Leveraging Continuous Feedback in Education to Increase C Code Quality

Max Schrötter, Bettina Schnor

University of Potsdam
Institute for Computer Science

March 15, 2024

Current Situation

- "How Secure are our Computer Systems Courses?" [1]
  - analyzed 760.000 lines of code of students and instructors from 20 top universities
  - students are not taught the security implications of using unsafe functions
  - by frequently using unsafe functions, instructors and textbooks are passively teaching students to use them
  - students can graduate with a CS major without taking any computer security course

"The real problem lies in students' lack of more fundamental knowledge and skills, such as paying attention to compiler and OS messages and carefully reading documentation." [2]

[1] Almansoori et. al., How Secure are our Computer Systems Courses in ICER '20, 2020

[2] Almansoori et. al., Towards finding the missing pieces to teach secure programming skills to students in SIGCSE 2023

Current Situation

- "How Secure are our Computer Systems Courses?" [1]
  - analyzed 760.000 lines of code of students and instructors from 20 top universities
  - students are not taught the security implications of using unsafe functions
  - by frequently using unsafe functions, instructors and textbooks are passively teaching students to use them
  - students can graduate with a CS major without taking any computer security course

- experiences at University of Potsdam
  - security problems not limited to unsafe functions
  - race conditions, memory safety vulnerabilities, non robust programs ...

[1] Almansoori et. al., How Secure are our Computer Systems Courses in ICER '20, 2020

Current Situation

- "How Secure are our Computer Systems Courses?" [1]
  - analyzed 760.000 lines of code of students and instructors from 20 top universities
  - students are not taught the security implications of using unsafe functions
  - by frequently using unsafe functions, instructors and textbooks are passively teaching students to use them
  - students can graduate with a CS major without taking any computer security course
- experiences at University of Potsdam
  - security problems not limited to unsafe functions
  - race conditions, memory safety vulnerabilities, non robust programs ...

→Secure and robust programming should be taught alongside the normal curriculum

[1] Almansoori et. al., How Secure are our Computer Systems Courses in ICER '20, 2020

CoFee: Continuous Feedback

- Many frameworks have been created[2]:
  - focus on functional tests
  - tightly coupled with analyzers
  - most frameworks are no longer maintained

[2] https://systemscorpus.strickroth.net

CoFee: Continuous Feedback

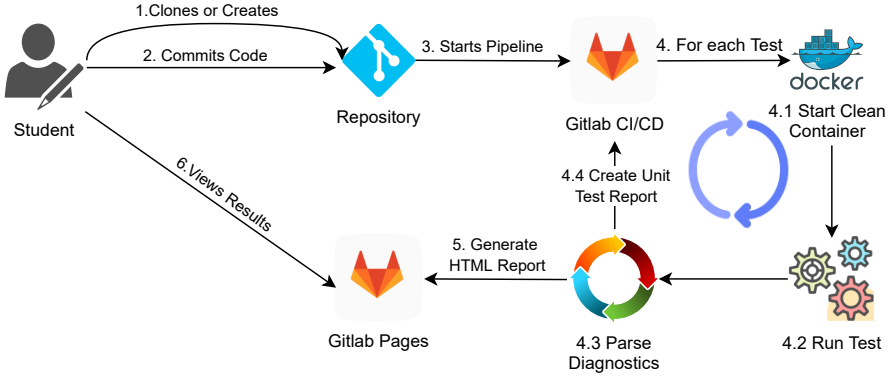
- Many frameworks have been created[2]:
  - focus on functional tests
  - tightly coupled with analyzers
  - most frameworks are no longer maintained

- **CoFee**
  - focuses on code security and robustness
  - offers fast and **continuous** feedback
  - adds meaningful hints for **undergraduate** students
  - **hides complex configurations** of static analysis tools
  - modular design and loosely **integrates various state of the art tools**
  - **situated learning:** integration of common software development workflows

[2] https://systemscorpus.strickroth.net

CoFee

Continuous Feedback



Evaluation of Code Analyzers for Education

Our test suite is based on Secure Coding Standards (SEI-CERT) and past experiences

1. **Memory Violations**: Use-After-Free, Memory Leak, Allocation Size, Pointer Manipulation, Null Pointer Dereference, Uninitialized Memory, etc.
2. **Unchecked Return Values**: CWE252, library function return values should be checked for errors (code robustness)
3. **Thread Safety**: Race Conditions, Mutex Deadlock, Improper Locking
4. **String & IO**: String Terminator, User Format Strings, File Handling, Accessesing Closed File Descriptor
5. **Misc**: One Definition Rule, Int Overflow, Signal Safety, Non-Reentrant, Inclusion of C-Files, Errno Handling

for full list see `https://doi.org/10.1109/CSCI58124.2022.00351`

Code Analyzer Results

- only considered tools withouth manual instrumentation
- focused on established open source tools
- including the three overall winners of the SV-Comp 2022[3]: Symbiotic, CPAChecker, UAutomizer

