



21º Congresso Latino-americano de  
Software Livre e Tecnologias Abertas

Realização



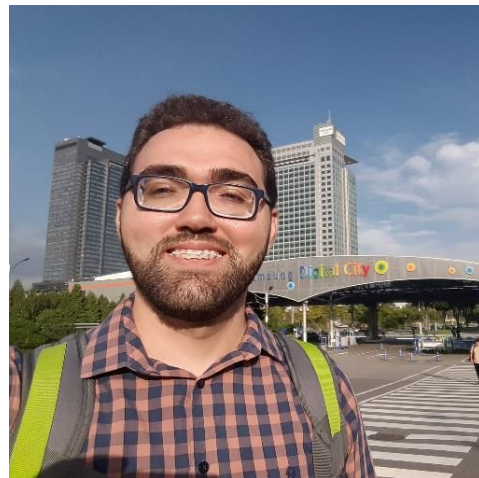
# Segurança em Software de Código Aberto: Detecção de Vulnerabilidades e Melhores Práticas de Mitigação

*Autor:*

**Janisley Oliveira de Sousa  
(SIDIA, UFAM)**

## Sobre Mim!

- Bacharel em Engenharia de Computação pela UFPB (2017).
- Especialização em Software Embarcado com foco em Android pela Cesar School (2018).
- Mestrado em Engenharia Elétrica pela UFAM (2023) na área de Verificação de Software.
- Doutorado em andamento em Engenharia Elétrica pela UFAM (2024-2027) na área de Segurança de Sistemas.
- Membro IEEE desde 2016.
- 9 anos de experiência na área de projetos de sistemas embarcados.
- Atualmente é líder técnico de P&D no Instituto SIDIA em Manaus trabalhando em projetos da Samsung para América Latina.



