XXIV Brazilian Symposium on Information
and Computational Systems Security
São José dos Campos/SP - 2024

Trust, but Verify: Evaluating Developer Behavior in Mitigating Security Vulnerabilities in Open-Source Software Projects

Janislley Oliveira de Sousa, Bruno Carvalho de Farias, Eddie Batista de Lima Filho, Lucas Carvalho Cordeiro

Email: janislley.sousa@sidia.com

ESBMC UFAM MANCHESTER 1824 The University of Manchester sidia //impact innovation SAMSUNG

XXIV Brazilian Symposium on Information and Computational Systems Security
São José dos Campos/SP - 2024

Research Team:

MSc. Janislley Oliveira
(UFAM/SIDIA)

Ph.D. Bruno Farias
(Manchester)

Dr. Eddie Batista
(UFAM/TPV)

Dr. Lucas Cordeiro
(UFAM/Manchester)

ESBMC

UFAM

MANCHESTER 1824
The University of Manchester

sidia
//impact innovation

SAMSUNG

Outline

1. **Motivation and Research Problem**: Investigating the effectiveness of vulnerability detection in critical open-source software projects to address security risks.

2. **Background and Methodology**: Utilizing bounded model checking and the LSVerifier tool to systematically identify and assess vulnerabilities in real-world OSS projects.

3. **Empirical Study Results and Key Findings**: Highlighting common vulnerabilities and providing actionable strategies to improve security practices and mitigate risks in OSS development.

1 - Introduction

- **More than 50%** of a software project's costs today are allocated not to the creative process of software development, but to the corrective tasks of debugging and fixing errors [1].

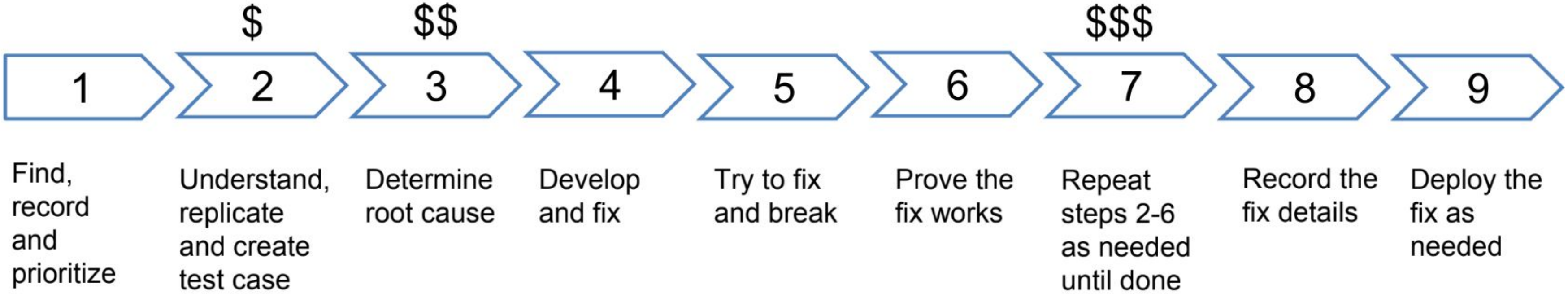
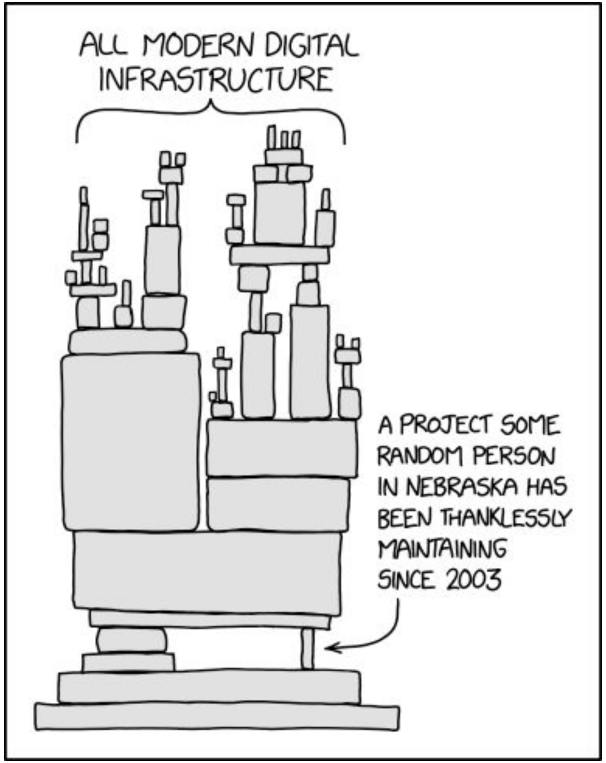


