Vulnerability Exploitability Prediction and Risk Evaluation

JIAO YIN

Bachelor of Engineering, 2010

Master of Engineering, 2013

A thesis submitted in total fulfilment of the requirements for the degree of
Doctor of Philosophy

Department of Computer Science and Information Technology

School of Engineering and Mathematical Sciences

La Trobe University

Victoria Australia

November 2021

Contents

List of Figures

List of Tables

Abstract

Software vulnerability evaluation and assessment is a fundamental and systematic review process to identify the severity levels of existing vulnerabilities in cybersecurity. It is essential for information systems to find out potential security weaknesses and thus to recommend remediation or mitigation actions in a timely manner. Existing studies and tools on vulnerability assessment are criticized for apparent flaws in vulnerability exploitability prediction and analysis. This thesis aims to enhance the performance of exploitability prediction and analysis by leveraging advanced deep learning and knowledge graph techniques, thus helping improve existing vulnerability risk evaluation and management systems.

The thesis first focuses on the binary vulnerability prediction problem and proposes a framework named ExBERT to predict if a vulnerability will be exploited or not. Further, a novel consecutive batch learning algorithm, called Real-time Dynamic Concept Adaptive Learning (RDCAL), is proposed to deal with concept drift and dynamic class imbalance problems in online learning scenarios. Then, the thesis predicts the probable vulnerability exploitation time as a multiclass online learning problem and accordingly proposes an Adaptive Sliding Window Weighted Learning (ASWWL) algorithm. Finally, this thesis explores the co-exploitation behaviour discovery problems by formulating it as a link prediction problem between vulnerability entities within a cybersecurity domain-specific knowledge graph. A general Modality-Aware Graph Convolutional Network module and a Graph Knowledge Transfer Learning strategy are designed for link prediction.

The thesis contributes to the deep learning community by developing more accurate and domain-specific algorithms and models for binary and multiclass

classification, online learning, data imbalance handling and graph embedding and representation. It also contributes to the cybersecurity community by providing more accurate vulnerability exploitability prediction and exploitation behaviour analysis, which are essential for vulnerability risk evaluation and management.

Statement of Authorship

This thesis includes work by the author that has been published or accepted for publication as described in the text. Except where reference is made in the text of the thesis, this thesis contains no other material published elsewhere or extracted in whole or in part from a thesis accepted for the award of any other degree or diploma. No other person's work has been used without due acknowledgement in the main text of the thesis. This thesis has not been submitted for the award of any degree or diploma in any other tertiary institution.

Jiao Yin
20 November 2021

Acknowledgements

This work was supported by a La Trobe University Postgraduate Research Scholarship and a La Trobe University Full Fee Research Scholarship.

During the completion of this dissertation and my candidature, many individuals have given me continuous support and help. I would like to express my hearty appreciation and gratitude to them, without which it would have been impossible to complete this thesis.

First and foremost, I am tremendously grateful to my principal supervisor, mentor and friend, A/Prof. Jinli Cao, who provided me with continuous assistance, guidance, support and encouragement throughout my candidature. As a supervisor and mentor, she organized regular meetings with me to discuss my research progress and give me specific suggestions and guidance. As a friend, she invited me to attend private holiday dinners with her family and friends, which warmed my heart during my journey of studying abroad. I feel so lucky and appreciated to have had a supervisor like Jinli in my life.

Special thanks must go to my co-supervisor, Dr MingJian Tang, a Principal Data Scientist with comprehensive industry experience in cybersecurity, advanced machine learning and deep artificial intelligence. Although busy with his work, he always made time to join the regular meetings with Jinli and me. His valuable research experience and professional knowledge in cybersecurity provided my research project with an in-depth industrial perspective. He also gave me many suggestions and guidance from a practical viewpoint.

I am grateful to have had the opportunity to learn from Prof. Hua Wang from Victoria University, a project collaborator of A/Prof. Jinli Cao. Thanks to Prof. Hua for inviting me to join the Applied Informatics Research group at Victoria

University and allowing me to attend their weekly general meetings and learn from their knowledgeable mentor team as well as their helpful and supportive peers. I would like to especially thank Professor Yanchun Zhang, Dr Siuly Siuly, Ms Jiaying Kou and Dr Fan Liu from their research team for providing me with valuable advice and support.

My sincere thanks to my co-supervisor, Dr Nasser Sabar and my Progress Committee Chair, Prof. Wenny Rahayu. They provided me with important suggestions and gave me their valuable time to help me improve the research quality.

I would like to acknowledge Prof. Phoebe Chen, Prof. Zhen He and Mr Haritha Thilakarathne for allowing me to use the La Trobe GPU research cluster. Mr Haritha Thilakarathne did volunteer work to maintain the system and provide help for the users. I appreciate his excellent job and valuable time.

I would like to express my gratitude to my friends and colleagues in the Department of Computer Science and Information Technology, La Trobe University, Dr Yuehua Liu, Mr Yong-Feng Ge, Mr Syed Qasim Abbas, Dr Wenjin Yu, Miss Xi Cao, Miss Yuehan Du, Mr Junqi Li, Dr Ming Li, Dr Caihao Cui, Dr Jing Meng, Mr Szu-Chi Chen, Dr Tu Doan Quang. Thank you for the companionship and numerous discussions during my research candidature.

Finally, I would like to express my indebtedness to my family. Thank you, my parents, for your upbringing and unconditional support of my education since childhood. Thanks, my dear husband, for your sacrifices, encouragement and support. I cannot imagine how I would have gone through these difficult lock-down days caused by COVID-19 if you had not left your job and moved to Australia to be with me. The love and support from my family always give me endless motivation and courage to face the challenges in my life and work.

Jiao Yin
Melbourne, November 2021

Publications

Some of the ideas, data, results and figures presented in this thesis have been published or are under review throughout my PhD candidature in journals, conference proceedings and book chapters. I declare that I, Jiao Yin, was the leading contributor and primary author for these works. The author has the permission of the publishers to reproduce the contents of these publications for academic purposes. See below for an exhaustive list of the aforementioned publications.

Published papers:

[1] **Yin, Jiao**, MingJian Tang, Jinli Cao, and Hua Wang. "Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description." Knowledge-Based Systems 210 (2020): 106529.

[2] **Yin, Jiao**, MingJian Tang, Jinli Cao, Hua Wang, Mingshan You, and Yongzheng Lin. "Adaptive Online Learning for Vulnerability Exploitation Time Prediction." In International Conference on Web Information Systems Engineering, pp. 252-266. Springer, Cham, 2020.

[3] **Yin, Jiao**, MingJian Tang, Jinli Cao, Hua Wang, Mingshan You, and Yongzheng Lin. "Vulnerability exploitation time prediction: an integrated framework for dynamic imbalanced learning." World Wide Web (2021): 1-23.

[4] **Yin, Jiao**, Ming Jian Tang, Jinli Cao, Hua Wang, and Mingshan You. "A Real-time Dynamic Concept Adaptive Learning Algorithm for Exploitability Prediction." Neurocomputing (2021).

Paper under review:

[5] **Yin, Jiao**, MingJian Tang, Jinli Cao, Mingshan You, Hua Wang, Mamoun Alazab. "Knowledge-Driven Cybersecurity Intelligence: Software Vulnerability Co-exploitation Behaviour Discovery", submitted to IEEE Transactions on Industrial Informatics (2022).

Accepted Book Chapter:

[6] **Yin, Jiao**, MingJian Tang, Jinli Cao, Mingshan You, Hua Wang. "Cybersecurity Applications in Software: Data-Driven Software Vulnerability Assessment and Management", to be included as a chapter in the "Cybersecurity Applications" book published by Springer (2022).

Chapter 1

Introduction

1.1 Motivation

As global data expands at exponential rates in terms of volume, velocity, variety and complexity, both commercial and personal data are increasingly exposed to the threat of network attacks caused by software vulnerabilities [1, 2]. Each year, thousands of software vulnerabilities are archived and disclosed to the public, and the scale and types of risks from cyber threats expand proportionately [3]. More than 160,000 vulnerabilities have been disclosed to the public through open-source databases so far. Bilge and Dumitras pointed out that once a vulnerability is disclosed, the opportunity of being exploited increases by five orders of magnitude[4].

With the increasing number of vulnerabilities, it is impossible to patch every vulnerability in a timely manner. To effectively and efficiently allocate budget and resources and make emergency plans in advance, vulnerability assessment and management has become a crucial measure for both a single organisation and the entire cybersecurity community to protect information systems and networks from cyberattacks, intrusions, malware and various types of data breaches [5, 6].

Based on the fact that only 10-–15% of vulnerabilities actually ever have a known exploit, and even fewer are weaponised as part of hacking toolkits [7, 8], a critical challenge for vulnerability risk management is to make a reasonable

trade-off between coverage and efficiency. On the one hand, firms are trying to patch as many disclosed vulnerabilities as possible to provide the highest level of protection. On the other hand, they have to deprioritise most relatively low-risk vulnerabilities and only focus on these high-risk ones due to limited vulnerability patching and remediation resources. Therefore, vulnerability exploitation analysis has become one of the most vital tasks for vulnerability assessment and management.

Considering the writing, testing and installation costs of patching software vulnerabilities, exploitation analysis is vital to identify the vulnerabilities most likely to be exploited accurately. The results of exploitation analysis would be used to evaluate the severity of software vulnerabilities and thus prioritise the order of patching.

However, existing solutions for exploitation analysis in both industry and academia still have a long way to go to meet the requirements of modern information systems. For example, the de facto industry standard for vulnerability assessment, Common Vulnerability Scoring System (CVSS), only provides an exploitability score between 0-10 to indicate the overall exploitability severity. This exploitability score is proven to be inconsistent with the real-world exploitation results [7]. In academia, researchers adopted a variety of machine learning and deep learning models and algorithms to predict the exploitability of vulnerabilities, where vulnerability features were extracted from publicly available information, including descriptions, CVSS metrics, social media streams, etc. [9, 10, 11]. Although promising results were achieved on the prediction of exploitability, more in-depth research is demanded on a varied range of topics, such as the exploitability prediction in online learning scenarios, exploitation time prediction and the analysis of interrelationships of vulnerabilities affected the same product.

To help build a better vulnerability assessment model and thus make better decisions on vulnerability management, this thesis focuses on vulnerability exploitation analysis. Specifically, this thesis tackles the following tasks.

(1) Vulnerability exploitability prediction. This thesis analyses the exploitability of a vulnerability based on its publicly available information. Further, this thesis builds a novel model along with comprehensive features to predict how likely a new software vulnerability will be exploited.

(2) Vulnerability exploitation time prediction. If a vulnerability will be exploited, this thesis analyses the possible exploitation time period of the vulnerability. Further, this thesis builds a novel model along with comprehensive features to predict the possible exploitation time period of a new software vulnerability.

(3) Vulnerability co-exploitation behaviour discovery. This thesis also analyses the relationships between different vulnerabilities and predicts if two or more vulnerabilities will be co-exploited by the same exploit. More specifically, this thesis proposes a novel model based on a knowledge graph to predict if co-exploitation links exist between different vulnerabilities to cause large scale and severe cyber-attacks.

1.2 Challenging Problems

This section analyses the challenging problems associated with each task separately.

1.2.1 Challenging Problems in Exploitability Prediction

Predicting the exploitability of vulnerabilities is crucial for decision-makers to prioritise their efforts and patch the most critical vulnerabilities.

The Common Vulnerability Scoring System (CVSS), the de facto industry standard for vulnerability assessment, is widely criticised for the inconsistency between CVSS scores and the exploitability of vulnerabilities.

In the academic community, researchers have tried to build binary classification models to accurately predict the exploitability of a vulnerability based on its description. However, the main challenge comes from the lack of an effective semantic feature extraction method due to the small size of description corpus.

Furthermore, previous studies built exploitability prediction with a batch learning mode, which actually made a strong assumption that the data distribution is static over time. Therefore, these studies failed to consider the concept drift problem due to the evolving user behaviours and system environments.

Therefore, better methods are expected for inventors and decision-makers to predict the exploitability of vulnerabilities.

1.2.2 Challenging Problems in Exploitation Time Prediction

Exploitation time is an essential factor for vulnerability assessment in cybersecurity management. Previous works always focus on predicting whether a vulnerability will be exploited or not and ignore predicting the possible exploitation time for exploitable vulnerabilities. The exploitation time prediction problem can be formulated as an online multiclass prediction problem, which faces at least three challenging problems.

Firstly, because of the increased data complexity and the imbalanced one-vs-all data distribution for each class, the multiclass classification task itself is inherently more challenging than binary classification problems [12].

The second challenge comes from the dynamic multiclass data imbalance status existing in an online learning setting. When samples with randomly distributed multiclass labels are fed into the classifier as a data stream, the real-time data imbalance statuses dynamically change over time [13]. Because of lacking prior knowledge of the whole data, some specifically designed techniques should be employed to detect and track the class imbalance pattern over time [14].

Thirdly, compared with offline learning, online learning may involve new concepts when new samples arrive [15]. Therefore, strategies to deal with possible concept drift problems should be designed to ensure no dramatic performance decrease occurs when making predictions on new samples.

1.2.3 Challenging Problems in Co-exploitation Behaviour Discovery

Co-exploitation behaviour can reveal intricate relationships between different vulnerabilities. However, to the best of our knowledge, existing works all make exploitation predictions and vulnerability risk analysis for a single vulnerability based on the available information of every single vulnerability. No previous works tried to make predictions on the relationships between different vulnerabilities. Traditional machine learning and deep learning models are powerful to extract latent features from multimodal information sources for every single

sample [16, 17, 18]. However, they are incapable of discovering and embedding relationships between different samples from either homologous or heterogeneous entities. Therefore, as the first research work on co-exploitation behaviour discovery, the possible challenging problems are as follows:

(1) How to formulate the co-exploitation behaviour discovery problem into a general machine learning or deep learning task? As previously mentioned, the exploitability prediction problem and exploitation time prediction problem can be formulated as a binary classification problem and a multiclass classification problem accordingly. However, as each co-exploitation behaviour may involve an indefinite number of vulnerabilities and exploits, it is inappropriate to simply formulate the co-exploitation behaviour discovery problem as a binary classification problem.

(2) How to construct a labelled dataset for the co-exploitation behaviour discovery problem? As the first work on co-exploitation behaviour discovery, there is no well-structured publicly available dataset for direct usage. Considering that co-exploitation behaviour may span several years and involve an uncertain number of vulnerabilities and exploits which are from different data sources, it is a challenge to collect and construct a labelled co-exploitation behaviour discovery dataset.

(3) How to embed features from vulnerabilities, exploits and their relationships into a unified feature space? To make decisions on whether co-exploitation behaviour exists, information on both the involved vulnerabilities and exploits, as well as their relationships, should be taken into consideration. A vast volume of information from diverse sources, including credible vendors, cybersecurity experts, technical posts and even social media, would be involved. This information may include videos, photos, speeches, reports, code snippets and texts. It is challenging to properly fuse and embed this multimodality information into a unified feature space.

1.3 Contributions

To partly solve the aforementioned problems, this thesis leverages advanced artificial intelligence techniques, including but not limited to machine learning, deep learning and knowledge graphs, to provide feasible solutions for effective and accurate exploitability prediction, exploitation time prediction and co-exploitation behaviour discovery. The contributions of this thesis are both theoretical and practical.

1.3.1 Theoretical Contributions

The thesis contributes to the artificial intelligence community by developing more accurate and domain-specific algorithms and models for binary and multiclass classification, online learning, data imbalance handling and graph embedding and representation. Specifically, the theoretical contributions of the thesis are as follows.

(1) The thesis proposes a binary classification framework named ExBERT to address the challenge of lacking an effective semantic feature extraction method due to the small size of the description corpus. ExBERT firstly applies a transfer learning strategy to fine-tune a pre-trained Bidirectional Encoder Representations from Transformers (BERT) model with a domain-specific corpus to extract semantic features. Then, a pooling layer and a long short term memory (LSTM) classification layer are added on top of the fine-tuned BERT model to make binary classification decisions. ExBERT can be used as a general text-based binary classification framework for scenarios where the size of the domain-specific corpus is too small to train a comprehensive natural language processing (NLP) model alone.

(2) The thesis proposes a general online binary classification framework called Real-time Dynamic Concept Adaptive Learning (RDCAL) to address the concept drift problem due to the evolving user behaviours and system environments. RDCAL is a consecutive batch learning framework containing a Class Rectification Strategy (CRS) and a Balanced Window Strategy (BWS) [19]. Under online learning scenarios, CRS is a general algorithm to handle the actual drift problem

in sample labels, and BWS is a general algorithm to tackle the dynamic class imbalance problem.

(3) To address the challenging problems in exploitation time prediction, this thesis formulates the exploitation time prediction problem as a multiclass classification problem. An Adaptive Sliding Window Weighted Learning (ASWWL) algorithm is designed to handle the combined challenges posed by multiclass classification, multiclass imbalance and online learning. Specifically, a Sliding Window Imbalance Factor (SWIF) is designed to trace the dynamic imbalanced status of each class in real-time.

(4) To handle the challenging problems in co-exploitation behaviour discovery, this thesis formulates the co-exploitation behaviour discovery problem as a link prediction problem. Then, a general Modality-Aware Graph Convolutional Network (MAGCN) module is proposed to enhance graph embedding and representation capability with multimodality node properties. Furthermore, this thesis also proposes a general Graph Knowledge Transfer Learning (GKTL) strategy to transfer node embeddings between different subgraphs extracted from the same knowledge graph. Both MAGCN and GKTL can be used in graph-based knowledge representation and reasoning applications.

1.3.2 Practical Contributions

From a practical perspective, the thesis contributes to the cybersecurity community by providing more accurate vulnerability exploitability prediction and exploitation behaviour analysis, which are essential for vulnerability risk evaluation and management. Specifically, the practical contributions of the thesis are as follows.

(1) The thesis successfully applies ExBERT to solve a practical problem in cybersecurity, i.e. the vulnerability exploitability prediction problem. The performance of ExBERT is verified on 46,176 real-world vulnerabilities.