Category:	**SAN**[4]	**CSA**	**Tidy**[5]	**GCC**	Valgrind	ESBMC	Symbiotic
Memory	52/98	66/98	62/98	16/98	60/98	52/98	52/98
Error Handling	0/9	0/9	0/9	0/9	0/9	0/9	0/9
Threads	9/10	1/10	0/10	0/10	0/10	0/10	0/10
IO	2/23	8/23	12/23	12/23	8/23	8/23	0/23
Misc	0/24	0/24	1/24	0/24	0/24	0/24	0/24
Ringbuf	1/1	0/1	0/1	0/1	0/1	1/1	0/1

[3] Dirk Beyer, Progress on Software Verification: SV-COMP 2022 in TACAS 2022
[4] Google Sanatizers: ASAN, MSAN, TSAN, UBSAN
[5] also enables some CSA checks

Detected Mistakes for the OS-Course 2022/23

Error Category	Number
segmentation fault	180
use of uninitialized variable	149
memory leak	126
null dereference	85
missing error check for allocation	**82**
buffer overflow	54
TCP segmentation violation	52
resource leak: unclosed streams	47

Error Category	Number
use after free	41
use of unsafe functions	36
incompatible pointer conversion	21
double free	19
data races	15
bugprone include	12
async signal violation	1

- in 2022 a simple static analyzer that detects if error handling is missing for malloc & calloc was added

Why is Error Handling Relevant?

- CVE-2019-15504, Linux Kernel, CVSS: 9.8

  Hui Peng and Mathias Payer discovered that the 91x Wi-Fi driver in the Linux kernel ***did not properly handle error conditions on initialization****, leading to a double-free vulnerability. A physically proximate attacker could use this to cause a denial of service (system crash).*

- CVE-2023-23004, Linux Kernel, CVS: 5.5

- CVE-2023-0401, OpenSSL, CVS: 7.5

  A NULL pointer can be dereferenced when signatures are being verified on PKCS7 signed or signedAndEnveloped data. In case the hash algorithm used for the signature is known to the OpenSSL library but the implementation of the hash algorithm is not available the digest initialization will fail. ***There is a missing check for the return value from the initialization function*** *which later leads to invalid usage of the digest API most likely leading to a crash.*

CoFee-Error-Handling-Analyzer (CoFee-EHA)

- error handling bugs need to be detected **before** results are used.
- function specification inspired by Glibc and POSIX function signatures
  - returns valid pointer or NULL or -1 on failure
  - returns a valid integer or -1 on failure
  - returns 0 on success or -1 on failure and results via parameter
  - returns 0 on success or -1 on failure and has no results (side effects: setuid)

```
malloc|1|-1|NULL;open|2|-1|-1;read|3|1|-1
```

- CoFee-EHA tracks result and error symbol
- If result is accessed while error symbol can be the error value
  →error handling is missing or faulty

CoFee-Error-Handling-Analyzer (CoFee-EHA)

- error handling bugs need to be detected before results function call are used.
- function specification inspired by Glibc and POSIX function signatures
- CoFee-EHA tracks result and error symbol
- If result is accessed while error symbol can be the error value →error handling is missing or faulty
- Relies on Clang Static Analyzers Symbolic Engine
  - `checkPostStmt` to register the symbols in the program state
  - `checkLocation` to check if the tracked symbols are loaded or assigned
  - `checkPreStmt (CheckPreCall)` to check if the tracked symbols is passed to a function
  - `checkDeadSymbols` to get notified if a symbol moves out of scope

Example

```
1  int main (){
2    int fd;
3    char* buf=malloc(64*sizeof(char));
4    if (buf == (void*)-1 ) { exit(EXIT_FAILURE); }
5    fd = open("/tmp/file", 0);
6    int res = read(fd, buf, 63);
7    buf[res] = '\0';
8    printf("%s\n", buf);
9  }
```

DEMO

Diagnostics Support

CoFee parses the following:

- text output of: make, ld, clang-tidy and LLVM sanitizers
- plist files from: Scan-Build, CodeChecker
- JUnit XML files
- Valgrind XML
- GCC json diagnostics

