André Eduardo Pacheco Dias, Gabriela Moreira Mafra e Lucas Schmitt Seidel

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

Joinville

2019

UNIVERSIDADE DO ESTADO DE SANTA CATARINA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

André Eduardo Pacheco Dias, Gabriela Moreira Mafra e Lucas Schmitt Seidel

ESTUDO COMPARATIVO ENTRE 3 ANALISADORES DE CÓDIGO C SOBRE AS AMEAÇAS DE NÍVEL 1 DO PADRÃO CERT C 2016

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Charles Christian Miers
Orientador

Joinville, Março de 2019

ESTUDO COMPARATIVO ENTRE 3 ANALISADORES DE CÓDIGO C SOBRE AS AMEAÇAS DE NÍVEL 1 DO PADRÃO CERT C 2016

André Eduardo Pacheco Dias, Gabriela Moreira Mafra e Lucas Schmitt Seidel

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação Integral do CCT/UDESC.

Banca Examinadora

Charles Christian Miers - Doutor (orientador)

Resumo

Esse trabalho compara quantitativamente a capacidade de detecção do descumprimento das regras de prioridade 27, 18 e 12 (Nível 1) do padrão de código CERT C de 2016 pelas ferramentas de código aberto: Cppcheck, FlawFinder e LGTM. Ao fim, é determinada a ferramenta de código aberto mais conforme à capacidade de impôr as diretrizes do padrão de código, levando em conta às regras de maior prioridade.

Palavras-chaves: Padrão de Código, Programação Segura, Ferramenta de código livre, Analisador de Código, CERT C

Abstract

This work makes a quantitative comparisson of the capacity for noncompliance detection of rules with priority 27, 18 and 12 (Level 1) from CERT C coding standard from 2016 by open source tools: Cppcheck, FlawFinder and LGTM. At the end, the open source tool that is most compliant with the coding standard directives is determined, considering the rules with most priority.

Keywords: Coding Standard, Secure Programming, Open Source Tool, Code Analyzer, CERT C

Contents

List of Figures

List of Tables

1 Introdução

1.1 Objetivos

1.1.1 Objetivos Específicos

- TO DO
- TO DO

2 Conceitos

2.1 Definição

Segundo a RedHat (REDHAT DEVELOPER, 2019), programação segura é um conjunto de tecnologias e boas práticas para tornar software tão seguro e estável quanto possível. Ela engloba conceitos desde criptografia, certificados e identidade federada até recomendações para movimentação de dados sensíveis, acesso a sistemas de arquivos e gerenciamento de memória.

Para definir o que faz um programa seguro, existem padrões de programação segura. Esses padrões estabelecem regras bem definidas para separar código conforme de não conforme. Para isso, os padrões são direcionados a uma linguagem de programação específica, e tratam de potenciais vulnerabilidades que emergem daquela linguagem.

2.2 Histórico

A ideia de um padrão CERT para programação segura surgiu nem um encontro do comitê de padrões C, em 2006 (SNAVELY, 2016). O padrão C é um documento oficial, porém mais direcionado a desenvolvedores de compiladores e é considerado obscuro pela comunidade de desenvolvedores mais geral. Um padrão de código seguro seria direcionado primariamente a programadores da linguagem C, guidando-os a programar de forma segura na linguagem.

Com essa ideia, foi criada uma *wiki* onde membros da comunidade e do próprio comitê contribuíram para, ao fim de dois anos e meio, a publicação do primeiro padrão de código seguro CERT C na forma de um livro em 2008. A wiki continuou em desenvolvimento, e gerou uma nova publicação em 2014. A última versão foi gerada em 2016, e está disponível em formato PDF.

2.3 Padrão CERT C

O Padrão SEI CERT C, edição 2016 (CARNEIGE MELLON UNIVERSITY, 2016), determina regras para programação segura na linguagem de programação C, contendo título, descrição, exemplos de não conformidade de a solução conforme. As regras são propostas com o objetivo de desenvolver sistemas seguros e confiáveis, como por meio da eliminação de comportamentos que podem levar a indefinições do programa, gerando vulnerabilidades exploráveis.

As regras determinadas por esse padrão são estabelecidas necessárias para um software que busca confiança e segurança, porém não são suficientes para tornar um programa confiável e seguro.

2.4 Analisadores de código

Ferramentas de análise de código fonte ou Testes Estáticos de Segurança da Aplicação (SAST - *Static Application Security Testing*) tem como objetivo analisar código fonte ou suas versões compiladas para encontrar falhas de segurança (OWASP, 2019). É interessante, para o ciclo de desenvolvimento de software, que essas análises possam ser feitas com alta frequência, diminuindo o tempo de feedback e detectando os problemas o quanto antes.

Os analisadores tendem a ser escaláveis, funcionando bem para códigos com muitas linhas, e a trazer informações completas para o programador, como o número da linha do código onde o problema acontece. São úteis para encontrar alguns problemas específicos com alto grau de confiança, como para erros de *Buffer Overflow*.

Muitos problemas de segurança, contudo, não podem ser encontrados de maneira automática e, portanto, não são detectáveis com esse tipo de ferramenta. Os analisadores disponíveis no momento da escrita desse trabalho são capazes de encontrar apenas uma minoria das falhas de segurança em aplicações no geral. As ferramentas ainda tendem a apresentar numerosos falsos positivos.

Apesar das análises feitas por essas ferramentas poderem ser feitas pelo próprio programador, manualmente, a automação do processo permite uma frequência maior de verificações e tende a ser menos suscetível a erros conforme o aumento da frequência de

análises.

2.5 Ferramentas escolhidas

2.5.1 Cppcheck

A ferramenta de análise de código C/C++ Cppcheck (MARJAMäKI, 2010) busca detectar problemas não identificados por compiladores. Sendo assim, ela não detecta problemas de sintaxe, buscando alertar potenciais vazamentos de memória, alocações sem respectivas desalocações e vice-versa, *buffer overrun*, e outros problemas dessa família.

O objetivo da ferramenta é anular todos os falsos positivos. Sendo assim, há vários problemas que deixam de ser detectados - mas aqueles que são tem uma probabilidade alta de serem problemas reais. Esse objetivo ainda não foi atingido e a ferramenta está em estado de desenvolvimento.

2.5.2 FlawFinder

O FlawFinder (WHEELER, 2010) é uma ferramenta que examina código C/C++ e reporta possíveis fragilidades de segurança ordenadas por um nível de vulnerabilidade. O objetivo é permitir uma checagem rápida por potenciais problemas de segurança. Algumas das vantagens apontadas pra essa ferramenta incluem a presença de um relatório amigável e detalhado, assim como a facilidade de instalação e uso.

2.5.3 LGTM

O LGTM (MOOR, 2019) é uma plataforma de análise de variantes que checa código em busca de vulnerabilidades. Ela combina uma busca profunda na semântica do código com informações encontradas com ciência de dados para relacionar os resultados mais importantes e mostrar alertas relevantes.

O conceito por trás do funcionamento do LGTM consiste na observação de que os mesmos problemas aparecem repetidas vezes no ciclo de vida de um software e na base de código. Essas repetições podem estar apresentadas em diferentes formas, chamadas variantes. Nesse sentido, sempre que um problema é encontrado, são buscadas variantes

dele e vulnerabilidades semelhantes.

2.6 Método de aplicação da ferramenta

Para realizar os testes de detecção se falhas em códigos não conformes, o seguinte procedimento é adotado:

1. Construção de um código funcional mínimo que inclua o exemplo de não conformidade provido na especificação da regra. O código deve incluir o mínimo de estrutura para ser compilado com sucesso.

2. Execução de cada ferramenta, tendo como parâmetro o arquivo do código contendo a não conformidade.

3. Levantamento de dados sobre o relatório gerado por cada ferramenta. Esse processo deve levar em conta os quatro quadrantes de uma matriz de confusão simples, isso é: quantidade de falsos positivos, falsos negativos, verdadeiros positivos e verdadeiros negativos. Para isso, é considerada que a única vulnerabilidade presente é aquela sendo avaliada.

2.7 Tipos de regras

O padrão CERT C faz avaliações de risco e custo de remediação para cada guia associado a uma regra e recomendação. Com o uso dessas informações, é definida uma prioridade para cada regra, o que pode ser usado como classificação em análises como a feita neste trabalho.

Em (CARNEIGE MELLON UNIVERSITY, 2016), a prioridade é definida com base em três outras avaliações conforme as definições: severidade - quão sérias são as consequências dá regra ser ignorada, conforme a Tabela 2.1; chance - quão provável é que uma falha introduzida por ignorar uma regra leve a uma vulnerabilidade explorável, conforme a Tabela 2.2; e custo de remediação - quão custoso é ficar conforme com a regra, conforme a tabela 2.3.

Os valores dessas três avaliações são múltiplicados para obter a prioridade de

Table 2.1: Severidade de uma regra

Valor	Significado	Exemplos de Vulnerabilidades
1	Baixo	Ataque de negação de serviço
2	Médio	Violação da integridade dos dados
3	Alto	Rodar um código arbitrário

Table 2.2: Chance de uma regra

Valor	Significado
1	Pouco provável
2	Provável
3	Muito provável

Table 2.3: Custo de remediação de uma regra

Valor	Significado	Detecção	Correção
1	Alto	Manual	Manual
2	Médio	Automático	Manual
3	Alto	Automático	Automático

uma regra. A Tabela 2.4 traz a classificação das possíveis prioridades em três níveis com possíveis interpretações para seus significados.

Table 2.4: Níveis de prioridades

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

O comparativo neste trabalho trata estritamente das regras classificadas como L1, ou seja, de prioridade 12, 18 ou 27.

3 Regras

3.1 EXP33-C

3.1.1 Descrição

3.1.2 Código não conforme analisado

3.2 EXP34-C

3.2.1 Descrição

3.2.2 Código não conforme analisado

3.3 ARR38-C

3.3.1 Descrição

3.3.2 Código não conforme analisado

3.4 STR31-C

3.4.1 Descrição

3.4.2 Código não conforme analisado

3.5 STR32-C

3.5.1 Descrição

3.5.2 Código não conforme analisado

3.6 STR38-C

4 Comparativo

4.1 Analisadores de código

4.1.1 Cppcheck

4.1.2 FlawFinder

4.1.3 LGTM

4.2 Método de aplicação da ferramenta

4.3 Resultados

Referências

CARNEIGE MELLON UNIVERSITY. *C Coding Standard*. 2016. Disponível em: <https://resources.sei.cmu.edu/downloads/secure-coding/assets/sei-cert-c-coding-standard-2016-v01.pdf>. Acesso em: 28 mar. 2019.

MARJAMäKI, D. *Cppcheck Design*. 2010. Disponível em: <https://sourceforge.net/projects/cppcheck/files/Articles/cppcheck-design-2010.pdf/download>. Acesso em: 28 mar. 2019.

MOOR, O. de. *Open results on LGTM: The key to securing open source*. 2019. Disponível em: <https://lgtm.com/blog/open_results_on_lgtm>. Acesso em: 05 jun. 2019.

OWASP. *Source Code Analysis Tools*. 2019. Disponível em: <https://www.owasp.org/index.php/Source_Code_Analysis_Tools>. Acesso em: 05 jun. 2019.

REDHAT DEVELOPER. *Secure Coding*. 2019. Disponível em: <https://developers.redhat.com/topics/secure-coding/>. Acesso em: 05 jun. 2019.

SNAVELY, W. *SEI CERT C Coding Standard History*. 2016. Disponível em: <https://wiki.sei.cmu.edu/confluence/display/c/History>. Acesso em: 05 jun. 2019.

WHEELER, D. A. *Flawfinder*. 2010. Disponível em: <https://dwheeler.com/flawfinder/>. Acesso em: 28 mar. 2019.