(2) The thesis successfully applies RDCAL, as well as CRS and BWS to solve the real-time vulnerability exploitability prediction problem. This is the first work to undertake exploitability prediction in an online learning mode to the best

of our knowledge. The performance of RDCAL is also verified by experiments conducted on real-world vulnerabilities.

(3) The thesis successfully applies ASWWL and SWIFT to solve the exploitation time prediction problem in cybersecurity. To the best of our knowledge, this is the first work to undertake exploitation time prediction in an online learning mode. The performance of ASWWL is verified by the results of experiments on a real-world dataset.

(4) The thesis formulates the vulnerability co-exploitation behaviour discovery problem as a link prediction problem between vulnerability entities within a cybersecurity domain-specific knowledge graph. This is the first work on vulnerability co-exploitation behaviour discovery to the best of our knowledge. The thesis successfully applies MAGCN and GKTL to boost the link prediction performance.

1.4 Thesis Outline

This thesis consists of 7 chapters, including this one. The rest of the thesis is organised as follows.

Chapter 2 introduces the background and foundations of vulnerability assessment and risk evaluation, including the terminologies in cybersecurity, the databases and datasets used in this thesis, the latest development and progress, as well as existing problems of vulnerability assessment and exploitability analysis in both industry and academia.

Chapter 3 elaborates a framework named ExBERT to predict if a vulnerability will be exploited or not accurately. ExBERT essentially is an improved BERT model for exploitability prediction. Results on 46,176 real-world vulnerabilities demonstrate that the proposed ExBERT framework achieves 91.12% for accuracy and 91.82% for precision, outperforming the state-of-the-art approach, which achieves 89.0% for accuracy and 81.8% for precision.

Chapter 4 presents a novel consecutive batch learning algorithm, called Real-time Dynamic Concept Adaptive Learning (RDCAL), to deal with concept drift

and dynamic class imbalance problems existing in exploitability prediction in online learning scenarios. In particular, a Class Rectification Strategy (CRS) is designed to handle the actual drift in sample labels, and a Balanced Window Strategy (BWS) is proposed to boost the minority class during real-time learning. The results of experiments conducted on real-world vulnerabilities collected between 1988 to 2020 show that the overall performance of classifiers, including neural networks, SVM, HoeffdingTree and logistic regression (LR), improves by over 3% by adopting the proposed RDCAL algorithm. Furthermore, RDCAL achieves state-of-the-art performance on exploitability prediction compared with other concept drift algorithms.

Chapter 5 generalises the consecutive batch learning framework introduced in Chapter 4 to multiclass classification problems and further proposes an Adaptive Sliding Window Weighted Learning (ASWWL) algorithm to tackle the dynamic multiclass imbalance problem existing in many industrial applications, including exploitation time prediction. The results of experiments carried out on a real-world dataset demonstrate the proposed ASWWL algorithm can significantly enhance the performance of the minority classes without compromising the performance of the majority class. Furthermore, the generalised consecutive batch learning framework achieves the most robust and state-of-the-art performance compared with the other five consecutive batch learning algorithms.

Chapter 6 presents a feasible and high-performance solution for the co-exploitation behaviour discovery problem, which is formulated as a link prediction problem between vulnerability entities within a cybersecurity domain-specific knowledge graph. A general Modality-Aware Graph Convolutional Network (MAGCN) module and a Graph Knowledge Transfer Learning (GKTL) strategy are proposed to boost the link prediction performance. The proposed solution is verified by the experiments conducted on a real-world cybersecurity knowledge graph consisting of co-exploitation incidents between 1995 to 2021.

Chapter 7 concludes the findings and contributions of this thesis and discusses possible challenges and potential for future work.

Chapter 2

Background and Foundations

With the ongoing adoption of information technology and its impact on national economies and society, software plays a key role in the daily life of both organisations and individuals [20]. However, a growing number of vulnerabilities caused by poor design or overlooked implementation are being disclosed nowadays [21]. The insecurity of information technology is often inevitable, which is a side effect brought by the use of information technology [22]. The scale, type and destructiveness of cyber threats and cyber attacks are increasing year by year as more and more software vulnerabilities are discovered and publicly disclosed. According to CVE details [23], more than 166,000 software vulnerabilities have been disclosed and archived from 1988 to the end of 2021. More vulnerabilities are available from various channels and venues (e.g., security bulletins, forums, social media, and so on). Bilge and Dumitras pointed out that once a vulnerability is disclosed, the chance of being exploited increases by five orders of magnitude[4, 7]. Obviously, unpatched known vulnerabilities impose significant security risks to modern society. Obviously, unpatched known vulnerabilities impose significant security risks to modern society.

2.1 Vulnerability and Vulnerability Disclosure

Vulnerability is a term referring to a system flaw that can leave it open to attack. According to the Common Vulnerabilities and Exposures (CVE) consortium, it is formally defined as a weakness in the computational logic (e.g., code) found in software and some hardware components (e.g., firmware) that, when exploited, results in a negative impact on confidentiality, integrity, or availability (CIA) [24, 25].

Vulnerability disclosure is the practice of reporting security flaws in computer software or hardware [26]. Vulnerability can be disclosed by multiple parties, including but not limited to third-party or internal software developers, vendors, suppliers, cybersecurity professionals and cybersecurity researchers [27]. Different parties have different preferences for vulnerability disclosure time. Software vendors, suppliers and related developers usually prefer to disclose vulnerabilities after the corresponding patches or remedies are available. By contrast, affected end-users, cybersecurity professionals and researchers tend to disclose vulnerabilities as early as possible.

2.2 Exploit and Exploitability

A typical exploit in the cybersecurity domain can be a piece of software, a chunk of data, or a sequence of commands, which takes advantage of a bug or vulnerability to cause unintended or unanticipated behaviour [28].

Exploitation is the behaviour of using an exploit to abuse software, hardware or other electronic equipment, including things like gaining control of a computer system, allowing privilege escalation, or launching a denial-of-service (DoS) attack [29].

Exploitability is the state or condition of being exploitable. In the cybersecurity domain, a vulnerability is identified as exploitable when the proof-of-concept of the corresponding exploit exists. Exploitability is an important vulnerability assessment metric to reflect the properties of the vulnerability that lead to a successful attack [30].

Table 2.1: Six vulnerability lifecycle events

Event	Occurred Time	Avalable to Public?
creation	t_{creat}	No
discovery	t_{disco}	No
exploit available	t_{explo}	No
disclosure	t_{discl}	Yes
patch available	t_{patch}	Yes
patch installation	t_{insta}	Yes

2.3 Lifecycle of a Vulnerability

Frei et al. described a typical vulnerability lifecycle in [31]. We simplify the main events of the lifecycle in Table 2.1. A typical vulnerability lifecycle consists of six events, namely, creation, discovery, exploit available, disclosure, patch available and patch installation. It should be noted that the order of occurrence of these six events may be slightly different for individual vulnerabilities. For example, vulnerability exploitation may occur before disclosure.

When a vulnerability is disclosed, the vulnerability information has three features, namely, free access, independence and validation [31]. Information about disclosed vulnerabilities is available to the public for free. Then, disclosed vulnerability information will be widely accepted and used by the entire cybersecurity community. Finally, the disclosed information undergoes a thorough assessment by a panel of security experts, and some assessment results will also be added to the disclosed vulnerability as basic risk ratings.

The time period between vulnerability discovery and disclosure is called the pre-disclosure phase, denoted as $\triangle\ t_{disco}(v)$, where $\triangle t_{disco}(v) = t_{discl}(v)$ - $t_{disco}(v)$; $t_{discl}(v)$ is the disclosure time of v and $t_{disco}(v)$ is the discovery time of v. At this stage, the newly discovered vulnerabilities remain largely private. If they are known by researchers or vendors, they can work to provide patches before they become exploitable or disclosed in public. However, once they are discovered by malicious intruders or cyber-criminals, the potential risk involved can be significantly elevated. However, in this pre-disclosure phase, few things can be done to

stop exploitation.

The time period from disclosure to patch available is another important phase, namely, the post-disclosure phase, denoted as $\triangle t_{patch}(v)$, where $\triangle t_{patch}(v) = t_{patch}(v)$ - $t_{discl}(v)$ and $t_{patch}(v)$ is the patch available time of v. At this stage, the risks of exploitation soar because more parties, including hackers and cyber-criminals, know of the existence of and have detailed information on the vulnerability. To make matters worse, end-users of the affected products will also be aware of the existence of this vulnerability, which will undoubtedly bring great pressure to vendors and service providers. Therefore, it is crucial for vendors and security information providers (SIPs) to provide a patch or give effective security advice. This research focuses on improving exploitability predictions and analysis to better inform decision-makers to prioritise the most urgent and risky vulnerabilities.

Similarly, post-patch phase refers to the time period between vulnerability patch available and patch installation, which is denoted as $\triangle t_{insta}(v)$, where $\triangle t_{insta}(v) = t_{insta}(v)$ - $t_{patch}(v)$ and $t_{insta}(v)$ is the patch instalment time of v. At this stage, if users are able to install the patch of v in a timely manner, the risks of exploitation can be mitigated.

2.4 Cybersecurity Ecosystem

Whenever a new vulnerability is discovered, various parties with different and often conflicting motivations and incentives become involved in a complex way [31, 32]. Participants include but are not limited to discoverers, security advisories, cyber-criminals, traders in white or underground black markets, vendors and the public. The so-called security ecosystem consists of these players and their interactions. Fig. 2.1provides a high-level view of a cybersecurity ecosystem.

As shown in Fig. 2.1, Path (A) and (B) in red are at high risk, while paths (D) and (E) in green have fewer security concerns. Most vulnerabilities go through the path (C). Once disclosed, the security of a vulnerability is uncertain, depending on whether it is exploited by attackers or patched by vendors. The risk of exploitation soars after being disclosed, as described in [4], 'after vulnerabilities

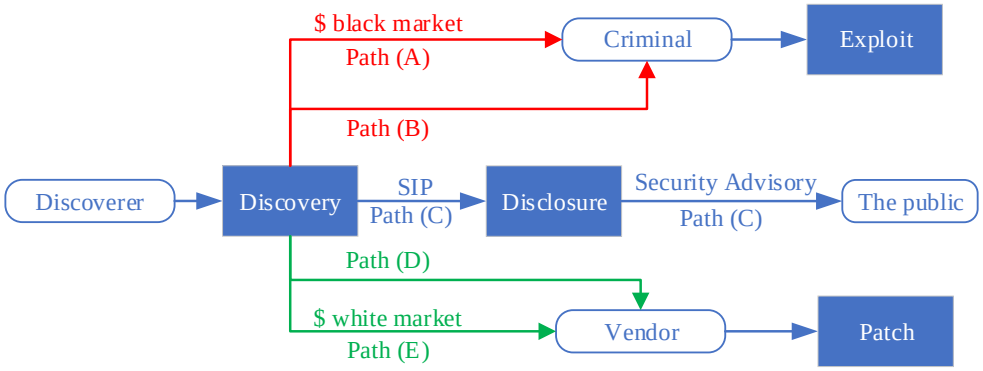


Fig. 2.1. Cybersecurity ecosystem illustration

are disclosed publicly, the volume of targeted attacks increases by five orders of magnitude.'

Vulnerabilities can be mitigated by patches, reconfigurations, and other workarounds. However, there is a consistent trend that the exploit availability continuously exceeds patch availability. Furthermore, most vulnerabilities are never actually exploited [33]. Thus, considering the development, test, distribution, and deployment costs of software patches, it is crucial for inventors to prioritise the remediation of vulnerabilities that are most likely to be exploited. In other words, vulnerability risk evaluation and exploitability prediction are important to enable inventors to make better decisions.

Based on the results of vulnerability risk evaluation, decision-makers such as vendors and cybersecurity specialists can prioritise vulnerabilities and allocate resources to patch and protect systems. Furthermore, an increasingly obvious trend is that hackers are more inclined to combine several vulnerabilities when launching attacks. Predicting the links between a newly added vulnerability and existing vulnerabilities will help avoid co-exploitation behaviour by patching any of these vulnerabilities. Despite the strong demand in the industry, only a few studies consider inferring exploitable links amongst new vulnerabilities and mitigating exploitations from the perspective of complex networks.

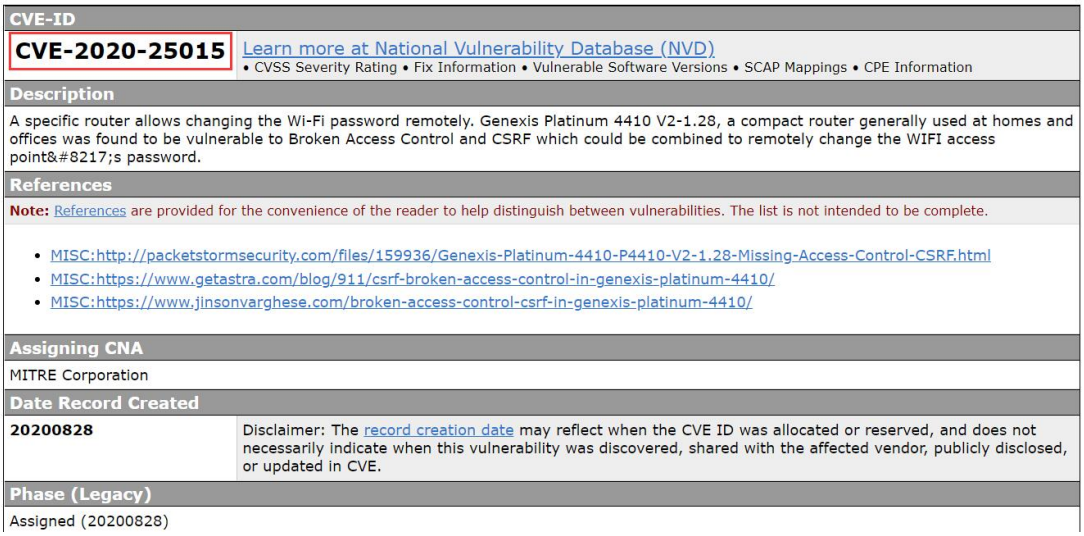
2.5 The Disclosed Vulnerabilities and Exploits

Historical vulnerability and exploit records are the most important and valuable digital assets for vulnerability assessment and management. Therefore, many commercial or non-profit organisations collect, store, and maintain their own vulnerabilities and exploit databases. Some of them are available to the public. The rest of this section introduces several well-known and publicly accessible databases and repositories. They provide comprehensive and credible information on vulnerabilities and potential links of detailed exploits (if exploitable).

2.5.1 CVE: Common Vulnerabilities and Exposures database

At present, vulnerability disclosure sources mainly include individual vendors, cybersecurity forums and open source databases. Each disclosed vulnerability will be assigned a unique identification code, CVE-ID. CVE-ID is widely accepted by both local individual information providers/repositories and multiple global vulnerability databases [9]. This unique CVE-ID of each vulnerability facilitates the fast and accurate integration of data across multiple information sources and databases. In other words, it can be used to retrieve and link various information of the same vulnerability from different databases. Apart from CVE-ID, vulnerability disclosure reports may also include disclosure date, the names and corresponding version numbers of affected software products, required permission, the scope of impact and repair suggestions etc. [9].

The CVE database is one of the most well-known vulnerability databases. It stores essential disclosed vulnerability information, such as the CVE-ID, description, one or more public reference links [34]. A vulnerability description is a brief paragraph on each vulnerability, which contains abundant details such as the vulnerability type, names of affected products and vendors, a summary of affected versions, the impact, the access that an attacker requires to exploit the vulnerability and the important code components or inputs that are involved [35]. Depending on the source of disclosure, the description of a software vulnerability is usually written by the party requesting its CVE-ID.



CVE-ID

CVE-2020-25015 Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

A specific router allows changing the Wi-Fi password remotely. Genexis Platinum 4410 V2-1.28, a compact router generally used at homes and offices was found to be vulnerable to Broken Access Control and CSRF which could be combined to remotely change the WIFI access point’s password.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:http://packetstormsecurity.com/files/159936/Genexis-Platinum-4410-P4410-V2-1.28-Missing-Access-Control-CSRF.html
- MISC:https://www.getastra.com/blog/911/csrf-broken-access-control-in-genexis-platinum-4410/
- MISC:https://www.jinsonvarghese.com/broken-access-control-csrf-in-genexis-platinum-4410/

Assigning CNA

MITRE Corporation

Date Record Created

20200828 Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Phase (Legacy)

Assigned (20200828)

Fig. 2.2. A vulnerability listed in the CVE database

The information in the CVE database serves as the baseline for vulnerability disclosure and is referenced by many vulnerability databases, security products and services. The vulnerability list in the CVE database organised by year is available for download in several formats, i.e. comma separated format, HTML, text, and XML. More than 160,000 vulnerability entries spanning over 20 years from 1999 to the present are included in the CVE database. The CVE database provides multiple attributes of each vulnerability for the public, such as Description, References, Assigning CNA, Date Record Created and Phase. Fig. 2.2 shows a screenshot of vulnerability information listed in the CVE database. For more information, refer to the official website of the CVE database https://cve.mitre.org/index.html.

