

# Estudo comparativo entre 3 analisadores de código C sobre as ameaças de nível 1 do padrão CERT C 2016

André Eduardo Pacheco Dias

Universidade do Estado de Santa Catarina

`andre.dias@msn.com`

Gabriela Moreira Mafra

Universidade do Estado de Santa Catarina

`gabrielamoreiramafra@gmail.com`

Lucas Schmitt Seidel

Universidade do Estado de Santa Catarina

`lucasschmittseidel@gmail.com`



Objetivo

Analísadores de código

Regras

Resultados e Considerações



# Programação segura

- Prática
- Mitigar vulnerabilidades
  - Geradas por erros de programação
- Importância crescente devido ao aumento de ativos resguardados por *software*.



# Programação segura

- Prática
- Mitigar vulnerabilidades
  - Geradas por erros de programação
- Importância crescente devido ao aumento de ativos resguardados por *software*.

## CERT C

### Padrão de programação segura em C/C++

- Dividido em Regras
- Classifica por prioridade
  - Grau de severidade
  - Probabilidade de exploração
  - Custo de remediação



# CERT C

Nível	Prioridades	Possível Interpretação
L1	12, 18, 27	Alta severidade, muito provável, sem custo de reparo
L2	6, 8, 9	Média severidade, provável, custo médio de reparo
L3	1, 2, 3, 4	Baixa severidade, pouco provável, custo alto de reparo

Table: Níveis de prioridades



# CERT C

Nível	Prioridades	Possível Interpretação
L1	12, 18, 27	Alta severidade, muito provável, sem custo de reparo
L2	6, 8, 9	Média severidade, provável, custo médio de reparo
L3	1, 2, 3, 4	Baixa severidade, pouco provável, custo alto de reparo

Table: Níveis de prioridades

## Escopo

Neste trabalho, foram analisadas apenas as regras de nível 1 (L1). Isso inclui 17 regras.



# Analísadores de código

Para evitar falha humana e economizar recurso, é interessante usar um analisador automático para encontrar falhas.

O padrão CERT C indica que seja usada a ferramenta capaz de detectar o maior número de não conformidades possível.



# Objetivo

Este trabalho tem como objetivo **comparar** três ferramentas analisadoras com código aberto recomendadas pela OWASP: Cppcheck, FlawFinder e LGTM; de acordo com sua **detecção de não conformidades** com as regras de nível 1 do padrão CERT C 2016.





# Cppcheck

- Objetivo de minimizar falsos positivos.
- Disponível para download, é executada sobre um diretório local.
- Aponta falhas de segurança e problemas de estilo de código.





# FlawFinder

- Reporta possíveis fragilidades de segurança ordenadas por um nível de vulnerabilidade.
- Disponível para download, é executada sobre um diretório local.
- Reconhecida pela CWE e *C/C++ Best Practices*.





# LGTM

- Analisa a semântica do código com métodos da ciência de dados.
- Procura por variantes no histórico do projeto.
- Serviço que se conecta como uma aplicação do GitHub.
- Suporta várias linguagens.





# Regras

As regras impostas pelo padrão CERT C são definidas com os elementos:

- Descrição da prática e explicação das consequências
- Exemplo(s) de código(s) não conforme(s)
- Versão(ões) conforme(s) do(s) mesmo(s)

## Método de aplicação

Para cada regra, escolheu-se arbitrariamente um exemplo não conforme para ser analisado.

O conjunto de exemplos não conformes foi agrupado em um diretório ou projeto, e enviado à ferramenta de análise.

```
projeto
├── ARR38-C.c
├── ENV32-C.c
├── ENV33-C.c
├── ERR33-C.c
├── EXP33-C.c
├── EXP34-C.c
├── FI030-C.c
├── FI034-C.c
├── FI037-C.c
├── MEM30-C.c
├── MEM34-C.c
├── MSC32-C.c
├── MSC33-C.c
└── SIG30-C.c
```

## Exemplo: Regra STR31-C (1/2)

```
1 #include <stddef.h>
2 void copy(size_t n, char src[n], char dest[n])
3 {
4     size_t i;
5     for (i = 0; src[i] && (i < n); ++i)
6     {
7         dest[i] = src[i];
8     }
9     dest[i] = '\0';
10 }
11 int main(){}

```

Figure: Código não conforme para a regra STR31-C



## Exemplo: Regra STR31-C (2/2)

### Alerta FlawFinder:

STR31-C.c:2: [2] (buffer) char: Statically-sized arrays can be improperly restricted, leading to potential overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.

### Alerta Cppcheck:

[STR31-C.c:5]: (style) Array index 'i' is used before limits check.

## Exemplo: Regra FIO30-C (1/2)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 void incorrect_password(const char *user){
5     int ret;
6     static const char msg_format[] =
7         "%s cannot be authenticated.\n";
8     size_t len = strlen(user) + sizeof(msg_format);
9     char *msg = (char *)malloc(len);
10    ret = snprintf(msg, len, msg_format, user);
11    fprintf(stderr, msg);
12    free(msg);
13 }
```

Figure: Código não conforme para a regra FIO30-C





## Exemplo: Regra FIO30-C (2/2)

### Alerta FlawFinder:

FIO30-C.c:24: [4] (format) fprintf: If format strings can be influenced by an attacker, they can be exploited (CWE-134). Use a constant for the format specification.



## Não conformidades detectadas por ferramenta (1/2)

Regra	Detecção (S/N)		
	Cppcheck	FlawFinder	LGTM
EXP33-C	N	N	N
EXP34-C	N	N	N
ARR38-C	N	S	N
STR31-C	S	S	N
STR32-C	N	S	N
STR38-C	N	S	N
MEM30-C	N	N	N
MEM34-C	N	N	N
FIO30-C	N	S	N



## Não conformidades detectadas por ferramenta (2/2)

Regra	Detecção (S/N)		
	Cppcheck	FlawFinder	LGTM
FIO34-C	N	N	N
FIO37-C	N	S	N
ENV32-C	N	N	N
ENV33-C	N	S	N
SIG30-C	N	N	N
ERR33-C	N	N	N
MSC32-C	N	N	N
MSC33-C	S	N	S



## Matriz de confusão: Cppcheck

	Detecção de falha	Não detecção de falha
Falha correspondente	2	15
Restante do código	0	17

Table: Matriz de confusão para detecção com Cppcheck



## Matriz de confusão: FlawFinder

	Detecção de falha	Não detecção de falha
Falha correspondente	7	10
Restante do código	4	13

Table: Matriz de confusão para detecção com FlawFinder



## Matriz de confusão: LGTM

	Detecção de falha	Não detecção de falha
Falha correspondente	1	14
Restante do código	0	17

Table: Matriz de confusão para detecção com LGTM

## Considerações

Verdadeiros positivos e falsos negativos são mais relevantes devido ao critério do padrão CERT C.

A ferramenta FlawFinder se mostrou mais adequada para análise de código C/C++ com objetivo de mitigar vulnerabilidades abordadas nas regras de nível 1 do padrão CERT C 2016.



## Considerações

Verdadeiros positivos e falsos negativos são mais relevantes devido ao critério do padrão CERT C.

A ferramenta FlawFinder se mostrou mais adequada para análise de código C/C++ com objetivo de mitigar vulnerabilidades abordadas nas regras de nível 1 do padrão CERT C 2016.

### Limitações

- Foram considerados apenas os códigos não conformes.
- As análises envolveram apenas um dos exemplos de não conformidade.





# Fim :D

André Eduardo Pacheco Dias  
Gabriela Moreira Mafra  
Lucas Schmitt Seidel