 @janisley

 janisley.me

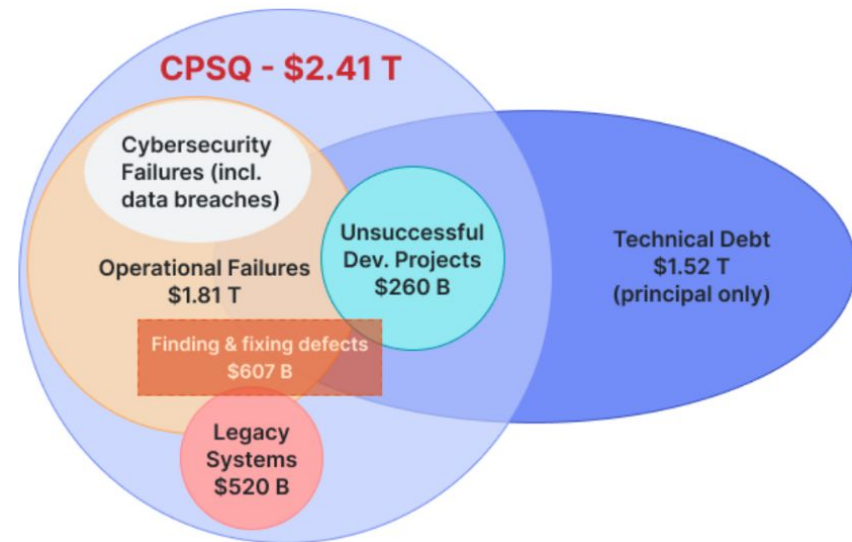


# Agenda

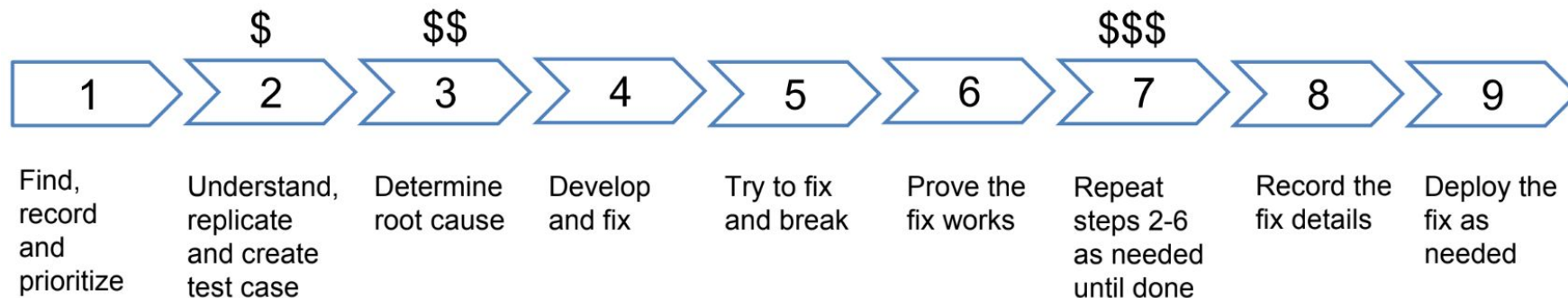
1. Introdução sobre segurança em software de código aberto (OSS).
2. Desafios de segurança no OSS.
3. Importância da detecção de vulnerabilidades.
4. Impacto das falhas de segurança em OSS.
5. Comportamento de desenvolvedores na mitigação de vulnerabilidades.
6. Necessidade de ferramentas eficazes.
7. Estratégias de mitigação eficazes.
8. Integração de ferramentas de verificação no fluxo de trabalho de desenvolvimento.
9. Recomendações para desenvolvedores e equipes de segurança.

# 1. Segurança em software de código aberto (OSS)

- Mais de 50% dos custos de um projeto de software atualmente não são alocados ao processo criativo de desenvolvimento de software, mas às tarefas corretivas de depuração, correção de erros e falhas de segurança [1].

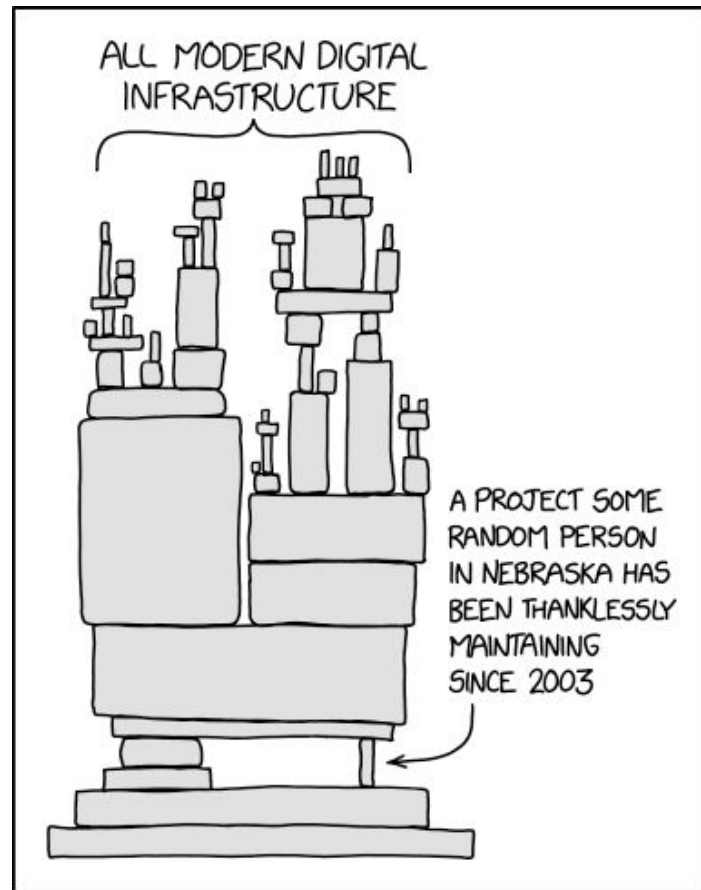


[1] KRASNER, Herb. The cost of poor software quality in the US: A 2024 Report.



