

# Artigo C++

## Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects



Grupo: Augusto Stambassi Duarte, Davi Cândido de Almeida, João Pedro Torres, Lucas Carneiro Nassau Malta

# Sumário

1. Artigo.....	3
2. Contexto.....	4
3. Materiais e Métodos.....	9
4. Resultados e Discussões.....	16
5. Conclusão.....	19
6. Considerações Finais.....	20
7. Apêndices.....	21
8. Referencias.....	26

# Artigo



PUC Minas

Caracterizando vulnerabilidades de estouro de buffer em grandes projetos C/C++

## Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects

Editora: [IEEE](#) (Intitute of Electrical and Electronics Engineers)

Periódico: [IEEE Access](#) (A3)

### Autores:



José D'Abruzzo Pereira 

Centre for Informatics and Systems of the  
University of Coimbra, Department of  
Informatics Engineering, University of  
Coimbra, Coimbra, Portugal



Marco Vieira 

Centre for Informatics and Systems of the  
University of Coimbra, Department of  
Informatics Engineering, University of  
Coimbra, Coimbra, Portugal



Naghmeh Ivaki 

Centre for Informatics and Systems of the  
University of Coimbra, Department of  
Informatics Engineering, University of  
Coimbra, Coimbra, Portugal

2021

8

Cites in  
Papers

2546

Full  
Text Views

# Contexto

## Vulnerabilidades de Software

- Vulnerabilidades são falhas **críticas**
  - Podem levar a **ataques de segurança e danos severos**
  - **Exemplos:** perda de dados confidenciais, prejuízos financeiros e violações de segurança
- Cenário Pós-Pandemia:
  - Maior exposição dos sistemas devido ao trabalho e serviços **online**
  - Aumento no investimento em cibersegurança (mais de **40% do orçamento** de TI em 2021)
- Linguagens C/C++ são as mais afetadas:
  - Responsáveis por **52% das vulnerabilidades** em código aberto (46% em C e 6% em C++)
  - *Buffer overflow* é o tipo de erro mais **frequente**

# Contexto

## C/C++ e o Problema da Memória

- Principais Vulnerabilidades em C/C++ (dados do estudo do Mend.io)
  - *Buffer errors* (CWE-119): Problemas no gerenciamento de memória que podem ser explorados para ataques **(Top 20)**
  - *Input validation* (CWE-20): Falhas na checagem de dados de entrada, abrindo brechas para injeção de código **(Top 12)**
  - *Information Leak / Disclosure* (CWE-200): Vazamento acidental de dados confidenciais do sistema **(Top 17)**
- Desafios na Detecção e Correção:
  - Falta de **conhecimento** e **experiência** na aplicação de boas práticas
  - Limitação nas técnicas existentes: Análise Estática de Código (SATs) e Métricas de Software (SMs)

# Contexto

## O que é o *Buffer Overflow*?

- Vulnerabilidade de Gerenciamento de Memória
  - Ocorre quando o software lê ou escreve dados **além do espaço de memória alocado**
  - Pode levar a **travamentos, vazamento de informações** ou **execução de código malicioso**
- Exemplo Prático: Função *strcpy* da biblioteca *string.h*
  - Copia dados **sem verificar** se o destino tem **espaço suficiente**
  - Código vulnerável no **Kernel do Linux** (antes da correção VE-2010-4527) utilizava *strcpy*
- Soluções para *strcpy* e Similares
  - Substituir por **funções mais seguras**, como *strncpy*
  - Adicionar **verificações de limite** antes de usar funções inseguras

```
1  n = num_mixer_volumes++;  
2  
3  strcpy(mixer_vols[n].name, name);  
4  
5  if (present)  
6      mixer_vols[n].num = n;  
7  else  
8      mixer_vols[n].num = -1;
```

# Contexto

## Técnicas de identificação de problemas

1. Correspondência de padrões sintáticos
2. Análise lexical
3. Análise de fluxo de dados
4. Análise sintática
5. Verificação de modelos e execução simbólica

