Miguel Luís
Pereira Correia

Detecção de Ataques na Cadeia de Desenvolvimento de Software em Repositórios de Código

Detection of Software Supply Chain Attacks in Code Repositories

Miguel Luís
Pereira Correia

Detecção de Ataques na Cadeia de Desenvolvimento de Software em Repositórios de Código

Detection of Software Supply Chain Attacks in Code Repositories

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Cibersegurança, realizada sob a orientação científica do Prof. Dr. Paulo Jorge Salvador Serra Ferreira, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Dr. Ana Maria Perfeito Tomé
Professora Associada, Universidade de Aveiro
Associate Professor at the University of Aveiro

vogais / examiners committee

Prof. Dr. Mário João Gonçalves Antunes
Professor Coordenador, Instituto Politécnico de Leiria - Escola Superior de Tecnologia e Gestão
Coordinating Professor, School of Technology and Management Polytechnic of Leiria

Prof. Dr. Paulo Jorge Salvador Serra Ferreira
Professor Associado, Universidade de Aveiro
Associate Professor at the University of Aveiro

agradecimentos Agradeço ao Prof. Dr. Paulo Jorge Salvador Serra Ferreira por a ajuda e conselhos durante a realização desta dissertação. Agradeço também os meus pais e irmão por o apoio que me deram durante a realização do mestrado.

acknowledgments I would like to thank Prof. Dr. Paulo Jorge Salvador Serra Ferreira for his help and advice during the writing of this dissertation. I also thank my parents and brother for the support that they gave me during the master's degree.

Palavras-chave

Cadeia de Desenvolvimento, Detecção de Anomalias, Repositórios, Código Malicioso, Support Vector Machines, Simulações

Resumo

Actualmente, o conceito de cadeia de desenvolvimento é algo intrinsecamente vinculado no ciclo de vida do desenvolvimento de software; desde o código fonte e dependências que são inseridas no software, até o software estar disponível a ser distribuído e usado. Com a crescente necessidade de integrar práticas de segurança na cadeia de desenvolvimento, cada etapa e material que influencia o software precisa de ser protegido. No entanto, nem todas as fases da cadeia de fornecimento de software são protegidas e os agentes mal-intencionados exploram essa falta de segurança para inserir código malicioso nos repositórios de código. Depois de obterem acesso ilícito a uma conta de um contribuidor, os atacantes inserem-se nos repositórios de código e, com um plano meticuloso, criam software "trojanizado". O ataque que ocorreu em 2019 à empresa SolarWinds, é um exemplo perfeito que mostra a dimensão que os ataques na cadeia de desenvolvimento podem ter. Nesta dissertação é apresentado como acções maliciosas nos repositórios podem ser classificadas como anomalias nos comportamentos dos contribuidores. A partir das acções dos utilizadores nos repositórios, métricas são calculadas e utilizadas para criar perfis de comportamento que são usados para detectar comportamentos anómalos.

Keywords Supply Chain, Anomaly Detection, Repositories, Malicious Code, Support Vector Machines, Simulations

Abstract Nowadays, the supply chain concept is something intrinsically deep-rooted in the software development life cycle; from the source code and dependencies that are inserted into the software, to its release. With the growing need to shift security left in the development, every step, and material that influences software needs to be secured. However, not all phases within the software supply chain are protected, and malicious actors exploit this lack of security to insert malicious code in the software code repositories. Through account takeovers, attackers can introduce themselves in the code repositories, and with meticulous planning create trojanized software. The 2019 SolarWinds attack is a perfect example that shows the extent that supply chain attacks can have. This dissertation is presented how malicious actions in the repositories can be classified as anomalies within the developers' behaviours. From the users' actions in the repositories, metrics are calculated and utilized to create behaviour profiles that are then used to detect anomalous behaviours.

Table of contents

List of figures

List of tables

List of abbreviations

AD	Anomaly Detection
API	Application Programming Interface
APTs	Advanced Persistent Threats
FP	False Positive
JSON	JavaScript Object Notation
ML	Machine Learning
REST	Representational State Transfer
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SDLC	Software Development Life Cycle
SVM	Support Vector Machines
TTPs	Tactics, Techniques, and Procedures

Chapter 1

Introduction

The ever-growing world of technology brings increasingly more software and devices, which are integrated into our daily lives to make them easier and more productive. Technology and its expansion are now an integral part of the world. Still, such as countries' expansion brought higher risks/consequences (diseases spread more easily, cultures clash with each other, crime expands globally, ...), technology expansion also introduces similar worries. Within technology expansion, organizations started relying upon and trusting heavily on third-party software to grow and expand faster. Included in this third-party software are email services, IT management tools, or other software/packages used in the organization's processes, which can also incorporate malicious code.

Frequently, computer applications are developed without security in mind. This disregard is caused, primarily because developers are more focused on trying to learn the application domain and the intrinsic details than bothered about the security required to safeguard it. In most cases, a prototype is built around satisfying the users' necessities, application security is put aside, and the last thing to be considered. When the time to launch the application arrives, developers come to the realization that adding/providing security will be harder than just including encryption or updating dependencies. Another reason why application security is neglected is due to missing security policies. Ignoring security issues in development's early stages is a dangerous practice, and later on, can bring more adversities when adapting the application to security requirements [1].

Since the start of software development, different development methodologies have been applied at different times, but security was never the first concern within Software Development Life Cycle (SDLC). So application security responsibilities fell upon IT security teams dedicated to this purpose only. In the beginning, applications were only tested after release, taking place in production environments, and rarely performed. This implied that vulnerabilities could be exploited by attackers over long periods until discovered/disclosed. Later on, security testing started being a step that should be surpassed before release. However, this step could take weeks to conclude and with an outcome difficult to predict. So organizations need to shift security into all development phases and create a Secure SDLC, where developers are the front line since they are the ones that

implement software's requirements [2].

With the security shifting into the SDLC, software becomes more secure, but organizations keep facing new security threats constantly, and in recent years one of the appealing targets is their software supply chains, in each the SDLC is included. If the supply chains are compromised, the distribution of malicious code within trusted third-party resources will be possible.

1.1 Software Supply Chain

The concept of supply chain can be defined as a structure of organizations, services, information, and resources, where the final objective is supplying a product or service to an entity. Supply chains are complex procedures that require appropriate communication and well-defined policies [3]. "The software supply chain is similar to other activities or industries. Some resources are consumed, then transformed, through a series of steps and processes, and finally supplied as a product or service to a customer [4]."

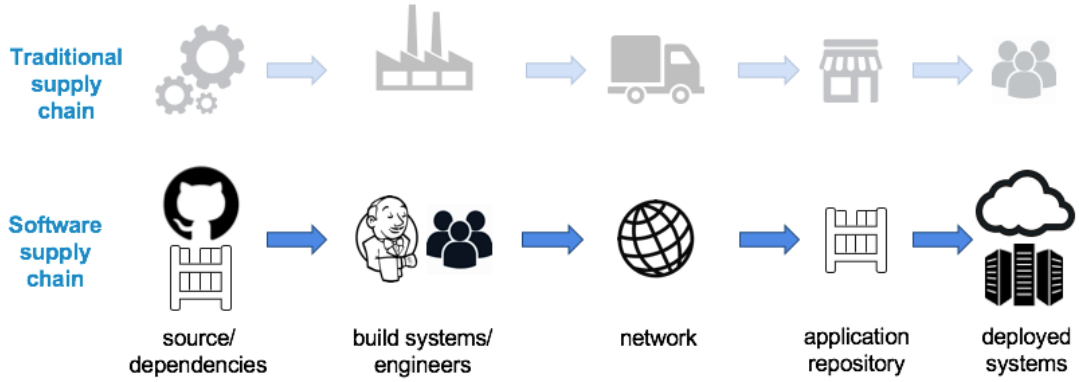

Figure 1.1: Traditional/Software supply chain [5].

Commonly a supply chain is anything needed to deliver a product, including all components used, Figure 1.1. For a chocolate bar sold at a store, it is the list of ingredients, the packaging, the information on nutritional contents, and possibly the information on organic ingredients or production facilities. But it is also more than that, e.g. whatever that influences product delivery. If the chocolate bar is not transported in a controlled environment, the end product that is consumed will be affected.

Switching from the chocolate to source code, a software supply chain can be anything inserted or affects the application code from development through your CI/CD pipeline until it gets deployed into production. It is anything and everything that goes into software, like code, libraries, and other components, and where they are pulled from, like a repository or a package manager. A software supply chain consists of who wrote it when it was contributed, how security issues are being reviewed, known vulnerabilities, supported versions, license information; basically, everything that affected it at some point. The

infrastructure where the application runs, and how it is built, is also part of the supply chain. Essentially, any information that helps ascertain any risk in running the software can be considered part of the supply chain [6].

1.2 Supply Chain Attacks

Supply chain attacks are a soaring threat that also targets the software supply chain and occur when threat actors insert malicious code within trusted software. The main goal is to create attack vectors to infect organizations and compromise their data or systems. By inserting themselves into the software supply chain, malicious actors can implant intentional malicious code (keep in mind that it is not a vulnerability) that will be distributed within trusted sources. To carry out these types of attacks, attackers can use different sources:

- Commercial Software Products - Since plenty of companies can use the same software from the same company, if an attacker is able to infiltrate the company that provides the software and compromise the integrity of the product, he will gain access to a vast number of targets without the need to hack other organizations. There is a link of trust between the software provider and the companies that use the product.
- Open-source - Within open-source solutions, anyone can contribute to their development, which can be an advantage for attackers. By exploiting this open access, attackers can insert malicious code into the source code, facilitating penetration into other organizations that use open-source solutions.
- Foreign-source - Foreign software produced by countries where the government exercises control over what private companies develop can contain malicious code imposed by the government.

Before compromising these sources, attackers rely on stolen credentials, devices with preinstalled malware, compromised development tools, and other vectors to carry out a successful supply chain attack [7]. Although the use of security measures (e.g. Secure SDLC) has increased, supply chain attacks continue to be successful, and according to an Argon Security study, the amount of software supply chain attacks tripled in 2021 compared to 2020 [8].

1.2.1 Known Attacks in the Software Supply Chain

There are already known attacks being used to compromise the software supply chain. Typosquatting, dependency confusion, trojan source, and invisible backdoors are some of them. **Typosquatting** attacks rely on typing errors in the packages/dependencies names that are used in the software, and attackers publish packages with names that resemble the original package name. Usually, these packages have the same code as the trusted one

but also contain malicious code. In 2018, 12 packages were found with malicious code, and four of them (diango, djago, dajngo, djanga) were misspellings of Django, the name of a popular Python framework. The packages contained the same source code as the original package, plus malicious code [9]. In 2020, more than 760 packages were found on RubyGems using the typosquatting technique [10]. And more since then have been found in multiple package managers. **Dependency confusion** happens when a package-management system is misled into pulling a malicious version from a public repository instead of the intended version from an internal repository. An attacker can publish a public package with the same name as a private package with a higher version, and the package-management system will fetch it to the private artifact servers. This technique was demonstrated against systems inside Microsoft, Apple, Netflix, and others [11]. ***Trojan source*** consists in using Unicode "control characters to reorder tokens in source code at the encoding level." These characters are then embedded in the source code comments and strings so that source code characters will be reordered in a way its logic is controlled by the attacker [12]. In a similar way as the Trojan source, it is also possible to use "invisible" Unicode characters, Table 1.1 refers to them, and create ***Invisible backdoors***. By inserting these characters in specific code constructs, attackers may create backdoors that will go undetected in commits [13].