# 1. Segurança em software de código aberto (OSS)

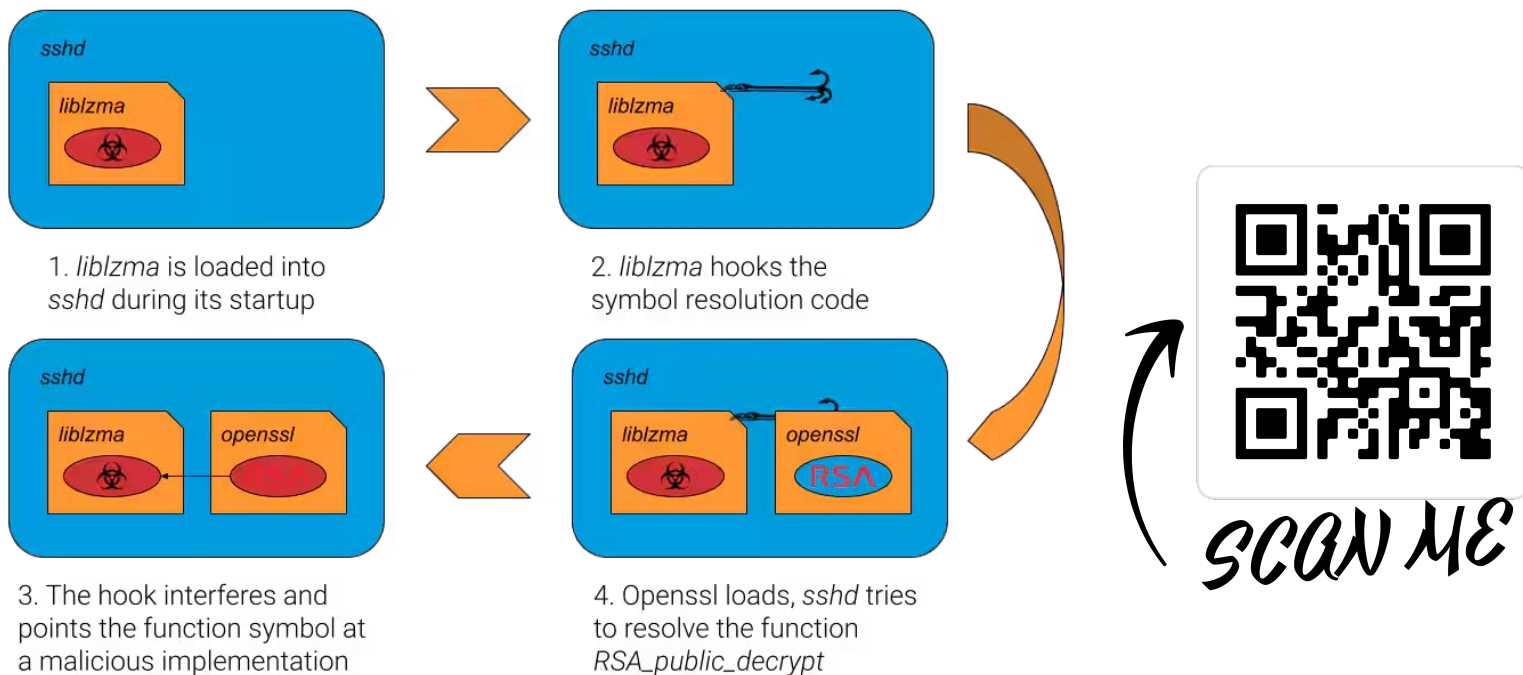
- **Desenvolvimento de software** utiliza um grande volume de **código de terceiros** proveniente de **bibliotecas externas**.
- **Ausência de verificação do código** em busca de bugs e problemas de segurança usando **ferramentas especializadas**.
- Como um projeto de software pode depender de várias bibliotecas de código aberto, a **análise de toda a árvore de dependências** de um projeto de software pode se tornar **muito complexa**.