**Cada problema identificado e relatado por um SAT é chamado de "alerta"**



# Objetivo

- Buscam estudar as **principais características de códigos com vulnerabilidades de estouro de buffer**, bem como as correções aplicadas para removê-las
  - i) estudar as **alterações necessárias aos código-fonte** para se corrigir as vulnerabilidades de estouro de buffer
  - ii) estudar a eficiência de **ferramentas de análise estática (SATs)**, ou seja, ferramentas de detecção de erros automática
  - iii) compreender a eventual **correlação na análise de Métricas de Software (SMs)** com a existência de vulnerabilidades de estouro de buffer



Relação entre, tamanho, complexidade, quantidade de comandos com a presença de vulnerabilidades



# Materiais e Métodos

1. Selecionar diversos projetos de software
2. Coletar os metadados de vulnerabilidade
3. Coletar o código fonte em várias versões
4. Classificar as vulnerabilidades usando o ODC
5. Coletar e analisar os alertas das SATs
6. Coletar e analisar as Métricas de Software (SMs)

# Materiais e Métodos

## 1. Selecionar diversos projetos de software

Escolha de 3 dos 5 projetos mantidos pelo dataset de Alves et al.

**Critério:** projetos com o maior número de vulnerabilidades conhecidas e com maior quantidade de ataques sofridos (CVE-Details)

**TABLE 1. Number of vulnerabilities: A) all vulnerabilities, B) vulnerabilities with the CWE-119 identifier, C) CWE-119 vulnerabilities with “overflow” in vulnerability type.**

	Linux Kernel	Mozilla	Xen	Total
<b>A</b>	867 (100.0%)	2441 (100.00%)	148 (100.0%)	3456 (100.0%)
<b>B</b>	123 (14.2%)	336 (13.8%)	18 (12.2%)	477 (13.8%)
<b>C</b>	98 (11.3%)	56 (2.3%)	5 (3.4%)	159 (4.6%)

# Materiais e Métodos

## 2. Coletar os metadados de vulnerabilidade

Informações sobre cada vulnerabilidade coletada pelo CVE-Details

- CVSS (Sistema de pontuação de vulnerabilidades comuns)
- Impacto
- Tipo da vulnerabilidade (Pode ter vários valores)
- CWE
- Todas as versões do Software que foram impactadas pelo erro

# Materiais e Métodos

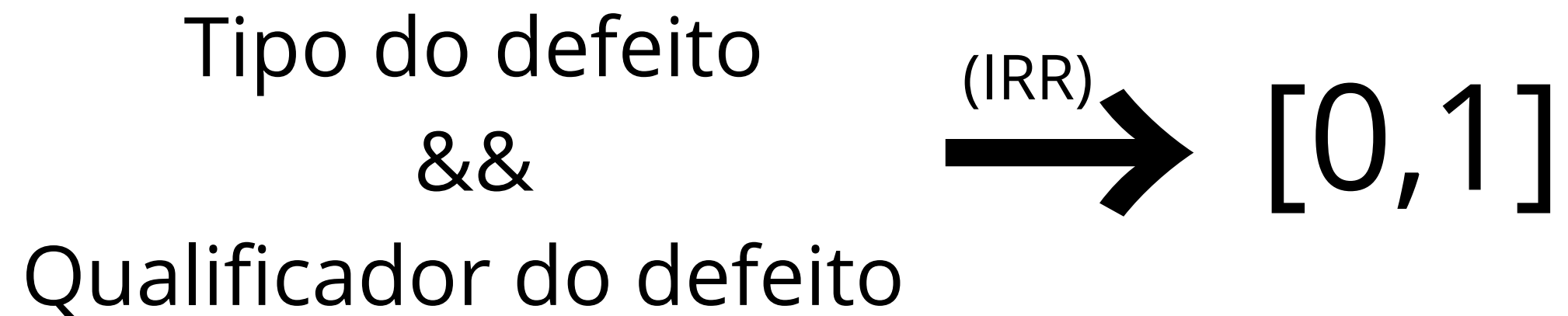
## 3. Coletar o código fonte em várias versões

