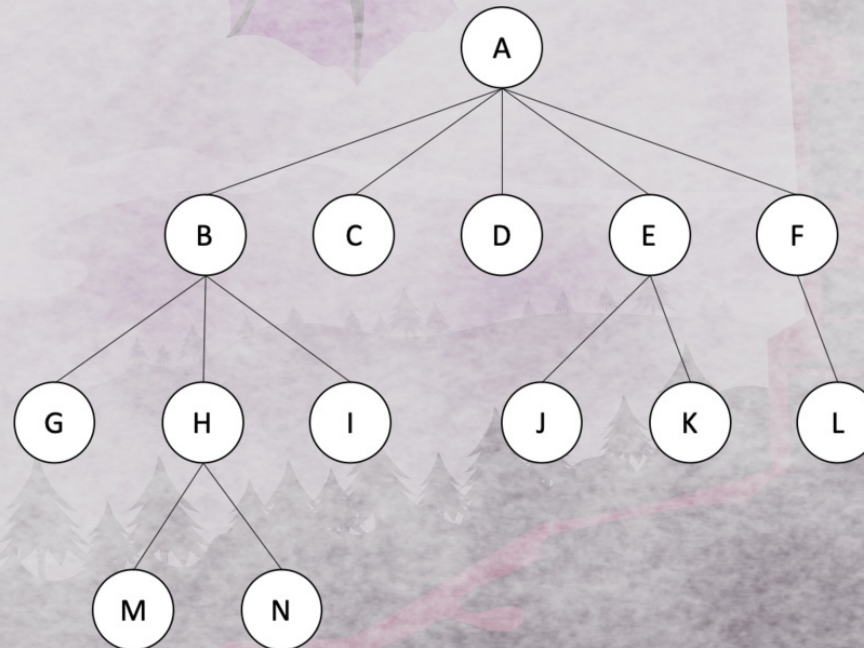
In Loop: **True**

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

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(void)
5 {
6     char *ch_ptr = NULL;
7
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     }
18
19     return 0;
20
21 }
```

# 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)

# Code Warrior

Path: /home/cooler/study/php-security-pitfalls

Sink search: \$var\_name

Module: PHP common fails



Extension: .php



Automaton: Stack overflow detector



Start

Sink

Tree

dlopen

[Edit modules](#)

Clear screen



# Codewarrior



**Title:** SHow inputs like get/post/cookies etc...

**Description:** notification

**Relevance:** Low

**Reference:** [search SQLi etc...](#)

**Match:** JREQUEST\_ALLOWHTML|JREQUEST\_ALLOWRAW|GET|POST|REQUEST|SESSION|COOKIE|FILES|SERVER

**Path:** /home/cooler/study/php-security-pitfalls/public\_html/xss/index.php

**Module:** eggs/php\_common\_fail.egg



```
1 Line: 14 - if ($_POST) {  
2 Line: 24 -     if ($validation->validate($_POST, $rules) == FALSE) {
```



**Title:** Possible XSS

**Description:** notification at possible XSS

**Relevance:** High

**Reference:** <http://guides.rubyonrails.org/security.html#cross-site-scripting-xss>

**Match:** (echo|print|print\_r|exit|die|printf|vprintf).\*?\s\*\\$\_((POST|GET)\|..\*?\|SERVER\|.?(REQUEST\_URI|QUERY\_STRING))

**Path:** /home/cooler/study/php-security-pitfalls/public\_html/xss/views/comment.php

**Module:** eggs/php\_common\_fail.egg



```
1 Line: 9 -     echo '<h1>Your name</h1>';  
2 Line: 10 -     echo escape($_POST['name']);  
3 Line: 11 -     echo '<h1>Your comment</h1>';  
4 Line: 12 -     echo escape($_POST['comment']);
```

# Codewarrior

Title: SHow inputs like get/post/cookies etc...

Description: notification

Relevance: Low

Reference: search SQLi etc...

Match: JREQUEST\_ALLOWHTML|JREQUEST\_ALLOWRAW|GET|\_POST|REQUEST|\_SESSION|\_COOKIE|\_FILES|\_SERVER

Path: /home/cooler/study/php-security-pitfalls/public\_html/xss/index.php

Module: eggs/php\_common\_fail.php

1 Line: 14 -  
2 Line: 24 -

Title: Possible XSS

Description: notific

Relevance: High

Reference: http://g

Match: (echo|print|

Path: /home/cooler

Module: eggs/php

1 Line: 9 -  
2 Line: 10 -  
3 Line: 11 - echo '<h1>Your comment</h1>';  
4 Line: 12 - echo escape(\$\_POST['comment']);

Path: /home/cooler/study/php-security-pitfalls/public\_html/xss/views/comment.php

```
01 <?php
02 /**
03  * This code is part of the Tutsplus course PHP Security Pitfalls.
04  * It is meant for demonstration purposes only.
05  * Do not use this code in a production environment!
06  */
07
08 if (!empty($_POST['comment'])) {
09     echo '<h1>Your name</h1>';
10     echo escape($_POST['name']);
11     echo '<h1>Your comment</h1>';
12     echo escape($_POST['comment']);
13 }
14
15 // plain text
16 // html text from CMS
```





# Questions ?





**Thank you!**

**Credits: CoolerVoid**

**Contact: [coolerlair@gmail.com](mailto:coolerlair@gmail.com)**

designed by  [freepik.com](https://www.freepik.com)