Fonte: <https://xkcd.com/2347/>

# 1. Segurança em software de código aberto (OSS)

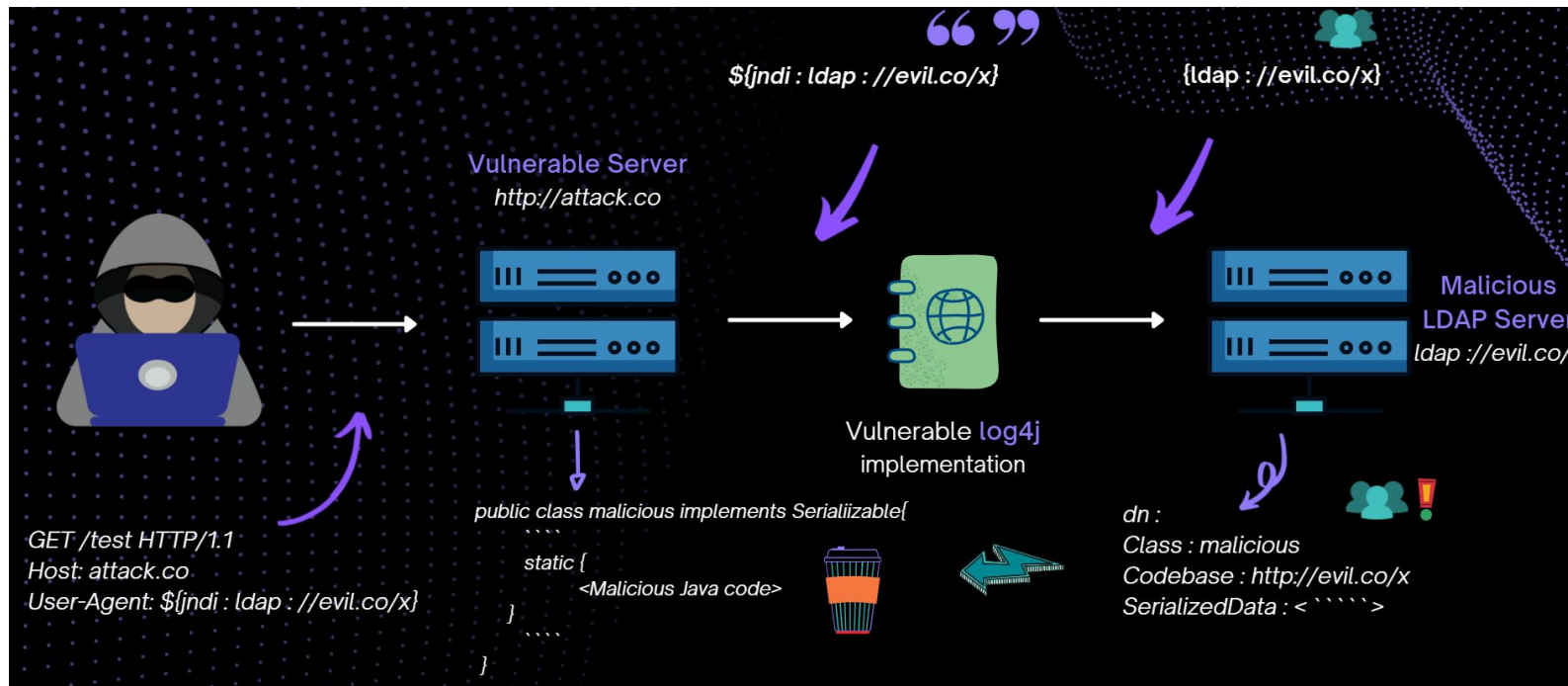
- Uma vulnerabilidade crítica (CVE-2024-3094) descoberta na **ferramenta de compressão "xz" do Linux**, onde um backdoor malicioso foi introduzido nas versões 5.6.0 e 5.6.1 da **biblioteca liblzma**, permitindo que atacantes obtenham execução remota de código.





# 1. Segurança em software de código aberto (OSS)

- **Log4Shell (CVE-2021-44228)** é uma vulnerabilidade de **zero-day** no Apache Log4j 2. Trata-se de uma vulnerabilidade de execução remota de código (RCE).



**O comportamento e as práticas dos desenvolvedores influenciam significativamente a mitigação de vulnerabilidades de segurança dentro de projetos de código aberto.**



## 2. Desafios de segurança no OSS

- **Vulnerabilidades não identificadas:** Dificuldade em identificar falhas de segurança.
- **Falta de atualização e manutenção:** Atrasos na correção de vulnerabilidades e na atualização de bibliotecas e pacotes.
- **Contribuições externas descontroladas:** Falta de revisão rigorosa de código, permitindo a introdução de vulnerabilidades por desenvolvedores externos.
- **Gerenciamento complexo de dependências:** A complexidade de gerenciar várias dependências e suas vulnerabilidades em projetos grandes.