Códigos coletado pelo GitHub e, usando o BugTracker, a versão pré e pós correção da vulnerabilidade

# Materiais e Métodos

## 4. Classificar as vulnerabilidades usando o ODC

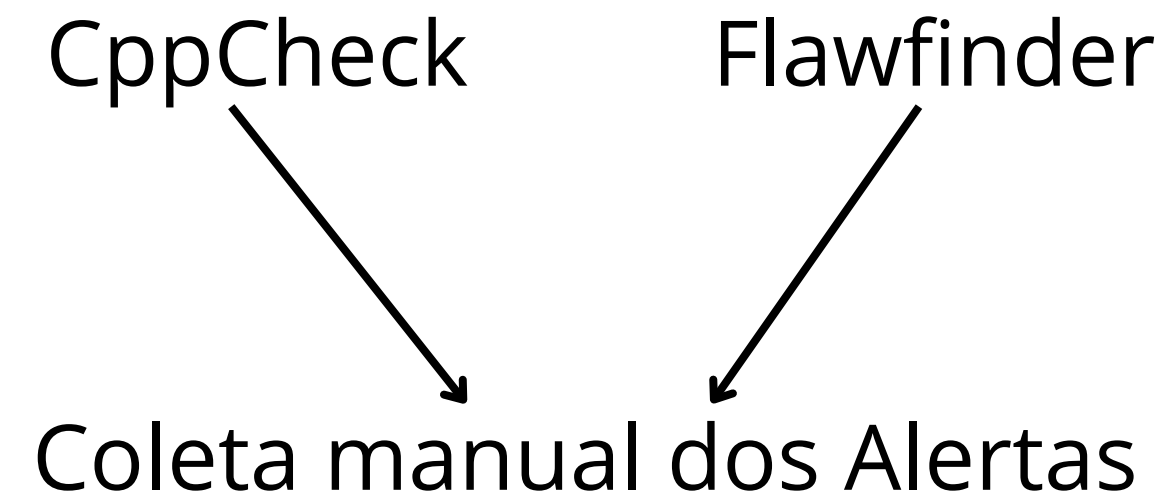
Orthogonal Defect Classification ou “Classificação Ortogonal de defeitos”



> 0 : Sem concordância | 0 - 0.20: Pouca concordância | 0.21 - 0.40: Concordância justa  
0.41 - 0.60: Moderada concordância | 0.61 - 0.80: Concordância substancial |  
0.81 - 1.00: Quase perfeito

# Materiais e Métodos

## 5. Coletar e analisar os alertas das SATs



# Materiais e Métodos

## 6. Coletar e analisar as Métricas de Software (SMs)

Uso da SciTools Understand para classificação de métricas de Software apenas para arquivos  
(Maior parte do dataset do Alves et al.)



# Resultados e Discussões

- **Resultados principais:**

- A maioria das vulnerabilidades pertence aos tipos **ODC Checking e Algorithm/Method**
- **SATs falham em detectar a maioria das vulnerabilidades**, especialmente verificações ausentes ou incorretas
- **Não foi possível encontrar relação causal** entre correções de buffer overflow e as métricas de software

Distribuição de vulnerabilidades entre o tipo de defeito ODC e os qualificadores.


Defect Type	Incorrect	Ausentes Missing	Estranho Extraneous	Total
<b>Checking</b>	31 (14.35%)	51 (23.61%)	3 (1.39%)	85 (39.35%)
<b>Algorithm/Method</b>	50 (23.15%)	12 (5.56%)	2 (0.93%)	64 (29.63%)
Assignment/Initialization	24 (11.11%)	18 (8.33%)	0 (0.00%)	42 (19.44%)
Interface	16 (7.41%)	3 (1.39%)	0 (0.00%)	19 (8.80%)
Function	2 (0.93%)	4 (1.85%)	0 (0.00%)	6 (2.78%)
Timing	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
<b>Total</b>	123 (56.94%)	88 (40.74%)	5 (2.31%)	216 (100.00%)

**68,98%** das vulnerabilidades


# Resultados e Discussões

- Resultados principais:
  - Os códigos pertencentes ao **kernel do linux** apresentaram **maior quantidade de vulnerabilidades**
  - Desenvolvedores **não antecipam a necessidade da "Checking"**
    - Difícil considerar todas as possibilidades que podem levar a um estouro de buffer

Distribuição de vulnerabilidades entre os tipos de defeitos do ODC para os projetos (Kernel Linux, Mozilla, Xen).