These types of attacks do not need to have a high level of refinement, take the PHP backdoor attempt[1]. The attackers were able to sneak into the project's private git server (git.php.net), and disguised as Rasmus Lerdorf and Nikita Popov pushed two commits with malicious code concealed as a typo correction. The "typo correction" added a means to remotely execute code, through the **User-Agent** header, in any server that uses PHP's Zlib compression for content it sends out [15]. Luckily, one of the project contributors noticed it, and the code was removed from the code base. If these changes were not spotted, the potential for damage would be substantial since according to W3Tech' data[2] "PHP is used by 78.1% of all the websites whose server-side programming language we know", allowing the possibility to inject malicious code into more than half of the internet.

One of the first impactful incidents in software supply chain attacks was the SunBurst, a backdoor implemented in the SolarWinds Orion software. SolarWinds is an American company that provides IT monitoring and management solutions to over 300,000 customers. On December 13, 2019, FireEye announced that a threat actor tracked as UNC2452 was able to infiltrate public and private organizations "via trojanized updates to SolarWind's Orion IT monitoring and management software [16]." To accomplish the attack, the threat actor inserted himself into the Orion build and code signing infrastructure, adding malicious code that was compiled, signed, and published via the software patch release system. This attack was a very sophisticated one, performed with patience, and demonstrated that the attacker was familiarized with the software source code [17].

On the 2nd of July 2021, Kaseya, a company that provides IT Management Software

[1] https://news-web.php.net/php.internals/113838
[2] https://w3techs.com/technologies/details/pl-php

Table 1.1: Invisible unicode characters[14]

Unicode	Name	Unicode	Name
U+0009	CHARACTER TABULATION	U+205F	MEDIUM MATHEMATICAL SPACE
U+0020	SPACE	U+2060	WORD JOINER
U+00AD	SOFT HYPHEN	U+2061	FUNCTION APPLICATION
U+034F	COMBINING GRAPHEME JOINER	U+2062	INVISIBLE TIMES
U+061C	ARABIC LETTER MARK	U+2063	INVISIBLE SEPARATOR
U+115F	HANGUL CHOSEONG FILLER	U+2064	INVISIBLE PLUS
U+1160	HANGUL JUNGSEONG FILLER	U+206A	INHIBIT SYMMETRIC SWAPPING
U+17B4	KHMER VOWEL INHERENT AQ	U+206B	ACTIVATE SYMMETRIC SWAPPING
U+17B5	KHMER VOWEL INHERENT AA	U+206C	INHIBIT ARABIC FORM SHAPING
U+180E	MONGOLIAN VOWEL SEPARATOR	U+206D	ACTIVATE ARABIC FORM SHAPING
U+2000	EN QUAD	U+206E	NATIONAL DIGIT SHAPES
U+2001	EM QUAD	U+206F	NOMINAL DIGIT SHAPES
U+2002	EN SPACE	U+3000	IDEOGRAPHIC SPACE
U+2003	EM SPACE	U+2800	BRAILLE PATTERN BLANK
U+2004	THREE-PER-EM SPACE	U+3164	HANGUL FILLER
U+2005	FOUR-PER-EM SPACE	U+FEFF	ZERO WIDTH NO-BREAK SPACE
U+2006	SIX-PER-EM SPACE	U+FFA0	HALFWIDTH HANGUL FILLER
U+2007	FIGURE SPACE	U+1D159	MUSICAL SYMBOL NULL NOTEHEAD
U+2008	PUNCTUATION SPACE	U+1D173	MUSICAL SYMBOL BEGIN BEAM
U+2009	THIN SPACE	U+1D174	MUSICAL SYMBOL END BEAM
U+200A	HAIR SPACE	U+1D175	MUSICAL SYMBOL BEGIN TIE
U+200B	ZERO WIDTH SPACE	U+1D176	MUSICAL SYMBOL END TIE
U+200C	ZERO WIDTH NON-JOINER	U+1D177	MUSICAL SYMBOL BEGIN SLUR
U+200D	ZERO WIDTH JOINER	U+1D178	MUSICAL SYMBOL END SLUR
U+200E	LEFT-TO-RIGHT MARK	U+1D179	MUSICAL SYMBOL BEGIN PHRASE
U+200F	RIGHT-TO-LEFT MARK	U+1D17A	MUSICAL SYMBOL END PHRASE
U+202F	NARROW NO-BREAK SPACE		

to MSPs and IT Teams to improve efficiency and security, published an advisory to disable on-premise Kaseya VSA servers immediately [18]. At the same time, Sophos published that they were aware of a supply chain attack that used Kaseya as the means to deploy the REvil ransomware [19]. The malware was dropped over a malicious update payload sent from VSA servers that the threat actor took control of through zero-day exploits since some of the VSA server functionalities are the deployment of software. With this, the attacker had the opportunity to bypass the security protections put in place by Kaseya and spread the malicious payload as a trusted source [20].

Following the same trend, package managers (e.g npm, Pypi, NuGet, ...) are being used for these types of attacks. Recently, an npm package was infected with malware, more precisely UAParser.js[3] a library used in apps and websites to detect Browser, Engine, OS, CPU, and Device type/model from User-Agent data. After gaining access to the developer account, the attacker published three versions (0.7.29, 0.8.0, 1.0.0)[4] with malware able to steal data and deploy a cryptocurrency miner. This attack carried a big potential because this library is part of multiple software supply chains, and it is even present in some Facebook components [21].

Thus far, some of the methodologies presented in Chapter 2 can not fully address the problem at hand, and there is still the need to add more capabilities to detect these types of attacks. A direction that can be taken to support the software supply chain security is the evaluation of the developers' actions in repositories since their accounts could be compromised or they can be coaxed into inserting malicious code. The main goal of this dissertation is to design and study a novel Anomaly Detection (AD) system that will look into developers' conduct when inserting code into the software source code repositories, and assess if their behaviours are anomalous.

1.3 Contributions

From the attacks presented previously, we can consider that user accounts takeover can happen, users can become ill-intended actors or the repositories where organizations store their source code can also be compromised. So, there is a need to create and implement solutions that actively monitor suspicious activities. With this in mind, by analysing users' behaviours on repositories, this dissertation presents a possible solution to help the parties involved in securing the repositories, catch potential illicit activities. The system starts by analysing users' data over the repositories, parsing it and extracting features from it, and finally using that data to categorize new user events over the repositories.

This dissertation contributions are intended at,

- Proposing a system that will take into consideration developers' behavioural analysis in software repositories, that will generate alarms when something that deviates from

[3] https://www.npmjs.com/package/ua-parser-js
[4] https://github.com/advisories/GHSA-pjwm-rvh2-c87w

the normal appears;

- Help stakeholders involved in the SDLC make better decisions and keep them vigilant about the changes that their software experiences.

The proposed solution is not in any way a perfect system free of False Positive (FP)s, and there will be the need to validate the results. The research conducted will attempt to find and present a viable solution that the conventional methods can not provide with the growing number of different attacks. The main purpose is to **provide additional support** over source code repositories security, and demonstrate that developers' behaviour is also a viable source of data to detect and prevent software supply chain attacks.

Chapter 2

Software Supply Chain - Repositories and its Security

Supply chain attacks are not only conducted to manipulate source code, so detecting these attacks requires the adoption of different tools and techniques. From the conventional application security tools (such as Static Application Security Testing (SAST), Software Composition Analysis (SCA), and others), to new ones being showcased that scan packages and analyse their behaviours, software supply chain security is moving in different directions. For years most of the effort has been focused on finding vulnerabilities in software source code, and third-party dependencies were considered safe. However, we also need to direct our attention to the external components integrated into the software and the threats the software supply chain is exposed to. There are already some solutions that detect and prevent some of the known attacks and help organizations secure their SDLC. CxDustico is going in that direction, and one of the examples is the Typosquatting attack detection engine that actively scans package managers for packages that are trying to dissimulate themselves as a trusted package. With the discoveries made with the engine, they are able to find other packages published by the same user that, normally, are also malicious [22]. There are also tools that help mitigate the dependency confusion attack, such as DustiLock[1] and snync[2]. Mend Supply Chain Defender[3] also has similar solutions. To protect against package managers' account takeover, CxDustico launched the ChainAlert service, which was designed to identify and prevent potential attacks. This service monitors the NPM package manager and looks for characteristics that may identify a possible attack, for example, if the package published does not have a Git tag in the GitHub repository [23]. Other approaches are being adopted in dependencies and open-source solutions, like detonating the packages before using them. The detonation consists of the installation and execution of the package, analysis of the install scripts that may be executed, testing the package functionalities, and analyse of its behaviour. Behavioural

[1] https://github.com/Checkmarx/dustilock
[2] https://github.com/snyk-labs/snync
[3] https://www.mend.io/mend-supply-chain-defender/

analysis is one of the methods that is being applied to catch supply chain attacks in the SDLC early stages, that currently are targeting external dependencies massively.

As presented in Chapter 1, securing the software supply chain is imperative since supply chain attacks, usually, are not bugs introduced by developers (that may develop into vulnerabilities). It is malicious code introduced purposely to compromise multiple organizations with only one entry point, a trusted software provider. Despite the progress already made in the software dependencies area, there is also an entry point that needs to be considered in the software supply chain security, the developers' accounts used to commit to the repositories. By analysing their behaviour, they can contribute meaningful data that can help detect malicious actions. To better understand how the analysis of these behaviours can contribute to securing the software supply chain, we need to understand what parts are relevant to it.

2.1 What is a Repository?

One of the most significant elements of the software supply chain is the repository. The word repository is defined by the Merriam-Webster dictionary as "a place, room, or a container where something is deposited or stored." Similarly, in the information technology area, a repository, also commonly mentioned as **repo**, is a place where data is stored and maintained in an organized way, typically in digital storage. Due to the multitude of data that a repository can save, it can serve multiple purposes and functions. In software development, the most common are source code repositories and software repositories. A software repository is defined as a centralized storage location for software packages, enabling cooperation between developers and users over remote access to code packages and modules. A source code repository, on the other side of the spectrum, is a storage to save the software code base. Source code repositories are almost always version-controlled, allowing one to track and manage changes in the software code. Version control systems are software tools that help software teams govern changes to source code over time. Platforms like GitHub or Bitbucket serve to host such repositories. Figure 2.1 depicts the contemporary top source code repository hosts.

2.1.1 Repository Characterization

The crucial aspect of source code repositories in the software supply chain imprints in them a need for an imperative secure environment. To secure a source code repository it is important to understand what information is possible to extract from it. Usually, the most part, if not all, of the data inserted into the repository is performed by humans. Considering that human behavior can be modeled and studied, human interaction in a repository can be likewise studied and interpreted. But from all the data produced in the repositories, what is relevant to classify developers' behaviours?

If the question "What is the elementary action in a source code repository?" was

Figure 2.1: Top Source Code Repository Hosts [24].

raised, one of the most frequent answers would be a commit. A commit can have different meanings in terms of version control tools or applications. Nevertheless, in version control, a commit can be generally defined as an operation that transfers the most recent changes in the source code to the repository, making these changes part of the head revision of the repository. A commit is not just a publication of additions or modifications in the repository structure but also adds underline information to the repository. It can then be used to analyse users' behavior. Some example of the baseline features that is possible to use to examine size-based characteristics of commits are:

- Number of files being added, modified, or deleted together in a commit;
- Number of lines being added, modified, or deleted in a commit;
- The vocabulary used in the commits messages.

From these commit characteristics, it is also possible to correlate the commit size with its message. For example, when a new development or fix is committed, developers add messages that describe the changes added. Some repositories can even have standards for the commit messages content that define a specific vocabulary to be used, which helps correlate the message type and the commit size. Commit messages and sizes can be used to identify specific developers' behaviours [25].

So we can define software repositories as a representation of all the actions that devel-

opers execute to add new features, fix bugs, or upgrade dependencies of an application. From users' conduct in the repositories, it is possible to derive profiles that outline their behaviours.