**A segurança em OSS está constantemente ameaçada por vulnerabilidades ocultas, atualizações negligenciadas, contribuições descontroladas e a complexidade crescente no gerenciamento de dependências.**

### 3. Importância da detecção de vulnerabilidades

- **Identificação antecipada de falhas em dependências:** Detectar vulnerabilidades em bibliotecas de terceiros antes que sejam exploradas.
- **Redução de riscos para usuários e colaboradores:** Impedir que falhas de segurança afetem usuários finais ou comprometam a integridade do projeto.
- **Manutenção da confiança na comunidade OSS:** Garantir que o software continue confiável, promovendo um ecossistema seguro e colaborativo.



## 4. Impacto das falhas de segurança em OSS

- **Exposição de dados sensíveis:** Falhas podem permitir o acesso não autorizado a informações pessoais, financeiras ou confidenciais de usuários.
- **Prejuízos financeiros:** Explorações de falhas podem resultar em custos elevados com recuperação, multas regulatórias e perda de receita.
- **Comprometimento da integridade do código:** Ataques podem modificar o código fonte ou adicionar backdoors, prejudicando a confiabilidade e segurança do software.
- **Danos à reputação do projeto:** Falhas de segurança podem manchar a imagem do projeto OSS, especialmente se forem amplamente divulgadas na mídia.

# 5. Comportamento de desenvolvedores na mitigação de vulnerabilidades


ANais do Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSEG)

SOL ▾

TODAS AS EDIÇÕES

SOBRE O EVENTO

EXPEDIENTE

 BUSCAR


INÍCIO / TODAS AS EDIÇÕES /  
2024: ANAIS DO XXIV SIMPÓSIO BRASILEIRO DE SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS /  
Artigos Completos

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

**Janisley Oliveira de Sousa**  
Sidia / UFAM

**Bruno Carvalho de Farias**  
University of Manchester

**Eddie Batista de Lima Filho**  
UFAM / TPV Technology

 PDF (ENGLISH)

PUBLICADO

16/09/2024

### IDIOMA

Português (Brasil)  
English

Artigo SBSEG'24:

SCAN ME

LATINOWARE 2024

27 a 29 de novembro de 2024

## 5. Comportamento de desenvolvedores na mitigação de vulnerabilidades

Este estudo identificou vulnerabilidades comuns de dependência em projetos de software de código aberto, incluindo:

- Problemas de desreferência de ponteiro, como erros de **double-free (CWE-415)** no **VLC**.
- Violações de acesso a arrays, como erros de **out-of-bounds (CWE-787)** no **RUFUS**.
- Ponteiros inválidos detectados no **CMake** e no **Wireshark - Invalid Pointer (CWE-824)**.
- Desreferências de ponteiros nulos (**Null Pointer**) no **Wireshark (CWE-476)**.



## 5. Comportamento de desenvolvedores na mitigação de vulnerabilidades

- O projeto SQLite destaca um problema comum no desenvolvimento de software: a tendência de desconsiderar os resultados das ferramentas de análise estática.

---

(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

---

## 5. Comportamento de desenvolvedores na mitigação de vulnerabilidades

- No caso do OpenSSL, uma desreferência de ponteiro inválido foi reportada, mas os desenvolvedores não a classificaram como uma vulnerabilidade ou erro.

paulidale commented on Jan 24, 2022

Contributor

...

Line 227 isn't dereferencing anything. `st` is de-referenced later, which will likely crash the caller but we don't consider this to be a vulnerability -- lots of the OpenSSL APIs crash if passed a null pointer.

I think I'm missing something here.

## 5. Comportamento de desenvolvedores na mitigação de vulnerabilidades

- Desenvolvimento orientado a segurança com **políticas e processos mais rígidos**.
- As vulnerabilidades não são incidentes isolados, mas **problemas recorrentes na gestão de dependências**.
- Há uma **necessidade de estratégias de mitigação mais sistemáticas e proativas** para garantir a segurança de projetos OSS.
- Ações dos desenvolvedores, como a **remoção de subsistemas obsoletos e a adição de etapas de verificação**, demonstram o papel crucial da manutenção proativa na mitigação de vulnerabilidades de segurança.