2.5.2 NVD: National Vulnerability Database

The NVD database is the U.S. government repository of standards-based vulnerability management data represented using the Security Content Automation Protocol (SCAP) [36]. It provides an analysis of CVE entries published in the CVE database. Based on the descriptions and references provided by the CVE database and other publicly accessed supplemental data, NVD expert panellists conduct an

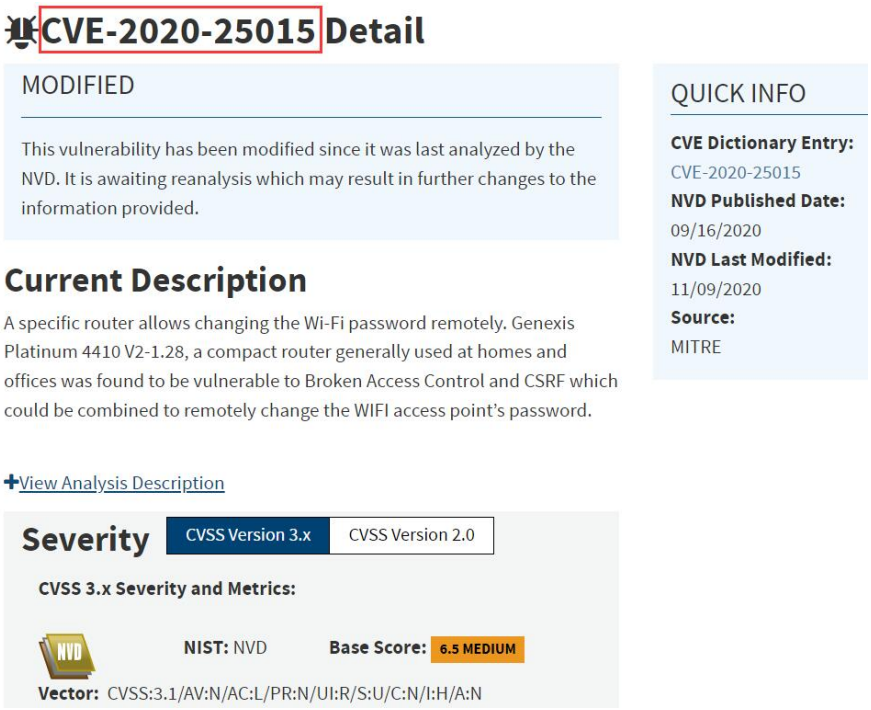
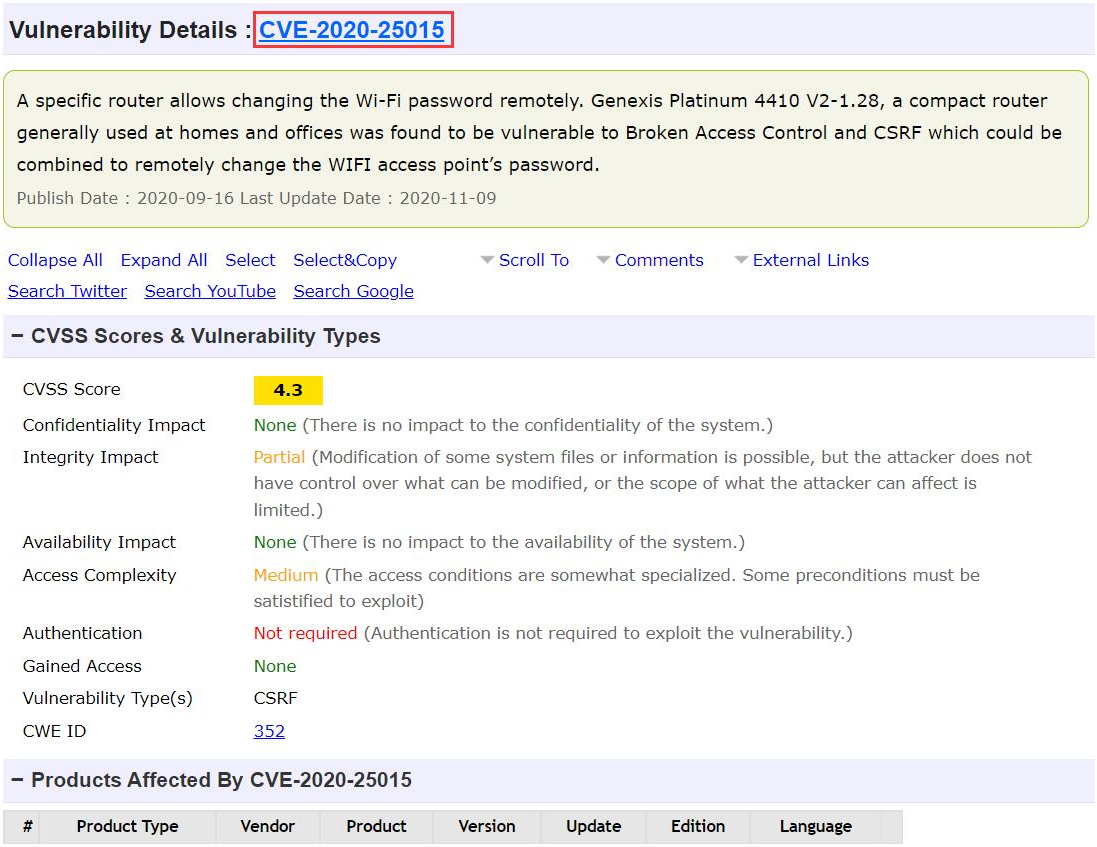


Fig. 2.3. A vulnerability listed in the NVD database

initial vulnerability assessment and give results based on certain standards, such as impact metrics (defined by Common Vulnerability Scoring System (CVSS)), applicability statements (defined by Common Platform Enumeration (CPE)), vulnerability types (defined by Common Weakness Enumeration (CWE)), and also other pertinent metadata [36]. Fig. 2.3 shows the screenshot of information on a vulnerability listed in the NVD database.

Most importantly, the NVD database keeps re-analysing vulnerabilities as time and resources change over time to ensure the information provided by NVD is up to date. The NVD database is updated periodically to maintain the accuracy and real-timeliness of vulnerability information. The data feeds in the NVD database are available to the public for free. [37].



Vulnerability Details : CVE-2020-25015

A specific router allows changing the Wi-Fi password remotely. Genexis Platinum 4410 V2-1.28, a compact router generally used at homes and offices was found to be vulnerable to Broken Access Control and CSRF which could be combined to remotely change the WIFI access point's password.

Publish Date : 2020-09-16 Last Update Date : 2020-11-09

Collapse All Expand All Select Select&Copy Scroll To Comments External Links

Search Twitter Search YouTube Search Google

– CVSS Scores & Vulnerability Types

CVSS Score	4.3
Confidentiality Impact	None (There is no impact to the confidentiality of the system.)
Integrity Impact	Partial (Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited.)
Availability Impact	None (There is no impact to the availability of the system.)
Access Complexity	Medium (The access conditions are somewhat specialized. Some preconditions must be satistified to exploit)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	CSRF
CWE ID	352

– Products Affected By CVE-2020-25015

#	Product Type	Vendor	Product	Version	Update	Edition	Language

Fig. 2.4. A vulnerability listed in CVE Details

2.5.3 CVE Details

CVE Details is a website developed by security consultant Serkan Özkan, who wanted to find an easy-to-use list of security vulnerabilities [23]. CVE Details contains information from multiple sources, including NVD XML data feeds, the Exploit Database [38], software vendor statements and additional vendor-supplied data, and Metasploit modules [23]. CVE Details presents each vulnerability entry on a single, easy-to-use web page. Fig. 2.4 shows an example of the vulnerability information listed in CVE Details. Some statistics on vulnerabilities, vendors, products, and exploits are also available in tables or figures [23].

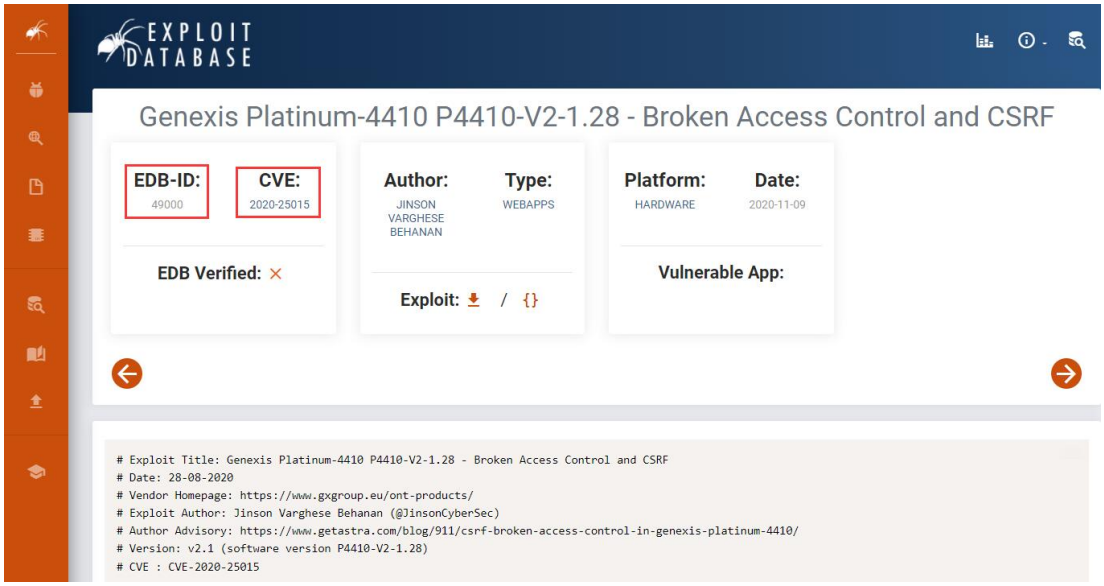


Fig. 2.5. An exploit listed in the EDB database

2.5.4 EDB: Exploit Database

The Exploit Database is an archive of public exploits and their targeted vulnerabilities, developed for use by penetration testers and vulnerability researchers [38]. The exploits in EDB are gathered from public sources and are freely available to the public. Each exploit in the EDB database has a unique EDB-ID for identification purposes.

The EDB database provides proofs-of-concept rather than advisories for vulnerabilities. Therefore, many researchers use the existence of exploits as the first sign of the exploitability of vulnerabilities [28, 33, 39], although exploitations always appear behind the existence of exploits. Fig. 2.5 shows a screenshot of information on an exploit listed in the EDB database. Apart from the proof-of-concepts' executive codes, other crucial information on an exploit is also provided, such as EDB-ID, CVE-ID, Author, Type and Platform, as shown in Fig. 2.5. The EDB database is also a CVE-compatible database, making it possible to link the information of vulnerabilities and exploits.

At present, most commercial vulnerability management systems regularly synchronise the vulnerability and exploit information from these mainstream

databases. Furthermore, the experimental data of most research papers on vulnerability risk assessment come from the integration results of these open-sourced mainstream databases [7, 19, 40, 41, 42]. To provide more examples for reference, Table 2.2 lists the Uniform Resource Locator (URL) of examples of more vulnerabilities or exploits contained in the aforementioned four databases. The URL is entered into a browser for detailed information corresponding to that vulnerability or exploit. It is worth mentioning that the exploit listed below each vulnerability in Table 2.2 is the specific exploit attacking that vulnerability.

2.6 Vulnerability Assessment and Management

Vulnerability management is a crucial measure for both organisations and the entire cybersecurity community to protect their information systems and networks from cyberattacks, intrusions, malware and various types of data breaches [42]. Since the availability of exploits is much more in quantity than the availability of patches [43], it is important for vulnerability management experts to assess the risk level of existing vulnerabilities accurately. For risk management and vulnerability repair of modern information systems, vulnerability assessment and prioritisation are the most basic steps in order to allocate budget and resources efficiently and effectively [44].

To date, various methods have been developed and introduced to assess software vulnerabilities and predict the trends of vulnerability outbreaks [42, 45]. The rest of this section introduces the de facto industry standard and some of the latest developments in the academic community on vulnerability assessment.

2.6.1 Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) is the de facto industry standard to assess the severity of security vulnerabilities. It originated from a research project which aimed to promote a common understanding of vulnerabilities and their impact through the development of a common vulnerability scoring system by the National Infrastructure Advisory Council (NIAC) in July 2003 [46].

Table 2.2: Examples of vulnerabilities and their corresponding exploits listed in CVE, NVD, CVE Details and EDB

CVE-ID/EDB-ID	Database	URL
CVE-2020-25015	CVE	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25015
	NVD	https://nvd.nist.gov/vuln/detail/CVE-2020-25015
	CVE Details	https://www.cvedetails.com/cve-details.php?cve_id=CVE-2020-25015
EDB-ID: 49000	EDB	https://www.exploit-db.com/exploits/49000
CVE-2021-24275	CVE	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24275
	NVD	https://nvd.nist.gov/vuln/detail/CVE-2021-24275
	CVE Details	https://www.cvedetails.com/cve-details.php?cve_id=CVE-2021-24275
EDB-ID: 50346	EDB	https://www.exploit-db.com/exploits/50346
CVE-2021-24287	CVE	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24287
	NVD	https://nvd.nist.gov/vuln/detail/CVE-2021-24287
	CVE Details	https://www.cvedetails.com/cve-details.php?cve_id=CVE-2021-24287
EDB-ID: 50349	EDB	https://www.exploit-db.com/exploits/50349
CVE-2021-24286	CVE	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-24286
	NVD	https://nvd.nist.gov/vuln/detail/CVE-2021-24286
	CVE Details	https://www.cvedetails.com/cve-details.php?cve_id=CVE-2021-24286
EDB-ID: 50350	EDB	https://www.exploit-db.com/exploits/50350

Table 2.3: Metrics in CVSS metric groups

Metric Group	Metric Name (and Abbreviated Form)
Base metric group	Attack Vector (AV), Attack Complexity (AC), Privileges Required (PR), User Interaction (UI), Scope (S), Confidentiality (C), Integrity (I), Availability.
Temporal metric group	Exploit Code Maturity (E), Remediation Level (RL), Report Confidence (RC).
Environmental metric group	Confidentiality Requirement (CR), Integrity Requirement (IR), Availability Requirement (AR), Modified Attack Vector (MAV), Modified Attack Complexity (MAC), Modified Privileges Required (MPR), Modified User Interaction (MUI), Modified Scope (MS), Modified Confidentiality (MC), Modified Integrity (MI), Modified Availability (MA).

CVSS is currently at version 3.1 and under the custody of the Forum of Incident Response and Security Teams (FIRST). FIRST is the premier organisation and recognised global leader in incident response. Currently, FIRST has more than 400 members ranging from government, commercial, and educational organisations, spread over Africa, the Americas, Asia, Europe and Oceania [47]. Nowadays, CVSS is recommended by a wide range of computer, networking and software vendors, such as Cisco, the NVD database, Microsoft and Oracle [48].

2.6.1.1 CVSS Metric Groups

CVSS defines three independent metric groups, namely, the base metric group, temporal metric group, and environmental metric group, whose detailed metric names are shown in Table 2.3 [30]. Only the base metric group is mandatory for calculating a vulnerability CVSS score.

2.6.1.2 CVSS Scores

The values of CVSS metrics shown in Table 2.3 are either a number between 0-10 or a discrete categorical value, which are given by a cybersecurity experts panel according to the basic information of disclosed vulnerabilities [30]. Based on these metric groups, CVSS then calculates an overall score between 0-10.0 as the final CVSS score of a vulnerability according to a specially designed formula, where 10.0 represents the highest risk [7]. The detailed calculation process can be found in [30].

In particular, CVSS includes a formula to calculate the exploitability score of a vulnerability, as shown in Equation (2.1) [30].

$$\text{Exploitability} = 8.22 \times \text{AV} \times \text{AC} \times \text{PR} \times \text{UI}, \tag{2.1}$$

where 8.22 is the coefficient assigned by a panel of CVSS cybersecurity experts; AV, AC, PR and UI are the abbreviated forms of the four base metrics listed in Table 2.3.

In addition to an overall score between 0 and 10, CVSS also provides a qualitative evaluation method for vulnerabilities by mapping the overall CVSS score to five risk levels, namely, none (0.0), low (0.1-3.9), medium (4.0-6.9), high (7.0-8.9) and critical (9.0-10.0).

CVE Details presents the current vulnerability distribution by CVSS scores based on 162,031 vulnerabilities, which shows the weighted average CVSS score for all disclosed vulnerabilities is 6.5 [23].

2.6.1.3 Limitations of CVSS

CVSS is a carefully designed scoring system based on expert knowledge and accepted by a wide range of organisations. However, it is widely questioned by researchers that an overall score calculated by combining multiple metric groups with fixed weights, such as a CVSS score, can accurately represent the risk level of different software vulnerabilities [7].

Furthermore, CVSS is widely criticised by the academic community for the inconsistency between CVSS scores and the exploitability of vulnerabilities [9,

28, 39]. The overall CVSS scores of existing disclosed vulnerabilities show that there is no significant correlation between the CVSS score of a vulnerability and the possibility of its exploitability.

To further validate the criticism of CVSS in the academic community, the work in [7] visualises two CVSS metrics, namely, base score and exploitability score, that are most relevant to the exploitability of vulnerabilities from two CVSS versions (CVSS V2.0 and V3.0), as shown in Fig. 2.6. The data samples in Fig. 2.6 are from all disclosed vulnerabilities recorded in the NVD database from 1988 to 2019. Figure 6 shows exploited vulnerabilities in dark orange and unexploited vulnerabilities in the navy. The Y-axis represents the comparison of the number of these two types of vulnerabilities with the same CVSS metric score. The X-axis indicates the value of the corresponding CVSS metrics and the larger the value, the greater the risk of the corresponding vulnerabilities. Taking the V2 exploitability score shown in subplot (b) as an example, the blue bar with a score between 8 and 10 is very high, indicating that the number of unexploited vulnerabilities in this interval is very large. Obviously, this contradicts the low probability of unexploited vulnerabilities.

Similarly, the contradiction between the CVSS metric score and the exploitability of vulnerabilities is also reflected in subplots (a), (c) and (d). It is worth noting that V3 is an improved CVSS version of V2. However, as can be seen from subplots (c) and (d), the deficiency that CVSS cannot effectively depict the exploitability of vulnerabilities has not been significantly improved.

Other concerns on the application of CVSS as a vulnerability assessment indicator include the following two points. Firstly, the value assignment of CVSS metrics relies on an expert panel, which is costly in time and money. Furthermore, it is difficult to ensure consistency when the personnel changes.

2.6.2 Vulnerability Assessment in Academia

Researchers in the academic community have tried to find a better way to evaluate the severity of vulnerabilities. In addition to the CVSS system, the authors in [39] considered two risk factors: (1) the existence of a public proof-of-concept exploit; (2) the existence of an exploit traded in the cybercrime black markets to evaluate

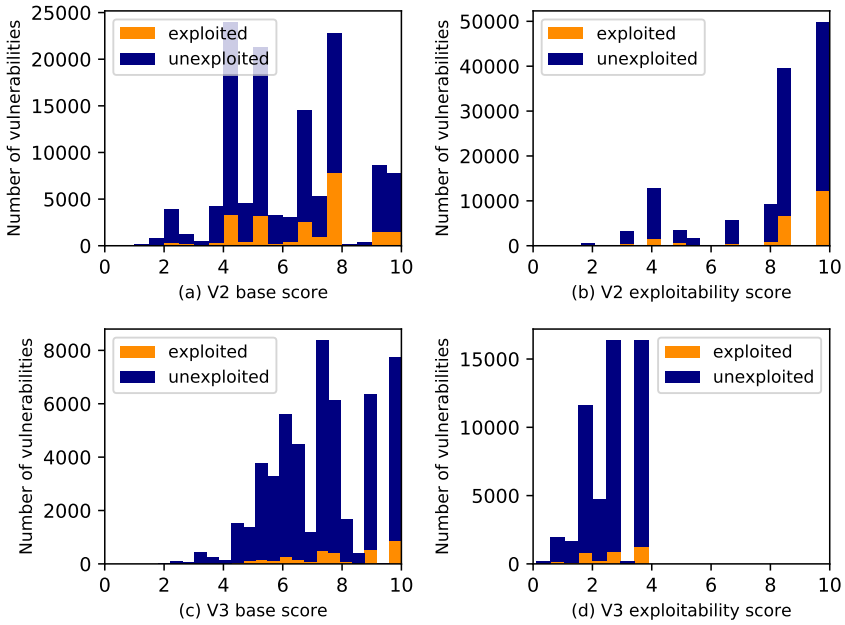


Fig. 2.6. The CVSS metric score distribution of vulnerabilities

the possibility of exploitation using a case-control study methodology. Although some improvements have been achieved, it is still not satisfactory and needs to be improved. The work in [49] applied time series analysis to build predictive models for all the reported vulnerabilities of five popular web browsers: Firefox, Chrome, Safari, Internet Explorer and Opera. A more recent study [21] proposed a GARCH model to investigate the long-term and time-invariant dependence relationships in vulnerabilities leveraging multivariate time series analysis, and then to predict the vulnerability disclosure trends and dependencies. However, these works concentrated on predicting the trends and the number of disclosed vulnerabilities for certain vendors or applications, instead of predicting the severity of vulnerabilities or the possibility of exploitation.

Other researchers focused on the prediction of vulnerability exploitability. The state-of-the-art machine learning algorithms [9, 10, 11] cast exploit prediction as a problem of binary classification. Based on the large-scale vulnerability databases, the work in [9] labelled each vulnerability as 'exploited' or 'not exploited'. The feature extraction algorithm in [9] is a kind of one-hot representation. More

specifically, they use a set of binary features to record whether or not particular tokens (e.g., 'buffer', 'heap', 'DNS') appear in specific text fields (e.g., 'title', 'solution', 'product name'). The classifier adopted in [9] is one of the most popular large margin classifiers, namely the linear support vector machine (LSVM) classifier. In the online experiment mode, where classifiers are repeatedly trained with all vulnerabilities seen up to a point in time, they achieved a classification accuracy of nearly 85%.

Sabottke, Suciu and Dumitras [10] proposed a Twitter-based exploit detector, predicting real-world exploitations. In this work, they extract features from NVD, OSVDB and Twitter streams and manually select features based on the mutual information between features and labels. The LSVM classifier is also used in their work. One of the contributions of this work is that it not only can predict if a vulnerability will be exploited, it can also predict the emergence of exploits on an average of two days in advance. However, the features extracted from Twitter streams are also manually selected statistical features (e.g., specific words, number of retweets and replies, information about the users posting these messages).

Along the same line, the work in [11] proposed an exploitability prediction method based on neural language models. Instead of extracting linguistic features using traditional TF-IDF-based representation, it adopts the neural language models to learn word embeddings based on the corpus collected from multiple sources. By capturing the linguistic regularities of human languages, such as syntactic, semantic similarity and logic analogy, the learned embeddings can better classify discussions about exploited vulnerabilities than traditional text analysis methods [11].

The work in [50] investigated the effectiveness of different features, including common words from vulnerability descriptions, external references and vendor products, CVSS scores and categorial attributes, and Common Weakness Enumeration (CWE) numbers, in predicting the exploitability of vulnerabilities. Jacobs, Romanosky et al. proposed an open, data-driven framework, called the Exploit Prediction Scoring System (EPSS), to estimate the probability of a vulnerability being exploited within the first twelve months after disclosure [51]. Paper [7] employed transfer learning to extract paragraph-level embeddings from vulnerabilities and built a high-performance exploitation predictive model.

In summary, although the use of machine learning and pattern recognition methods to predict exploitability has received increasing attention in past decades, this field is still not satisfactory in terms of research methods and research results. More research needs to be directed to create a more secure cyber environment. Furthermore, some of the latest research advances in machine learning and deep learning, such as word embedding, graph representation can be introduced into this field to make a difference.

Chapter 3

Vulnerability Exploitability Prediction

From the feature extraction and selection strategies to the selection of classifiers, there are enormous choices in building machine learning (ML) or deep learning (DL) based exploitability prediction models. This chapter focuses on the vulnerability description-based exploitability prediction problem and proposes a novel exploitability prediction framework, ExBERT, based on transfer learning [52].

This chapter starts with an elaborate analysis of existing problems of currently description-based exploitability prediction models in Section 3.1, followed by an introduction of the related works, including transfer learning, the Bidirectional Encoder Representations from Transformers (BERT) model and wordpiece tokenization in Section 3.2. Then, section 3.3 formulates the exploitability prediction problem and clarifies its optimization objective. Section 3.4 elaborates the proposed ExBERT framework, followed by the experiment results and analysis in Section 3.5. Finally, a conclusion is provided in Section 3.6.

3.1 Introduction

Considering the limitation of CVSS scores in the exploitability prediction of vulnerabilities described in Section 2.6.1.3, researchers have been trying to build

more accurate exploitability prediction models, leveraging the emerging machine learning and deep learning techniques [45, 53, 54]. In particular, to reduce dependence on domain knowledge and time delay problems existing in CVSS, an increasing number of scholars assess vulnerabilities and predict exploitability by mining vulnerability descriptions [9, 11, 53, 55, 56]. Although some promising results were reported in previous studies, the following problems still exist:

(1) Failure to consider the polysemy problem in natural language processing (NLP) and the special technical terms in cybersecurity. For example, there are two sentences: sentence A 'A buffer overflow in lsof allows local users to obtain root privilege' and sentence B 'Roses will not root in such acid soil'. The meaning of the word 'root' in sentences A and B are different. Furthermore, the word 'lsof' in sentence A is a term in cybersecurity which means 'list open files'. Terms such as 'lsof' are essential to capture the semantic meaning in vulnerability descriptions and should not be ignored. In previous description-based exploitability prediction studies, researchers used the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm[9, 11], the rule-based statistical method [50, 56, 57] and word-embedding [11, 55, 58] techniques to extract semantic features from descriptions. These feature extraction methods use identical representations or methods to deal with the same word or phrase. Therefore, they cannot capture the polysemy of words in different contexts. Furthermore, the cybersecurity domain corpus contains many low-frequency domain-specific words, such as package names, tool names, variable names and other technical terms. None of the previous studies addresses this problem.

(2) Failure to consider the dependencies between the features extracted from descriptions when choosing classifiers. As sequential texts, dependencies exist in vulnerability descriptions. For example, the meaning of 'root' in the phrase 'root privilege' depends on the word 'privilege'. Existing studies applied support vector machine (SVM) [9, 11, 50], random forest (RF) [11, 59], naive Bayes [56] and fully-connected neural networks (denoted as DenseNN) [55] as the classifiers for exploitability prediction. These classifiers treat features as individuals with no dependency. Therefore, they fail to capture the dependencies within each description.

(3) No publicly accessed unified datasets exist in the exploitability prediction community. Publicly accessed unified datasets are a very important factor in developing machine learning or deep learning algorithms. Although researchers collect data from the same open-source dataset, i.e. NVD [37] and EDB [38], the chosen vulnerabilities, the exploitability status at a specific time, and the total amount of vulnerabilities vary in different studies. To the best of our knowledge, no publicly accessed unified dataset exists for exploitability prediction so far. The major reason is that vulnerability-related databases are dynamically increasing, and the exploitability of vulnerabilities also changes over time. Researchers always try to obtain the latest data to support their research. As a result, all existing studies use self-collected datasets when predicting the exploitability of vulnerabilities [9, 11, 50, 55, 56].

Compared with previous works, the main merits of the proposed framework ExBERT can be summarized as follows.

(1) ExBERT can extract context-aware token embeddings from vulnerability descriptions and embed low-frequency cybersecurity technical terms. The BERT model provides different token embeddings for the same token according to its context, which means it can understand word polysemy. Inspired by transfer learning, ExBERT fine-tunes a pre-trained BERT model using vulnerability descriptions instead of training a BERT from scratch, to overcome the insufficiency of the domain corpus. ExBERT also inherits the capability of polysemy embedding from BERT, which means it can extract context-aware token embeddings. Furthermore, ExBERT adopts a wordpiece tokenization algorithm to deal with low-frequency cybersecurity technical terms to gain better semantic features.

(2) ExBERT can extract comprehensive sentence-level semantic features instead of token-level features by designing a pooling layer on top of the fine-tuned BERT. The concept of the pooling layer was originally used in convolutional neural networks (CNN) to reduce dimensionality and extract high-level features. In this chapter, ExBERT introduces a pooling layer to extract sentence-level semantic features from token-level features extracted from the fine-tuned BERT model.

(3) ExBERT applies the long short term memory (LSTM) model as the classifier in exploitability prediction instead of using SVM, RF, CNN, etc., to deal with

the existing long-term dependencies in vulnerability descriptions. Descriptions are sequential inputs with long term dependencies, recurrent neural networks (RNNs) and their variation LSTM have been proven to be good at capturing the long-term dependencies in sequential data and dealing with natural language [60]. ExBERT is the first to apply LSTM in exploitability prediction.

3.2 Related Works

The most related techniques involved in this chapter are briefly reviewed and presented in this section.

3.2.1 Transfer Learning

Transfer learning is an emerging learning strategy in machine learning (ML) that focuses on learning and storing knowledge gained from one domain and applying it to a different but related domain [61, 62, 63]. A Domain $\mathcal{D}$ consists of two components: a feature space χ and a marginal probability distribution $P(X)$, where $X = x_1, \ldots, x_n \in \chi$. In general, if two domains are different, then they may have different feature spaces χ or different marginal probability distributions $P(X)$. Given a specific domain, $D = \{\chi, P(X)\}$, a task $\tau = \{\mathcal{Y}, f(\cdot)\}$ consists of two components: a label space $\mathcal{Y}$ and a predictive function $f(\cdot)$, which can be written as $P(y|x)$. Labels can be $y = \{0, 1\}$ in binary classification. The predictive function $f(\cdot)$ is not observed, but can be estimated from the training data $\{x_i, y_i\}$ $(x_i \in X, y_i \in \mathcal{Y})$.

We further denote $D_s = (x_{s_1}, y_{s_1}), \ldots, (x_{s_n}, y_{s_n})$ as the source domain, where $x_{s_i} \in \chi_s$ and $y_{s_i} \in \mathcal{Y}_s$ representing features and corresponding labels. Similarly, the target domain is denoted as $D_t = \{(x_{t_1}, y_{t_1}), \ldots, (x_{t_m}, y_{t_m})\}$, where $x_{t_i} \in \chi_t$ and $y_{t_i} \in \mathcal{Y}_t$. In general, $0 < n \ll m$.

Given all the defined notations, transfer learning is formally defined as [61]:

Transfer Learning: Given a source domain $\mathcal{D}_s$ and a learning task τ_s, a target domain $\mathcal{D}_t$ and a learning task τ_t, transfer learning aims to help improving the learning of the target predictive function $f_t(\cdot)$ in $\mathcal{D}_t$ using the knowledge learnt in $\mathcal{D}_s$ and τ_t, where $\mathcal{D}_s \neq \mathcal{D}_t$ or $\tau_s \neq \tau_t$.

3.2.2 BERT Model

In 2018, Devlin and Chang et al. at Google AI Language proposed an NLP model named BERT for language understanding [64]. It has caused a stir in the deep learning community by achieving the state-of-the-art results on eleven NLP tasks instead of on a single task or domain, including Natural Language Inference (NLI) and Question Answering (SQuAD v1.1). BERT is developed based on a popular Transformer model[65] and it trains a multi-layer Transformer in a bidirectional way.

To enable bidirectional training, BERT defined two training tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). The training process of MLM is shown in Fig. 3.1. Given an input sentence, 15% tokens are randomly replaced with a [MASK] token, then train the BERT model to predict the masked tokens. Tokens [CLS] and [SEP] in Fig. 3.1 are manually added as the beginning marker and ending marker of an input sentence. BERT consists of 12 stacked Transfer layers, named as Transfer Layer -1, ..., -11, -12. The output of each Transfer Layer corresponding to the token embeddings of the input tokens. For example, the output of the first node in each Transfer Layer is the token embedding of [CLS]. In the original paper[64], only the token embedding corresponding to [CLS] in Transfer Layer -1 is used to predict the value of the masked token, other token embeddings are discarded as shown in Fig. 3.1. Then, a DenseNN is used as a classifier on top of the [CLS] token embedding to calculate all possibilities on the vocabulary of BERT.

Similarly, the NSP training process is shown in Fig. 3.2. Given two sentences, [CLS] is added as the beginning of the first sentence and two [SEP] tokens are added at the end of these two sentences. BERT is trained to predict if Sentence B is the sub-sequent sentence of Sentence A. Also, in NSP task, only the token embedding of [CLS] is used to do classification, other token embeddings are discarded.

Because BERT has become ubiquitous recently for NLP tasks and our implementation is effectively identical to the original paper except for some tiny data pre-processing, we omit an exhaustive description of the model architecture and refer readers to [64] as well as its GitHub resources [66].

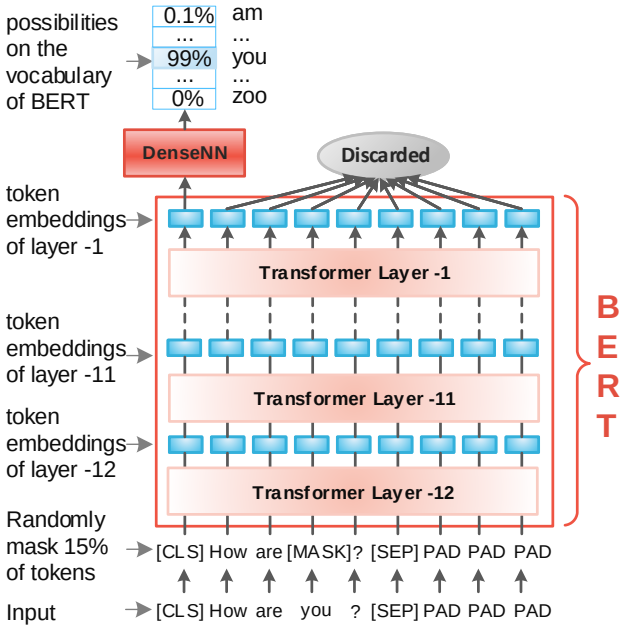


Fig. 3.1. The training process of Masked Language Model

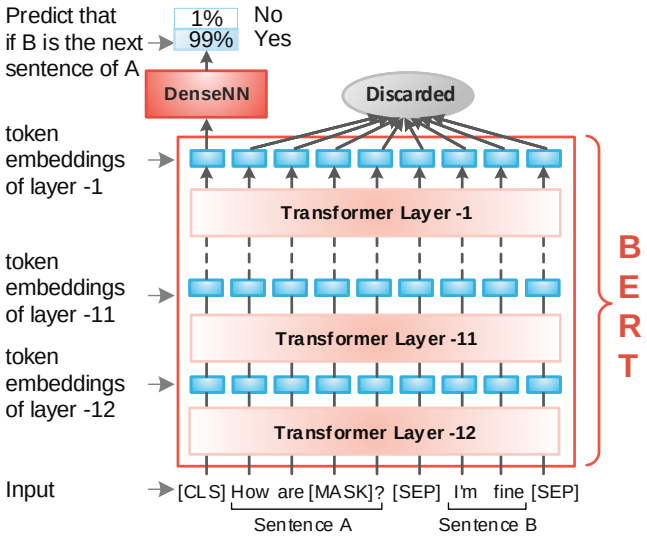


Fig. 3.2. The training process of Next Sentence Prediction

Although the performance of BERT is outstanding, we cannot apply it directly to exploitability prediction. For one thing, training BERT from scratch is slow and resource-consuming. It is infeasible and unnecessary for most downstream applications to train such a comprehensive NLP model from scratch. For another, the corpora size in most downstream applications including exploitability prediction are too small to train such a complicated NLP model. For example, the total words in descriptions collected from published vulnerabilities between 1999 and 2019 is only about 5 million. Fortunately, Google has released several versions of pre-trained models, which are well trained on BooksCorpus (800 million words) [67] and English Wikipedia (2,500 million words).

Applying transfer learning to BERT is a feasible solution for exploitability prediction. Paper [68] demonstrated that the pretraining-then-fine-tuning paradigm has an even better performance and generalization capability than training from scratch for BERT model in a visualization way. Therefore, we can fine-tune a pre-trained BERT model using cybersecurity domain corpus and transfer the knowledge learnt from the super-large pre-training corpora to the new cybersecurity task.

3.2.3 Wordpiece Tokenization

Wordpiece tokenization was first proposed by Wu et al. in 2016 to handle rare words in Google's Neural Machine Translation System [69]. It divides low-frequency words to a set of common sub-word units (wordpieces) if these words are not in the vocabulary. For example, Table 3.1 lists some wordpiece tokenized words when adopting the same vocabulary (contains 30,255 tokens) with BERT. We can see that for some common words contained in the vocabulary like 'windows', 'linux', 'root' and 'leaves', the tokenized tokens are the same with the original words. For some low-frequency words which are not in the vocabulary like 'spyware', 'malware', 'vulnerabilities' and 'overflow', they are tokenized into several wordpieces like ['spy', '##ware'], ['mal', '##ware'], ['vu','##ln', '##era', '##bilities'] and ['over', '##flow'] based on the vocabulary. The rest part followed the '##' in a token should be attached to the previous token without space when reversing the tokenization.

Table 3.1: Examples of wordpiece tokenization

Originial words	Tokenized wordpieces
windows	'windows'
linux	'linux'
root	'root'
leaves	'leaves'
spyware	'spy', '##ware'
malware	'mal', '##ware'
vulnerabilities	'vu', '##ln', '##era', '##bilities'
overflow	'over', '##flow'

Wordpiece tokenization is a powerful tool to deal with rare words or domain-specific terms. Because it can deal with words which are not contained in a vocabulary. The tokenized word is a fixed token combination once the vocabulary is fixed. Many NLP models including BERT can leverage the bidirectional information of a sequence. Therefore, the model can understand the semantic meaning of a fixed token combination after training [70, 71], which means the tokenized word can still be identified and represented by the corresponding NLP model.

The corpus for exploitability prediction contains around 5 million words, which is too small to train a robust NLP model from scratch. Therefore, we need to apply transfer learning to the pre-trained BERT model and transfer the knowledge learnt from the huge pre-training corpus to the cybersecurity domain. When applying transfer learning, it is important to use the identical wordpiece tokenizaiton method including the same vocabulary as the pre-trained NLP model. Because it will split the the downstream-task's corpus into tokens the same way with the pretraining corpus, and the pre-trained semantic knowledge will be inherited by downstream applicaitons. Furthermore, technology is evolving and new technical terms are emerging every day. Therefore, it is impossible to keep changing the vocabulary of the NLP model. In fact, different vocabulary will only cause different forms of token conbinations. As long as keeping consistency, it makes no difference for a machine to understand 'overflow', { 'over', '##flow'}

or { ‘ov’, ‘##erf’, ‘##low’}. Therefore, instead of building a cybersecurity domain vocabulary for exploitability prediction, we use the identical vocabulary with BERT.

3.3 Problem Formulation

Exploitability prediction is a classification problem in the machine learning domain. The input is a vulnerability description and the output is the likelihood of the input vulnerability being exploitable. As text information, descriptions cannot be used as the input of a classification model directly and it must be transformed into numerical vectors using semantic feature extraction algorithms such as one-hot, Bag-of-Words [72], Skip-gram [72], GloVe [73] and BERT. We formulate the exploitability prediction model as (3.1),

$$\hat{y} = f(\Theta, x), \tag{3.1}$$

where x is the input feature extracted from a vulnerability description; Θ is the model parameters and $f(\cdot)$ is the predictive function determined by the predictive model structure and $\hat{y}$ is the predicted probability in a closed interval [0,1].

We denote the dataset for exploitability prediction as D, Y, where D=[$d^{(1)}$, $d^{(2)}$, $\cdots$, $d^{(m)}$]; $d^{(i)}$ is the i-th description of the i-th vulnerability; m is the total number of vulnerabilities in D. Accordingly, Y=[$y^{(1)}$, $y^{(2)}$, $\cdots$, $y^{(m)}$] is the ground truth label vector and $y^{(i)} \in \{1, 0\}$ indicates if the i-th sample is exploitable or not. To implement exploitability prediction, D will be transformed into a numerical feature matrix X=[$x^{(1)}$,$x^{(2)}$, $\cdots$, $x^{(m)}$], where $x^{(i)} \in \mathbb{R}^n$ is the feature extracted from $d^{(i)}$ and n is the feature dimension. According to (3.1), the predictive output corresponding to X is $\hat{Y}$=$f(\Theta, X)$, where $\hat{Y}$=[$\hat{y}^{(1)}$, $\hat{y}^{(2)}$, $\cdots$, $\hat{y}^{(m)}$]; $\hat{y}^{(i)}$ is the predicted probability corresponding to $x^{(i)}$.

Since exploitability prediction is a binary classification task, according to the conventions in machine learning, the training objective for optimizing the predictive model parameter Θ is to minimise the binary cross-entropy loss function expressed as (3.2) [74, 75].

$$\begin{aligned} &\min_{\Theta} \quad E_{(x,y)\sim(X,Y)}[L_{(x,y)}(\Theta)]) \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \bullet log(\hat{y}_i) + (1-y^{(i)}) \bullet log(1-\hat{y}_i)]. \end{aligned} \tag{3.2}$$

3.4 Methodology

To predict the exploitability of vulnerabilities using descriptions, we propose a framework ExBERT based on transfer learning. This section specifies the architecture of ExBERT and how it works in detail.

3.4.1 Architecture of ExBERT

ExBERT mainly consists of two stages, i.e., BERT transfer learning and exploitability prediction application, as shown in Fig. 3.3. The BERT transfer learning process will finally generate a fine-tuned BERT model, which will be used in the exploitability prediction application stage. Exploitability prediction application process consists of four steps, namely, tokenization, token embedding, sentence embedding and exploitability prediction. Fig. 3.3 shows the sequential relationship of these steps. Firstly, in the tokenization step, vulnerability descriptions are split into tokens with wordpiece tokenization algorithm. Then, in token embedding step, the tokenized tokens are put into the fine-tuned BERT model. The outputs of the fine-tuned BERT at the l-th (l=-1,-2,$\cdots$,-L) layer are the extracted token embeddings, where L is the total number of Transfer layers in BERT and the symbol '-' indicates that we count the layer from the output layer to the input layer. Next, in the sentence embedding step, instead of picking [CLS] token embedding directly as the sentence embedding, we design a Pooling Layer to extract the sentence-level semantic features based on all token embeddings. Finally, a Classification Layer with an LSTM classifier is designed to calculate the predicted likelihood of exploitability $\hat{y}$. We state how the proposed framework works in detail in the rest of this section.

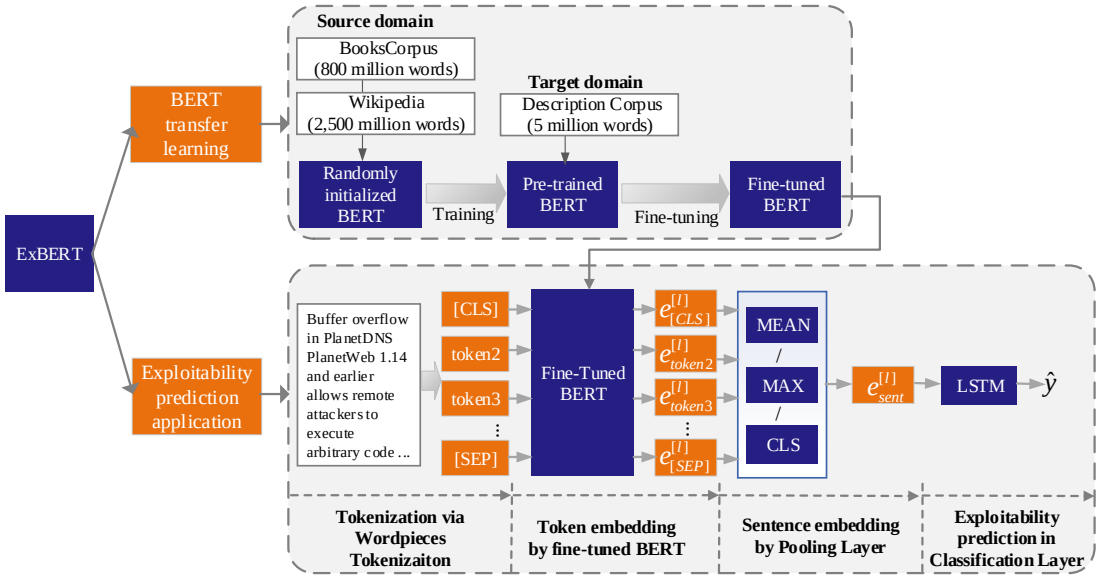


Fig. 3.3. The architecture of the proposed framework ExBERT

3.4.2 BERT Transfer Learning

BERT transfer learning means fine-tuning a pre-trained BERT into a fine-tuned BERT using the collected description corpus. The pre-trained BERT is trained from a randomly initialized BERT on the two pre-training corpora BooksCorpus and English Wikipedia performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total) [64]. Google AI has trained and released many versions of pre-trained BERT to satisfy different applications.

The first thing for BERT transfer learning is to download a proper pre-trained BERT model. We choose the uncased BERTbase [66] as the pre-trained model, which consists of 12 Transfer layers. Then, all vulnerability descriptions in the NVD database from 1999 to 2019 are collected as the domain corpus.

We simplify the input-output relationship of the pre-trained BERT model as formula (3.3),

$$\boldsymbol{e}^{[l]} = f_{BERT}(\Theta_{BERTpret}, \boldsymbol{t}), \tag{3.3}$$

where $\boldsymbol{t}$=[t_1, t_2, $\cdots$, t_n] is a token list with n tokens which is tokenised from a vulnerability description; $\boldsymbol{e}^{[l]}$=[$e_1^{[l]}$, $e_2^{[l]}$, $\cdots$, $e_n^{[l]}$] is the corresponding l-th layer's token embeddings extracted from the pre-trained BERT model; $\Theta_{BERTpret}$ is

the pre-trained BERT model parameters; $f_{BERT}(\cdot)$ is the forward propagation mapping function between $\boldsymbol{t}$ and $\boldsymbol{e^{[l]}}$, which is decided by the BERT structure; $e_j^{[l]} \in \mathbb{R}^{H_{BERT}^{[l]}}$ is the l-th layer's token embedding for the j-th token t_j, where $H_{BERT}^{[l]}$ is the hidden size of the l-th layer.

After transfer learning, parameters of BERT are updated from the pre-trained state $\Theta_{BERTpret}$ to the fine-tuned state Θ_{BERTft}. Compared with training a BERT model from scratch, applying transfer learning to BERT model can not only maintain the high performance brought by the comprehensive model, but also avoid the extremely high training cost and the lack of domain data.

3.4.3 Exploitability Prediction Application

3.4.3.1 Tokenization via Wordpiece Tokenization

Tokenization via Wordpiece Tokenization is a data pre-processing process for NLP. Descriptions D are tokenized into token lists T=[$\boldsymbol{t^{(1)}}$,$\boldsymbol{t^{(2)}}$, $\cdots$, $\boldsymbol{t^{(m)}}$], where $\boldsymbol{t^{(i)}}$=[$t_1^{(i)}$, $t_2^{(i)}$, $\cdots$, $t_n^{(i)}$] is a token list split from description $d^{(i)}$; n is the pre-set max sequence length for tokenized descriptions. The notation $t_j^{(i)}$ means the j-th token split from the i-th description $d^{(i)}$, where $i \in$\{1,2,$\cdots$, m\} and $j \in$\{1, 2, $\cdots$, n\}.

3.4.3.2 Token Embedding by Fine-tuned BERT

The output of tokenizaiton is the input of token embedding. When inputting a token list $\boldsymbol{t}$, the fine-tuned BERT will output different level of token embeddings through different BERT Transfer Layers. For example, the l-th layer's token embeddings extracted from the fine-tuned BERT is expressed as below:

$$\boldsymbol{e^{[l]}} = f_{BERT}(\Theta_{BERTft}, \boldsymbol{t}). \tag{3.4}$$

Similar with equation (3.3), $\boldsymbol{t}$=[t_1, t_2, $\cdots$, t_n] is a token list with n tokens, $\boldsymbol{e^{[l]}}$=[$e_1^{[l]}$, $e_2^{[l]}$, $\cdots$, $e_n^{[l]}$] is the corresponding l-th layer's token embeddings extracted from fine-tuned BERT model. $e_j^{[l]} \in \mathbb{R}^{H_{BERT}^{[l]}}$ is the l-th layer's token embedding for the j-th token t_j, where $H_{BERT}^{[l]}$ is the hidden size of the l-th layer of BERT. Especially, the embedding of the first token [CLS] is $e_1^{[-1]}$. Generally speaking,

the closer l is to the output layer, the more specific and application-relevant the corresponding token embeddings are. In the original paper, they always extract the last layer's token embedding for [CLS] $e_1^{[-1]}$ as the semantic feature to do classification task.

3.4.3.3 Sentence Embedding by Pooling Layer

In order to leverage the abundant semantic information contained in all token embeddings and more layers, instead of using $e_1^{[-1]}$ only like the original paper, we design a Pooling Layer for ExBERT to obtain the sentence-level embeddings on top of the fine-tuned BERT. The input of the Pooling Layer is the token embeddings $\boldsymbol{e^{[l]}}$ extracted from fine-tuned BERT. The output of the Pooling Layer is decided by the adopted pooling strategy. Common pooling strategies for NLP tasks include 'Mean' and 'Max'. Additionally, paper [64] has demonstrated that the token embedding for [CLS] $e_1^{[l]}$ can be used as the sentence embedding for downstream tasks directly. Therefore, we define the output of the Pooling Layer as a piecewise function formulated in (3.5),

$$e_{sent}^{[l]} = \begin{cases} mean(e^{[l]}), & pooling\ strategy\ is\ Mean \\ max(e^{[l]}), & pooling\ strategy\ is\ Max \\ e_1^{[l]}, & pooling\ strategy\ is\ [CLS] \end{cases}, \tag{3.5}$$

where l can be any single layer in BERT or a combination of several layers; $e_{sent}^{[l]}$ means the extracted sentence-level embedding.

3.4.3.4 Exploitability Prediction in Classification Layer

Following the Pooling Layer is a Classification Layer, which contains a hidden layer and an output layer. We adopt a LSTM layer instead of a DenseNN as the hidden layer to capture the dependencies amongst extracted sentence embeddings. The output of the hidden layer is formulated in (3.6),

$$a_h = f_{LSTM}(\Theta_{LSTM}, e_{sent}), \tag{3.6}$$

where $a_h \in \mathbb{R}^{H_h}$ is the activation result and H_h is the number of hidden nodes in the hidden LSTM layer; Θ_{LSTM} is the parameters for the LSTM layer; $f_{LSTM}(\cdot)$ is the mapping functions for LSTM.

The final output layer is a DenseNN layer with one sigmoid node and the final exploitability prediction result is calculated as below,

$$\hat{y} = \sigma(W a_h + b), \tag{3.7}$$

where weights W and bias b are the parameters of the final DenseNN layer, σ is the sigmoid activation function and $\hat{y}$ is the final possibility of vulnerability exploitability.

3.5 Experiments and Results

All experiments in this work are implemented on an Ubuntu 18.04 operating system with two NVIDIA GeForce RTX 2080 Ti GPUs. The deep learning framework used is Keras [76] with a TensorFlow backend.

3.5.1 Dataset Collection and Experimental Setting

The datasets used in this work are downloaded from NVD [37] and EDB [38], containing all vulnerabilities and exploits published between 1999 and 2019. Different vulnerabilities and exploits are identified by CVE-IDs and EDB-IDs accordingly. Most studies including this work mark a vulnerability as exploitable when it has a corresponding proof-of-concept exploit identified by an EDB-ID. As shown in Fig. 3.4, we obtain 'CVE-ID/Description' pairs from NVD database and 'EDB-ID/CVE-ID' pairs from EDB website. The two databases are integrated into 'Description/Exploitability' pairs through CVE-ID matching. If a CVE-ID can be found in the 'EDB-ID/CVE-ID' pairs, the corresponding exploitability is set to 1 and the vulnerability is exploitable. Otherwise, the corresponding exploitability is set to 0 and the vulnerability is unexploitable. Finally, the collected dataset is denoted as $\{D,Y\}$. In total, we download 123,254 'CVE-ID/Description' pairs from the NVD website and 41,365 'EDB-ID/CVE-ID' pairs from the EDB website.

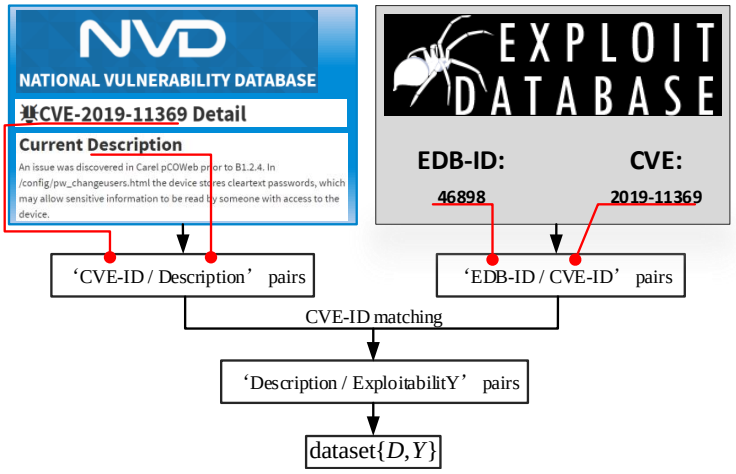


Fig. 3.4. Dataset collection process

After data de-duplication and integration, we finally get 118,209 samples. Among them, 23,073 (19.52%) unique vulnerabilities have corresponding exploits which are exploitable. We randomly selected 23,073 non-exploitable vulnerabilities, along with all exploitable vulnerabilities, to form a balanced dataset with 46,176 vulnerabilities. The dataset is divided to a training set, a validation set and a test set according to a ratio of 70%: 15%: 15%. The training set is used to train classifiers; validation set is used to select hyper-parameters and test set is used to evaluate the final performance of ExBERT.

We apply wordpiece tokenization on all descriptions and listed the distribution of the word counts and token counts in Fig. 3.5, in which we omit word bins $>$ 300 and token bins $>$ 600 to give more details on the valid data. In fact, 99.16% of descriptions is $\leq$ 128 words. After tokenization, 99.11% of descriptions is $\leq$ 254 tokens. Considering the added tokens [CLS] and [SPE], the hyper-parameter max sequence length is set to 256.

The pre-trained BERT model used in this chapter is a downloaded BERTbase [66] model, whose Transfer layer L=12 and hidden size $H^{[l]}_{BERT}$=768. All 118,209 vulnerability descriptions are used as the fine-tuning corpus, and the training epochs are set to 3. It takes about 3 hours and 20 minutes to finish the fine-tuning process on 2 NVIDIA GeForce RTX 2080 Ti GPUs.

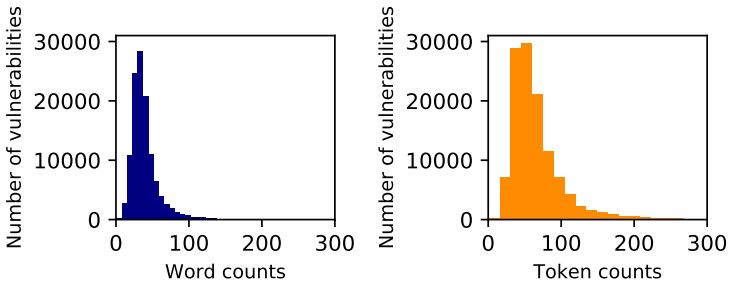


Fig. 3.5. The word count and token count distributions for descriptions

3.5.2 Ablation Study

ExBERT is an improved framework based on BERT and we propose three improvements in token embedding stage, Pooling Layer and Classification Layer. Compared with the original BERT model, the proposed improved factors and their possible values are listed as below.

(1) In token embedding stage, the BERT model can be fine-tuned BERT or pre-trained BERT.

(2) In Pooling Layer, pooling strategy can be Mean, Max or [CLS] and BERT token embedding layer l can be -1,-2, et al.

(3) In Classification Layer, the classifier can be DenseNN, RNN and LSTM.

When implementing ExBERT as a whole, those improvements make contributions at the same time. To evaluate the effect and influence of every single factor in ExBERT, we conduct a series of ablation studies and list the results in this section.

3.5.2.1 Wordpiece Tokenization Effect Analysis

As described in Section 3.2.3, we adopt wordpiece tokenization method to deal with low-frequency cybersecurity domain-specific words. To verify if the semantic meaning of low-frequency words which are not contained in the vocabulary can be learnt, we select several words in Table 3.1 to visualise their word embeddings in a two-dimensional space as shown in Fig. 3.6.

To get the data points in Fig. 3.6, we first feed each word into the pre-trained BERT and fine-tuned BERT separately. Although the words ‘spyware’,

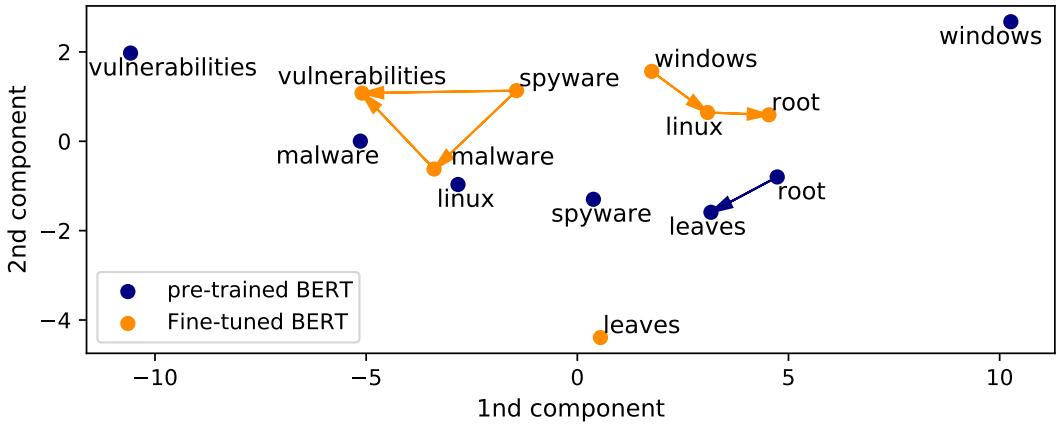


Fig. 3.6. Word embeddings in a two-dimensional space

'malware' and 'vulnerabilities' are not contained in the vocabulary, they can be tokenized in token sequences. BERT has the capacity to learn semantic knowledge bidirectionally from a token sequence. Specifically, their word embeddings can be represented by their corresponding [CLS] token embeddings in the last layer of the pre-trained or fine-tuned BERT model. After that, a Principal Component Analysis (PCA) algorthm is used to reduce those token embeddings to a two-dimensional space.

In the cybersecurity domain, both 'spyware' and 'malware' are harmful software that can take advantage of software 'vulnerabilities' to attack a user's system. As shown in Fig. 3.6, both 'spyware' and 'malware' have a relatively shorter distance to 'vulnerabilities' with fine-tuned BERT model, compared with pre-trained BERT model. This example shows that, after fine-tuning with cybersecurity corpus, the fine-tuned model is indeed better at understanding the semantics in the cybersecurity context than the pre-trained model.

This conclusion also holds for words in the vocabulary, such as 'windows', 'linux', 'root' and 'leaves'. In a general context, 'windows' mean an opening in a wall or roof, so it has a long distance with 'linux', which is an operating system, in a semantic space established by BERT model. However, a fine-tuned BERT model learnt that both 'windows' and 'linux' are operating systems, so they are very close in the fine-tuned cybersecurity semantic space. Similarly, in a general

Table 3.2: Word distances in a two-dimensional space

Word pairs	In vocabulary	Distances (BERT)	Distances (Fine-tuned BERT)
spyware→vulnerabilities	No	11.44	**3.66**
malware→vulnerabilities	No	5.79	**2.41**
spyware→malware	No	5.67	**2.62**
windows→linux	Yes	13.59	**1.61**
linux→root	Yes	7.56	**1.45**
root→leaves	Yes	**1.76**	6.38

context, both 'root' and 'leaves' are a part of plants, so they are quite close in BERT semantic space. In contrast, 'root' has a much closer relationship with 'linux' than with 'leaves' in the cybersecurity context.

Table 3.2 lists the Euclidean distances between word pairs in Fig. 3.6, in which the minimum distance of each word pair is emphasised in bold.

3.5.2.2 Transfer Learning Effect Analysis

To evaluate the effect and influence of transfer learning on the performance of exploitability prediction, we fix pooling strategy as [CLS], BERT token embedding layer l=-1 and classifier in Classification Layer as DenseNN. The only variable factor is the BERT model in token embedding stage.

Token Embedding Visualization Comparison. Fig. 3.7 shows the [CLS] token embeddings of 1000 randomly chosen samples extracted from fine-tuned BERT and pre-trained BERT. The original [CLS] token embedding's dimension is reduced from $H_{BERT}^{(-1)}$=768 to 2 using Principal Component Analysis (PCA). The horizontal axis is the 1st principal component and the vertical axis is the 2nd principal component of [CLS] token embedding. The dots in navy are non-exploited samples while the dots in dark orange are exploited samples. It is obvious that [CLS] token embeddings extracted from fine-tuned BERT are more promising to separate exploitable and unexploitable vulnerabilities compared with pre-trained BERT.

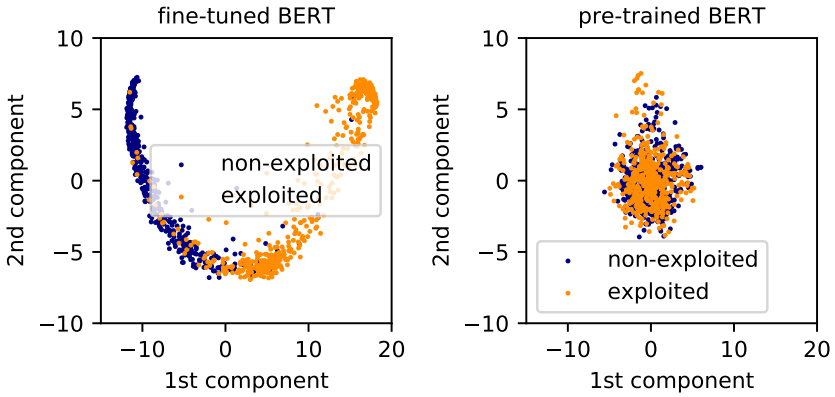


Fig. 3.7. [CLS] token embeddings visualization comparison

Table 3.3: Comparison on fine-tuned BERT and pre-trained BERT

Models	Fine-tuned BERT	Pre-trained BERT	Improvement(%)
Best epoch	**11.63**	53.47	78.24%
Loss	**0.2348**	0.4516	48.01%
Accuracy	**0.9098**	0.7924	14.54%
F1 score	**0.9106**	0.7943	14.71%
Precision	**0.9134**	0.7963	14.53%
Recall	**0.9078**	0.7926	14.53%

Classification performance Comparison. To make quantitative comparison, we input the [CLS] token-embeddings extracted from fine-tuned and pre-trained BERT to a DenseNN classifier and compare their classification performance in Fig. 3.8 and Table 3.3.

Fig. 3.8 shows the results of one training process. The left subplots are the results on fine-tuned BERT while the right subplots are on the pre-trained BERT. The x-axes represent the number of training epochs and the y-axes are the metrics specified by the y-labels. The lines in navy are the results on the training set while the lines in dark orange are on the validation set. Fig. 3.8 shows that using fine-tuned BERT can achieve approximately 0.9 on the accuracy, F1 score, precision and recall on the validation set, while using the pre-trained BERT can

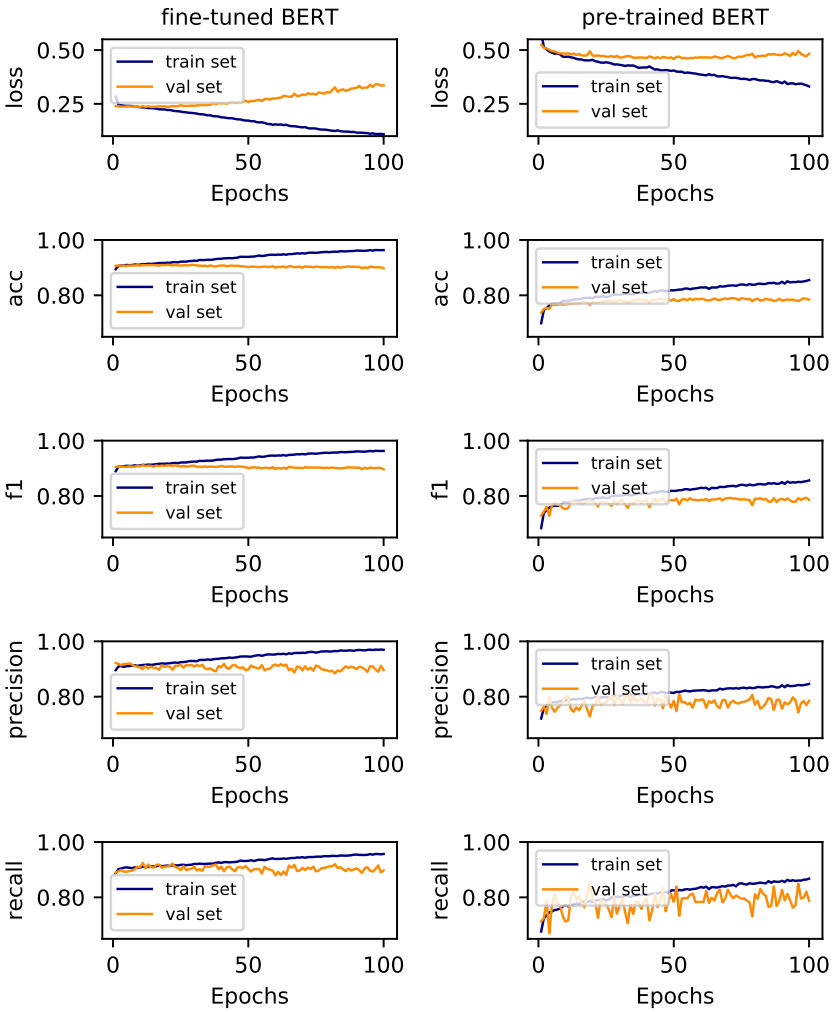


Fig. 3.8. Training process comparison on fine-tuned BERT and pre-trained BERT

Table 3.4: Comparison on pooling strategies

Pooling strategies	Mean	Max	[CLS]
Best epoch	16.17	22.53	**11.63**
Loss	0.2386	0.2421	**0.2348**
Accuracy	0.9063	0.9036	**0.9098**
F1 score	0.9070	0.9044	**0.9106**
Precision	0.9106	0.9069	**0.9134**
Recall	0.9034	0.9019	**0.9078**

only get around 0.8 on all criteria.

Table 3.3 shows the performance comparison on the test set. To reduce the influence of randomness, we repeat the training and test experiments 30 times and place the mean values into Table 3.3. Best epoch is the training epoch number on which the model achieves the smallest validation loss. Results on the test set demonstrate the effectiveness of the transfer learning. The accuracy has improved by 14.54% from 0.7924 to 0.9098 via transfer learning. The F1 score, precision and recall also have an equivalent improvement as shown in Table 3.3. The average best epoch is decreased by 78.24% from 53.47 to 11.63, which indicates the fine-tuned BERT model are more presentative in the exploitability prediction domain and therefore needs less training epochs to distinguish the exploitable and unexploitable vulnerabilities.

3.5.2.3 Pooling Layer Effect Analysis

There are two variable factors in Pooling Layer, i.e., pooling strategy and BERT token embedding layer. These two factors are analysed separately below.

Pooling Strategy Comparison. Similarly, to evaluate the effect and influence of pooling strategy on the performance of exploitability prediction, we fix token embedding model as fine-tuned BERT, BERT token embedding layer l=-1 and classifier in Classification Layer as DenseNN. The only variable factor is the pooling strategy. Evaluation results are shown in Fig. 3.9 and Table 3.4.

Fig. 3.9 shows the results of one training process. All left subplots are the results of 'Mean' pooling strategy, all middle subplots are for 'Max' and all right subplots are for [CLS]. Similarly, all x-axes represent the number of epochs for training and all y-axes are the metrics specified by y-labels. The lines in navy are results on the training set while the lines in dark orange are on the validation set. The results in Table 3.4 are also the average results of 30 independent experiments to eliminate the influence of randomness.

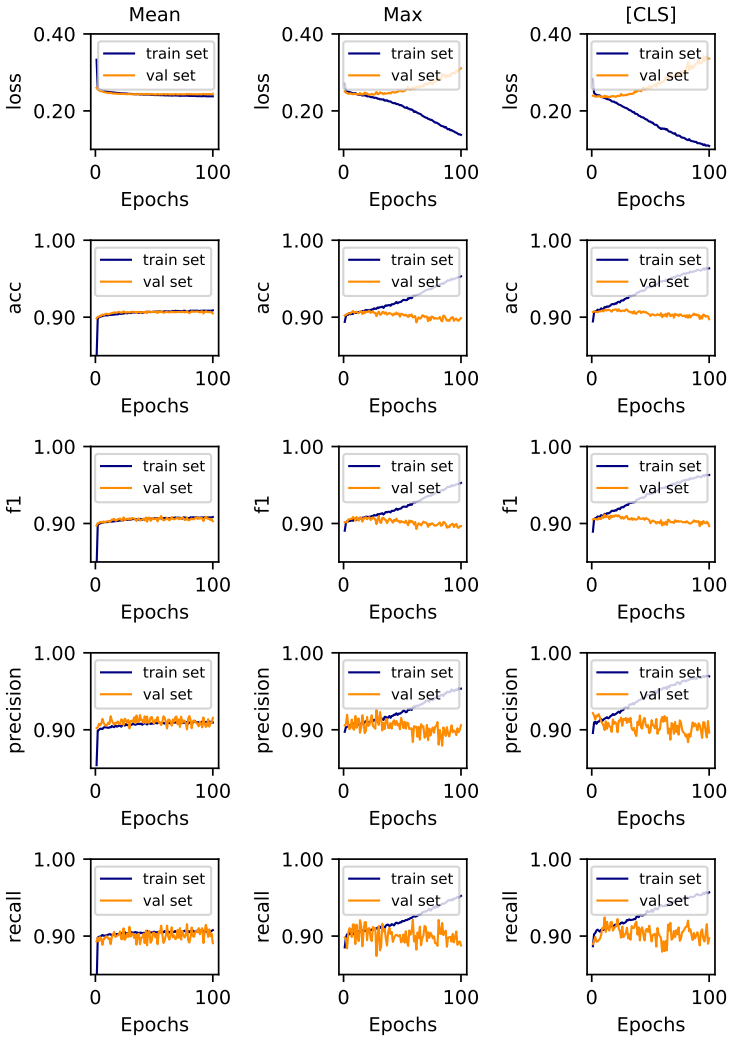


Fig. 3.9. Training process comparison on pooling strategies

Table 3.5: Comparison on BERT token embedding layers

BERT layers	Best epoch	Loss	Accuracy	F1 score	Precision	Recall
-1	11.63	0.2348	**0.9098**	**0.9106**	0.9134	0.9078
-2	8.13	0.2370	0.9068	0.9076	0.9103	0.9049
-1,-2	10.17	0.2352	0.9092	0.9099	**0.9136**	0.9062
-3,-4	9.10	0.2444	0.9030	0.9039	0.9053	0.9026
-1,-2,-3,-4	**8.10**	0.2343	0.9087	0.9096	0.9112	**0.9079**

Furthermore, Fig. 3.9 shows that although pooling strategy 'Mean', 'Max' and [CLS] can achieve equivalent good performance on their best epochs, 'Mean' has a more stable performance on both training set and validation set when the training epoch increases. However, both 'Max' and [CLS] will become overfitting when training epoch increases, which means the performance increases on the training set but decreases on the validation set, thus the generalization performance will become worse.

Table 3.4 indicates all pooling strategies can achieve very good performance. However, [CLS] has a weak advantage.

BERT Token Embedding Layer Comparison. Similarly, to evaluate the effect and influence of BERT token embedding layer on the performance of exploitability prediction, we fix token embedding model as fine-tuned BERT, pooling strategy as [CLS] and classifier in Classification Layer as DenseNN. The only variable factor is the BERT token embedding layer in Pooling Layer. Setting BERT token embedding layer $l \in$ {-1,-2,[-1,-2],[-3,-4],[-1,-2,-3,4]}, the evaluation results for BERT token embedding Layer are listed in Table 3.5. When BERT layer contains more than one layer (i.e. [-1,-2]), the token embedding for each token is the concatenation of token embeddings in each layer.

Table 3.5 also presents the average results of 30 independent experiments to reduce the influence of randomness. As shown in Table 3.5, layer -1 achieves the best accuracy 0.9098 and F1 score 0.9106, layer [-1,-2] gains the best precision 0.9136 and layer [-1,-2,-3,-4] reaches the smallest best epoch 8.1 and the best recall 0.9079. However, generally speaking, different BERT token embedding layers have an equivalent performance on exploitability prediction and none of them has

Table 3.6: Classifier Comparison Results

Classifiers	Best epoch	loss	Accuracy	F1 score	Precision	Recall
DenseNN	11.63	0.2348	0.9098	0.9106	0.9134	0.9078
RNN	14.73	0.2350	0.9102	0.9109	0.9136	**0.9083**
LSTM	**11.57**	0.2354	**0.9112**	**0.9116**	**0.9182**	0.9051

an overwhelming advantage. To decrease the computing complexity caused by concatenating token embedding between different layers, finally, ExBERT fixed l to -1.

3.5.2.4 Classifier Effect Analysis

Similarly, to evaluate the effect and influence of classifier on the performance of exploitability prediction, we fix token embedding model as fine-tuned BERT, pooling strategy as [CLS] and BERT token embedding layer l=-1. The only variable factor is the classifier in Classification Layer. The evaluation results for different classifiers are listed in Table 3.6, which also presents the average results of 30 independent experiments to reduce the influence of randomness.

As shown in Table 3.6, classifier RNN achieves the best recall 0.9083 and LSTM obtains the best accuracy 0.9112, F1 score 0.9116 and precision 0.9182. Therefore, LSTM has a weak advantage compared with DenseNN and RNN.

3.5.3 Remarks

In Section 3.5.2, we analyse the effect and influence of various design factors separately. In this section, we put things together and list the exploitability prediction results on ExBERT and BERT as well as 8 variants for both of them (ExBERT_V1 to ExBERT_V8 and BERT_V1 to BERT_V8) in Fig. 3.10 and Table 3.7. Obviously, ExBERT and its variants outperformed BERT and its variants with a large margin, improved by almost 14% on accuracy, F1 score, precision and recall.

Table 3.7: Performance comparison on ExBERT and BERT as well as their variants

BERT	Pooling strat.	Classifiers	Model	Best epoch	Loss	Acc.	F1	Prec.	Recall
Fine-tuned BERT	Mean	DenseNN	ExBERT_V1	16.17	0.2386	0.9063	0.9070	0.9106	0.9034
		RNN	ExBERT_V2	31.30	0.2391	0.9071	0.9079	0.9103	0.9055
		LSTM	ExBERT_V3	15.47	0.2385	0.9068	0.9076	0.9100	0.9053
	Max	DenseNN	ExBERT_V4	22.53	0.2421	0.9036	0.9044	0.9069	0.9019
		RNN	ExBERT_V5	17.57	0.2429	0.9034	0.9042	0.9074	0.9011
		LSTM	ExBERT_V6	28.27	0.2418	0.9034	0.9042	0.9066	0.9019
	[CLS]	DenseNN	ExBERT_V7	11.63	0.2348	0.9098	0.9106	0.9134	0.9078
		RNN	ExBERT_V8	14.73	0.2350	0.9102	0.9109	0.9136	**0.9083**
		LSTM	**ExBERT**	**11.57**	0.2354	**0.9112**	**0.9116**	**0.9182**	0.9051
Pre-trained BERT	**Mean**	DenseNN	BERT	65.30	0.4389	0.7996	**0.8032**	0.7978	**0.8088**
		RNN	BERT_V1	96.83	0.4510	0.7945	0.7959	0.7994	0.7925
		LSTM	BERT_V2	54.57	**0.4386**	**0.8000**	0.8029	**0.8002**	0.8058
	Max	DenseNN	BERT_V3	90.20	0.4670	0.7758	0.7735	0.7909	0.7573
		RNN	BERT_V4	88.26	0.4750	0.7749	0.7750	0.7837	0.7666
		LSTM	BERT_V5	94.40	0.4648	0.7795	0.7794	0.7889	0.7705
	[CLS]	DenseNN	BERT_V6	53.47	0.4516	0.7924	0.7943	0.7963	0.7926
		RNN	BERT_V7	95.90	0.4535	0.7879	0.7890	0.7939	0.7843
		LSTM	BERT_V8	**38.00**	0.4521	0.7917	0.7935	0.7957	0.7916

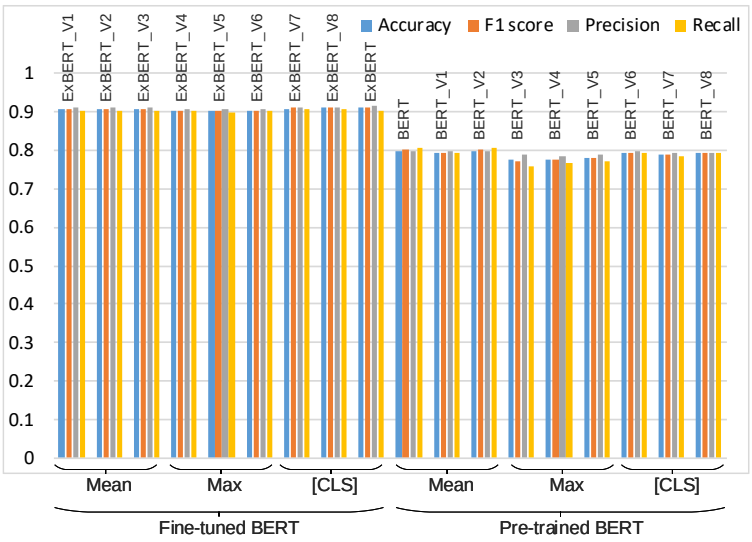


Fig. 3.10. Performance comparison on ExBERT and BERT as well as their variants

3.5.4 Comparison With Other Similar Works

We compare ExBERT with several existing algorithms from the aspects of feature extraction method, classification algorithm and four classification metrics in Table 3.8. All algorithms listed in Table 3.8 use vulnerability descriptions from the NVD database as the input feature to predict the exploitability of vulnerabilities. The results of other works are obtained from the original papers. Results in Table 3.8 show that ExBERT has greatly improved the performance of vulnerability exploitability prediction on accuracy, F1 score, precision and recall.

3.5.5 Result Analysis

Based on results presented in Fig. 3.8 to Fig. 3.10 and Table 3.3 to Table 3.7, we can draw the following conclusions.

(1) The proposed framework ExBERT achieves the state-of-the-art results for exploitability prediction problem. ExBERT can achieve 91.12% on the accuracy, 0.9116 on F1 score, 91.82% on precision and 90.51% on recall.

(2) The dominating factor for performance improvement is BERT fine-tuning

Table 3.8: Performance comparison with other similar works

Researchers	Feature extraction	Classifier	Acc.	F1	Prec.	Recall
Bozorgi[9]	TF-IDF	SVM	0.890	-	-	-
Nazgol Tavabi [11]	TF-IDF/Doc2Vec	SVM/RF	-	0.660	-	-
Zhuobing Han [55]	word embedding	CNN	0.816	0.816	0.818	0.815
Michel Edkrantz [50]	10,000 MCWs	SVM	0.833	-	0.825	0.834
ExBERT	**Fine-tuned BERT**	**LSTM**	**0.911**	**0.912**	**0.918**	**0.905**

via transfer learning other than pooling strategies or classifiers. Fig. 3.7, Fig. 3.8 and Table 3.3 have verified that only extracting [CLS] token embedding from the fine-tuned BERT can achieve desirable performance. Keeping other settings identical, extracting features from the pre-trained BERT can only get around 0.8 on all metrics.

(3) As demonstrated in Fig. 3.10 and Table 3.7, pooling strategy [CLS] can achieve the best performance on fine-tuned BERT, while 'Mean' is the best for the pre-trained BERT. 'Max' is the worst for both of them. The reason why [CLS] can achieve best results on the fine-tuned BERT is that [CLS] itself is a token embedding extracted from the fine-tuned BERT and it learns some semantic information of the entire sentence during the fine-tuning process.

(4) LSTM is the best classifier for ExBERT, as shown in Table 3.6 and 3.7. This verifies that LSTM is suitable for dealing with sequence inputs by capturing the dependencies within inputs.

3.6 Conclusion

To find the most possible exploitable vulnerabilities, this chapter proposes an exploitability prediction framework ExBERT to accurately predict if a vulnerability will be exploited or not. The experiment results on 46,176 real-world vulnerabilities show that ExBERT can achieve state-of-the-art performance on exploitability prediction.

Chapter 4

Online Vulnerability Exploitability Prediction

ExBERT, proposed in Chapter 3, has achieved good performance on vulnerability exploitability prediction. However, two drawbacks still exist. Firstly, it is implemented under the assumption that the data distribution and patterns for exploitability prediction are static, so ExBERT is trained and validated in an offline learning mode on randomly shuffled data between 1999 and 2020. Furthermore, ExBERT focuses on learning the semantic features from vulnerability descriptions but ignores other available sources of related information.

This chapter proposes a novel consecutive batch learning algorithm, called Real-time Dynamic Concept Adaptive Learning (RDCAL), to predict vulnerability exploitability in an online learning mode. RDCAL can deal with possible concept drift and dynamic class imbalance problems in a real-time exploitability prediction scenario. Furthermore, in addition to vulnerability description, multiple sources of vulnerability information are taken into account when extracting features, including information on user privilege, user interaction, availability, authentication, confidentiality, severity, etc.

This chapter starts with a literature review in Section 4.1, followed by the related works on concept drift and concept drift learning in Section 4.2. Then a detailed description of RDCAL is presented in Section 4.3. The real-time ex-

ploitability prediction results of RDCAL on real-world vulnerabilities from 1988 to 2020 are provided in Section 4.4, followed by a comparison with other baseline algorithms and a discussion. Section 4.5 concludes this chapter.

4.1 Introduction

In industry, most organisations prioritise their remediation efforts by overly relying on the Common Vulnerability Scoring System (CVSS) [30, 77]. However, CVSS has been found to be sub-optimal as an exploitability indicator. In some cases, it is no better than randomly choosing vulnerabilities to remediate [8, 78].

To complement CVSS, researchers seek to construct machine-learning and deep-learning-based predictive models by making use of a large collection of multiple open-source datasets together [45]. For example, Bozorgi et al. adopted an SVM classifier to predict whether vulnerabilities will be exploited within t ($t \geq 0$) days, operating on high dimensional feature vectors extracted from the text fields, time stamps, cross-references and other entries in the existing vulnerability disclosure reports[9]. Finally, the best performance of 79.82% was recorded in 2010 [9]. The work in [50] investigated the binary classification performance of exploitability prediction on a wide range of machine learning (ML) classifiers, including SVM, k-nearest neighbours (KNN), naive Bayes and random forests, achieving the best testing accuracy of 83% with the SVM algorithm on data collected from the NVD[37] and EDB [38] databases. In these previous works, text-related features are usually extracted using traditional statistical text processing techniques, such as the Term Frequency-–Inverse Document Frequency (TF—IDF) algorithm and common word counting, without capturing the context and obtaining semantic features at a high level.

With recent advances in natural language processing (NLP) [79, 80], techniques such as word embedding, sen2vec and Bidirectional Encoder Representations from Transformers (BERT) are employed to extract semantic features from vulnerability descriptions[11, 56], having achieved great success in classification performance. For example, by applying transfer learning to a pre-trained BERT model, the work in [7] achieved an accuracy of 91% in exploitability prediction.

Despite the advances, a close inspection reveals that the existing approaches make strong assumptions with respect to data distributions. In other words, they assume that all available data share the same data distribution, using batch learning to train the classifier and adopt hold-out evaluation to evaluate their models in a randomly separated test set. Due to the evolving system behaviours and environments, concept drift exists in the data distribution of both vulnerabilities and exploits [6]. As a result, traditional batch learning and hold-out evaluation can lead to an inflated performance because of unveiling unseen data in the future to construct the predictive model.

Considering the real-world applications of exploitability prediction, this chapter conducts concept drift learning and trains the classifier incrementally as new data becomes available. Furthermore, a prequential evaluation or an interleaved-test-then-train evaluation mode will be used to assess the real-time performance of the classifier. This means that new data would be used to test the classifier's performance before training the classifier. Undoubtedly, as an online learning mode, concept drift learning is more in line with a practical application than batch learning or offline learning [81]. It also enables the classifier to capture new concepts when new data arrive. Therefore, it can achieve more reliable exploitability prediction performance.

On the other hand, compared with batch learning, predicting exploitability with concept drift learning faces the following challenges due to the dynamic and incomplete nature of evolving data.

(1) Class label drift problem. A unique trait of vulnerabilities is that their exploitability is chronologically variable. In other words, at one time slice, a vulnerability may be labelled as unexploitable, since no corresponding published exploits exist. However, several months or years later, the vulnerability can become exploitable with proof-of-concept exploits available. In the batch learning scenario, both vulnerabilities and exploits are collected at a certain date. Thus, the time variation factor and label drift problem are ignored by previous studies. However, with concept drift learning, class label drift is a problem that has to be considered when collecting the data on vulnerabilities and exploits, evaluating and updating the classifier over time.

(2) Dynamic class imbalance problem. All data is available in batch learning scenarios, and the class imbalance status is static and determined. Therefore, existing solutions, such as resampling samples, generating synthetic samples and penalizing misclassification samples, can be applied directly. However, the magnitude of data imbalance is dynamically changing in the data stream. Probably, the minority class may become the majority class in a certain time slice. Therefore, more flexible and sensitive strategies are urged for handling the dynamic class imbalance problem in concept drift learning.

To partly solve the aforementioned problems, this chapter proposes an online learning algorithm called RDCAL to address these two challenges in concept drift learning and improve the practicability and classification performance for the exploitability prediction task. RDCAL comprises two strategies. One is the Class Rectification Strategy (CRS), which is designed to handle the actual drift in sample labels, and the other is the Balanced Window Strategy (BWS), which is used to boost the prediction performance of the minority class during real-time learning. The proposed RDCAL learning algorithm with a fully-connected neural networks (DenseNN) classifier achieves state-of-the-art performance on exploitability prediction in data stream learning scenarios, compared with the other five adaptive data stream learning algorithms.

4.2 Related Work

The online exploitability prediction problem is formulated as a concept drift learning problem. This section discusses some of the most important and related literature in data stream learning.

4.2.1 Concept Drift

Concept drift is a phenomenon in which the statistical properties of a target domain change over time in an arbitrary way [82]. Given a set of samples at a period of time $[0,\ t]$, denoted as $S_{0,t} = \{d_0,\ d_1,\ \cdots,\ d_t\}$, where $d_i{=}\{X_i,\ y_i\}$ is one data sample or instance observed at time step i, $X_i \in \mathbb{R}^n$ is a feature

vector in an n-dimensional feature space $\mathcal{X}$ and y_i is the corresponding label. Let $S_{0,t}$ follows a certain distribution $F_{0,t}(X, y)$, if $F_{0,t}(X, y) \neq F_{t+1,\infty}(X, y)$, concept drift occurs at time step t+1. The term concept drift at time step t can be defined as the change of joint probability of X and y at time step t, denoted as $\exists t : P_t(X, y) \neq P_{(t+1)}(X, y)$ [82]. Considering that $P_t(X, y)$ is determined by two parts as $P_t(X, y)$=$P_t(X) \times P_t(y|X)$, there are three main sources triggering a concept drift.

(1) Virtual drift: $P_t(X) \neq P_{(t+1)}(X)$ while $P_t(y|X)$=$P_{t+1}(y|X)$. Since $P_t(X)$ drift doesn't affect the decision boundary, it has been considered a virtual drift [82, 83].

(2) Actual drift: $P_t(y|X) \neq P_{t+1}(y|X)$ while $P_t(X) = P_{(t+1)}(X)$. When actual drift happens, the actual decision boundary changes. If the classifier cannot update accordingly, the performance will decrease.

(3) Hybrid drift: a mixture of virtual drift and actual drift.

4.2.2 Concept Drift Learning

According to when to handle concept drift, there are generally two learning strategies, namely, lazy and active. For the lazy strategy, concept drift learning consists of a drift detection process and a drift adaptation process [84]. When new data arrives, either data-distribution-based or error-rate-based detection algorithms are used to detect the occurrence of concept drift. ADaptive sliding WINdow (ADWIN) [85], for example, is a data-distribution-based detection algorithm, by calculating the absolute value of some statistics over two windows and comparing it with a pre-defined threshold to determine if drift occurs. PageHinkley [86], another example of data-distribution-based method, employs a Page-Hinkley test as a drift detector to monitor the features' magnitude of changes. Kolmogorov-Smirnov Windowing (KSWIN) is a concept change detection method based on the Kolmogorov-Smirnov (KS) statistical test [87]. Drift Detection Method (DDM) [88], Early Drift Detection Method (EDDM) [89] and Drift Detection Method based on Hoeffding's bounds (HDDM) [90] are examples of concept change detection methods based on learner's error rate. Once drift occurs, drift adaptation

algorithms will adjust the classifier model accordingly. For active strategy, the classifier updates constantly and incrementally when new data is available.

Generally speaking, the performance of lazy-strategy-based concept drift learning algorithms are more effective than active-strategy-based methods, due to less frequency of updating classifier. However, the drift detection process itself is also resource-consuming. Furthermore, the performance of lazy strategy algorithms is greatly limited to the sensitivity of drift detection algorithms.