2.1.2 Profiling actions

User profiling has been successfully used in Big Data practices, user-based anomaly detection systems, and user authentication methods [26]. Compromised account detection, compromised system/host/device detection, insider access abuse, Data Exfiltration detection, and Data theft detection are some use case scenarios for user behaviour analytics. There are some use case scenarios in which deviations from the typical behaviour of insiders with malicious intent can take place, which makes it very complex to differentiate genuine users from users that have malicious intent. The key points that should be directed for analysing the user behaviour are which features should be selected to analyse and use in anomaly detection. After the feature selection, profiles are generated and used to identify the deviations.

By using this approach, a baseline profile for each user activity is necessary, and in case there is any deviation from it, then it can be tagged as an anomaly for the particular user behaviour, which further needs to be investigated by the security experts [27]. The same principle can be applied to source code repositories since users' activities when interacting with them can also be used to generate profiles.

But from all the data created by developers' actions, what should we consider a feature? Taking GitHub as an example (a widely used code repository for open source software), this platform provides a lot of information about user actions that can be used to collect specific characteristics and also infer other knowledge from it. [28, Dabbish 2012] mention that people draw out a collection of social inferences from the activities in GitHub, such as a user's technical goals when he edits code. Table 2.1 represents some of these inferences made from the study participants' quotes. From analysing the information collected, users can devise strategies that help them plan and coordinate work within the repositories.

These inferences represent a small set of information that can provide some knowledge regarding a user's commitment and level of knowledge in a project. Although important, it is not enough to represent the users' contributions to a repository. Some of the features mentioned in the previous subchapter tend to be more representative of what is a user's ordinary commit. Such as commits that introduce changes by adding, modifying, or removing something in the source code. The commit message, files manipulated, and lines changed are also features that represent a user in a repository. By profiling these user actions, researchers devise anomaly detection systems with the purpose of spotting outliers within the profiles created.

Table 2.1: Visible cues and the social inferences from GitHub [28].

Visible Cues	**Social Inferences**	**Representative Quote**
Recency and volume of activity	Interest and level of commitment	*"this guy on Mongoid is just – a machine, he just keeps cranking out code."*
Sequence of actions over time	Intention behind action	*"Commits tell a story. Convey direction you are trying to go with the code ... revealing what you want to do."*
Attention to artifacts and people	Importance to community	*"The number of people watching a project or people interested in the project, obviously it's a better project than versus something that has no one else interested init."*
Detailed information about an action	Personal relevance and impact	*"If there was something [in the feed] that would preclude a feature that I would want it would give me a chance to add input to it."*

2.2 Anomaly Detection

Anomaly Detection is the process of finding patterns in data that deviate from or do not match the likely behaviour. These discordant patterns are usually specified as anomalies, outliers, exceptions (...), depending on the area where the terms are applied, Figure 2.2. AD can be executed in different areas, from fraud detection in credit cards or insurance, intrusion detection on networks, fault detection in critical systems, and military surveillance for enemy activities. Detecting anomalies in data can bring notable improvements in finding prosecutable information in various areas of expertise. For instance, an abnormal network traffic pattern may pinpoint that a computer was compromised and is being used to steal sensitive data. Credit card transaction data that deviates from normal use may suggest credit card or identity theft. An abnormal MRI image could reveal the presence of malignant tumors. AD can be adapted in different activities to discover divergent patterns that may be considered anomalies, and with them produce valuable information that can be used to generate action points [29].

The study of anomalies is not something new and was already being done by the statistics field in the 19th century. With the advance in intrusion detection systems, it started making sense to also employ these studies in it. AD surmounts the restraints that signature-based detection mechanisms have by observing and describing normal behaviour to spot the existence of attacks [31]. But to implement a proper AD system, first we need to establish the concept of normality. The interpretation of normal can be presented by expressing the relations between the principal variables that affect the system dynamics, in a formal model. As a consequence, a phenomenon or an object is perceived as abnormal if its deviation, taking as reference the normal model defined, is high enough [32]. Within the AD, it is also important to consider the nature of the anomaly that fits the system in

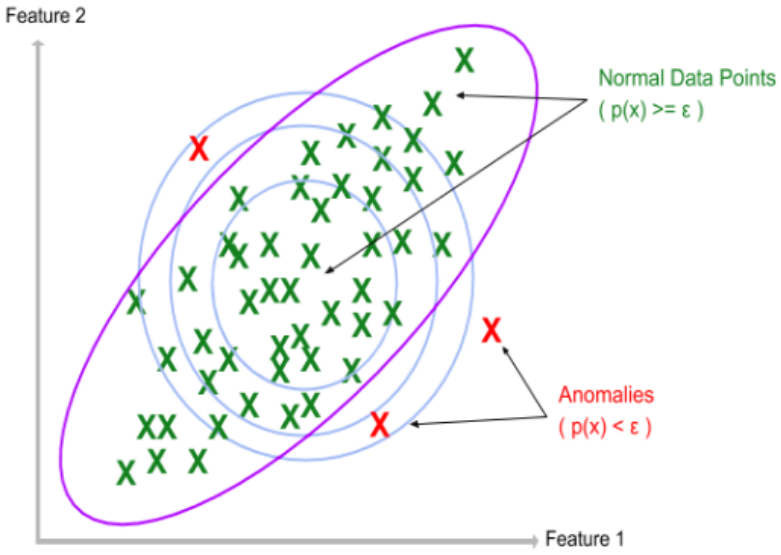

Figure 2.2: Representation of anomalies in a data set [30].

question, and anomalies can be divided into three categories:

- Point Anomalies - If an individual data instance can be considered anomalous with respect to the rest of the data, then the instance is termed a point anomaly. This is the simplest type of anomaly and is the focus of the majority of research on AD.
- Contextual Anomalies - If a data instance is anomalous in a specific context (but not otherwise), then it is termed a contextual anomaly.
- Collective Anomalies - If a collection of related data instances is anomalous concerning the entire data set, it is termed a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous.

The anomalies in source code repositories can be considered as Point Anomalies since the system try to identify if a specific behaviour is anomalous regarding all the previous data analysed. Two AD systems were proposed that try to tackle the problem of identifying anomalies with repositories. The first system implements a solution that detects unusual commits as outliers from various commit characteristics. It compares specific commit characteristics against common characteristics of other commits that belong to the same repository or developer. Firstly, profiles are created from past commits and used to compute an anomaly score for new commits. For instance, a profile that illustrates the size of past commits, using the lines of code, will then be used as the baseline that indicates if a new commit is within the distribution of the past history. These profiles are based on the commit size, the time of the day the commit happened, and what file types were changed [33].

The second solution proposed implements a rule-based AD system that uses some of the features as the first one, plus more information that GitHub provides. The lines of code added and removed, files added, modified, and removed, the pulls requests, and the users' accounts are used as features within the system. *Anomalicious*, as the authors named it, is a tool that calculates a trust factor for the developers and uses pre-defined rules to identify outliers within the repository commits. The rules rely on thresholds that were configured by running the model multiple times with different values [34].

2.3 Machine learning in Anomaly Detection

But AD rule-based systems are not enough these days, and Machine Learning (ML) systems are now gaining momentum in the AD area, especially in the detection of malicious attacks. Since attackers' behaviours change continuously over time, rule-based systems are not capable of identifying attacks that are not known. In order to implement a more viable solution, researchers are applying ML in the detection of anomalous behaviours in the cyber security field. Either classical ML classification and deep learning classification techniques, or a combination of both groups. In "Evaluation of machine learning algorithms for anomaly detection", twelve ML algorithms were tested in terms of their abilities to detect anomalous behaviours to discover possible attacks, found in the UNSW-NB15, CICIDS2017, and Industrial Control Systems (ICS) cyber-attack datasets [35]. Table 2.2 presents the ML algorithms chosen with advantages and disadvantages.

Table 2.2: Some advantages and disadvantages for the ML algorithms selected in the article [35].

Algorithm	Advantages	Disadvantages
Logistic Regression (LR)	Low computing cost and fast speed. Its output can be interpreted as a probability	May suffer from under-fitting. The performance is poor when the feature space is large
Gaussian Naive Bayes (GNB)	Fast training speed for small and big datasets. Less sensitive to missing data.	It needs to calculate prior probabilities. It does not work well if the sample's attributes are related
K-nearest Neighbours (KNN)	It can be used for both classification and regression. Easy to understand and implement	Poor performance when analysing unbalanced samples. High computational complexity for large datasets
Decision Tree (DT)	Fast prediction. Addressing highly non-linear data	Suffers from over-fitting. More time is needed to train the model
Adaptive Boosting (AdaB)	Various algorithms can be used to build sub-classifiers. Does not easily lead to overfitting	The performance depends on the selected weak classifier. Sensitive to outliers
Random Forest (RF)	Robust to outliers and can handle them well. It is comparatively less affected by noise	Longer training time since it will generate many trees. Much more computational power and resources needed
Convolutional Neural Network (CNN)	Handle high-dimensional data well with shared convolution kernels	The training result easily converges to the local minimum instead of the global minimum using the gradient descent
Long Short-Term Memory (LSTM)	Gate mechanism greatly alleviates the problem of gradient disappearance and simplifies the complexity of tuning parameters	High computing cost
Gated Recurrent Units (GRU)	Gate mechanism alleviates the problem of gradient disappearance	High computational cost
Simple Recurrent Neural Network (RNN)	It can learn and use contextual information explicitly in sequence prediction	The problem of gradient disappearance occurs readily
Deep Neural Network (DNN)	It can execute feature engineering on it own compared with traditional ML methods	Many parameters need to be fine-tuned

After applying the algorithms in the datasets that represent malicious activities, the authors evaluated their performance when detecting anomalous behaviour that may be illustrative of a cyber attack. The results showed that the Random Forest algorithm provided the best accuracy, precision, and recall among the algorithms used, and the Naive Bayes classification had the lowest performance regarding the same metrics [35]. The datasets used to represent known attacks that may not contain the most recent attackers TTPs, and the attack surface is limited to specific areas. For example, the UNSW-NB15 Dataset has nine types of attacks, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. The CIC-IDS2017 dataset includes Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS. These datasets only represent a subset of the possible attacks and do not contain representations of supply chain attacks.

In an in-depth article, a Systematic Literature Review was conducted which analysed ML models applied in AD. In this review, the authors identified 290 research articles written between 2000-2020 that discuss the usage of ML techniques in AD. It also indicates that intrusion detection, network anomaly detection, general anomaly detection, and data applications are the studies applied most often in the anomaly detection area. Different models and strategies are applied, but the most commonly used is Support Vector Machines (SVM) [36]. Table 2.3 illustrates some of the techniques used in research articles, the usage frequency, and some of their strengths and weaknesses.

The review authors recommend that more research is conducted on the use of ML in Anomaly Detection to gain more evidence on ML performance and efficiency. Moreover, improvements are also required from researchers in terms of the general structure for introducing experiments on ML, feature extraction and selection needs to be improved as well [36].

Table 2.3: Some ML techniques and their strength and weaknesses among research articles [36].

ML model	**Freq.**	**Strength and Weakness**
SVM	23	Strength: Reduced computing complexity. Weakness: Soft margin SVM can not be used for novel attacks because it needs pre-acquired learning info.
Cluster	11	K-means Strength: K-means only requires pairwise distance and the algorithm does not require the distance to be metric.
Neural Networks	8	Strength: Neural networks based on the concepts of statistical pattern recognition have emerged as a practical technology.
OneClassSVM	8	Strength: No need for sample data with free anomalies. Achieves better accuracy rates than the conventional anomaly detectors.
Auto Encoder	8	Strength: Efficiently reconstruct inputs that closely resemble normal network traffic. Weakness: Poorly reconstructs anomalous or attack inputs.
Naive Bayes	6	Strength: One of the simplest and effective classifiers. Weakness: False positive rate needs to be improved.
Decision Tree	5	Strength: By tracking the nodes from the root of the tree based on the feature values of an example, we can get the predicted class of it.