**Os desenvolvedores podem reduzir significativamente os riscos de segurança ao diminuir dependências desnecessárias, selecionar bibliotecas bem verificadas e monitorar e gerenciar continuamente as dependências.**

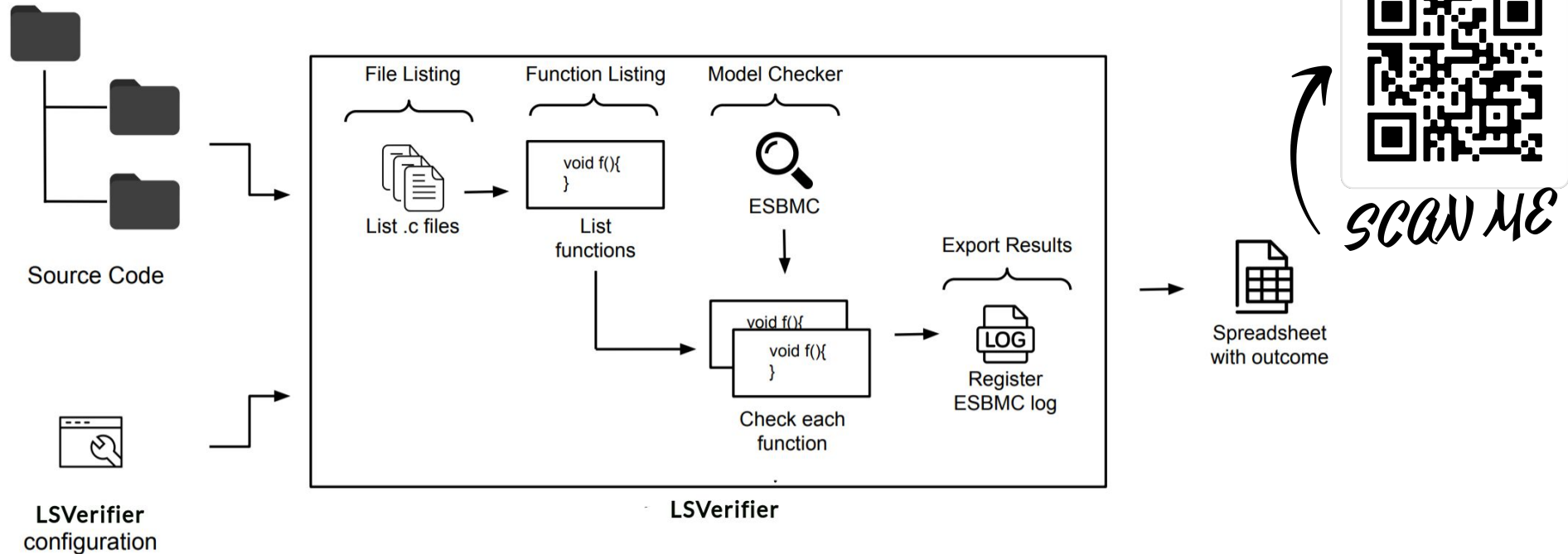
## 6. Necessidade de ferramentas eficazes

- **Análise Estática:** Ferramentas que analisam o código fonte sem executá-lo, detectando vulnerabilidades e erros de segurança antes de serem implementados.
- **Verificação Formal:** Técnicas de verificação formal garantem que o sistema atenda a propriedades de segurança específicas.
- **Lógica Fuzzy:** A lógica fuzzy pode ser aplicada para detectar comportamentos imprevistos no sistema e modelando incertezas.
- **Inteligência Artificial (IA):** Ferramentas baseadas em IA utilizam aprendizado de máquina para detectar vulnerabilidades em grandes bases de código e sugerir melhorias automáticas.



## 6. Necessidade de ferramentas eficazes

- **LSVerifier**

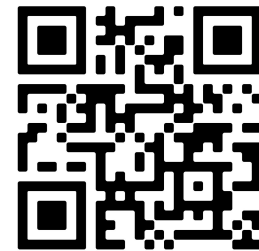




## 6. Necessidade de ferramentas eficazes

- **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.

