How to read code

A Primer for Security Practitioners

Hello!

I am Samy

Staff Product Security Engineer @ Okta

OSCP, OSWE, CISSP

What we'll cover?

1. **Soft Skills**
2. **Tools**

Why do we need to read code?

- Extend it
  - Add a feature
  - Contribute to OSS
- Fix a bug
- Find a bug
  - Vulnerability research
- Understand it better
  - Learn
  - Code review
  - Threat modeling

Reading vs Writing Code

Reading code is much harder than writing code:

- There are many solutions to a problem
  - Multiplication
- Reading and writing are tightly coupled in natural languages
- Reading == Understanding
  - Not the case in formal languages

The Situatoin

You are given a code base

Now what?

1

What problem is it trying to solve?

What does this program do?

```
int main(int argc, char **argv) {
  initialize_main(&argc, &argv);
  set_program_name(argv[0]);
  setlocale(LC_ALL, "");
  bindtextdomain(PACKAGE, LOCALEDIR);
  textdomain(PACKAGE);

  atexit(close_stdout);

  parse_gnu_standard_options_only(argc, argv,
PROGRAM_NAME, PACKAGE_NAME, Version, true, usage,
AUTHORS, (char const *)NULL);

  char **operands = argv + optind;
  char **operand_lim = argv + argc;
  if (optind == argc) *operand_lim++ =
bad_cast("y");

  /* Buffer data locally once, rather than having
the
     large overhead of stdio buffering each
item.  */
  size_t bufalloc = 0;
  bool reuse_operand_strings = true;
  char **operandp = operands;
  do {
    size_t operand_len = strlen(*operandp);
    bufalloc += operand_len + 1;
    if (operandp + 1 < operand_lim && *operandp +
operand_len + 1 != operandp[1])
reuse_operand_strings = false;
  } while (++operandp < operand_lim);
```

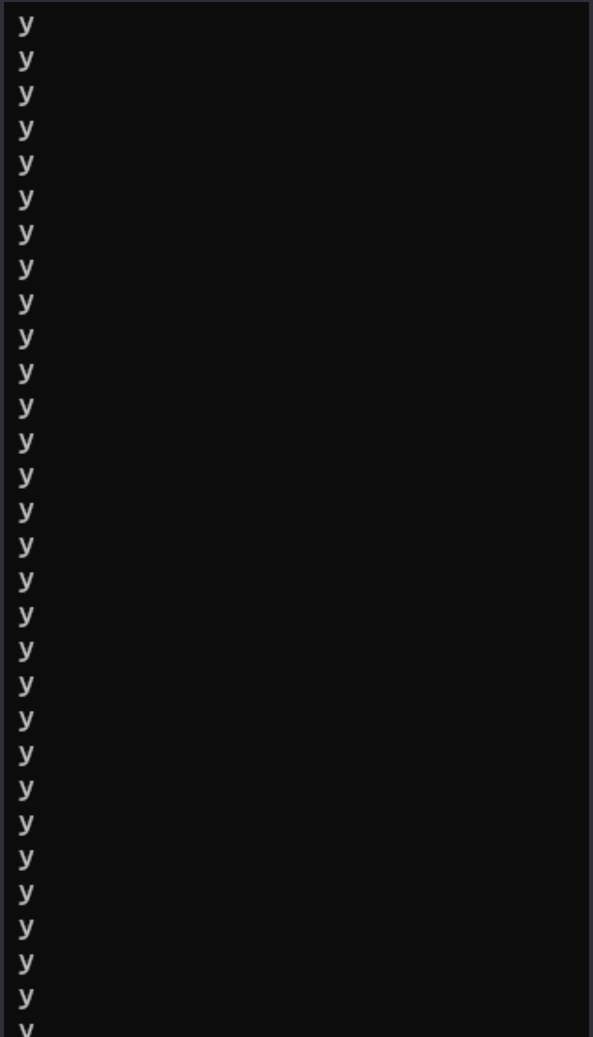
```
/* Improve performance by using a buffer size greater than BUFSIZ
/ 2.  */
  if (bufalloc <= BUFSIZ / 2) {
    bufalloc = BUFSIZ;
    reuse_operand_strings = false;
  }

  /* Fill the buffer with one copy of the output.  If possible,
reuse
     the operands strings; this wins when the buffer would be
large.  */
  char *buf = reuse_operand_strings ? *operands :
xmalloc(bufalloc);
  size_t bufused = 0;
  operandp = operands;
  do {
    size_t operand_len = strlen(*operandp);
    if (!reuse_operand_strings) memcpy(buf + bufused, *operandp,
operand_len);
    bufused += operand_len;
    buf[bufused++] = ' ';
  } while (++operandp < operand_lim);
  buf[bufused - 1] = '\n';

  /* If a larger buffer was allocated, fill it by repeating the
buffer
     contents.  */
  size_t copysize = bufused;
  for (size_t copies = bufalloc / copysize; --copies;) {
    memcpy(buf + bufused, buf, copysize);
    bufused += copysize;
  }

  /* Repeatedly output the buffer until there is a write error;
then fail.  */
  while (full_write(STDOUT_FILENO, buf, bufused) == bufused)
continue;
  error(0, errno, _("standard output"));
  main_exit(EXIT_FAILURE);
}
```