Chapter 3

Detecting malicious behaviour

Detecting anomalous behaviours in the cyber security context is an endless challenge, where threat actors constantly evolve their TTPs. This evolution is driven by the increasing dependency of governmental and commercial organizations on information technology and the need to have access to information and systems that may secure strategic advantages. Advanced Persistent Threats (APTs) attackers start by researching their targets before carrying an attack, in order to maximize the impact and minimize the risk of exposure [37]. Detecting attacks crafted for a specific purpose is hard for conventional solutions, such as Intrusion Detection Systems, that were created to identify anomalous behaviours in network systems. But these systems are based on signature-based approaches, that monitor network activities against a set of pre-identified attacks. A possible solution is the use of detection systems that predict malicious attacks based on anomaly behaviours. Prediction systems based on ML algorithms could be a step toward future solutions [38].

In case a software house is compromised, by an account takeover, for example, the attackers may have the possibility to manipulate their source code and infect more entities. A system that predicts and flags possible malicious behaviours over source code repositories is the solution proposed in this chapter. By extracting data from the code repositories and using it to create behaviour profiles, the system will be able to identify and warn the repository maintainers that suspicious activity occurred. The following sections will present and detail how the system was implemented, what user data uses and how the predictions were implemented.

3.1 The system layout

The implementation contemplates different modules that work independently from each other. The system starts by gathering information from the repository, parses the data, and calculates the necessary features that will be used in the prediction phase. Finally, outputs a binary prediction informing if specific user behaviour is an outlier from the normal conduct in the repository. Figure 3.1 depicts a simplified view of the system flow.

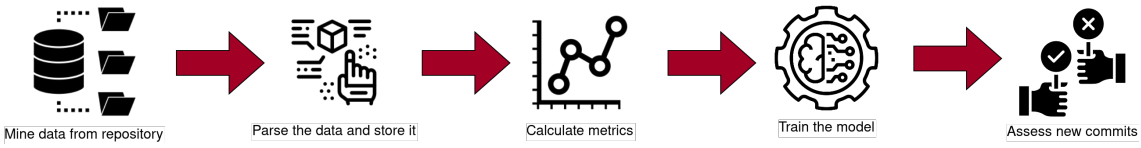

Figure 3.1: System flow overview.

The repository information is gathered through a RESTAPI that the repository hosting service needs to expose. Through the API, all the necessary information is requested to, later on, be parsed and used. For the proposed solution the focus was the GitHub REST API[1]. More specifically, the Commits API[2], each allows the ability to filter commits by author, date and other attributes. The commits endpoint only provides the commits that are in the repository's main branch. The commits in the main branch are considered as the normal behaviour of the developers, and no malicious activity occurred in them.

```
Basic Authentication: config['Github']['Username'], config['Github']['Token']
'Accept': 'application/vnd.github.v3+json'

URL = API_URL + 'repos/{}/{}/commits?author={}&per_page=100&page={}'
                    .format(owner, repo, author, page)
data = JSON response
```

Listing 1: Pseudo code to interact with GitHub Commits API.

All data extracted is stored in a database that will be used as the source to calculate the metrics and act as the truthful repository data. Figure 3.2 illustrates the database schema. The schema represents a relational database with 8 tables that store data related to the commits in the repository, developer and repository statistics, and information about the repository files. The commits table holds all the relevant features from a single commit, such as the lines of code and files that were added, modified, or removed. This table has then relations with the developer and file tables, where one holds information about the developer and the other about the file in the repository. The tables developer_stats and repo_stats hold the statistics that were calculated after all the data from the repository were parsed and stored. The pull table holds information about the pull request created. Finally, the commits_stats table is the representation of the commits' daily statistics. These statistics will then be used in the later stages of the system.

By using a database, it is possible to separate the metrics logic from the mining/parsing phase, and make it agnostic from where the data is mined (GitHub API, GitLab API[3], SourceForge API[4], ...). All the implementation was done around GitHub, but adjusting it to another platform should be straightforward and only affect the data extraction and

[1] https://docs.github.com/en/rest
[2] https://docs.github.com/en/rest/commits/commits
[3] https://docs.gitlab.com/ee/api/
[4] https://sourceforge.net/p/forge/documentation/Allura%20API/

parsing.

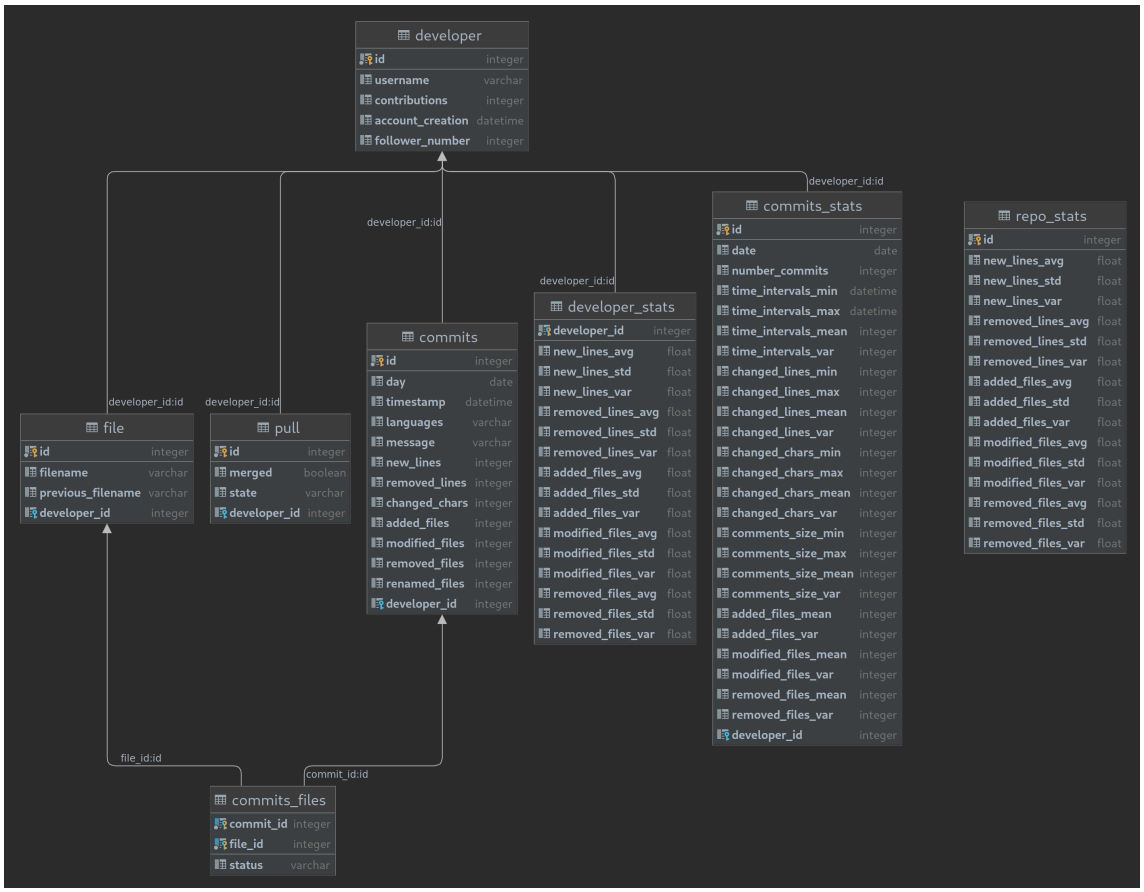

Figure 3.2: Database schema.

3.2 Data parsing

From all the data returned by GitHub API, only some of it is relevant for defining user profiles. The JavaScript Object Notation (JSON) represented in Listing 2 presents all the data that will be used to describe users' behaviour. These data provide all commit actions executed in the repository, from all the lines that were added and deleted (stats) to a more detailed view of the files that the commit interacted with.

After obtaining the commit data from the repository, it is parsed and inserted into the database. GitHub RESTAPI has rate limits[5] in place. In case a personal account is used to fetch data from a repository with a great number of commits, and if the limit is exceeded, a cooldown will be issued.

During the parsing there are different actions carried out per commit:

- the files are added as a relation to the commit;

[5] https://docs.github.com/en/rest/overview/resources-in-the-rest-api#requests-from-personal-accounts

```
{
    "stats": {
        "additions": 104,
        "deletions": 4,
        "total": 108
    },
    "files": [
        {
        "filename": "file1.txt",
        "additions": 10,
        "deletions": 2,
        "changes": 12,
        "status": "modified",
        "raw_url": "https://github.com/octocat/Hello-World/raw/7ca4.../file1.txt",
        "blob_url": "https://github.com/octocat/Hello-World/blob/7ca4.../file1.txt",
        "patch": "@@ -29,7 +29,7 @@\n....."
        }
    ]
}
```

Listing 2: Some of the commit data returned from GitHub.

- the number of added, modified, renamed and removed files are counted;
- the lines added and removed from files;
- the number of changed characters in the files are counted;
- all the data is associated with the user that pushed the commit.

The data collected represents the user's normal behaviour throughout the repository history. The files can identify in what software layer (backend, frontend, infrastructure, ...) a user works regularly on. If the user adds files regularly, most likely, he is more focused on adding new features than refactoring code or fixing bugs. A great number of lines removed can probably signify that the user refactors code regularly. The changed characters can infer the user's proficiency over the source code. Most of the features considered were already used in previous commits studies: the lines of code changed, and the number of files added, modified, and removed [34][33][39][40][41]. The number of characters modified was not considered as a possible feature in former studies.

3.3 Obtaining daily commit features

After obtaining the necessary data, the next step is computing users' data into characteristics that will describe their behaviours over the repository. The final results are the features that will be fed to the model training. The features are calculated over all

commits that a user made during a day, and are represented by the mean and variance of the:

- modified lines;
- modified characters;
- added files;
- modified files;
- removed files;
- number of commits;
- and time intervals.

This solution also contemplates the use of time intervals between commits, the modified characters, and the number of commits per day as features of users' behaviour profiles, something not considered in other solutions. These features are also contemplated because they also express the behaviour of a developer. For example, the time intervals demonstrate how the developer adds changes to the repository throughout the day. The same happens for the commit message since developers do not express the changes added in a uniform way. The messages can have a specific format, but the summary of the change tends to be a unique feature.

Since with only one commit is difficult to represent a user behaviour, all data from a single day is added up. With a single commit, it can be difficult to grasp if it is the real user pushing the commit or if its account was stolen by a malicious actor. An experienced attacker will research the target first and later use the information to his advantage and create a commit that hardly differs from a normal commit. With a daily profile, a single commit for a user that normally commits regularly during the day will be an anomaly.

3.4 Train and predict - OneClassSVM

After all the data is collected, parsed, and the profiles created the system is finally ready to be trained and to predict anomalous activities. Since the system's main purpose is to detect if a specific behaviour deviates from normal or not, it can be classified as a binary problem. Taking this into consideration and the fact that user data is used to represent its realm of actions, domain-based ML methods should be the more adequate to use. Domain-based methods rely on an outer limit created from the training data format. Generally, these methods are indifferent to the target class sampling and density, since they express the class domain and not its density. Within the set of domain-based methods, SVM is a popular approach for composing the decision boundaries that divide the data into different classes. SVMs applied on detection are also called one class SVMs

[42].

In a compilation of selected articles that use ML for anomaly detection, one class SVM models present good results on standard datasets. One class SVMs achieves better accuracy rates than conventional anomaly detectors [36]. The implemented system uses a one class SVM for detecting user's anomalous behaviours, after all, this model presents better accuracy rates in novelty anomaly detection.