Figure 1: Poor software quality costs.

[1] KRASNER, Herb.The cost of poor software quality in the US: A 2024 Report.

1.1 - Motivation

- Modern software development often employs **extensive third-party code** from **external libraries** to save time, which usually comes from **open-source software projects**.
- While developers usually review their code for bugs and security issues using specialized tools, they often **skip checking** or **not check third-party libraries** due to the **extra effort involved** in their evaluation or bad practices during development process.
- Since a software project may **depend on several open-source libraries**, analysis of a software project's entire **dependency tree can become very complex**.



Source: https://xkcd.com/2347/

Research problem

Challenges and motivations.

- The **C programming language** lacks protection mechanisms such as **bound checking and memory safety**.
- Developers are responsible for **memory and resource management**.
- **Software developers** frequently **use open-source libraries to speed up development cycles**.
- These libraries can contain **security vulnerabilities**, leading to high-profile incidents (**Java's Log4Shell**, **Windows CrowdStrike**).
- **Developers' behaviors** and **practices** significantly influence the mitigation of security vulnerabilities in third-party libraries within OSS projects.

Research Questions

What will be done?

This study aims to answer the following research questions:

- **RQ1**: What are the Common Types and Prevalence of Dependency Vulnerabilities in Open-Source Software Projects?
- **RQ2**: How do developers’ behaviors and practices influence the mitigation of security vulnerabilities?
- **RQ3**: What is the most effective strategy for mitigating risks from dependency vulnerabilities in open-source software projects?

Background

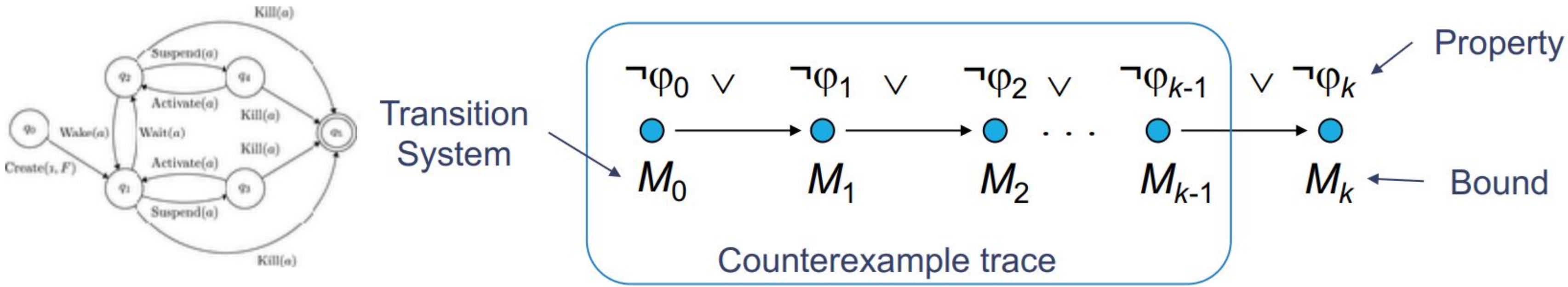
Key concepts and technologies

- Bounded Model Checking (BMC)
- LSVerifier Tool
- ESBMC module

2.1 - Bounded Model Checking (BMC)