context_1.cpp



```
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
y
```

How to gain more context?

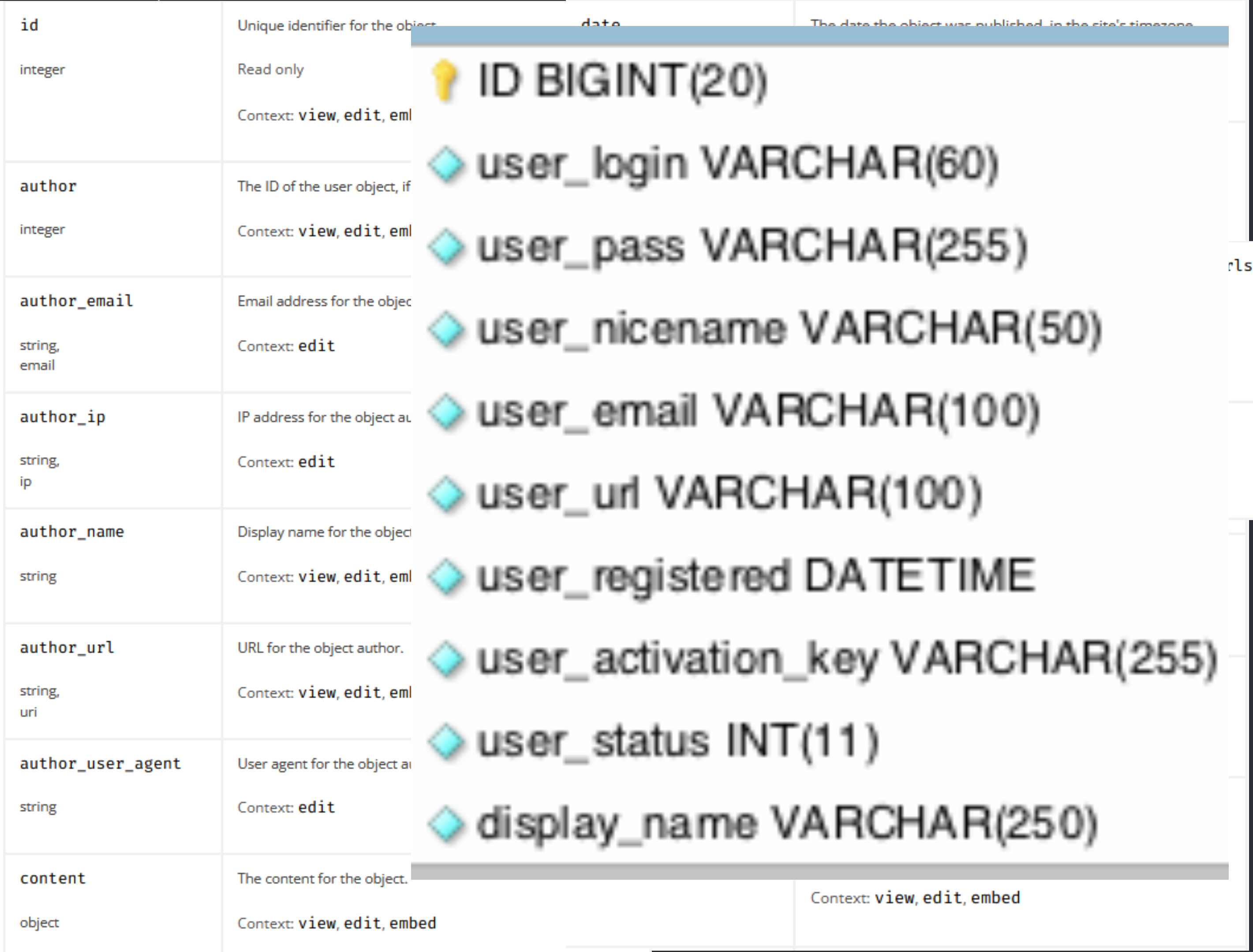
Become a user

Use the software you are trying to analyze

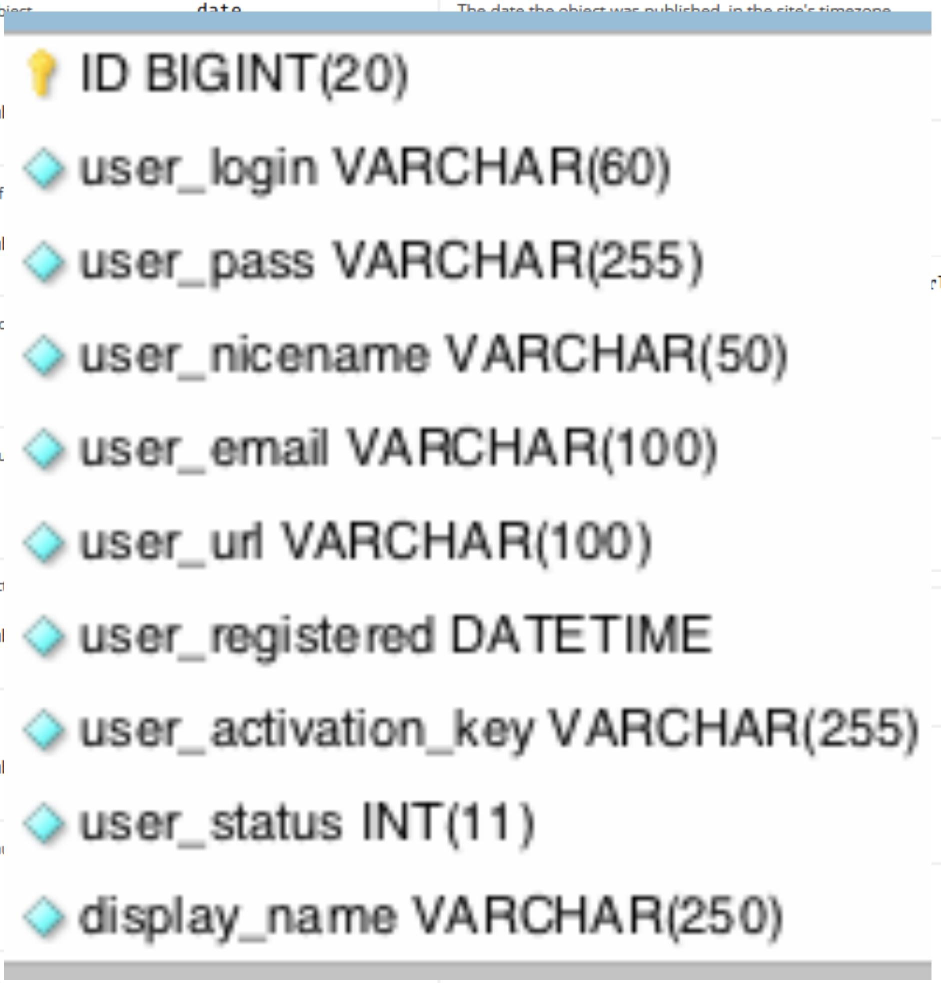
- Go through a simple use case

Set up a working development environment

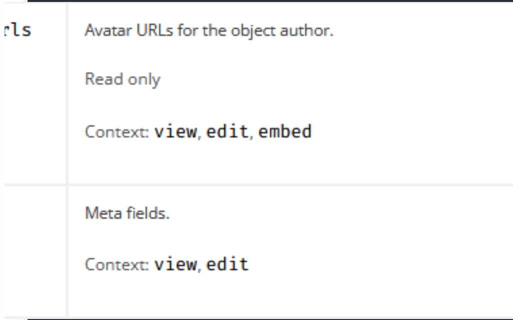
- Most projects come with a set up guide
- Use a proper setup
  - Search
  - Go to Implementation
  - Find usage
  - Debugger
  - Bookmarks (Optional, but highly recommended)