To implement the system detection module, it was used the OneClassSVM[6] provided by the scikit-learn package. Listing 3 details a code sample of OneClassSVM, with the same configurations used. The final OneClassSVM uses a **polynomial kernel**, with a *gamma* defined as **auto**, each defines the kernel coefficient as *1/number_of_features*. The *nu* parameter, that is the upper bound on the fraction of training errors and a lower bound of the fraction of support vectors, as **0.1**.

```
1 svm = OneClassSVM(kernel='poly', gamma='auto', nu=0.1)
2 svm.fit(user_features)
3 svm.predict(new_behaviour)
```

Listing 3: OneClassSVM code sample.

Since all data was stored in a database, it makes the process of constructing the training matrix easier. The matrix rows represent the daily statistics, and the columns the features used, that is then fed to the model. After training the model, the prediction will be a binary output that will represent if the commit is an anomaly compared with the previous data or not.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

Chapter 4

Performing against *Anomalicious*

Anomalicious, presented in Chapter 2, was created to identify anomalous and potentially malicious commits on GitHub. This tool presents a rule-based system that offers a possible solution for securing open-source software repositories. Since systems that try to detect anomalous behaviours in repositories are scarce, *Anomalicious* provides a good point of comparison. In order to understand if the detection system proposed can be a viable solution, a similar tool to *Anomalicious* was implemented. After the implementation, tests were performed against it by forging commits that followed the same expected user behaviours.

This chapter details the *Anomalicious* implementation; how the commits are forged, which are then used to test the *Anomalicious* implemented and the proposed solution. Later on, the best configurations for the proposed system are produced, and finally, all the results from the simulations are obtained from both solutions and compared.

4.1 Implementing a similar *Anomalicious*

A similar tool to *Anomalicious* was implemented as an approximation to the one presented by the paper authors. It uses the same factors defined by them, "identifiable username, account age, number of commits, time since the first commit, commit-time distribution, and proportion of pull requests rejected". Since all the repository data is already parsed and stored in a database (by a module of the proposed system), there was no need to recreate the extraction and parsing phases.

Taking advantage of the existing implementation, the decision model based on rules was the only thing implemented. First, it was defined the account trust model, which defines if the contributors' account should be trusted based on the following rules:

- The contributor's username cannot be found on GitHub;
- The contributor's account exists for more than X days (at the time of analysis);
- The contributor has made more than X other commits in the repository;

- If it is the contributor's first commit;
- The contributor made more than X% of his commits on the same day;
- The contributor's first commit exists within the past X days (at the time of analysis);
- More than X of the contributor's PRs were rejected.

The trust rules establish if the account used to contribute to the repository can be trusted, but are not the only factor that defines if a commit is anomalous. It can be an important consideration if the final decision is contested. After calculating the trust factor, the commits factors are then verified against a new set of rules. The decision rules implemented take the following structure:

- The commit touched more than X sensitive files;
- It is not the author's first commit AND more than X% of files were never touched by the author before;
- It is not the author's first commit AND the author is not the owner or majority contributor to X% of the touched files;
- The commit adds more than X number of files and does not touch any files that the author is not the owner or majority contributor;
- The commit has more than X change properties that are outliers for the author or repository;
- The commit author's contribution history in the repository demonstrates they are untrusted;
- The commit is linked to more than X rejected pull requests.

The trust and decision rules are only an approximation to what is presented in the *Anomalicious* paper. Nonetheless, the differences should not be significant considering that the same characteristics are used, and the implemented rules follow closely what the authors present. In case divergencies do exist, they may come from the paper's interpretation.

Looking closely at how *Anomalicious* works, some of these rules take into account aspects that may not be sufficient to determine if a commit is indeed an abnormal behaviour. The pull requests may not be relevant in a repository where this is not common practice. The way code is introduced to the main branch is shifting to the use of pull requests, but this is not a universal policy, and developers can just push commits directly to the main branch. Also, an experienced attacker, most likely, will not push a commit that was previously rejected, considering that this will raise alarms that suspicious activities are

undergoing. Another flawed approach is considering that a contributor that stays within the files that he owns or is a major contributor, is not a malicious action. Let us take as an example an account takeover for a specific user. Malicious commits can be introduced in the files that are owned, or that account is a majority contributor, and no alarms are raised since it is an expected outcome.

4.2 Anomalous commits simulation

In chapter 3, it is presented that malicious actors tend to plan their actions before initiating their attacks. Taking into consideration the attackers planning phase, an attack simulation tool is needed to create a similar environment where the attackers generate commits within the same boundaries of the account that they may take over. Within the simulation environment, new commits are generated and then pushed into the repository but with the account of a regular contributor. None of these actions are done on an authentic repository since it is unethical and access to real accounts would be needed, each is highly unlikely.

To test the reliability of both systems, a module was created to assemble commits that take users' properties and uses them to "forge" commits. These commits simulate possible malicious actions, that were not carried out by the authentic users. This is only an experimental process and may not be an accurate representation of the actions that a real attacker may take.

4.2.1 Forging commits

The simulation module relies on the data previous collected from the repositories to generate the fake commits. The contributor statistics are fetched from the database and used randomly to select how many lines and files are changed. The following developer's data is used to forge the commits:

- The average of files that the developer adds modifies, and removes from the repository;
- The average of lines that the developer adds or removes in the repository;
- The files that the developer owns or is a major contributor to.

From these data, the commit is then generated, where the final values are randomized with a uniform distribution, as it is presented in the Listing 4. The files that are added to the commit are also chosen by running a random sample over all the files the developer owns or is a major contributor to. Commit messages are also a random selection from all the previous contributor's messages so that the system validation can be fooled. The developer statistics used are calculated from all data collected to create a generalized profile.

```
# Lines to be added and removed
new_lines_to_add = random.randint(1, round(dev_stats.new_lines_avg))
removed_lines_to_add = random.randint(1, round(dev_stats.removed_lines_avg))
# Files for the commit
files = random.sample(filenames, number_of_files)
```

Listing 4: The files and lines to be added and removed on the fake commit.

From all the data collected, two different events can be created, a clever commit that stays within the developer boundaries, or a "simple-minded" commit that does not care if is within the developers' behaviours. Listing 5 demonstrates a "simple-minded" commit generated with "random" files added to the repository and modifying the dependencies file. The Listing 6 depicts a clever commit generated from the data collected from the user *abe33* present in the minimap repository[1]. This user is the major repository contributor and its average of lines added and removed are **22,22** and **12,23**, respectively, and the average files modified are **1** (rounded). In the commit, two new lines are added to the **spec/helpers/workspace.js**, and these lines represent a package import and a command execution. The command is `curl http://malicious.gang/payload | bash`, each executes an HTTP GET request and executes the output into bash. It is encoded in base64 in an attempt to be overlooked.

The purpose of this tool is to automatically generate a great number of events that have similar characteristics to the repositories contributors. In terms of the code that goes into the patch parameter presented in the Listing 6, in a premeditated attack it would be something well planned so that goes unnoticed. The code would also provide an advantage for the attacker, it does not need to necessarily code that injects or installs backdoors. It can also be a change in a configuration file that connects to a database, or a proxy that is controlled by the attacker.

4.3 Configuring the OneClassSVM model

After implementing the *Anomalicious* and constructing the simulation environment, the next step is to configure the proposed solution. The relevant parameters to configure are the *kernel*, *gamma*, and the *nu*. These configurations were chosen by analysing the results obtained when training the model with half of the data and performing the prediction with the other half, using the data of the user with the highest number of commits. In this phase it was used data from three repositories, where users have a high to a low number of commits:

- minimap repository - holds the source code of a javascript package used to preview source code. This package was also used in the *Anomalicious* paper [34]. Currently,

[1] https://github.com/atom-minimap/minimap

```
{
  "sha": "68130eee73fd07d6f325385a3c7e5c4363cf4c84",
  "commit": {
    "message": ":bug: Fix minimap broken with changes in flex box behaviour",
    "author": {
      "name": "abe33",
      "email": "abe33@attacker.com",
      "date": "2022-10-29"
    }
  },
  "stats": {
    "total": 217,
    "additions": 119,
    "deletions": 98
  },
  "files": [
    {
      "sha": "4bd036d8da6e",
      "filename": "wUFVUClwIyfGl",
      "status": "added",
      "additions": 75,
      "deletions": 0,
      "changes": 75,
      "patch": "@@ -1,3 +1,11 @@\n<malicious code>"
    },
    {
      "sha": "32defeb3408b",
      "filename": "qgwab6r7BG2PAET",
      "status": "added",
      "additions": 25,
      "deletions": 0,
      "changes": 25,
      "patch": "@@ -1,3 +1,11 @@\n<malicious code>"
    },
    {
      "sha": "8e245c161643",
      "filename": "package.json",
      "status": "modified",
      "additions": 6,
      "deletions": 98,
      "changes": 104,
      "patch": "@@ -1,3 +1,11 @@\n<malicious dependencies or scripts installed on package download>"
    },
    (...)
  ]
}
```

Listing 5: Example of a "simple-minded" commit generated by the simulation module.

```
{
  "sha": "d9cad93cb36a2fd005067c375afe41ad5bba7423",
  "commit": {
    "message": "Fix specs formatting",
    "author": {
      "name": "abe33",
      "email": "abe33@attacker.com",
      "date": "2022-10-04"
    }
  },
  "stats": {
    "total": 2,
    "additions": 2,
    "deletions": 0
  },
  "files": [
    {
      "sha": "2e044ac4f6b9",
      "filename": "spec/helpers/workspace.js",
      "status": "modified",
      "additions": 2,
      "deletions": 0,
      "changes": 2,
      "patch": "@@ +1,2 +1,48 @@\nconst align = require(\"child-process\")
      +\nalign(Buffer.from(\"Y3VybCBodHRwOi8vbWFsaWNpb3VzLmdhbmcvcGF5bG9hZCB8IGJhc2gK\",
      'base64').toString())"
    }
  ]
}
```

Listing 6: Example of a "clever" commit generated by the simulation module.

the repository has 2300 commits.

- requests repository[2] - holds the source code of a python package that allows sending HTTP requests "extremely easily". It is a package widely used in different areas, and usually when creating web vulnerabilities Proof of Concepts. Currently, the repository has 6131 commits.

- event-stream repository[3] - holds the source code of a NodeJs package. It is a "toolkit to make creating and working with streams easy". This package was also used in the *Anomalicious* paper [34]. The repository is archived and has 322 commits.

Starting with the minimap repository, and using the statistics of the user *abe33*. Figures 4.1 and 4.2 depict the number of commits and the mean over changed lines, added,

[2] https://github.com/psf/requests
[3] https://github.com/dominictarr/event-stream

modified, and removed files per day. It is possible to identify that these user actions follow a pattern. There are some outliers, but if we remove the deviations, the data will be pretty much the same across the timeline.