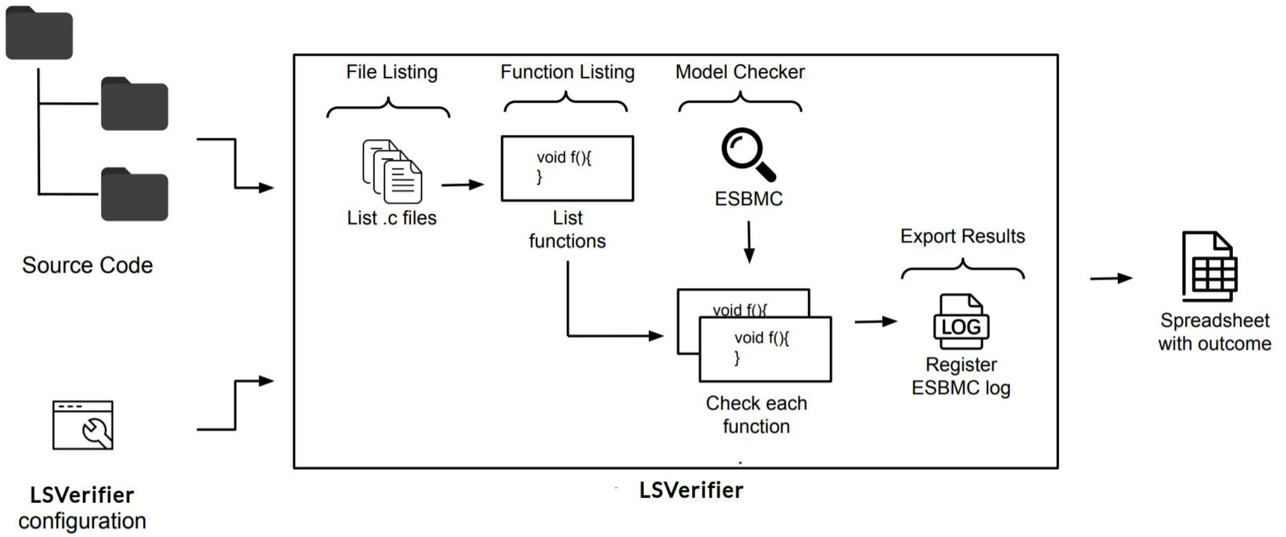
- **Basic Idea:** given a transition system **M**, check negation of a given property **φ** up to given depth **k**.

$$BMC_{\Phi}(k) = I(s_1) \wedge \left(\bigwedge_{i=1}^{k-1} T(s_i, s_{i+1}) \right) \wedge \left(\bigvee_{i=1}^{k} \neg\phi(s_i) \right)$$



- Bounded model checkers "slice" the state space in depth.
- It is aimed to find bugs and can only prove correctness if all states are reachable within the bound.
- Exhaustively explores all executions.
- Can be bounded to limit number of iterations and context-switch.
- Report errors as traces.

2.2 - Large Systems Verifier (LSVerifier) Architecture



SBSeg'23 Paper:

- The Tool takes a **source-code directory**, a **software project**, and **dependencies configuration** as inputs.
- The verification outcomes are compiled into a **report (logs, CSV files)**.
- Tool repository: https://github.com/janislley/LSVerifier - **Apache 2.0 Licence**.

2.2 - ESBMC module

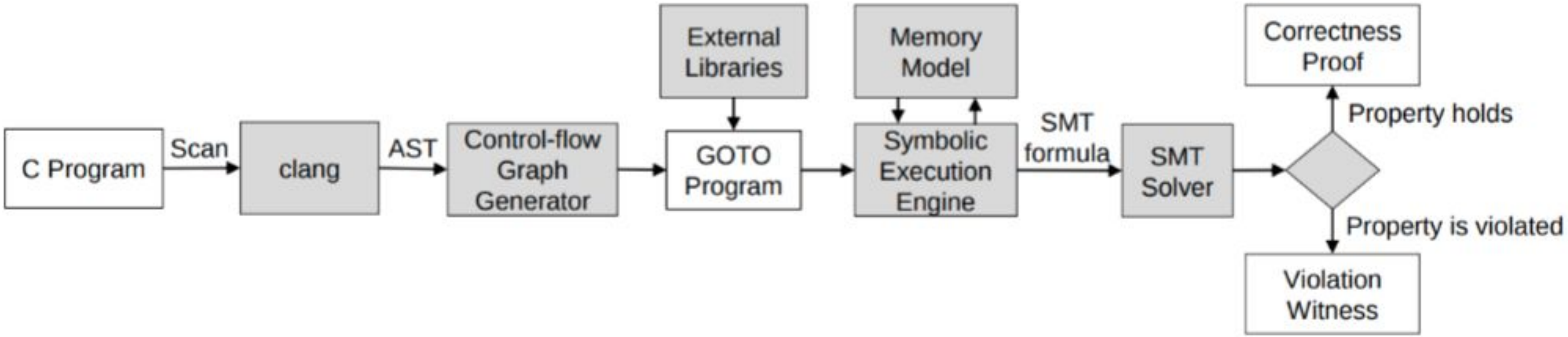
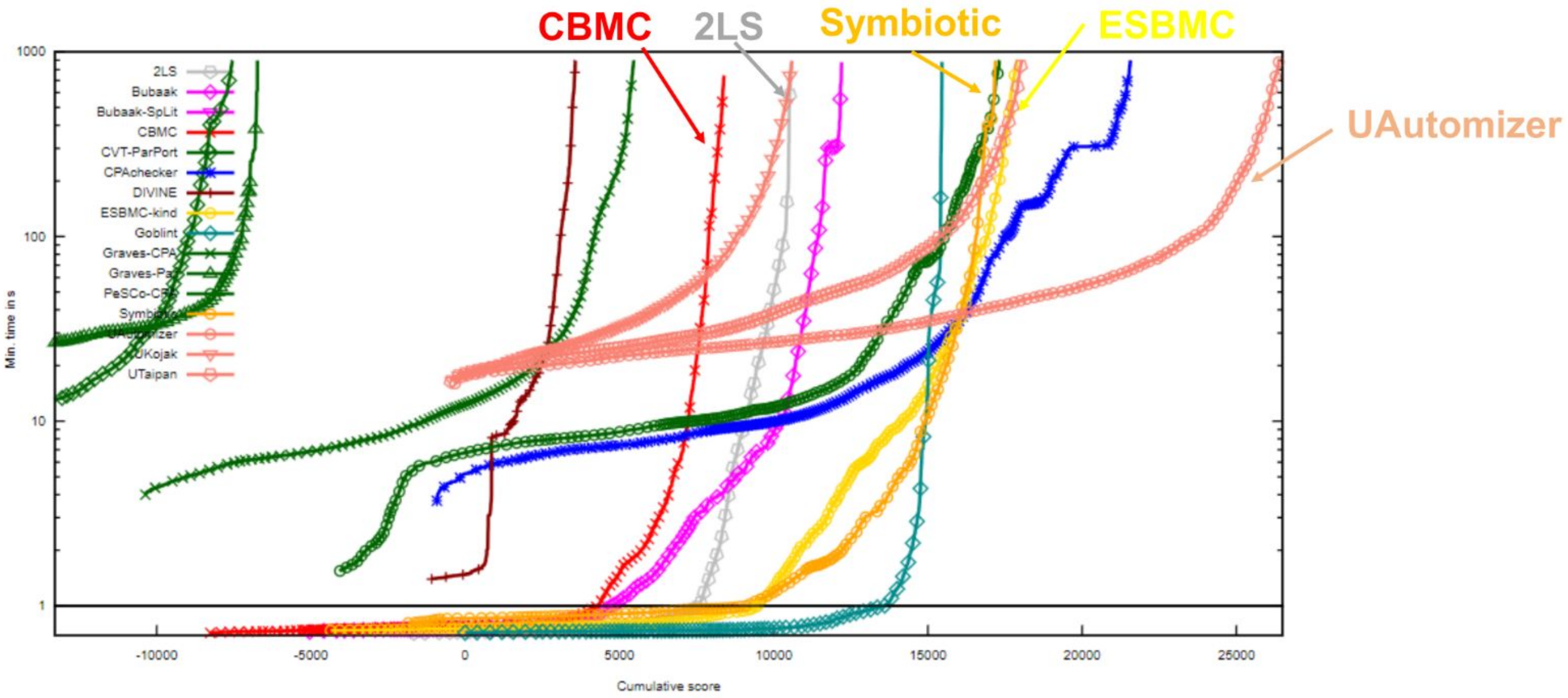


Figure 2: ESBMC module used to process the source-code.