`id` integer	Unique identifier for the ob Read only Context: `view`, `edit`, `em`	`date`	The date the object was published, in the site's timezone.
`author` integer	The ID of the user object, if Context: `view`, `edit`, `em`		
`author_email` string, email	Email address for the objec Context: `edit`		
`author_ip` string, ip	IP address for the object au Context: `edit`		
`author_name` string	Display name for the objec Context: `view`, `edit`, `em`		
`author_url` string, uri	URL for the object author. Context: `view`, `edit`, `em`		
`author_user_agent` string	User agent for the object a Context: `edit`		
`content` object	The content for the object. Context: `view`, `edit`, `embed`		Context: `view`, `edit`, `embed`



```
ID BIGINT(20)
user_login VARCHAR(60)
user_pass VARCHAR(255)
user_nicename VARCHAR(50)
user_email VARCHAR(100)
user_url VARCHAR(100)
user_registered DATETIME
user_activation_key VARCHAR(255)
user_status INT(11)
display_name VARCHAR(250)
```



`rls`	Avatar URLs for the object author. Read only Context: `view`, `edit`, `embed`
	Meta fields. Context: `view`, `edit`

Find the inputs

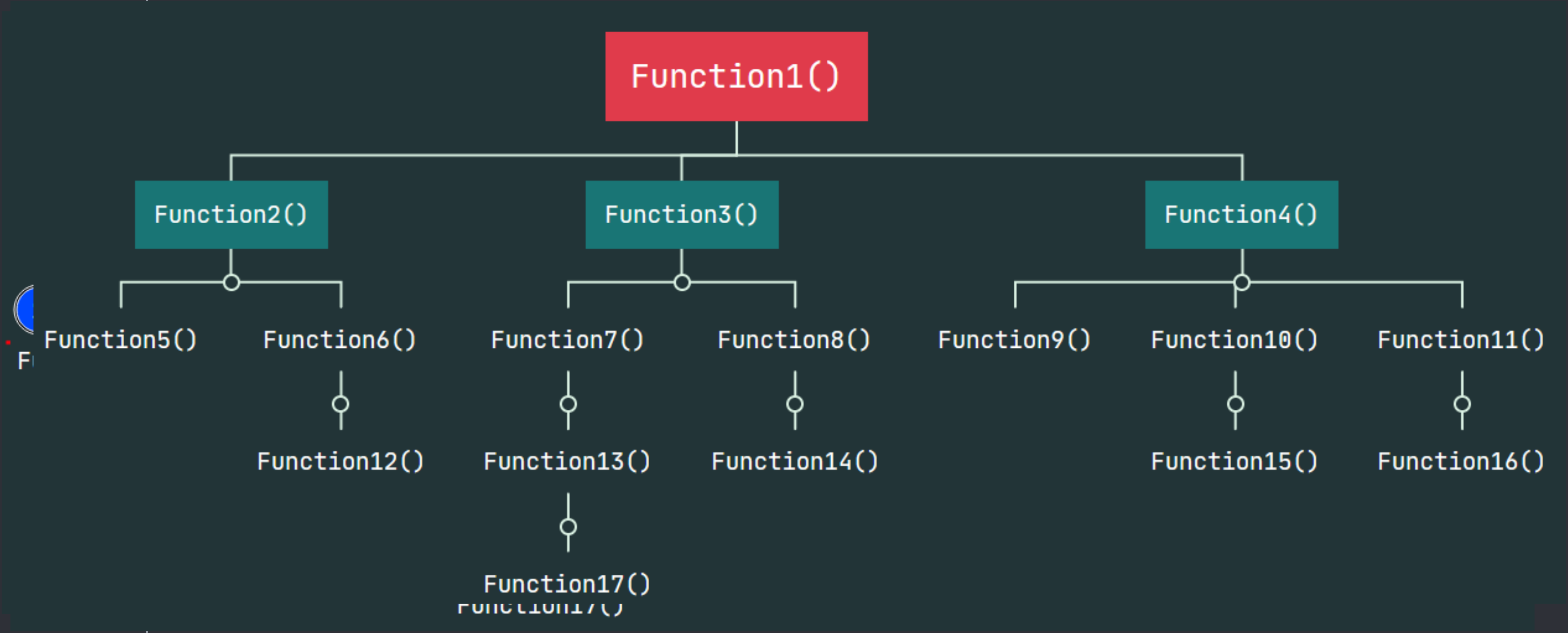
Where do the inputs come from?