Defect Type	Linux Kernel	Mozilla	Xen
Checking	45 (37.50%)	39 (42.86%)	1 (20.00%)
Algorithm/Method	33 (27.50%)	27 (29.67%)	4 (80.00%)
Assignment/Initialization	27 (22.50%)	15 (16.48%)	0 (0.00%)
Interface	11 (9.17%)	8 (8.79%)	0 (0.00%)
Function	4 (3.33%)	2 (2.20%)	0 (0.00%)
Timing	0 (0.00%)	0 (0.00%)	0 (0.00%)
Total	120 (100.00%)	91 (100.00%)	5 (100.00%)



Ineficiência em  
detecção por  
SATs

# Limitações

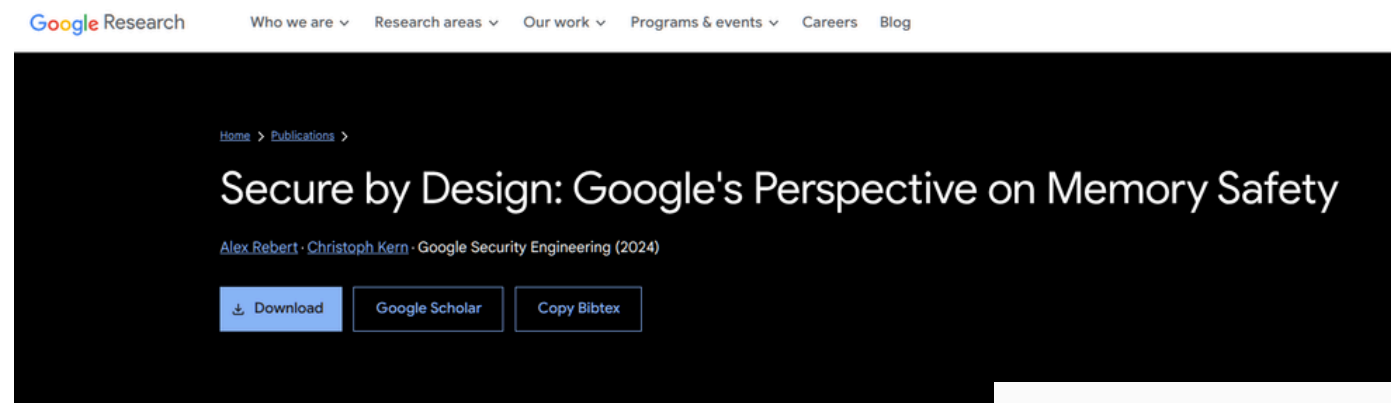
1. Metodologia abordando somente um tipo de erro
2. O artigo se limita a uma análise nas linguagens C e C++
3. Uso de apenas duas SATs, CppCheck e Flawfinder
4. Dependência de validação manual, na classificação ODC e nas identificações SATs

# Conclusão

- **Estudo de vulnerabilidades de estouro de buffer em C/C++**
  - Foram analisadas vulnerabilidades de buffer overflow em três grandes projetos C/C++ Linux Kernel, Mozilla e Xen
  - Cada vulnerabilidade foi classificada usando ODC (Orthogonal Defect Classification)
  - Foram comparadas versões vulneráveis e corrigidas dos códigos, utilizando:
    - Ferramentas de Análise Estática (SATs): CppCheck e Flawfinder
    - Métricas de Software (SMs)
- Trabalhos futuros buscam **ampliar o escopo de análise** para incluir:
  - Outros tipos de vulnerabilidades além de buffer overflow
  - Vulnerabilidades mais recentes e projetos mais modernos
  - Criar novas regras de identificação do buffer overflow para as SATs
  - Criar um mecanismo de priorização para equipes revisoras