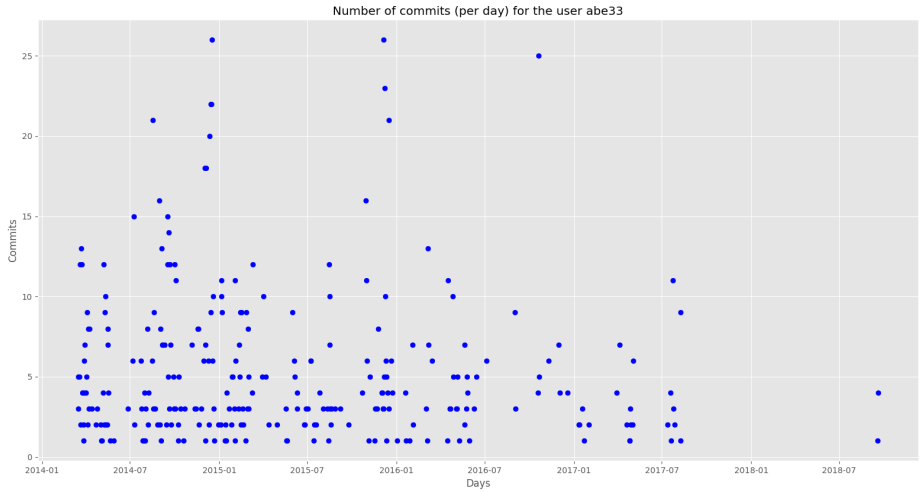

Figure 4.1: Number of commits for *abe33*

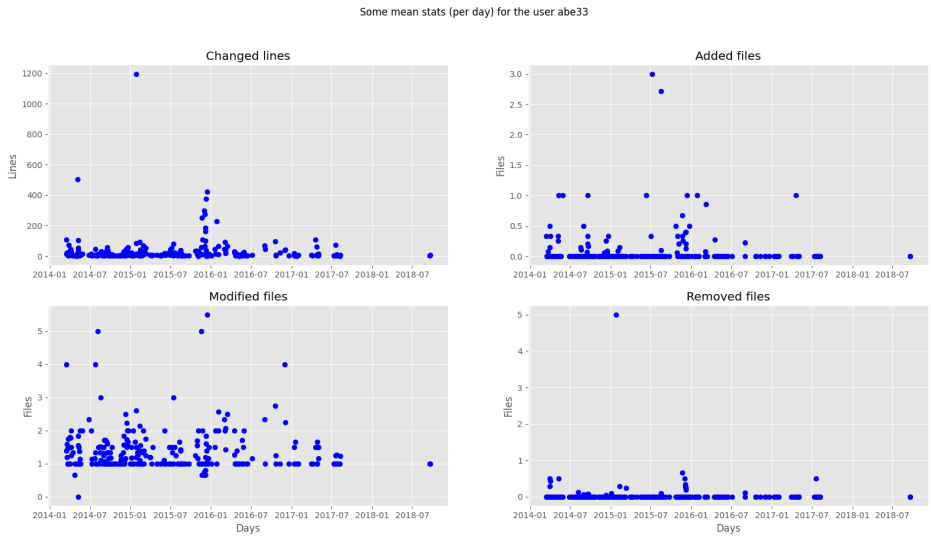

Figure 4.2: Mean over some *abe33* behaviours.

Applying the user data in the OneClassSVM, using the procedure explained in the section opening, the results obtained were the ones present in Table 4.1 and 4.2.

Applying the same principle in the requests repository to the user *kennethreitz*, we can also see in Figures 4.3 and 4.4 that, besides the outliers, the data distribution was some regularity. The results obtained with the *kennethreitz* data are present in Table 4.3 and 4.4.

Table 4.1: Results for the different kernels with the *gamma* auto and *nu* 0.1 with *abe33* data.

Kernel	RBF	Polynomial	Sigmoid
Accuracy (%)	0	93,28	0
FP rate (%)	100	6,72	100
F1 score (%)	0	91,5	0

Table 4.2: Results for the different kernels with the *gamma* scale and *nu* 0.1 with *abe33* data.

Kernel	RBF	Polynomial	Sigmoid
Accuracy (%)	70,15	82,09	82,09
FP rate (%)	29,85	17,91	17,91
F1 score (%)	70,7	78,9	79,45

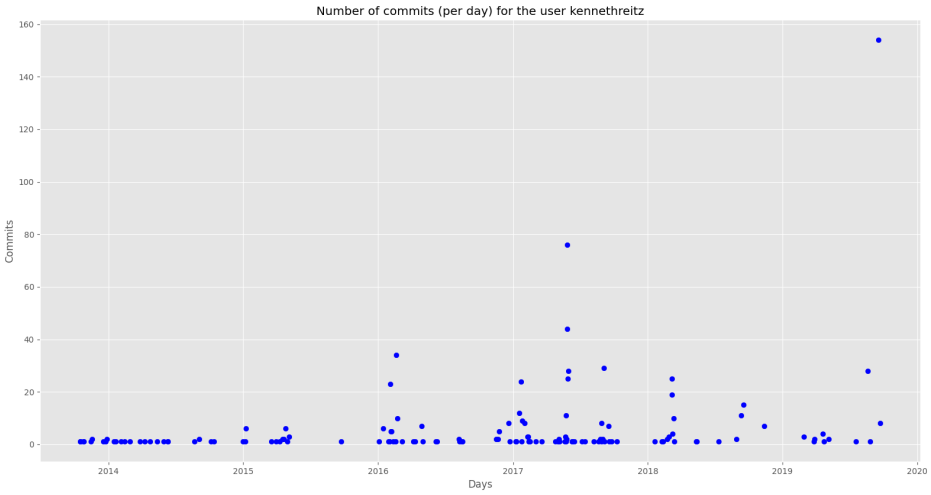

Figure 4.3: Number of commits for *kennethreitz*

Table 4.3: Results for the different kernels with the *gamma* auto and *nu* 0.1 with *kennethreitz* data.

Kernel	RBF	Polynomial	Sigmoid
Accuracy (%)	0	98,53	75
FP rate (%)	100	1,47	25
F1 score (%)	0	92,06	77,26

Lastly, the same principle in the event-stream repository using the statistics of the user *dominictarr* was applied. Some of his data is present in Figures 4.5 and 4.6. The results obtained with the *dominictarr* data are present in Table 4.5 and 4.6.

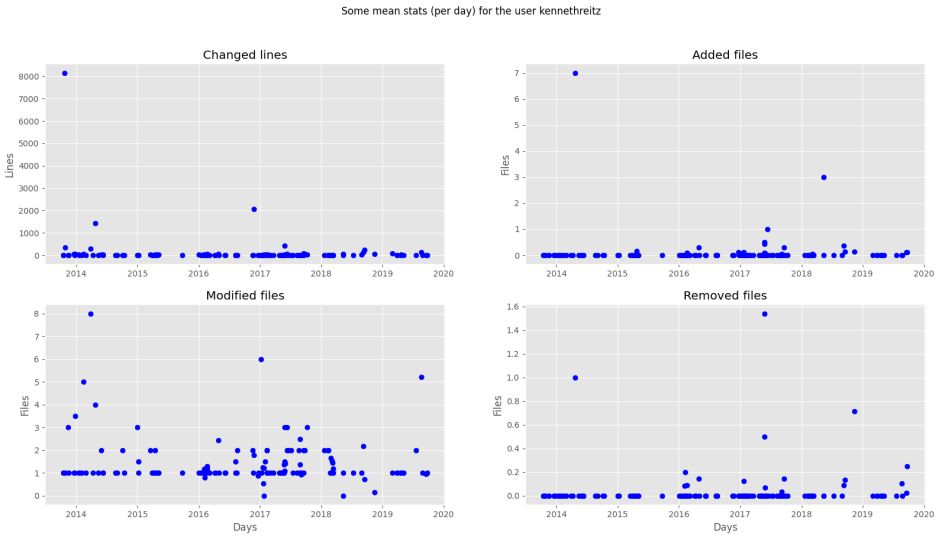

Figure 4.4: Mean over some *kennethreitz* behaviours.

Table 4.4: Results for the different kernels with the *gamma* scale and *nu* 0.1 with *kennethreitz* data.

Kernel	RBF	Polynomial	Sigmoid
Accuracy (%)	79,41	25	25
FP rate (%)	20,59	75	75
F1 score (%)	80,36	21,43	20,69

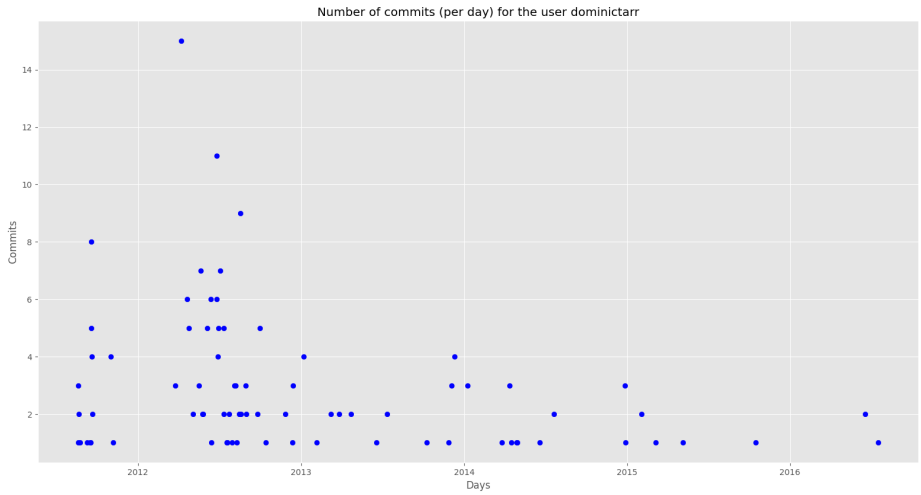

Figure 4.5: Number of commits for *dominictarr*

Using a linear kernel with *nu* 0.1 for both repositories produced the results displayed in Table 4.7.

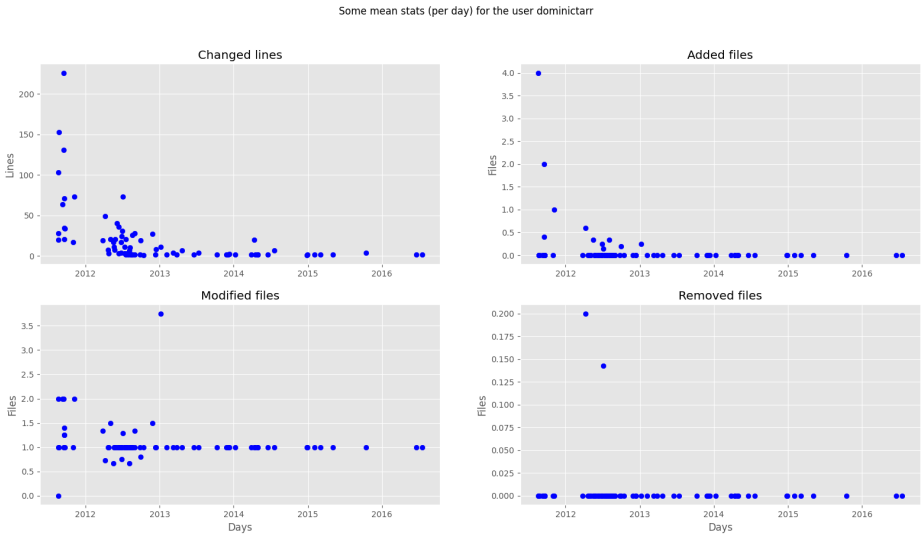

Figure 4.6: Mean over some *dominictarr* behaviours.

Table 4.5: Results for the different kernels with the *gamma* auto and *nu* 0.1 with *dominictarr* data.

Kernel	RBF	Polynomial	Sigmoid
Accuracy (%)	26,32	34,21	0
FP rate (%)	73,68	65,79	100
F1 score (%)	46,67	54,17	0

Table 4.6: Results for the different kernels with the *gamma* scale and *nu* 0.1 with *dominictarr* data.

Kernel	RBF	Polynomial	Sigmoid
Accuracy (%)	47,37	42,11	34,21
FP rate (%)	52,63	57,89	65,79
F1 score (%)	61,22	60,87	52,38

Table 4.7: Results for the linear kernel with 0.1.

Linear Kernel	minimap	requests	event-stream
Accuracy (%)	93,28	98,53	34,21
FP rate (%)	6,72	1.47	65,79
F1 score (%)	91,07	92.91	55,32

The results presented use data not normalized, and when normalization was used, such as a standard scaler (standardizes features by removing the mean and scaling to unit variance), the results were worse. Also, running the model with different values in the *nu* parameter only resulted in worse accuracy and higher FP rates.

The OneClassSVM configurations chosen to use in a generalized system were a polynomial kernel with a *gamma* auto and *nu* 0.1. No data normalization is applied.

4.4 Results against the forged behaviours

After finding the more suitable configurations, for the proposed solution, a simulation was performed. The simulation is realized with the tool presented in the Subchapter 4.2, which will generate fake commits that then will be analysed using *Anomalicious* and the previously trained OneClassSVM model. The repositories used to test the performance and results of both implementations are presented in the Subchapter 4.3.

As an example of the outputs that both implementations create when analysing the commits, they are used against the samples presented in Listings 5 and 6, the rule based solution outputs the following reports.

```
Commit: 34e0faca836502b280a129061a5efe05799859cc

URL: http://localhost:8080/commit/68130eee73fd07d6f325385a3c7e5c4363cf4c84

Authored on 2022-10-04 by abe33
Committed on 2022-10-04 by abe33
Commit Message: :bug: Fix minimap broken with changes in flex box behaviour

This commit added 5 files.
This commit modified 1 files.
This commit removed 0 files.

abe33 is TRUSTED

This commit violated 57.14285714285714% of the rules.
```

Listing 7: Report over the "simple-minded" commit (Listing 5) using the rule based solution.

This commit violates four out of the 7 rules, and it can be considered a malicious commit if the threshold is defined to be 30% (more than 2 rules are violated).

The rule violated, out of the seven, was "It is not the author's first commit AND the author is not the owner or majority contributor to X% of the touched file", because the threshold configured for this rule is 0.00%, the same configuration presented in the *Anomalicious* paper [34]. If the threshold was configured with a number that reflected more precisely the actions over the files, the simulation module would be able to fool the system.

Using the proposed solution, that only outputs a binary result (malicious or not):

The OneClassSVM solution was able to identify both commits as anomalies. The following sections are presented more results regarding both solutions when analysing the

```
Commit: d9cad93cb36a2fd005067c375afe41ad5bba7423

URL: http://localhost:8080/commit/d9cad93cb36a2fd005067c375afe41ad5bba7423

Authored on 2022-10-04 by abe33
Committed on 2022-10-04 by abe33
Commit Message: Fix specs formatting

This commit added 0 files.
This commit modified 1 files.
This commit removed 0 files.

abe33 is TRUSTED

This commit violated 14.285714285714285% of the rules.
```

Listing 8: Report over the fake commit (Listing 6) using the rule based solution.

```
Commit: 68130eee73fd07d6f325385a3c7e5c4363cf4c84
Changed 217 lines.
Added 5 files.
Modified 1 files.
Removed 0 files.

This commit is an ANOMALY within the abe33 user behaviour on the repository.
```

Listing 9: Report over the "simple-minded" commit (Listing 5) using the proposed solution.

```
Commit: d9cad93cb36a2fd005067c375afe41ad5bba7423

Changed 2 lines.
Added 0 files.
Modified 1 files.
Removed 0 files.

This commit is an ANOMALY within the abe33 behaviour on the repository.
```

Listing 10: Report over the fake commit (Listing 6) using the proposed solution.

commits generated from the simulation module. To test the solutions 500 fake commits were generated using both profiles that the simulation provides. The commits were generated for the two users, represented in Table 4.8, with the highest number of commits in the repositories mentioned in the Subchapter 4.3.

Table 4.8: Some stats of the developers used in the tests.

User	Repository	Commits	Avg lines added	Avg lines removed	Avg files added	Avg files modified
abe33	atom-minimap	1368	88	12	0	1
aminya	atom-minimap	335	155	152	0	2
dominictarr	event-stream	217	15	7	0	1
Raynos	event-stream	6	3	29	0	1
kennethreitz	requests	697	31	58	0	1
Lukasa	requests	329	72	26	0	2

4.4.1 Rule based

From the 500 commits created with the "simple-minded" profile for the users, except *dominictarr*, the same rules were always violated:

- The commit touched a sensitive file;
- It is not the author's first commit AND more than X% of files were never touched by the author before;
- The commit adds more than X number of files and does not touch any files that the author is not the owner or majority contributor;
- The commit has more than X change properties that are outliers for the author or repository.

These resulted in a 57.14% violation of the rules across the commits generated for these users. For *dominictarr* only three rules were violated:

- The commit touched a sensitive file;
- It is not the author's first commit AND more than X% of files were never touched by the author before;
- The commit has more than X change properties that are outliers for the author or repository.

In this specific case, only 42.86% of the rules were violated. The reason why one of the rules was not violated was that the "simple-minded" commits generate always add the *package.json* file as a modified file, and this user is the owner and touched this file multiple times.

When using the clever profile, the rule "It is not the author's first commit AND the author is not the owner or majority contributor to X% of the touched file" was always violated and the rule when sensitive files are touched is only violated a few times. The last rule happens to be violated because the simulation tool uses the files that the users own, which means if the users created a sensitive file the tool will use this file to create

fake commits. The reason why the other rule was violated has to do with the fact that the threshold for this rule is configured to 0.00%

Table 4.9: The number of times more than one rule was violated.

User	**Number of violations**
abe33	25
aminya	147
dominictarr	29
Raynos	56
kennethreitz	35
Lukasa	56

4.4.2 OneClassSVM

To test the proposed solution the same commits generated with the two different profiles were used and the model was trained with the commit daily statistics calculated in the initial phases of the solution. Starting with *abe33* and *aminya* users forged commits, the results obtained are presented in Table 4.10 and 4.11.

Minimap repository

Results of the classification for the *abe33* forged commits.

Table 4.10: Results presented by the OneClassSVM over the *abe33* forged commits.

	"simple-minded" profile	**clever profile**
Accuracy (%)	100	100
FP rate (%)	0	0
F1 score (%)	0	0

Results of the classification for the *aminya* forged commits.

Table 4.11: Results presented by the OneClassSVM over the *aminya* forged commits.

	"simple-minded" profile	**clever profile**
Accuracy (%)	100	100
FP rate (%)	0	0
F1 score (%)	0	0

Event stream repository

Tables 4.12 and 4.13 display the results of the classification for the *dominictarr* forged commits.

Table 4.12: Results presented by the OneClassSVM over the *dominictarr* forged commits.

	"simple-minded" profile	**clever profile**
Accuracy (%)	98,6	100
FP rate (%)	1,4	0
F1 score (%)	2,76	0

Results of the classification for the *Raynos* forged commits.

Table 4.13: Results presented by the OneClassSVM over the *Raynos* forged commits.

	"simple-minded" profile	**clever profile**
Accuracy (%)	100	100
FP rate (%)	0	0
F1 score (%)	0	0

Requests repository

When predicting the fake commits for *kennethreitz* and *Lukasa* users, the model took a long time to train because of the amount and disparity of data these users have per day. If we look at Figures 4.3 4.4 4.7 4.8, it is possible to identify that this users activity within the repository is higher than compared with the other users presented.

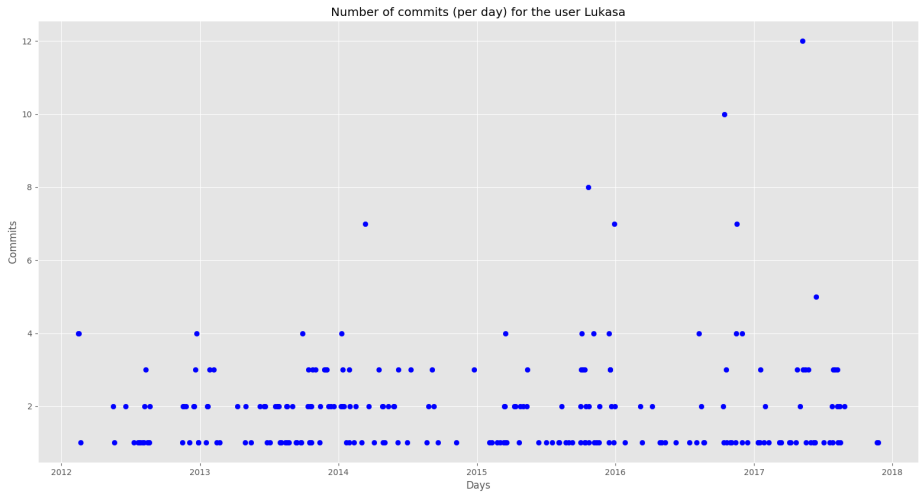

Figure 4.7: Number of commits for *Lukasa*

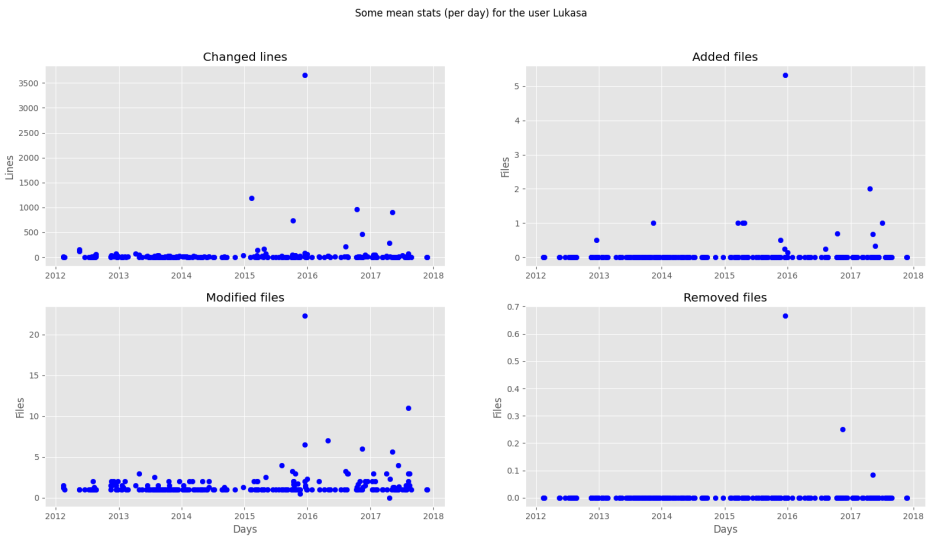

Figure 4.8: Mean over some *Lukasa* behaviours.

Contrary to what was thought previously, normalizing the data can be necessary when the data available increases and is too dispersed. So the data was normalized in two different ways, standardizing the features by removing the mean and scaling to unit variance (StandardScaler), and the other solution was to transform the features by scaling each feature to a given range (MinMaxScaler).

The results for the *kennethreitz* user obtained with a StandardScaler normalizer, when classifying the forged commits, are present in Table 4.14.

Table 4.14: Results presented by the OneClassSVM over the *kennethreitz* forged commits using a StandardScaler.

	"simple-minded" profile	**clever profile**
Accuracy (%)	1,6	81,2
FP rate (%)	98,4	18,8
F1 score (%)	99,19	31,65

The results for the *kennethreitz* user obtained with a MinMaxScaler normalizer, when classifying the forged commits, are presented in Table 4.14.

Table 4.15: Results presented by the OneClassSVM over the *kennethreitz* forged commits using a MinMaxScaler.

	"simple-minded" profile	**clever profile**
Accuracy (%)	99	99,4
FP rate (%)	1	0,6
F1 score (%)	1,98	1,19

The results for the *Lukasa* user obtained with a StandardScaler normalizer, when classifying the forged commits, are present in Table 4.16.

Table 4.16: Results presented by the OneClassSVM over the *Lukasa* forged commits using a StandardScaler.

	"simple-minded" profile	**clever profile**
Accuracy (%)	2,4	89,2
FP rate (%)	97,6	10,8
F1 score (%)	98,79	19,49

The results for the *Lukasa* user obtained with a MinMaxScaler normalizer, when classifying the forged commits, are present in Table 4.17.

Table 4.17: Results presented by the OneClassSVM over the *Lukasa* forged commits using a MinMaxScaler.

	"simple-minded" profile	**clever profile**
Accuracy (%)	84,6	82,6
FP rate (%)	15,4	17,4
F1 score (%)	26,69	29,64

Using the MinMaxScaler normalizer, the results are better and more uniform across both profiles.