Artigo STTT:



*SCAN ME*

## 7. Estratégias de mitigação eficazes

1. **Realizar Análise e Verificação:** Analisar o sistema e garantir conformidade com as propriedades de segurança.
2. **Analisar Violações de Propriedades:** Identificar e categorizar as violações com base em sua natureza e severidade.
3. **Identificar Potenciais Vulnerabilidades:** Avaliar se as violações identificadas são ameaças de segurança reais.
4. **Abrir uma Issue no Projeto OSS:** Relatar a falha com uma violação de propriedade válida que pode causar uma vulnerabilidade potencial.
5. **Discutir a Solução com Desenvolvedores e Mantenedores:** Explorar correções e soluções para a vulnerabilidade.



## 7. Estratégias de mitigação eficazes

- **Auditoria de Código Regular:** Realizar revisões contínuas e profundas do código, identificando e corrigindo vulnerabilidades de segurança.
- **Uso de Ferramentas de Análise Estática e Dinâmica:** Integrar ferramentas de análise estática para detectar falhas em tempo de desenvolvimento e execução.
- **Gestão de Dependências Segura:** Monitorar e atualizar regularmente bibliotecas e dependências.
- **Práticas de Programação Segura:** Adotar padrões de codificação segura.
- **Treinamento e Conscientização sobre Segurança:** Promover uma cultura de segurança entre desenvolvedores e contribuidores.

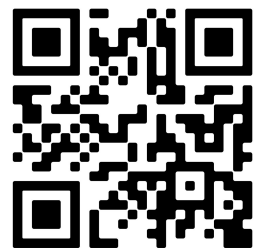
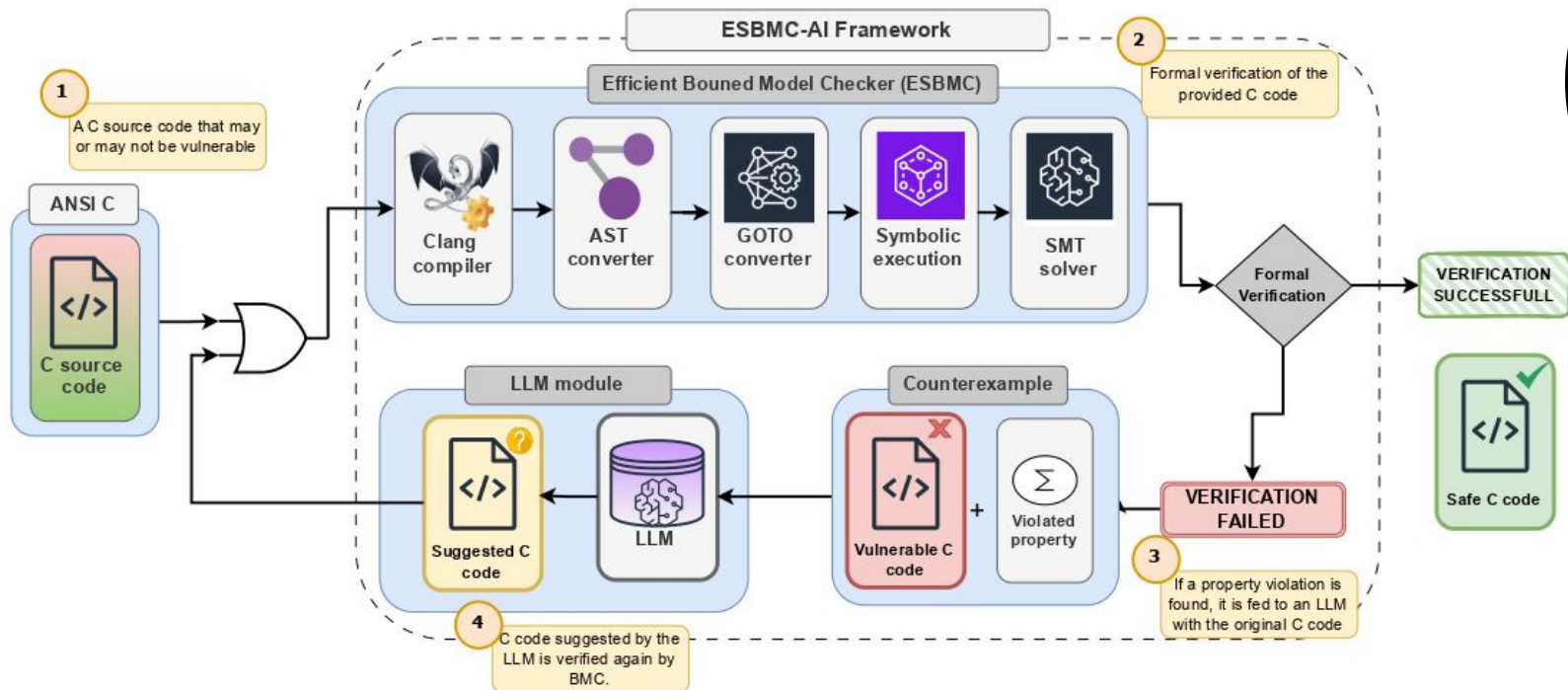
## 8. Integração de ferramentas de verificação no fluxo de trabalho de desenvolvimento