2.2 - ESBMC module - SV-COMP 2024



2.2 - LSVerifier: Property Verification Support

- **LSVerifier tool has support** to exploit the following properties violations:
  - Out-of-bounds array access;
  - Illegal pointer dereferences (null dereferencing, out-of-bounds dereferencing, double free, and misaligned memory access);
  - Arithmetic overflow;
  - Buffer overflow;
  - Not a number (NaN) occurrences in floating-point;
  - Division by zero;
  - Memory leak;
  - Dynamic memory allocation;
  - Data races;
  - Deadlock;
  - Atomicity violations at visible assignments.

STTT Paper:

Verification Methodology

Inputs and definitions for the proposed approach validation.

Main topics:

- Vulnerability Detection Process
- Experiment Setup

The data collected from this verification methodology is used to address the research questions (RQs).

3.1 - Vulnerability Detection Process

1. **Perform Formal Verification Analysis:**
   - Analyze the system and ensure compliance with security properties.
2. **Analyze Property Violations:**
   - Identify and categorize violations based on their nature and severity.
3. **Identify Potential Vulnerabilities:**
   - Assess whether identified violations are actual security threats.
4. **Open a Issue in the OSS Project:**
   - Issue Reporting with a valid property violation that can cause a potential vulnerability.
5. **Discuss the solution with Developers and Maintainers:**
   - Explore fixes and solutions for the vulnerability.

Figure 3. Verification methodology using LSVerifier.

3.2 - Experiment Setup

- LSVerifier was used on the entire set of OSS projects:

```
$ lsverifier -r -f -l dep.txt
```

Where,

-r enables recursive verification, ensuring that the verification process includes all nested functions and dependencies.
-f enables function verification, verifying individual functions within a codebase.
-l dep.txt specifies a file containing paths for including header files from dependencies.

- OSS projects verified:
    - VideoLAN Client (**VLC**) in **version 3.0.18**;
    - VI improved (**VIM**) in **version 9.0.1672**;
    - Terminal multiplexer (**Tmux**) in **version 3.3a**;
    - Reliable USB Formatting Utility System (**RUFUS**) in **version 4.1**;
    - OpenBSD secure shell (**OpenSSH**) in **version 9.3**;
    - Cross-platform Make (**CMake**) in version **3.27.0-rc4**;
    - Network Data (**Netdata**) in **version 1.40.1**;
    - Open Secure Sockets Layer (**OpenSSL**) in **version 3.1.1**;
    - Structured Query Language lightweight (**SQLite**) in **version 3.42.0**;
    - Remote dictionary server (**Redis**) in **version 7.0.11**;

Empirical Study Results

Data collected and analyzed.

- Analysis of vulnerabilities in OSS project dependencies.
- The impact on OSS project security.
- Analysis of developers' behaviors and practices that influence vulnerability mitigation.

4.1 - OSS Projects Exploitation

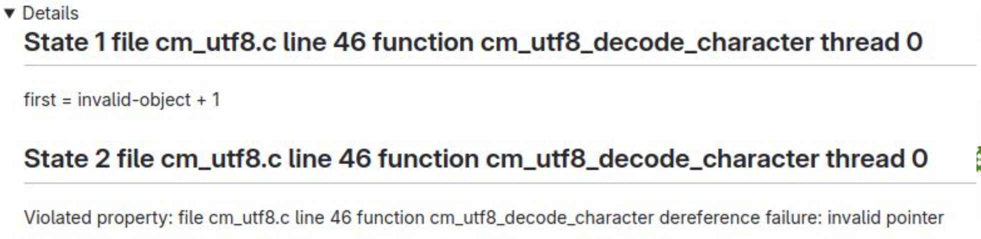
- With verification logs report (counterexample traces) provided by LSVerifier, we reported the issues found to the respective OSS projects according to verification methodology.