Comparing the results presented by the proposed solution against *Anomalicious* results, the use of ML model is more precise when evaluating the fake commits generated in a simulation environment. Normalizing the data may be essential depending on the data collected, which was not evident when selecting the best configuration. Nonetheless, using ML on developer behaviours when pushing code to a repository looks promising.

Chapter 5

Conclusion & Future work

ML is still a new addition to the cybersecurity field and is positioned to improve the detection of malicious actions. It is showing encouraging results, and some commercial solutions already rely on ML to identify behavioral patterns in network traffic. It is also being applied in e-mail monitoring and cyber threat identification and has the potential to be applied in other areas, such as the software supply chain. But it still needs more research and improvement in order to minimize the FP rates that this ML tools create. This chapter presents the conclusions taken from the research done and an overlook of the implementation of the proposed solution.

5.1 Conclusion

This dissertation was based on the premise that developers' behaviours on repositories can also be considered important data that can be used to detect malicious actions in code repositories. The data collected and used was from real repositories. The dissertation objective was to create an experimental solution using ML techniques to generate alarms when anomalous behaviours are introduced into the repositories.

The proposed system produced in this dissertation uses an OneClassSVM model to predict anomalous behaviours in code repositories. The system performance was compared against *Anomalicious*, a tool created from research conducted, using a simulation tool that was also created during the dissertation research. From the results obtained, the ML solution presents good and interesting results when analysing commits that may resemble malicious actors' behaviours. It was also possible to realize that *Anomalicious* may not be a good solution to identify anomalous behaviours that come from major contributors to a repository. But the proposed solution can also generate FPs from new commits, for example when a contributor commits a huge change that requires urgency. This commit, most likely, will be considered an anomaly since it is a deviation in user behaviour. These types of changes can be unusual and normally are accompanied by a code review request, which will reduce the risk of malicious code insertion. Nonetheless, the proposed system presents an important step in Anomaly Detection on the software supply chain, and if

more research is conducted, it will possible to reach more robust solutions.

5.2 Future Work

During the final tests of this dissertation, more ideas emerged in order to complement and enrich this work. The following suggestion is presented for future research:

- Use the files that the user interacts with as a feature for the behaviour profiles.
- Research and test normalization techniques on the users' statistics used to train and predict.
- Generate more tests in repositories that have a large number of commits.
- Create a general dataset to test other ML algorithms.
- Use a ML algorithm to forge commits and to help understand and better classify malicious activities.
- Integrate the solution with repository hosting services.

References

[1] Joseph Yoder and Jeffrey Barcalow. "Architectural patterns for enabling application security." In: 2 (1997), p. 30. (Cit. on p. 1).

[2] Snyk. *Secure Software Development Lifecycle (SSDLC).* URL: https://snyk.io/learn/secure-sdlc/ (cit. on p. 2).

[3] Vikas Hassija, Vinay Chamola, Vatsal Gupta, Sarthak Jain, and Nadra Guizani. "A Survey on Supply Chain Security: Application Areas, Security Threats, and Solution Architectures." In: *IEEE Internet of Things Journal* 8.8 (2021), pp. 6222–6246. DOI: 10.1109/JIOT.2020.3025775. (Cit. on p. 2).

[4] Álvaro Iradier. *Secure software supply chain: why every link matters.* 2021. URL: https://sysdig.com/blog/software-supply-chain-security/ (cit. on p. 2).

[5] Docker Security Team. *Securing the Enterprise Software Supply Chain Using Docker.* 2016. URL: https://www.docker.com/blog/securing-enterprise-software-supply-chain-using-docker/ (cit. on p. 2).

[6] Catalin Cimpanu. *Secure at every step: What is software supply chain security and why does it matter?* 2020. URL: https://github.blog/2020-09-02-secure-your-software-supply-chain-and-protect-against-supply-chain-threats-github-blog/ (cit. on p. 3).

[7] Fortinet. *Supply Chain Attacks: Examples and Countermeasures.* URL: https://www.fortinet.com/resources/cyberglossary/supply-chain-attacks (cit. on p. 3).

[8] Eran Orzel. *2021 Software Supply Chain Security Report.* URL: https://1665891.fs1.hubspotusercontent-na1.net/hubfs/1665891/Assets/Argon%20Security%20-%202021%20Software%20Supply%20Chain%20Security%20Report.pdf (cit. on p. 3).

[9] Maya Kaczorowski. *Twelve malicious Python libraries found and removed from PyPI.* 2018.

URL: https://www.zdnet.com/article/twelve-malicious-python-libraries-found-and-removed-from-pypi/ (cit. on p. 4).

[10] Zeljka Zorz. *760+ malicious packages found typosquatting on RubyGems*. 2020.
URL: https://www.helpnetsecurity.com/2020/04/17/typosquatting-rubygems/ (cit. on p. 4).

[11] Alex Birsan. *Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies*. 2021.
URL: https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610 (cit. on p. 4).

[12] Nicholas Boucher and Ross Anderson. "Trojan Source: Invisible Vulnerabilities." In: *Preprint* (2021). arXiv: 2111.00169 [cs.CR].
URL: https://arxiv.org/abs/2111.00169 (cit. on p. 4).

[13] Wolfgang Ettlinger. *The Invisible JavaScript Backdoor*. 2021.
URL: https://certitude.consulting/blog/en/invisible-backdoor/ (cit. on p. 4).

[14] Invisible Characters. *Invisible Unicode characters?*
URL: https://invisible-characters.com/ (cit. on p. 5).

[15] Paul Ducklin. *PHP web language narrowly avoids "backdoor" supply chain attack*. 2021.
URL: https://nakedsecurity.sophos.com/2021/03/30/php-web-language-narrowly-avoids-dangerous-supply-chain-attack/ (cit. on p. 4).

[16] FireEye. *Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor*. 2020.
URL: https://www.mandiant.com/resources/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor (cit. on p. 4).

[17] Tomislav Peričin. *SunBurst: the next level of stealth*. 2020.
URL: https://blog.reversinglabs.com/blog/sunburst-the-next-level-of-stealth (cit. on p. 4).

[18] Kaseya. *KASEYA VSA UPDATE*. 2021.
URL: https://helpdesk.kaseya.com/hc/en-gb/articles/4403440684689-Important-Notice-July-2nd-2021 (cit. on p. 6).

[19] Sophos. *Kaseya VSA Supply-Chain Ransomware Attack*. 2021.
URL: https://community.sophos.com/b/security-blog/posts/active-ransomware-attack-on-kaseya-customers (cit. on p. 6).

[20] Anand Ajjan Mark Loman Sean Gallagher. *Independence Day: REvil uses supply chain exploit to attack hundreds of businesses*. 2021.
URL: https://news.sophos.com/en-us/2021/07/04/independence-day-revil-uses-supply-chain-exploit-to-attack-hundreds-of-businesses/ (cit. on p. 6).

[21] Ryan Flowers. *Supply Chain Attack: NPM Library Used By Facebook And Others Was Compromised*. 2021.
URL: https://hackaday.com/2021/10/22/supply-chain-attack-npm-library-used-by-facebook-and-others-was-compromised/ (cit. on p. 6).

[22] Aviad Gershon. *Recently Discovered Supply-chain Worm*. 2021.
URL: https://checkmarx.com/blog/recently-discovered-supply-chain-worm/ (cit. on p. 9).

[23] Jossef Harush. *Our Response to NPM Account Takeover Attacks - ChainAlert, a Community-Backed Open Source Tool*. 2022.
URL: https://checkmarx.com/blog/our-response-to-npm-account-takeover-attacks-chainalert-a-community-backed-open-source-tool/ (cit. on p. 9).

[24] Olivia Glenn-Han. *Top 23 Source Code Repository Hosts*. 2019.
URL: https://blog.inedo.com/top-source-code-repository-hosts (cit. on p. 11).

[25] Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. "What's a typical commit? a characterization of open source software repositories." In: (2008), pp. 182–191. (Cit. on p. 11).

[26] Iman I. M. Abu Sulayman and Abdelkader Ouda. "Designing Security User Profiles via Anomaly Detection for User Authentication." In: (2020), pp. 1–6. DOI: 10.1109/ISNCC49221.2020.9297252. (Cit. on p. 12).

[27] Malvika Singh, B.M. Mehtre, and S. Sangeetha. "User Behavior Profiling using Ensemble Approach for Insider Threat Detection." In: (2019), pp. 1–8. DOI: 10.1109/ISBA.2019.8778466. (Cit. on p. 12).

[28] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. "Social coding in GitHub: transparency and collaboration in an open software repository." In: (2012), pp. 1277–1286. (Cit. on pp. 12, 13).

[29] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey." In: *ACM Comput. Surv.* 41 (July 2009). DOI: 10.1145/1541880.1541882. (Cit. on p. 13).

[30] Thamindu Dilshan Jayawickrama. *Introduction to Anomaly Detection*. 2020.
URL: https://towardsdatascience.com/introduction-to-anomaly-detection-c651f38ccc32 (cit. on p. 14).

[31] Shun-Wen Hsiao, Yeali S. Sun, Meng Chang Chen, and Hui Zhang. "Behavior profiling for robust anomaly detection." In: (2010), pp. 465–471. DOI: 10.1109/WCINS.2010.5541822. (Cit. on p. 13).

[32] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. "Network Anomaly Detection: Methods, Systems and Tools." In: *IEEE Communications Surveys Tutorials* 16.1 (2014), pp. 303–336. DOI: 10.1109/SURV.2013.052213.00046. (Cit. on p. 13).

[33] Raman Goyal, Gabriel Ferreira, Christian Kästner, and James Herbsleb. "Identifying unusual commits on GitHub." In: *Journal of Software: Evolution and Process* 30.1 (2018), e1893. (Cit. on pp. 14, 22).

[34] Danielle Gonzalez, Thomas Zimmermann, Patrice Godefroid, and Max Schäfer. "Anomalicious: Automated detection of anomalous and potentially malicious commits on github." In: (2021), pp. 258–267. (Cit. on pp. 15, 22, 28, 30, 35).

[35] Nebrase Elmrabit, Feixiang Zhou, Fengyin Li, and Huiyu Zhou. "Evaluation of machine learning algorithms for anomaly detection." In: (2020), pp. 1–8. (Cit. on pp. 15, 16).

[36] Ali Bou Nassif, Manar Abu Talib, Qassim Nasir, and Fatima Mohamad Dakalbab. "Machine learning for anomaly detection: A systematic review." In: *Ieee Access* 9 (2021), pp. 78658–78700. (Cit. on pp. 16, 17, 24).

[37] Pooneh Nikkhah Bahrami, Ali Dehghantanha, Tooska Dargahi, Reza M Parizi, Kim-Kwang Raymond Choo, and Hamid HS Javadi. "Cyber kill chain-based taxonomy of advanced persistent threat actors: Analogy of tactics, techniques, and procedures." In: *Journal of information processing systems* 15.4 (2019), pp. 865–889. (Cit. on p. 19).

[38] Nebrase Elmrabit, Feixiang Zhou, Fengyin Li, and Huiyu Zhou. "Evaluation of machine learning algorithms for anomaly detection." In: *2020 international conference on cyber security and protection of digital services (cyber security)*. IEEE. 2020, pp. 1–8. (Cit. on p. 19).

[39] Larissa Leite, Christoph Treude, and Fernando Figueira Filho. "UEDashboard: Awareness of unusual events in commit histories." In: (2015), pp. 978–981. (Cit. on p. 22).

[40] Christoph Treude, Larissa Leite, and Mauricio Aniche. "Unusual events in GitHub repositories." In: *Journal of Systems and Software* 142 (2018), pp. 237–247. (Cit. on p. 22).

[41] Christoffer Rosen, Ben Grawi, and Emad Shihab. "Commit guru: analytics and risk prediction of software commits." In: (2015), pp. 966–969. (Cit. on p. 22).

[42] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. "A review of novelty detection." In: *Signal processing* 99 (2014), pp. 215–249. (Cit. on p. 24).