CoFee EHA

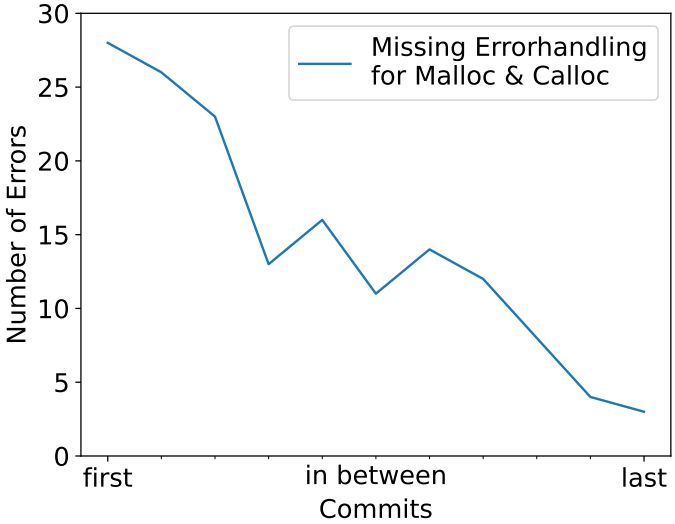
- Analyzed on Student Submissions in Operating Systems from University of Potsdam in 2022
- 101 Submissions
- manual analysis found 178 errors
- CoFee EHA detected 148 correctly →recall **83.15%**
- CoFee EHA reported 9 false positives →precision **94.27%**

CoFee EHA

- Analyzed on Student Submissions in Operating Systems from University of Potsdam in 2022
- 101 Submissions
- manual analysis found 178 errors
- CoFee EHA detected 148 correctly →recall **83.15%**
- CoFee EHA reported 9 false positives →precision **94.27%**
    - Problem: error handling with recovery & merged control flow

```
rc = setsockopt(s,SOL_SOCKET,SO_RCVBUF,(char *) &option_value,sizeof(int));
if (rc < 0) {
  perror("setsockopt(SO_RCVBUF) failed");
  fprintf(stderr, "Continuing with smaller buffer.\n");
}
```

Results


Missing Errorhandling
for Malloc & Calloc
Number of Errors
30
25
20
15
10
5
0
first
in between
Commits
last

Evaluation

Exercise	Average points in % without CoFee	Average points in % with CoFee	Difference
Ring Buffer	63	81	+18
Process Table	64	70	+6
Process Creation	77	74	-3
POSIX Threads	68	72	+4
IPC	76	88	+12
Signal Handler	49	84	+35
Reader/Writer Locks	53	70	+17

Conclusion & Future Work

Conclusion:

- CoFee is a modular framework intended to increase code quality
- tool configuration is hidden from students
- situated learning: introduces common software engineering workflows early
- evaluation shows a significant uplift in points
- lightweight: no database or other infrastructure than gitlab needed

Future Work:

- support Sarif-v2 (incl. editor integration)
- improving hints: include Error Values (pedagogical Question)
- reduce false positives

Demo: `https://gitlab.com/cofee-demo/c-demo`
CoFee: `https://gitlab.com/schrc3b6/cofee_up`

Thank You for Your Attention

Thanks to all contributors:

- Matthias Habich
- Maximilian Falk
- Kai Schlabitz
- all Tutors and participating Students

References I

[1] M. Schrötter, M. Falk **and** B. Schnor, “Automated detection of bugs in error handling for teaching secure c programming,” **in***Proceedings of the Sixth Workshop ’Automatische Bewertung von Programmieraufgaben’ (ABP 2023)* Gesellschaft für Informatik e.V., 2023. DOI: 10.18420/abp2023-1.

[2] M. Schrötter **and** B. Schnor, “Leveraging continuous feedback in education to increase c code quality,” **in***2022 International Conference on Computational Science and Computational Intelligence (CSCI)* 2022, **pages** 1950–1956. DOI: 10.1109/CSCI58124.2022.00351.

References II

[3] M. Almansoori, J. Lam, E. Fang, A. G. Soosai Raj **and** R. Chatterjee, "Towards finding the missing pieces to teach secure programming skills to students," **in***Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* **jourser** SIGCSE 2023, New York, NY, USA: Association for Computing Machinery, 2023, **pages** 973–979, ISBN: 9781450394314. DOI: 10.1145/3545945.3569730. **url**: https://doi.org/10.1145/3545945.3569730.

[4] S. Strickroth **and** M. Striewe, "Building a corpus of task-based grading and feedback systems for learning and teaching programming," *International Journal of Engineering Pedagogy (iJEP)*, **jourvol** 12, **number** 5, pp. 26–41, **november** 2022. DOI: 10.3991/ijep.v12i5.31283.

References III

[5] M. Almansoori, J. Lam, E. Fang, K. Mulligan, A. G. Soosai Raj **and** R. Chatterjee, “How secure are our computer systems courses?” **in***Proceedings of the 2020 ACM Conference on International Computing Education Research* **jourser** ICER ’20, Virtual Event, New Zealand: Association for Computing Machinery, 2020, **pages** 271–281, ISBN: 9781450370929. DOI: 10.1145/3372782.3406266.