- User input
- API Endpoints
- Files
- Env vars

● Choose a path

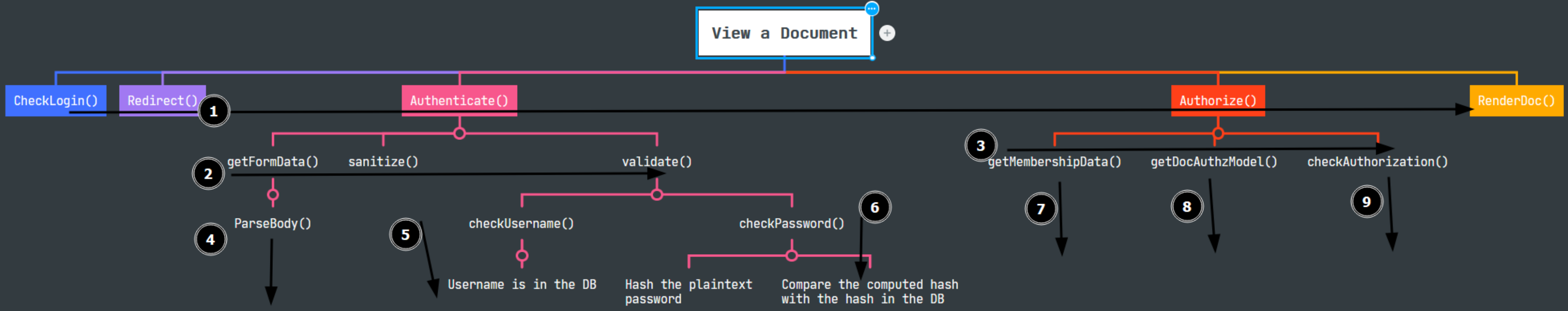
○ Trace!

DFS vs BFS



Function1()
Function2()
Function3()
Function4()
Function5()
Function6()
Function7()
Function8()
Function9()
Function10()
Function11()
Function12()
Function13()
Function14()
Function15()
Function16()
Function17()

DFS vs BFS


View a Document
CheckLogin()
Redirect()
Authenticate()
Authorize()
RenderDoc()
1
getFormData()
sanitize()
validate()
2
ParseBody()
4
5
checkUsername()
checkPassword()
6
Username is in the DB
Hash the plaintext password
Compare the computed hash with the hash in the DB
3
getMembershipData()
getDocAuthzModel()
checkAuthorization()
7
8
9

Set up a playground

- Set up a playground in your IDE
- Use online REPLs
- Demo 1
- Demo 2
- Demo 3

Debugger

- See the execution context
- Helps you slow down the time

  Print statement is NOT a debugger!

Other sources of insight

- Git (or any other form of version control)
- Unit Tests
- Comments

Demo time!

And the last technique

Ask for help!

?

Question time!