# Considerações Finais

- Relevância do assunto:



## Governo dos EUA quer que linguagens como C e C++ deixem de ser usadas

Linguagens de programação como C e C++ são consideradas inseguras por FBI e CISA; recomendações incluem mudança para linguagens como Rust, Go e Python

Por Emerson Alecrim  
08/11/2024 às 18:23

notícias

## Firefox está refazendo o gerador de relatórios de erros em Rust

25/04/2024

Home > Notícias > Software > Linux

## Linus Torvalds confirma Rust no kernel do Linux 6.1

Por Kaique Lima • Editado por Claudio Yuge | 21/09/2022 às 18:20



Movimento gradual rumo a uma infraestrutura de software mais segura

# Apêndice

## Augusto

- Diversas ferramentas para achar vulnerabilidades, como CppCheck e Flawfinder
- Diferentes métricas de software

# Apêndice

## Davi

- **Natureza dos problemas** computáveis e seu impacto na detecção de vulnerabilidades
  - Muitos problemas possuem uma difícil detecção dos possíveis erros, muitas vezes devido a natureza do problema o que dificulta sua abordagem e solução
- Substituição de sistemas feitos em C/C++ por LPs com maior foco em segurança como Rust
  - Natureza permissiva em relação ao gerenciamento de memória, influencia o **uso crescente de Rust** em projetos como o **kernel Linux, navegadores como o Firefox, e sistemas embarcados.**



# Apêndice

## Lucas

- Relevância e Contexto Atual
  - A discussão sobre vulnerabilidades é crucial, especialmente com o aumento da exposição online no cenário pós-pandemia e o crescente investimento em cibersegurança.
- Foco Preciso na Problemática
  - A apresentação direciona a atenção para a alta incidência de vulnerabilidades em C/C++ e a prevalência de buffer overflows, indicando a área específica de estudo.
- Clareza na Definição de Conceitos
  - A explicação do buffer overflow é direta e reforçada por um exemplo prático (função strcpy), tornando um conceito técnico acessível.

# Apêndice

## João Pedro

- Como as vulnerabilidades interferem nas linguagens
  - Apesar de C/C++ serem ambas linguagens com alto nível de desempenho e acesso para o seu desenvolvedor, essas deixam a desejar na segurança dos seus códigos, deixando tudo como responsabilidade do programador, o que faz que esses busquem novas ferramentas.



```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      cout << "Obrigado pela atenção!" << endl;
6
7      return 0;
8  }
```

# Referências

1. PEREIRA, José D'Abruzzo; IVAKI, Naghmeh; VIEIRA, Marco. Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects. IEEE Access, v. 9 (2021), p. 142879–142892, 2021. DOI: 10.1109/ACCESS.2021.3120349.
2. ALVES, Douglas. Linus Torvalds confirma Rust no kernel do Linux 6.1. Canaltech, 13 out. 2022. Disponível em: <https://canaltech.com.br/linux/linus-torvalds-confirma-rust-no-kernel-do-linux-61-225803/>. Acesso em: 15 jun. 2025
3. SILVA, Ricardo. Firefox agora utiliza Rust para lidar com relatórios de erros. Diolinux, 27 fev. 2023. Disponível em: <https://diolinux.com.br/noticias/firefox-relatorios-de-erros-em-rust.html>. Acesso em: 15 jun. 2025.
4. LUPETTI, Emerson. Governo dos EUA quer que linguagens como C e C++ deixem de ser usadas. Tecnoblog, 04 nov. 2022. Disponível em: <https://tecnoblog.net/noticias/governo-dos-eua-quer-que-linguagens-como-c-e-c-deixem-de-ser-usadas/>. Acesso em: 15 jun. 2025.