Table 1. Issues reported to the open-source software project repositories.

OSS project	Issues reported	Issues fixed
VLC	Issue 1[a]	1
VIM	Issue 1[b]	0
RUFUS	Issue 1[c], Issue 2[d]	1
OpenSSH	Issue 1[e], Issue 2[f]	0
CMake	Issue 1[g]	1
Netdata	Issue 1[h], Issue 2[i]	0
Wireshark	Issue 1[j]	1
OpenSSL	Issue 1[k]	0
SQLite	Issue 1[l], Issue 2[m]	0
Redis	Issue 1[n], Issue 2[o]	0

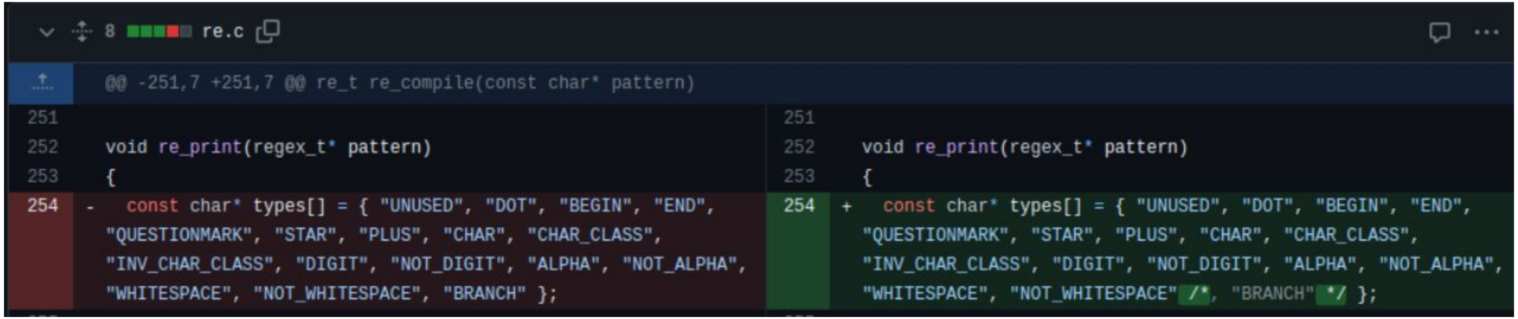
4.2 - Results - RQ1: What are the Common Types and Prevalence of Dependency Vulnerabilities in Open-Source Software Projects?

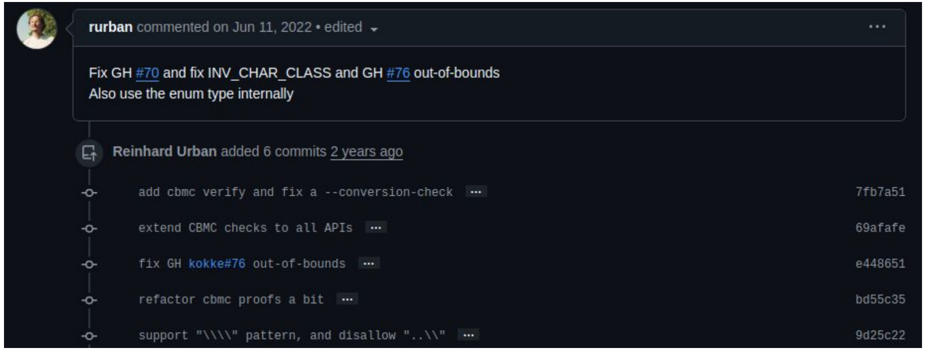
- Cmake Vulnerability: **dereference failure** caused by **invalid pointer** (const char* first).