A integração de ferramentas de verificação no fluxo de trabalho de desenvolvimento é essencial para melhorar a segurança e a qualidade do código em projetos de código aberto (OSS). Algumas ferramentas chave incluem:

- **CI/CD**: Jenkins, GitLab CI, CircleCI e GitHub Actions para automação de testes e entregas contínuas.
- **Análise Estática**: SonarQube, Coverity, Clang Static Analyzer, LSVerifier e Checkmarx para detectar falhas de segurança no código.
- **Análise Dinâmica**: Valgrind e AddressSanitizer para detectar erros de memória e vulnerabilidades em tempo de execução.
- **Modelos de verificação (LLM) baseados em IA**: GitHub Copilot, ESBMC-AI e Snyk são exemplos.

## 8. Integração de ferramentas de verificação no fluxo de trabalho de desenvolvimento

- Modelos de verificação (LLM) baseados em IA: ESBMC-AI



SCAN ME

## 8. Integração de ferramentas de verificação no fluxo de trabalho de desenvolvimento

O uso de IA e automação aprimora ainda mais a segurança:

- **Inteligência Artificial:** ESBMC-AI, DeepCode e Codota oferecem análise de código baseada em aprendizado de máquina para detectar vulnerabilidades.
- **Gerenciamento de Dependências:** Github Dependabot e Renovate CLI automatizam a atualização de dependências, corrigindo vulnerabilidades em bibliotecas desatualizadas.
- **Ferramentas de Segurança:** OWASP Dependency-Check para verificar vulnerabilidades em dependências de código aberto.



**Em projetos de código aberto (OSS), ignorar os resultados das ferramentas de análise estática é como deixar portas abertas para ataques – um risco que não pode ser negligenciado.**

## 9. Recomendações para desenvolvedores e equipes de segurança

Melhores práticas que os desenvolvedores e a comunidade OSS podem adotar para fortalecer significativamente as medidas de segurança incluem:

- Promover uma cultura de segurança em primeiro lugar;
- Fornecer um gerenciamento abrangente de dependências;
- Garantir atualizações constantes dos sistemas;
- Integrar ferramentas de verificação e análise de código;
- Utilizar bibliotecas bem estabelecidas;
- Exigir auditorias e revisões de segurança regulares.

**Promover uma mentalidade consciente de segurança e incorporar as melhores práticas no processo de desenvolvimento de Software é essencial para garantir a segurança e a longevidade dos projetos OSS.**

# Grupo de Pesquisa



Ph.D. Janisley Oliveira  
(UFAM/SIDIA)



Ph.D. Bruno Farias  
(Manchester)



Dr. Eddie Batista  
(UFAM/TPV)



Dr. Lucas Cordeiro  
(UFAM/Manchester)



# Dúvidas?

*Contato:*

[janislley@ieee.org](mailto:janislley@ieee.org)



**ESBMC**



**UFAM**



The University of Manchester

**sidia** **SAMSUNG**  
//impact innovation



**LATINOWARE 2024**



Realização:

