Corso di Laurea Magistrale in Informatica

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

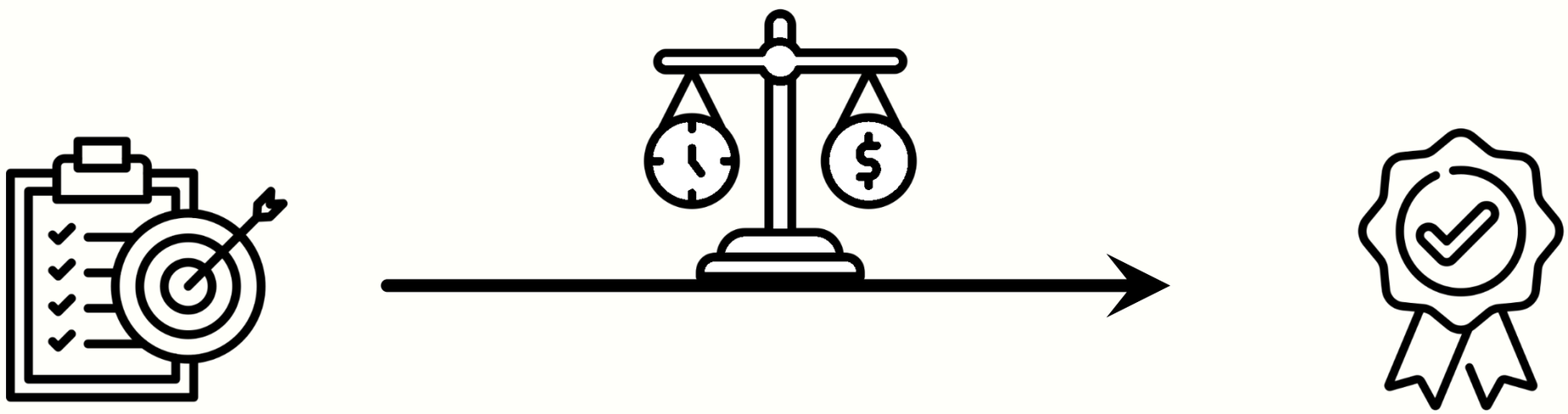
Prof. Fabio Palomba
Dott. Emanuele Iannone

Grazia Varone
Mat.: 0522501064

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

PDF tesi —>

Introduzione e Background

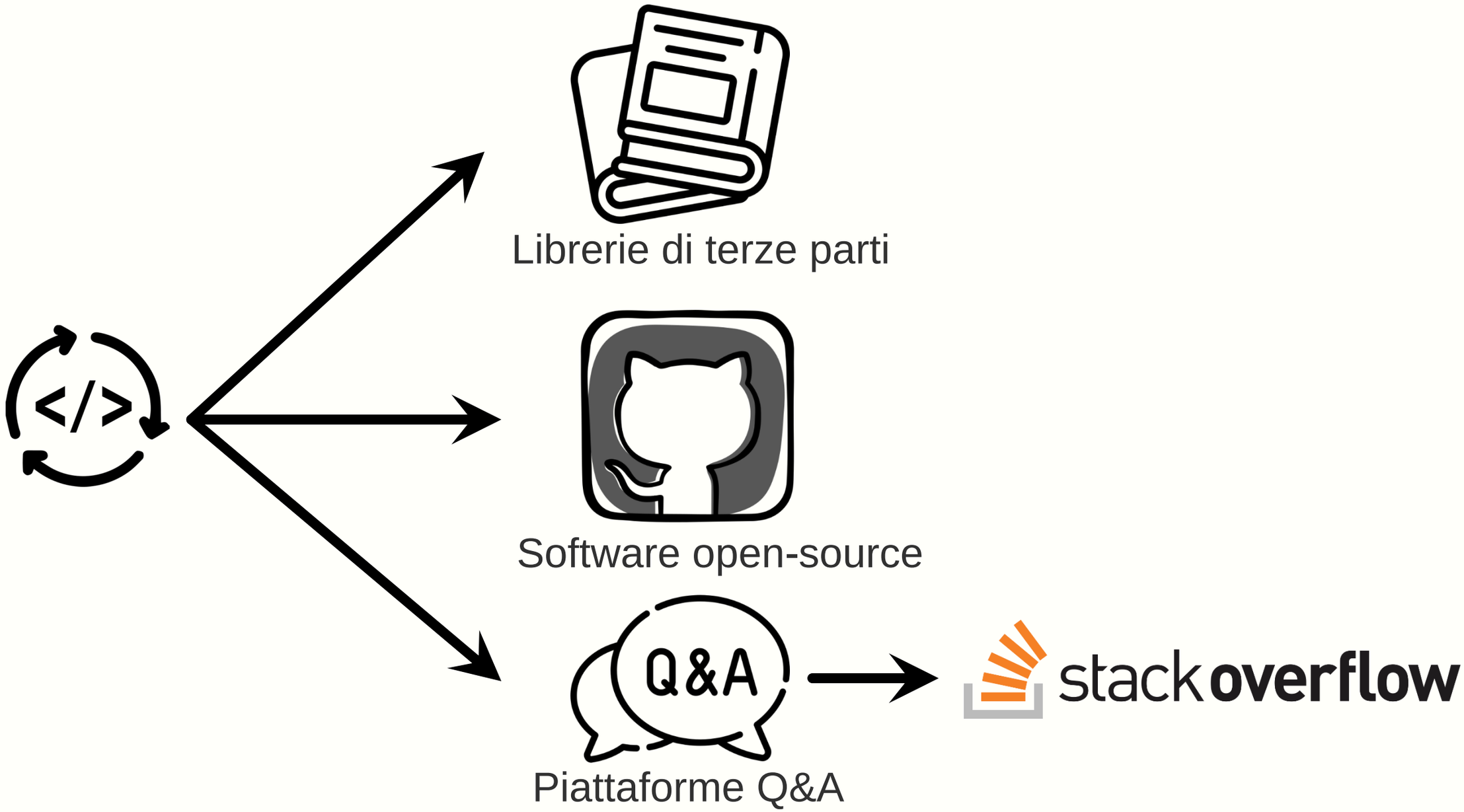


Sviluppo software

Software di alta qualità

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Introduzione e Background



g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Introduzione e Background

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Introduzione e Background

How do I execute a command and get the output of the command within C++ using POSIX?

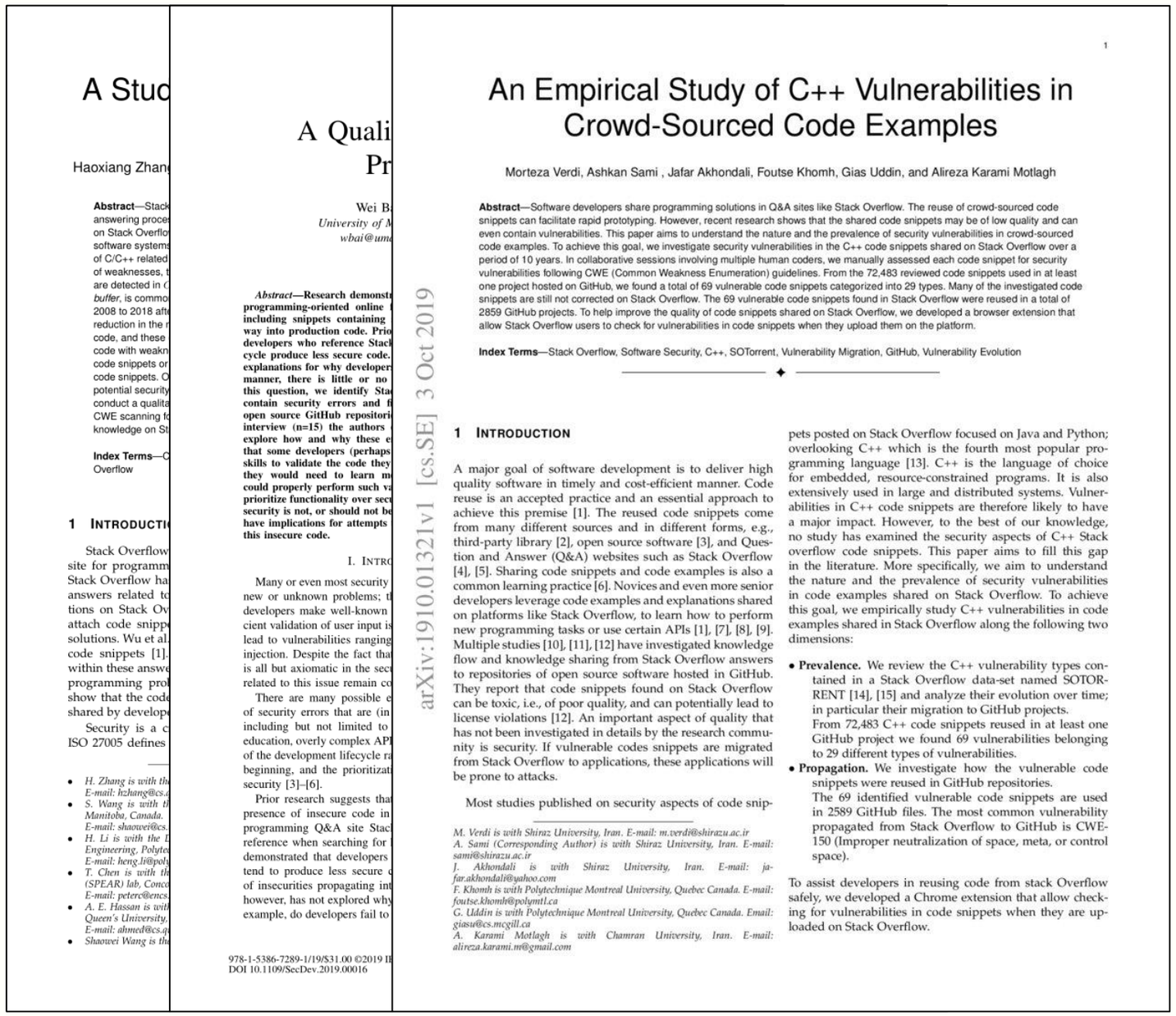
763

```cpp
#include <cstdio>
#include <iostream>
#include <memory>
#include <stdexcept>
#include <string>
#include <array>

std::string exec(const char* cmd) {
    std::array<char, 128> buffer;
    std::string result;
    std::unique_ptr<FILE, decltype(&pclose)> pipe(popen(cmd, "r"), pclose);
    if (!pipe) {
        throw std::runtime_error("popen() failed!");
    }
    while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr) {
        result += buffer.data();
    }
    return result;
}
```

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Introduzione e Background



An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples

Morteza Verdi, Ashkan Sami , Jafar Akhondali, Foutse Khomh, Gias Uddin, and Alireza Karami Motlagh

Abstract—Software developers share programming solutions in Q&A sites like Stack Overflow. The reuse of crowd-sourced code snippets can facilitate rapid prototyping. However, recent research shows that the shared code snippets may be of low quality and can even contain vulnerabilities. This paper aims to understand the nature and the prevalence of security vulnerabilities in crowd-sourced code examples. To achieve this goal, we investigate security vulnerabilities in the C++ code snippets shared on Stack Overflow over a period of 10 years. In collaborative sessions involving multiple human coders, we manually assessed each code snippet for security vulnerabilities following CWE (Common Weakness Enumeration) guidelines. From the 72,483 reviewed code snippets used in at least one project hosted on GitHub, we found a total of 69 vulnerable code snippets categorized into 29 types. Many of the investigated code snippets are still not corrected on Stack Overflow. The 69 vulnerable code snippets found in Stack Overflow were reused in a total of 2859 GitHub projects. To help improve the quality of code snippets shared on Stack Overflow, we developed a browser extension that allow Stack Overflow users to check for vulnerabilities in code snippets when they upload them on the platform.

Index Terms—Stack Overflow, Software Security, C++, SOTorrent, Vulnerability Migration, GitHub, Vulnerability Evolution

1 Introduction

A major goal of software development is to deliver high quality software in timely and cost-efficient manner. Code reuse is an accepted practice and an essential approach to achieve this premise [1]. The reused code snippets come from many different sources and in different forms, e.g., third-party library [2], open source software [3], and Question and Answer (Q&A) websites such as Stack Overflow [4], [5]. Sharing code snippets and code examples is also a common learning practice [6]. Novices and even more senior developers leverage code examples and explanations shared on platforms like Stack Overflow, to learn how to perform new programming tasks or use certain APIs [1], [7], [8], [9]. Multiple studies [10], [11], [12] have investigated knowledge flow and knowledge sharing from Stack Overflow answers to repositories of open source software hosted in GitHub. They report that code snippets found on Stack Overflow can be toxic, i.e., of poor quality, and can potentially lead to license violations [12]. An important aspect of quality that has not been investigated in details by the research community is security. If vulnerable codes snippets are migrated from Stack Overflow to applications, these applications will be prone to attacks.

Most studies published on security aspects of code snippets posted on Stack Overflow focused on Java and Python; overlooking C++ which is the fourth most popular programming language [13]. C++ is the language of choice for embedded, resource-constrained programs. It is also extensively used in large and distributed systems. Vulnerabilities in C++ code snippets are therefore likely to have a major impact. However, to the best of our knowledge, no study has examined the security aspects of C++ Stack overflow code snippets. This paper aims to fill this gap in the literature. More specifically, we aim to understand the nature and the prevalence of security vulnerabilities in code examples shared on Stack Overflow. To achieve this goal, we empirically study C++ vulnerabilities in code examples shared in Stack Overflow along the following two dimensions:

- **Prevalence.** We review the C++ vulnerability types contained in a Stack Overflow data-set named SOTORRENT [14], [15] and analyze their evolution over time; in particular their migration to GitHub projects. From 72,483 C++ code snippets reused in at least one GitHub project we found 69 vulnerabilities belonging to 29 different types of vulnerabilities.
- **Propagation.** We investigate how the vulnerable code snippets were reused in GitHub repositories. The 69 identified vulnerable code snippets are used in 2589 GitHub files. The most common vulnerability propagated from Stack Overflow to GitHub is CWE-150 (Improper neutralization of space, meta, or control space).

To assist developers in reusing code from stack Overflow safely, we developed a Chrome extension that allow checking for vulnerabilities in code snippets when they are uploaded on Stack Overflow.

M. Verdi is with Shiraz University, Iran. E-mail: m.verdi@shirazu.ac.ir
A. Sami (Corresponding Author) is with Shiraz University, Iran. E-mail: sami@shirazu.ac.ir
J. Akhondali is with Shiraz University, Iran. E-mail: jafar.akhondali@yahoo.com
F. Khomh is with Polytechnique Montreal University, Quebec Canada. E-mail: foutse.khomh@polymtl.ca
G. Uddin is with Polytechnique Montreal University, Quebec Canada. Email: giasu@cs.mcgill.ca
A. Karami Motlagh is with Chamran University, Iran. E-mail: alireza.karami.m@gmail.com

978-1-5386-7289-1/19/$31.00 ©2019 IE
DOI 10.1109/SecDev.2019.00016

- Analisi statica ☑
- Machine learning ☐
- Propagazione di codici vulnerabili da Stack Overflow a GitHub ☑
- Evoluzione dei frammenti di codice insicuri copiati da Stack Overflow ☐

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Metodologia di ricerca

RQ_1 Qual è l'approccio migliore per rilevare i frammenti di codice insicuri su Stack Overflow?

RQ_2 Qual è la tecnica più adatta per il rilevamento di frammenti di codice copiati e incollati da Stack Overflow?

RQ_3 Come evolvono i frammenti di codice C/C++ copiati da Stack Overflow nei progetti GitHub?

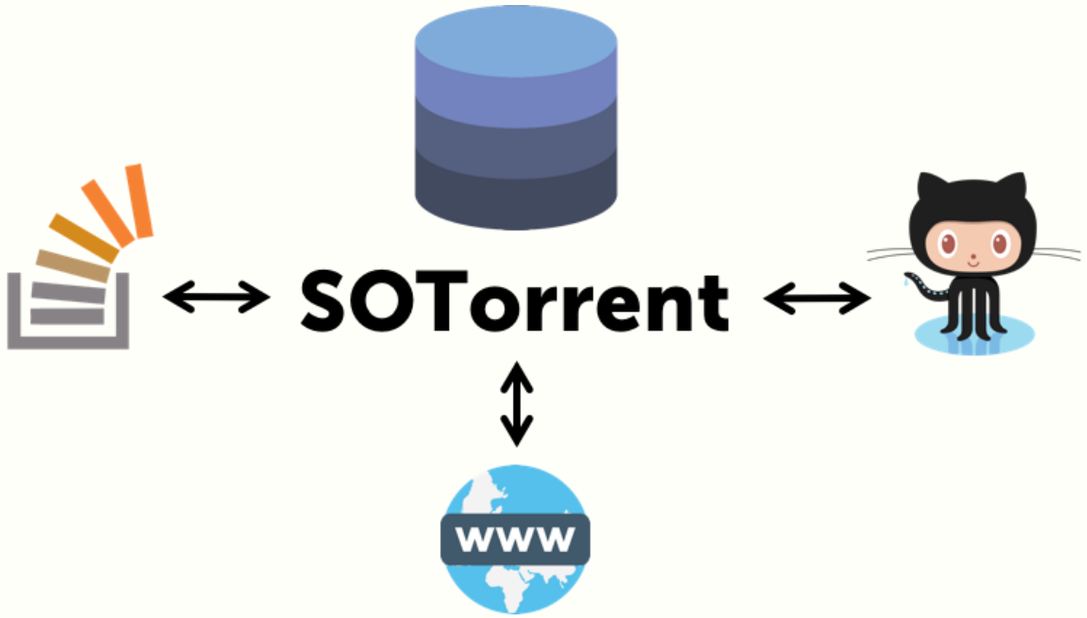
g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Contesto

452,671 Stack Overflow code snippets C/C++ collezionati dal dataset SOTorrent, nel periodo dal 2015 al 2020.

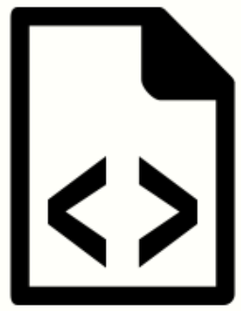


g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

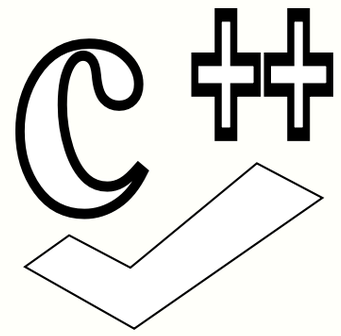
Metodologia di ricerca (RQ_1)



Stack Overflow code snippets

Codice contiene almeno una delle keywords di interesse

privilege
denial-of-service,
out-of-bounds,
authentication,
[…]*



Detection di vulnerabilità tramite analisi statica

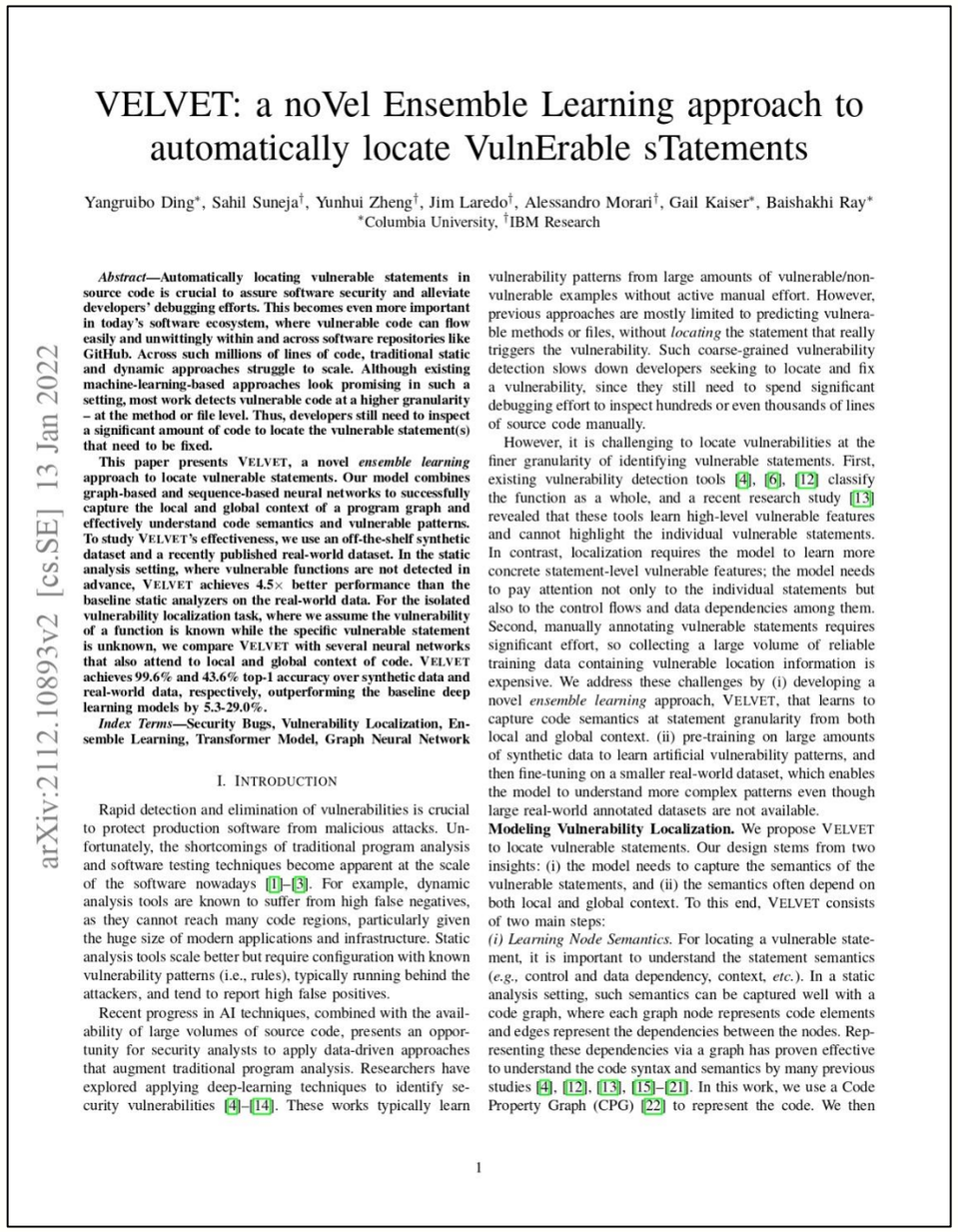
* H. Hong, S.Woo, and H. Lee, “Dicos: Discovering insecure code snippets from stack overflow posts by leveraging user discussions” in ACSAC ’21: Annual Computer Security Applications Conference, pp. 194–206, 2021.

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Metodologia di ricerca (RQ_1)



arXiv:2112.10893v2 [cs.SE] 13 Jan 2022

VELVET: a noVel Ensemble Learning approach to automatically locate VulnErable sTatements

Yangruibo Ding*, Sahil Suneja†, Yunhui Zheng†, Jim Laredo†, Alessandro Morari†, Gail Kaiser*, Baishakhi Ray*
*Columbia University, †IBM Research

***Abstract*—Automatically locating vulnerable statements in source code is crucial to assure software security and alleviate developers' debugging efforts. This becomes even more important in today's software ecosystem, where vulnerable code can flow easily and unwittingly within and across software repositories like GitHub. Across such millions of lines of code, traditional static and dynamic approaches struggle to scale. Although existing machine-learning-based approaches look promising in such a setting, most work detects vulnerable code at a higher granularity – at the method or file level. Thus, developers still need to inspect a significant amount of code to locate the vulnerable statement(s) that need to be fixed.**

This paper presents VELVET, a novel *ensemble learning* approach to locate vulnerable statements. Our model combines graph-based and sequence-based neural networks to successfully capture the local and global context of a program graph and effectively understand code semantics and vulnerable patterns. To study VELVET's effectiveness, we use an off-the-shelf synthetic dataset and a recently published real-world dataset. In the static analysis setting, where vulnerable functions are not detected in advance, VELVET achieves 4.5× better performance than the baseline static analyzers on the real-world data. For the isolated vulnerability localization task, where we assume the vulnerability of a function is known while the specific vulnerable statement is unknown, we compare VELVET with several neural networks that also attend to local and global context of code. VELVET achieves 99.6% and 43.6% top-1 accuracy over synthetic data and real-world data, respectively, outperforming the baseline deep learning models by 5.3-29.0%.

***Index Terms*—Security Bugs, Vulnerability Localization, Ensemble Learning, Transformer Model, Graph Neural Network**

I. Introduction

Rapid detection and elimination of vulnerabilities is crucial to protect production software from malicious attacks. Unfortunately, the shortcomings of traditional program analysis and software testing techniques become apparent at the scale of the software nowadays [1]–[3]. For example, dynamic analysis tools are known to suffer from high false negatives, as they cannot reach many code regions, particularly given the huge size of modern applications and infrastructure. Static analysis tools scale better but require configuration with known vulnerability patterns (i.e., rules), typically running behind the attackers, and tend to report high false positives.

Recent progress in AI techniques, combined with the availability of large volumes of source code, presents an opportunity for security analysts to apply data-driven approaches that augment traditional program analysis. Researchers have explored applying deep-learning techniques to identify security vulnerabilities [4]–[14]. These works typically learn vulnerability patterns from large amounts of vulnerable/non-vulnerable examples without active manual effort. However, previous approaches are mostly limited to predicting vulnerable methods or files, without *locating* the statement that really triggers the vulnerability. Such coarse-grained vulnerability detection slows down developers seeking to locate and fix a vulnerability, since they still need to spend significant debugging effort to inspect hundreds or even thousands of lines of source code manually.

However, it is challenging to locate vulnerabilities at the finer granularity of identifying vulnerable statements. First, existing vulnerability detection tools [4], [6], [12] classify the function as a whole, and a recent research study [13] revealed that these tools learn high-level vulnerable features and cannot highlight the individual vulnerable statements. In contrast, localization requires the model to learn more concrete statement-level vulnerable features; the model needs to pay attention not only to the individual statements but also to the control flows and data dependencies among them. Second, manually annotating vulnerable statements requires significant effort, so collecting a large volume of reliable training data containing vulnerable location information is expensive. We address these challenges by (i) developing a novel *ensemble learning* approach, VELVET, that learns to capture code semantics at statement granularity from both local and global context. (ii) pre-training on large amounts of synthetic data to learn artificial vulnerability patterns, and then fine-tuning on a smaller real-world dataset, which enables the model to understand more complex patterns even though large real-world annotated datasets are not available.

Modeling Vulnerability Localization. We propose VELVET to locate vulnerable statements. Our design stems from two insights: (i) the model needs to capture the semantics of the vulnerable statements, and (ii) the semantics often depend on both local and global context. To this end, VELVET consists of two main steps:

(i) Learning Node Semantics. For locating a vulnerable statement, it is important to understand the statement semantics (*e.g.,* control and data dependency, context, *etc.*). In a static analysis setting, such semantics can be captured well with a code graph, where each graph node represents code elements and edges represent the dependencies between the nodes. Representing these dependencies via a graph has proven effective to understand the code syntax and semantics by many previous studies [4], [12], [13], [15]–[21]. In this work, we use a Code Property Graph (CPG) [22] to represent the code. We then

Replica framework VELVET

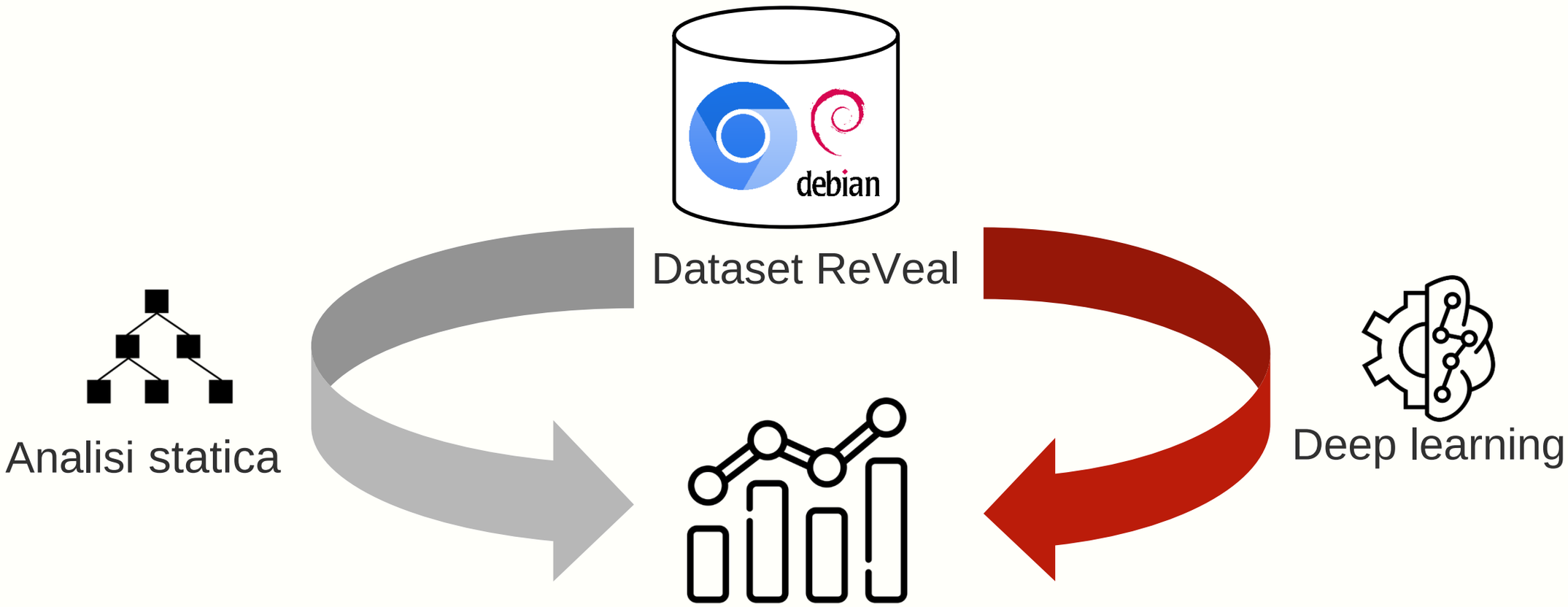
Rileva le istruzioni vulnerabili nel codice sorgente tramite ensemble learning (Gated Graph Neural Network e Transformer)

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Metodologia di ricerca (RQ_1)



g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Risultati (RQ_1)

	Precision	Recall	F1 Score	Label
Analisi statica	100%	65%	79%	Non vulnerabile
	100%	57%	73%	Vulnerabile
Ensemble learning	100%	100%	100%	Non vulnerabile
	0%	0%	0%	Vulnerabile
Transformer	100%	68%	81%	Non vulnerabile
	100%	15%	26%	Vulnerabile

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Risultati (RQ_1)

	Precision	Recall	F1 Score	Label
Analisi statica	100%	65%	79%	Non vulnerabile
Ensemble learning	100%	100%	100%	Non vulnerabile
Transformer	100%	68%	81%	Non vulnerabile
	100%	15%	26%	Vulnerabile

RQ_1

Qual è l'approccio migliore per rilevare i frammenti di codice insicuri su Stack Overflow?

Sintesi dei risultati

L'approccio migliore per rilevare frammenti di codice insicuri è l'analisi statica.

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Metodologia di ricerca (RQ_2)

Riuso di code snippets vulnerabili
in **133** repositories GitHub
(progetti C/C++, #releases >= **10**, #stars >= **10,000**)

Dataset SOTorrent

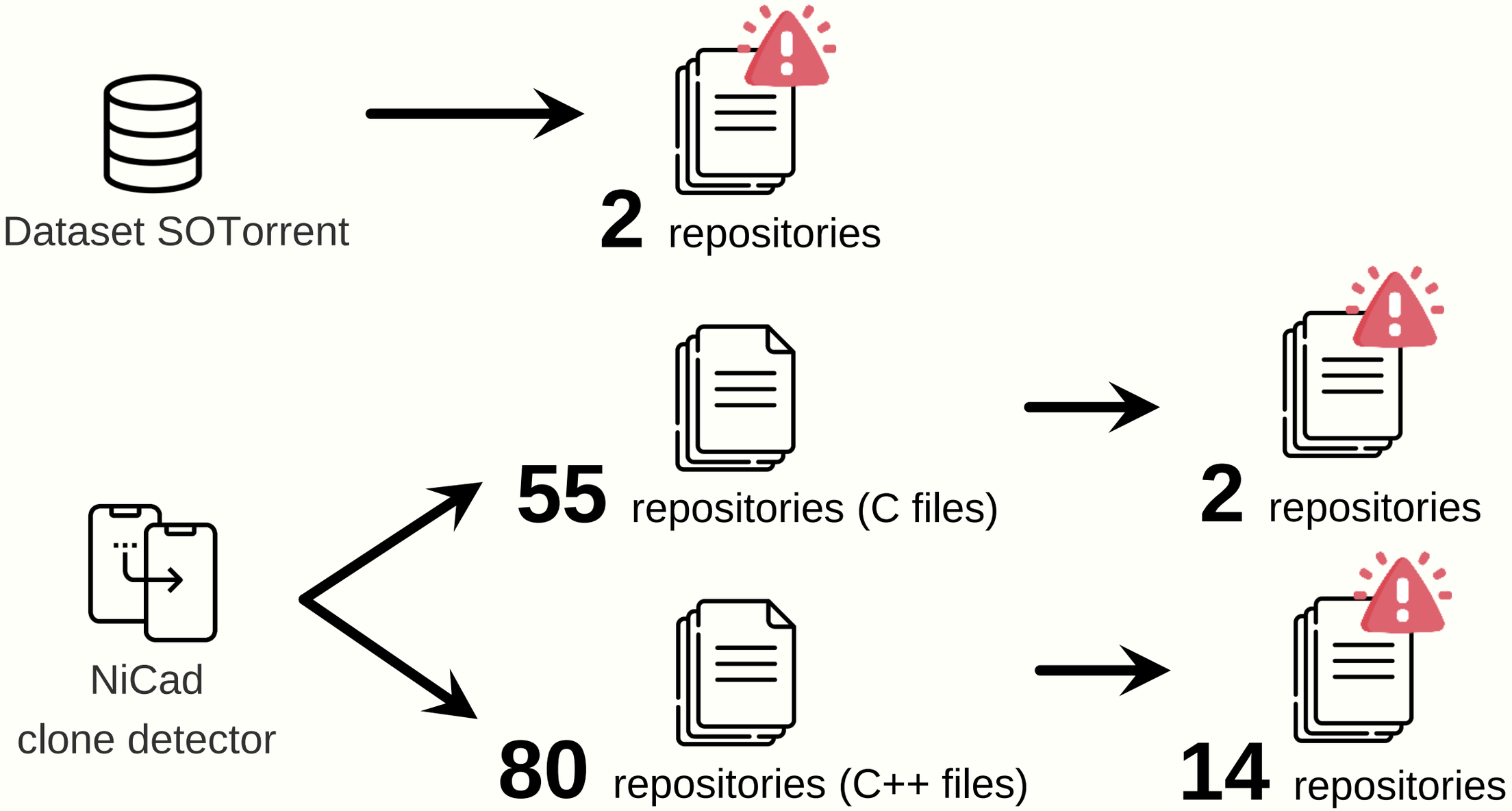
NiCad clone detector

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Risultati (RQ_2)

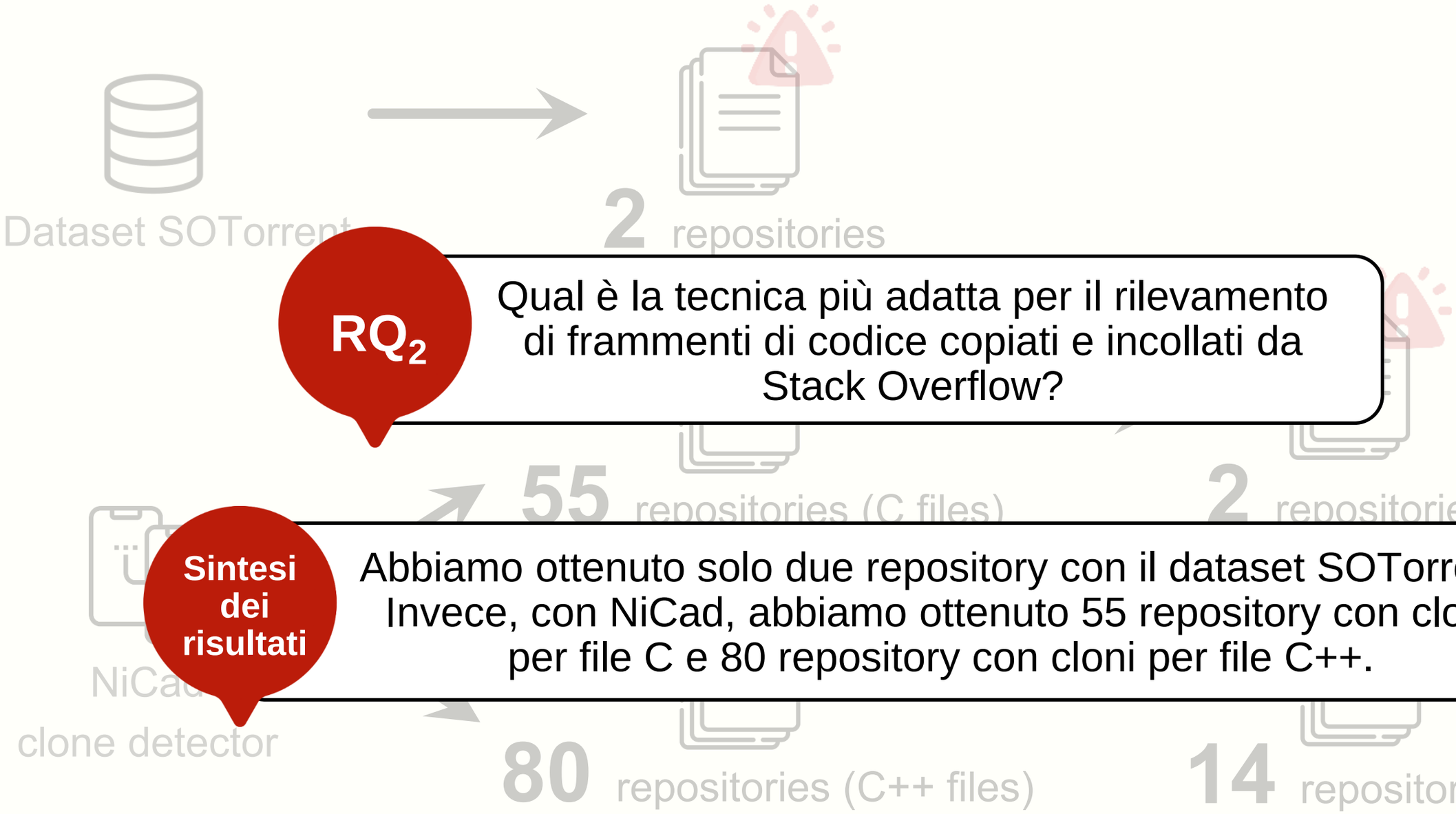


g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Risultati (RQ_2)



RQ_2

Qual è la tecnica più adatta per il rilevamento di frammenti di codice copiati e incollati da Stack Overflow?

Sintesi dei risultati

Abbiamo ottenuto solo due repository con il dataset SOTorrent. Invece, con NiCad, abbiamo ottenuto 55 repository con cloni per file C e 80 repository con cloni per file C++.

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Metodologia di ricerca (RQ_3)

```
1. // Code
2. // Code
3.   initPlayback($elem, $wrapper, moveTemplate, opts.autoPlay, opts.still)
4.   updateNavi($navi, 0);
5.   updateButtons($elem, 0, amount);
6. if(opts.resize) updateHeight($slideContainer, $slides, pos);
7. + disableSelection($elem)
8.   function moveTemplate(indexCb, animCb) {
9.   ....
10. - return function () {
11.  ...                    1.
12.  ...
13.  ..
14.  ..
15. + function disableSelection($e) {
16. +       // http://stackoverflow.com/questions/2700000/how-to-disable-text-selection-using-jquery
17. +       return $e.each(function() {
18. +           $(this).attr('unselectable', 'on').css({
19. +               '-moz-user-select':'none',
20. +               '-webkit-user-select':'none',
21. +               'user-select':'none'
22. +           }).each(function() {
23. +               this.onselectstart = function() { return false; };
24. +           });
25. +       });
26. +   };
27. +
```

```
1. // Code
2. // Code
3.   initPlayback($elem, $wrapper, moveTemplate, opts.autoPlay, opts.still)
4.   updateNavi($navi, 0);
5.   updateButtons($elem, 0, amount);
6. - disableSelection($elem)
7. + if(opts.resize) {
8. +     disableSelection($elem)
9. + }
10.  function moveTemplate(indexCb, animCb) {
11.  ....
12.  ..
13.  ..
14.  function disableSelection($e) {
15.      // http://stackoverflow.com/questions/2700000/how-to-disable-text-selection-using-jquery
16.      return $e.each(function() {
17.          $(this).attr('unselectable', 'on').css({
18.              '-moz-user-select':'none',
19.              '-webkit-user-select':'none',
20.              'user-select':'none'
21.          }).each(function() {
22.              this.onselectstart = function() { return false; };
23.          });
24.      });
25.  };
26.
27.
```

Commit C_0 → Commit C_i

t

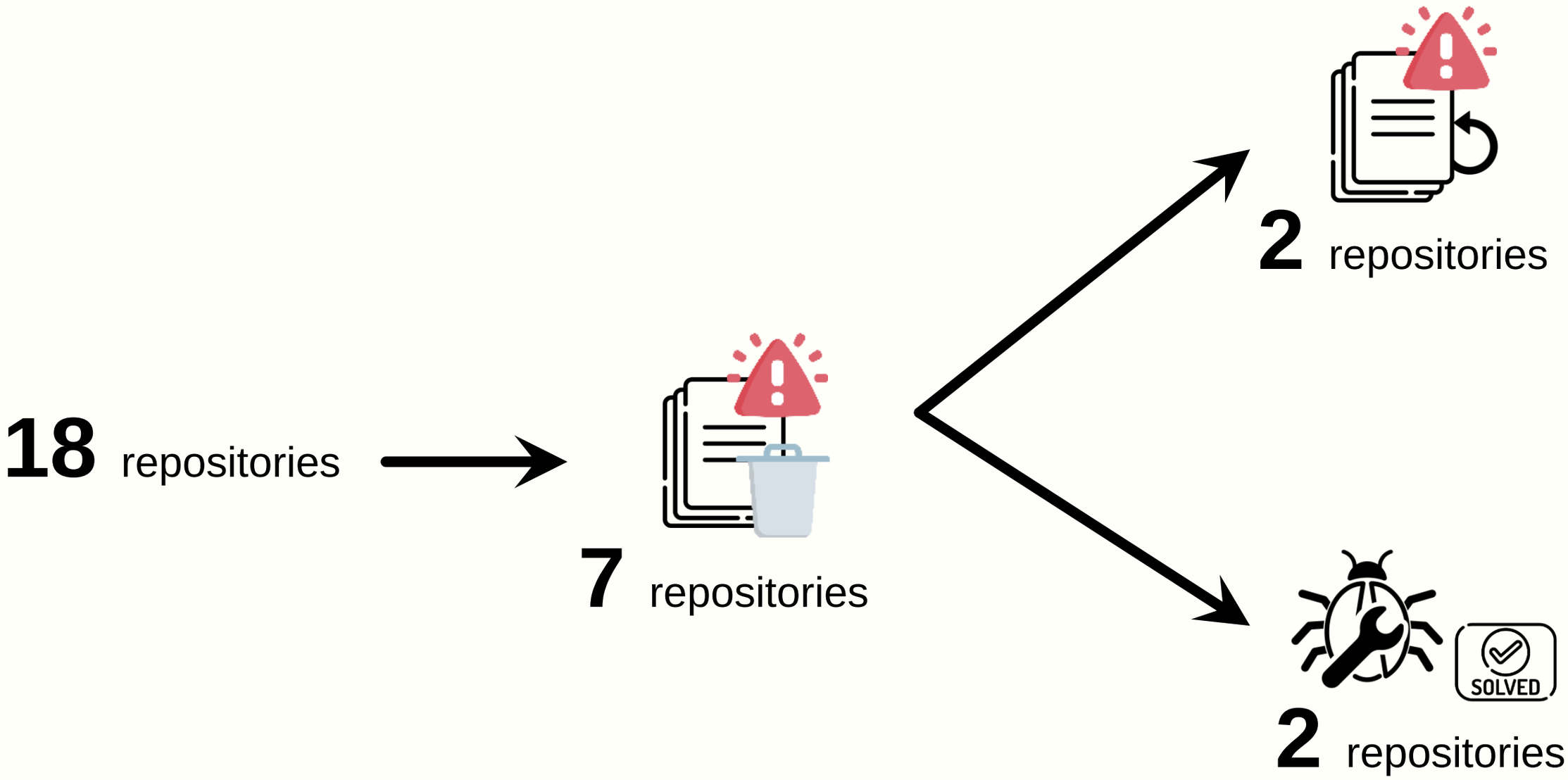
Code context | Righe aggiunte | Righe eliminate

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Risultati (RQ_3)



g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

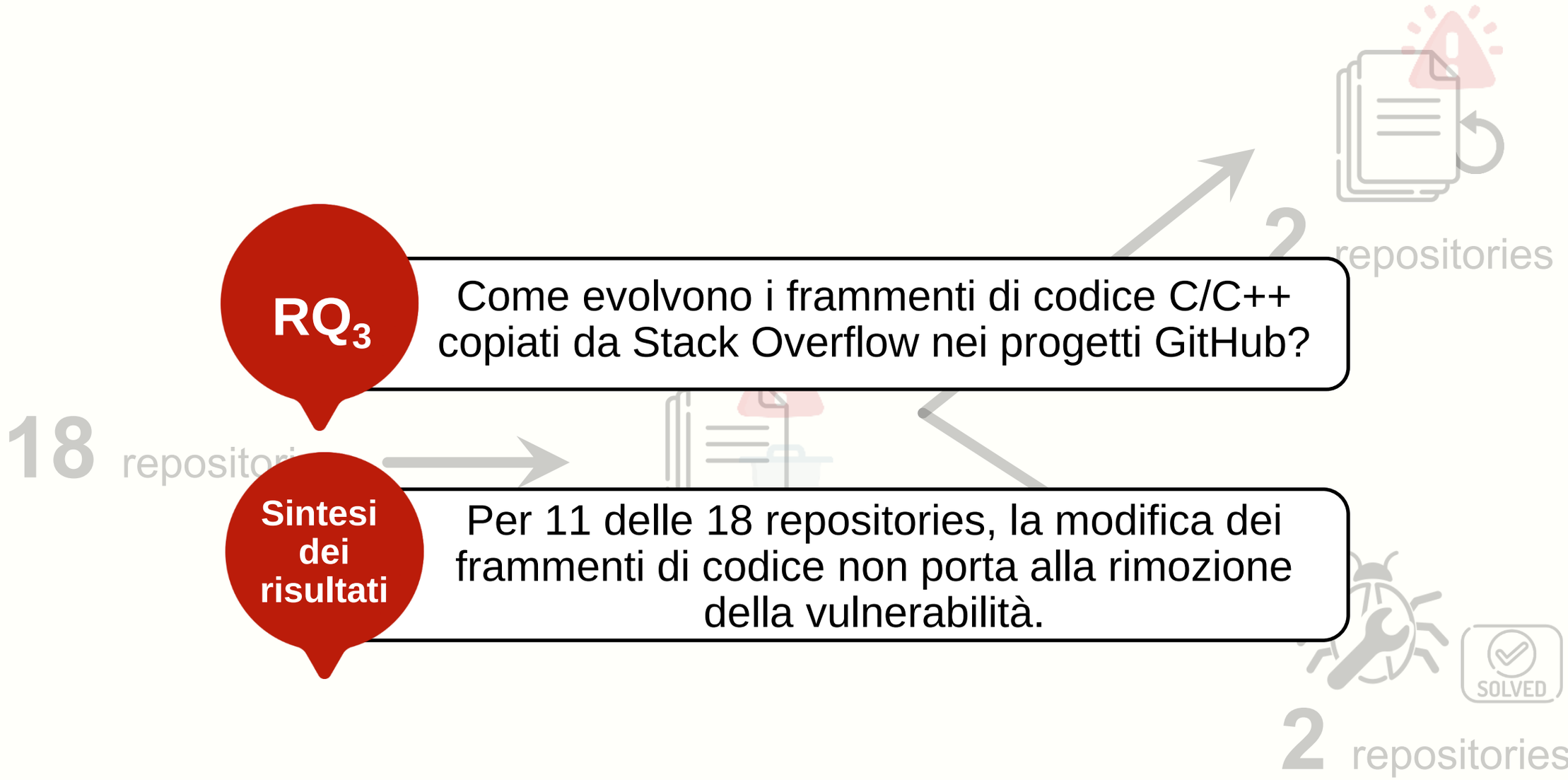
Risultati (RQ_3)

RQ_3

Come evolvono i frammenti di codice C/C++ copiati da Stack Overflow nei progetti GitHub?

Sintesi dei risultati

Per 11 delle 18 repositories, la modifica dei frammenti di codice non porta alla rimozione della vulnerabilità.



g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Conclusioni e sviluppi futuri

Conclusioni

Poca comprensione dei code snippets

Poca importanza alla sicurezza

Sviluppi futuri

Estensione browser

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazia Varone
Università degli Studi di Salerno

Introduzione e Background

How do I execute a command and get the output of the command within C++ using POSIX?

763

```cpp
#include <cstdio>
#include <iostream>
#include <memory>
#include <stdexcept>
#include <string>
#include <array>

std::string exec(const char* cmd) {
    std::array<char, 128> buffer;
    std::string result;
    std::unique_ptr<FILE, decltype(&pclose)> pipe(popen(cmd, "r"), pclose);
    if (!pipe) {
        throw std::runtime_error("popen() failed!");
    }
    while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr) {
        result += buffer.data();
    }
    return result;
}
```

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Metodologia di ricerca

RQ_1 Qual è l'approccio migliore per rilevare i frammenti di codice insicuri su Stack Overflow?

RQ_2 Qual è la tecnica più adatta per il rilevamento di frammenti di codice copiati e incollati da Stack Overflow?

RQ_3 Come evolvono i frammenti di codice C/C++ copiati da Stack Overflow nei progetti GitHub?

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow
Grazia Varone
Università degli Studi di Salerno

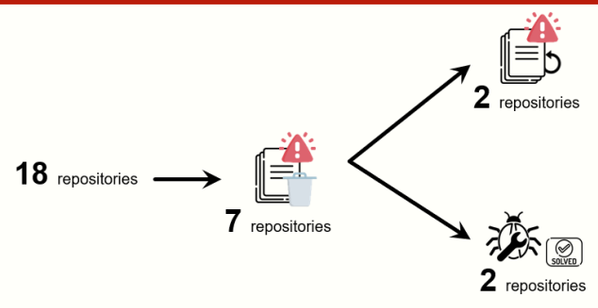
Risultati

	Precision	Recall	F1 Score	Label
Analisi statica	100%	65%	79%	Non vulnerabile
	100%	57%	73%	Vulnerabile
Ensemble learning	100%	100%	100%	Non vulnerabile
	0%	0%	0%	Vulnerabile
Transformer	100%	68%	81%	Non vulnerabile
	100%	15%	26%	Vulnerabile

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow
Grazia Varone
Università degli Studi di Salerno

Risultati



g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow
Grazia Varone
Università degli Studi di Salerno

treedom®
let's green the planet

Questa tesi ha contribuito a piantare un albero in Ghana

Better Safe than Sorry: Investigating the Evolution of Vulnerable Code Snippets Copied from Stack Overflow

Grazie!

Grazia Varone

g.varone9@studenti.unisa.it
https://github.com/graziavarone
grazia-varone