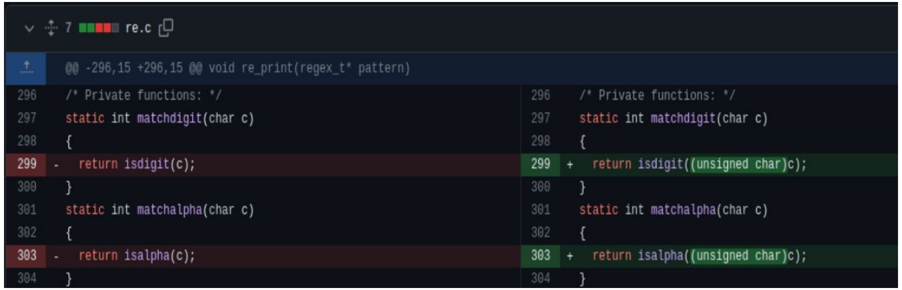


4.2 - Results - RQ1: What are the Common Types and Prevalence of Dependency Vulnerabilities in Open-Source Software Projects?

- **Array Out of Bounds violated**: array `types' upper bound fix for **tyne-regex third-party library.**







4.2 - Results - RQ1: What are the Common Types and Prevalence of Dependency Vulnerabilities in Open-Source Software Projects?

- **Dereference failure (invalid pointer and Null pointer)** issues were found in the **NPL third-party library** used by Wireshark.

4.2 - Results - RQ1: What are the Common Types and Prevalence of Dependency Vulnerabilities in Open-Source Software Projects?

- **[Finding 01]** This study identified common dependency vulnerabilities in open-source software projects, including:
  - **Pointer dereference issues** like **double-free errors (CWE-415)** in **VLC.**
  - **Array access violations** such as **out-of-bounds errors (CWE-787)** in **RUFUS**.
  - **Invalid pointers** were detected in **CMake** and **Wireshark (CWE-824)**,
  - **Null pointer dereferences** in **Wireshark (CWE-476)**.

- These findings demonstrate:
  - Vulnerabilities are not isolated incidents but **recurring issues in dependency management**.
  - Need for **more systematic and proactive mitigation strategies** to ensure OSS project security.

- **[Finding 02]** Developers' actions, such as **removing deprecated subsystems** and **adding verification steps**, demonstrate the **critical role of proactive maintenance in mitigating security vulnerabilities**.

4.2 - Results - RQ2: How do developers' behaviors and practices influence the mitigation of security vulnerabilities?

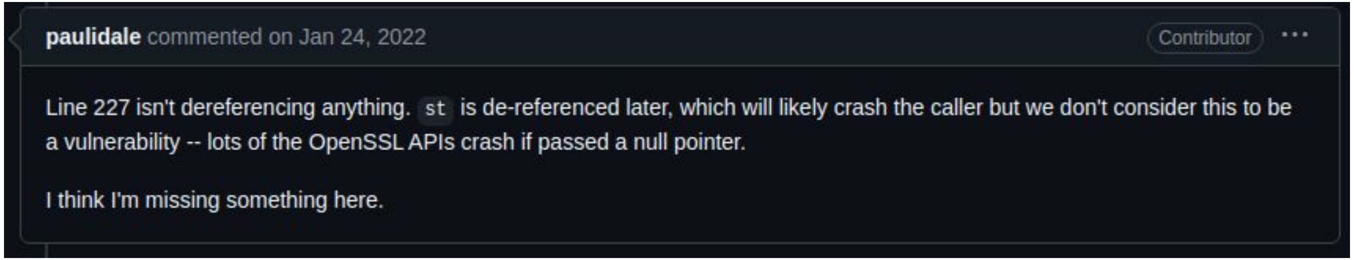
- The SQLite project highlights a common issue in software development: the **tendency to dismiss static analyzer results**.

> **(2) By Richard Hipp (drh) on 2023-10-29 01:14:07 in reply to 1 [link] [source]**
>
> All of the problems you report are almost certainly false-positives generated by a static analyzer. Static analyzers are notorious about spewing forth a fountain of false-positives.
>
> If you have an SQL script or a bit of code that will generate a problem, that's great. Please report it. But if all you have to show us is the output of a static analyzer, your reports will be ignored.
>
> Reply

- **[Finding 3]** Although static analyzers may generate false positives, they often identify **legitimate issues** that may be **missed during manual code reviews**. Also, **formal verifiers, supported by mathematical proofs, ensure higher accuracy.**

4.2 - Results - RQ2: How do developers' behaviors and practices influence the mitigation of security vulnerabilities?

- In the case of OpenSSL, an invalid pointer dereference was reported, but developers did not classify it as a vulnerability or error.



- This perspective reveals a problematic practice: **developers frequently assume that certain conditions will never occur**, dismissing potential vulnerabilities.

- **[Finding 4]** Dismissing potential issues identified by static analysis or formal verification tools, without thorough investigation, exposes software to significant security risks.

4.2 - Results - RQ3: What is the most effective strategy for mitigating risks from dependency vulnerabilities in open-source software projects?

- **[Finding 5]** Thorough verification of false positives is crucial, as dismissing them without proper investigation **can result in overlooked vulnerabilities** that compromise software security. **Rigorous validation of potential false positives** is essential to prevent unintended security weaknesses from entering the codebase.

- **[Finding 6]** Our analysis indicates that functions from dependency libraries, especially in C programs, where pointers are frequently used to access arrays, pose **serious security risks if not carefully verified**.

- **[Finding 7]** Our results demonstrate that **effective library management** plays a more crucial role in mitigating dependency vulnerabilities in OSS projects.

5 - Conclusion

- Developers can significantly **lower security risks by reducing unnecessary dependencies**, selecting well-vetted libraries, and **continuously monitoring and managing dependencies**.
- By addressing our three research questions, we have identified key best practices that developers and the OSS community can adopt to strengthen security measures significantly, as follows:
  - **Providing comprehensive dependency management;**
  - **Integrating formal verification tools and static analysis;**
  - **Fostering a security-first culture;**
  - **Using well-established libraries;**
  - **Enforcing regular security audits and reviews.**
- In summary, **fostering a security-conscious mindset** and **embedding best practices into the development process** is essential for ensuring the **security and longevity of OSS projects**.

References

- [Tang et al. 2022] Tang, W., Xu, Z., Liu, C., Wu, J., Yang, S., Li, Y., Luo, P., and Liu, Y.(2022). Towards understanding third-party library dependency in c/c++ ecosystem. In **37th IEEE/ACM ASE**, pages 1–12.
- [de Sousa et al. 2023] de Sousa, J. O., de Farias, B. C., da Silva, T. A., de Lima Filho, E. B., and Cordeiro, L. C. (2023b). Lsverifier: A bmc approach to identify security vulnerabilities in c open-source software projects. In **XXIII SBSeg**, pages 17–24. SBC.
- [de Sousa et al. 2024] de Sousa, J. O., de Farias, B. C., da Silva, T. A., Cordeiro, L. C., et al. (2024). Finding software vulnerabilities in open-source c projects via bounded model checking. **STTT**. arXiv preprint arXiv:2311.05281.
- [Gadelha et al. 2021] Gadelha, M. R., Menezes, R. S., and Cordeiro, L. C. (2021). Esbmc 6.1: automated test case generation using bounded model checking. **STTT**, 23(6): 857–861.
- [Menezes et al. 2024] Menezes, R. S., Aldughaim, M., Farias, B., Li, X., Manino, E., Shmarov, F., Song, K., Brauße, F., Gadelha, M. R., Tihanyi, N., et al. (2024). Es-bmc v7. 4: Harnessing the power of intervals: (competition contribution). In **TACAS**, pages 376–380. Springer.

XXIV Brazilian Symposium on Information and Computational Systems Security
São José dos Campos/SP - 2024

Trust, but Verify: Evaluating Developer Behavior in Mitigating Security Vulnerabilities in Open-Source Software Projects

Obrigado! Thank You!

Email: janislley.sousa@sidia.com