For both learning strategies, basically, regarding how to handle concept drift there are four strategies. Firstly, redesign base classifiers, such as redesigning the nodes of decision tree [91] or the structures of Neural Networks [15]. Secondly, retrain or fine-tune the parameters or hyperparameters of the learner[41, 92]. Thirdly, adaptively change the training set formation methods, such as adjusting training windows, training sample selection strategies and training sample weights adaptively[93, 94, 95]. Lastly, fusion rules or classifier ensemble algorithms are also good choices for drift adaptation [92, 96, 97].

Among these four strategies, ensemble algorithms are the most popular to reach state-of-the-art performance. On the other hand, it also has a higher computational complexity. It is worth noting that these four strategies are not separated from each other. Instead, they are often combined with each other to achieve better performance.

4.3 Real-time Dynamic Concept Adaptive Learning

To avoid either the possible omission of concept drift detection with lazy strategy or the frequent classifier updating with active strategy, we make a trade-off. Specifically, this work adopts a consecutive batch learning strategy as the learning framework for exploitability prediction. With respect to concept drift adaptation, the proposed RDCAL algorithm involves a combination of classifier parameter fine-tuning and training set reformation.

4.3.1 Consecutive Batch Learning Framework

The workflow of the consecutive batch learning framework adopted by this work is shown in Fig. 4.1. As a general concept drift learning framework, it is algorithm-agnostic. In other words, the feature extraction algorithms, feature selection algorithms, classifier models and classifier updating strategies used in this framework can be flexibly selected without affecting how the whole framework works. We introduce the involved notations and main processes in the following subsections.

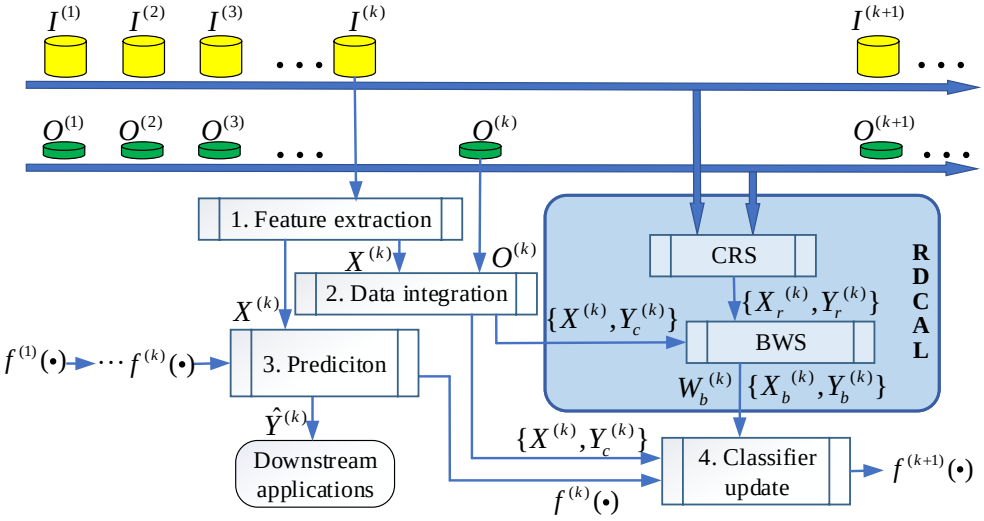


Fig. 4.1. Consecutive batch learning framework.

4.3.1.1 Feature extraction

As shown in Fig. 4.1, $\mathcal{I}=\{I^{(1)}, \cdots, I^{(k)}, \cdots\}$ is a sequence of consecutive raw input data batches in chronological order. Each data batch $I^{(k)}$ contains N_a raw samples arriving in a time slice $T^{(k)}$. Different feature extraction and feature selection algorithms can be used to extract the numerical features $\mathcal{X}=\{X^{(1)}, \cdots, X^{(k)}, \cdots\}$ from $\mathcal{I}$, where $X^{(k)}=\{X_1^{(k)}, X_2^{(k)}, \cdots, X_{N_a}^{(k)}\}$ is the feature set extracted from the k-th input raw data batch $I^{(k)}$; $X^{(k)} \in \mathbb{R}^{N_a \times n}$; N_a is the number of samples in the data batch; n is the dimension of the extracted sample feature; $X_i^{(k)} \in \mathbb{R}^n$ $(i \in [1, N_a])$ is the i-th sample in the data batch $X^{(k)}$.

4.3.1.2 Prediciton

The notation in Fig. 4.1 $f^{(1)}(\cdot), \cdots, f^{(k)}(\cdot), \cdots$ represents a sequence of different status of the same classifier with different parameters, where $f^{(k)}(\cdot)$ represents the classifier used to predict the output label at the k-th time slice. $f^{(k+1)}$ is the sequential status fine-tuned from $f^{(k)}$ based on the labelled data in the k-th time slice and the learning strategies adopted. If no prior data or knowledge is available, $f^{(1)}(\cdot)$ can be initialized with random parameters. Otherwise, it can be initialized with a pre-trained model.

Once feature $X^{(k)}$ is extracted from raw data batch $I^{(k)}$, the corresponding predicted label $\widehat{Y}^{(k)}$ can be calculated by (4.1).

$$\widehat{Y}^{(k)} = f^{(k)}(X^{(k)}), (k \geq 1). \tag{4.1}$$

$\widehat{\mathcal{Y}}$=$\{\widehat{Y}^{(1)}, \widehat{Y}^{(2)}, \cdots, \widehat{Y}^{(k)}, \cdots\}$ represents the sequence of consecutive predicted label batches, which can be used by downstream applications before real labels are available.

4.3.1.3 Data integration

The notation $\mathcal{O}$=$\{O^{(1)}, \cdots, O^{(k)}, \cdots\}$ in Fig. 4.1 represents a sequence of raw output data stream, where $O^{(k)}$ is the batch collected within $T^{(k)}$, the same time period with $I^{(k)}$. Note that, the size of $O^{(k)}$ is not necessarily the same as $I^{(k)}$ and the samples in $I^{(k)}$ and $O^{(k)}$ are not in one-to-one correspondence.

Considering the situation in cybersecurity, let $I^{(k)}$= $\{I_1^{(k)}, \cdots, I_{N_a}^{(k)}\}$ be a batch of latest disclosed vulnerabilities within the time period of $T^{(k)}$, where $I_i^{(k)}$ (i=1, $\cdots$, N_a) is the i-th vulnerabilities. $O^{(k)}$ is a batch of exploits published in the same time period $T^{(k)}$. Obviously, $O^{(k)}$ can exploit vulnerabilities in $I^{(k)}$ and other historical vulnerabilities in $I^{(1)}, \cdots, I^{(k-1)}$ as well as some unknown vulnerabilities not included in $\mathcal{O}$. Exploits in $O^{(k)}$ contain the CVE-ID (a globally unique vulnerability identifier) information of the exploited vulnerabilities, making it possible to integrate the exploit data batch and the vulnerability data batch. Specifically, if the vulnerabilities in $I^{(k)}$ are exploited by exploits in $O^{(k)}$, the corresponding labels are 1 (exploitable), otherwise, are 0 (unexploitable).

Generally speaking, each raw data in $I^{(k)}$ has a globally unique Sample Identification (SID), which is also inherited by the data in $X^{(k)}$. Through integrating $\mathcal{I}$ and $\mathcal{O}$ with the SIDs of raw data, a sequence of class labels in batches $\mathcal{Y}_c$=$\{Y_c^{(1)}$, $\cdots$, $Y_c^{(k)}$, $\cdots\}$ can be obtained. The subscript c is the capital of 'current', which means the labels are obtained by integrating the current output data batch $O^{(k)}$ collected in the current time period T^k. $Y_c^{(k)}$=$\{Y_{c1}^{(k)}$, $\cdots$, $Y_{cN_a}^{(k)}\}$ is the batch labels corresponding to $I^{(k)}$ and $X^{(k)}$. The value of $Y_{ci}^{(k)}$ (i=1, 2, $\cdots$, N_a) is calculated by (4.2).

$$Y_{ci}^{(k)}(i = 1, 2, \cdots, N_a) = \begin{cases} 1, \text{ if } f_{SID}(X_i^{(k)}) \in f_{SID}(O^{(k)}) \\ 0, \text{ if } f_{SID}(X_i^{(k)}) \notin f_{SID}(O^{(k)}) \end{cases}, \tag{4.2}$$

where $X_i^{(k)}$ is the i-th data in $X^{(k)}$ and the function of $f_{SID}(\cdot)$ is to find out the appeared SID set.

4.3.1.4 Classifier update

Let $D_t^{(k)}$=$\{X^{(k)}$, $Y_c^{(k)}\}$ be a labelled training set for the k-th time slice and $W^{(k)}$ = **ones**(N_a, 1) be the corresponding batch sample weight. The function **ones**($\cdot$) means to generate an array according to the specified dimensions, filled with 1. If no learning strategies are applied to optimise the performance of the consecutive batch learning framework, classifier would be updated from $f^{(k)}(\cdot)$ to $f^{(k+1)}(\cdot)$ by fitting $D_t^{(k)}$ with a sample weight of $W^{(k)}$.

Algorithm 4.1 is the pseudocode of the above-mentioned consecutive batch learning framework. Line 3, 4, 5 and 16 are four main steps executed at each data batch. Lines 6 to 15 related to the RDCAL learning strategy will be covered in Section 4.3.2.

Algorithm 4.1 Consecutive batch learning framework

Input: $\mathfrak{I}=\{I^{(1)}, \cdots, I^{(k)}, \cdots\}$; N_a; $\mathfrak{O}=\{O^{(1)}, \cdots, O^{(k)}, \cdots\}$; $f^{(1)}(\cdot)$. **Output:** $\mathfrak{X}=\{X^{(1)}, \cdots, X^{(k)}, \cdots\}$; $\widehat{\mathfrak{y}}=\{\widehat{Y}^{(1)}, \widehat{Y}^{(2)}, \cdots, \widehat{Y}^{(k)}, \cdots\}$; $\mathfrak{y}_c=\{Y_c^{(1)}, \cdots, Y_c^{(k)}, \cdots\}$; $\mathfrak{D}_t=\{D_t^{(1)}, \cdots, D_t^{(k)}, \cdots\}$; $f^{(2)}(\cdot)$, $f^{(3)}(\cdot)$, $\cdots$, $f^{(k+1)}(\cdot)$, $\cdots$.

1: $W^{(k)}$=**ones**(N_a,1); $D_t^{(k)}=\varnothing$
2: **for** each $k \geq 1$ **do**
3: Feature extraction: extract and select features $X^{(k)} \in \mathbb{R}^{N_a \times n}$ from $I^{(k)}$
4: Prediciton: predict labels $\widehat{Y}^{(k)}$ by calculating $\widehat{Y}^{(k)}=f^{(k)}(X^{(k)})$
5: Data integration: integrate $X^{(k)}$ and $O^{(k)}$ to obtain $D_t^{(k)} = \{X^{(k)}, Y_c{}^{(k)}\}$
6: **if** RDCAL== True **then**
7: **if** CRS== True **then**
8: run Algorithm 4.2 and get $D_r^{(k)}$
9: $D_t^{(k)} = D_t^{(k)} \cup D_r^{(k)}$
10: **end if**
11: **if** BWS== True **then**
12: run Algorithm 4.3 and get $D_b^{(k)}$ and $W_b^{(k)}$
13: $D_t^{(k)} = D_b^{(k)}$; $W^{(k)}=W_b^{(k)}$
14: **end if**
15: **end if**
16: Classifier update: update $f^{(k)}$ to $f^{(k+1)}$ based on $D_t^{(k)}$ and $W^{(k)}$
17: **end for**

4.3.2 Real-time Dynamic Concept Adaptive Learning

RDCAL is a general learning strategy used to improve the performance of consecutive batch learning framework, when existing class label drift and dynamic class imbalance. Specifically, RDCAL employs a Class Rectification Strategy (CRS) to handle the actual drift problem and a Balanced Window Strategy (BWS) to deal with the dynamic class imbalance problem. RDCAL achieves better performance by optimizing the labelled training set $D_t^{(k)}$ and the corresponding sample weight $W^{(k)}$ applied to update the classifier over time.

As shown in Fig. 4.1, the blue quadrangle named RDCAL is the proposed learning strategy. The specific implementation of RDCAL is listed in line 6-15 of

Algorithm 4.1. It is worth noting that CRS and BWS are optional for RDCAL. In real-world applications, they can be implemented separately or combinedly, depending on the existing problem in the corresponding data stream. However, if both of them are employed, CRS should be applied before BWS. The detailed implementations of CRS and BWS are given in Section 4.3.3 and 4.3.4 accordingly.

4.3.3 Class Rectification Strategy

CRS is designed to handle the class drift problem, which is also described as an actual drift in Section 4.2.1.

To adjust the classifier in real-time, the sample labels $Y_c^{(k)}$ in the training set $D_t^{(k)}$ of batch k ($k \geq 1$) are the current labels determined by the current output data $O^{(k)}$. Once the label of a sample has changed in a later time, according to a vanilla consecutive batch learning framework, the classifier has no chance to learn from the actually drifted samples, where vanilla means a naive version without any learning strategy, i.e. Algorithm 4.1 when RDCAL==False.

CRS is a strategy to rectify the label of the historical data. Once a class label drift is detected, the corresponding sample will be added into a rectified set. The rectified set works as a supplement to the original training set to finetune the classifier in real-time. Specifically, as shown in Algorithm 4.1, in a general consecutive batch learning framework, if RDCAL==True and CRS==True, Algorithm 4.1 will go to Algorithm 4.2.

For each k-th time slice, the input for Algorithm 4.2 includes historical extracted feature batches $\mathcal{X}_h^{(k)} = \{X^{(1)}, \cdots, X^{(k-1)}\}$, historical label batches $\mathcal{Y}_{hc}^{(k)} = \{Y_c^{(1)}, \cdots, Y_c^{(k-1)}\}$, and the current output batch $O^{(k)}$.

To start with, initialise $S{=}\varnothing$ and $D_r^{(k)}{=}\varnothing$, where S is a temporary set to hold the SIDs found in the current output batch $O^{(k)}$; $D_r^{(k)}$ is used to hold all rectified samples in the current time slice. If k equals 1, because no historical samples exist to rectify, CRS returns an empty $D_r^{(k)}$.

When $k > 1$, there are two steps to form a rectified set. Lines 6-11 in Algorithm 4.2 specify the first step, obtaining all SIDs appeared in current output data $O^{(k)}$. The historical data corresponding to these SIDs may have a class label drift. Step 2, described in lines 13-20 of Algorithm 4.2, rectifies the historical samples one

by one. Once a rectified sample x_h is identified, set its rectified label y_r to 1 and add this rectified sample $\{x_h, y_r\}$ to $D_r^{(k)}$.

Finally, a rectified set $D_r^{(k)}$ for time slice k is returned to Algorithm 4.1 line 8 and will be merged with the original $D_t^{(k)}$ and from a new $D_t^{(k)}$ as discribed in Algorithm 4.1 line 9.

Algorithm 4.2 Class Rectification Strategy

Input: $\mathcal{X}_h^{(k)}=\{X^{(1)}, \cdots, X^{(k-1)}\}$; $\mathcal{Y}_{hc}^{(k)}=\{Y_c^{(1)}, \cdots, Y_c^{(k-1)}\}$; $O^{(k)}$.
Output: $D_r^{(k)}$.

1: $S=\varnothing$; $D_r^{(k)}=\varnothing$
2: **if** $k==1$ **then**
3: **break**
4: **else**
5: # step (1): obtain SIDs in $O^{(k)}$.
6: **for** $O_i^{(k)}$ in $O^{(k)}$ **do**
7: $s=f_{SID}(O_i^{(k)})$
8: **if** $s \neq \varnothing$ **then**
9: $S=S \cup \{s\}$
10: **end if**
11: **end for**
12: # step (2): rectify history dataset $\{\mathcal{X}_h^{(k)}, \mathcal{Y}_{hc}^{(k)}\}$.
13: **for** s in S **do**
14: **for** x_h, y_{hc} in **zip**$(\mathcal{X}_h^{(k)}, \mathcal{Y}_{hc}^{(k)})$ **do**
15: **if** $s == f_{SID}(x_h)$ **and** $y_{hc}==0$ **then**
16: $y_r=1$
17: $D_r^{(k)}=D_r^{(k)} \cup \{x_h, y_r\}$
18: **end if**
19: **end for**
20: **end for**
21: **end if**
22: **return** $D_r^{(k)}$

4.3.4 Balanced Window Strategy

BWS is designed to cope with the dynamic class imbalance problem in stream learning. The basic idea behind BWS is to keep a balanced training set $D_b^{(k)}$ at each time slice k.

Basically, in the vanilla consecutive batch learning framework, the training set $D_t^{(k)}$ used to update classifier from $f^{(k)}(\cdot)$ to $f^{(k+1)}(\cdot)$ is imbalanced, no matter applying CRS or not. To make things worse, the imbalance status within $D_t^{(k)}$ changes irregularly and dynamically over time.

BWS is a strategy to keep a balanced window to dynamically hold the latest N_b negative samples and N_b positive samples as the balanced training set $D_b^{(k)}$ for the current k-th time slice. The class balance size N_b ($N_b \geq N_a$) is a hyperparameter of BWS to control the size of each class within $D_b^{(k)}$.

To keep $D_b^{(k)}$ balanced, the samples belonging to the current minority class will stay more time slices in the balanced window. To avoid the possible over-fitting caused by multiple-times training on the same minority class samples, BWS designs a balanced sample weight $W_b^{(k)}$ to control the training weight of each sample. When it is the first time slice at which a sample occurs in $D_b^{(k)}$, its corresponding balanced sample weight $w_b^{(k)}$ =1. After that, $w_b^{(k)}$ is related to N, the number of time slices that the corresponding sample has stayed in the balanced window. Specifically, $w_b^{(k)}$ can be calculated by (4.3).

$$w_b^{(k)} = \alpha^{N-1}, (N \geqslant 1), \tag{4.3}$$

where the time decay factor α ($\alpha \in (0,1]$) is another hyperparameter for BWS.

Since classes in $D_b^{(k)}$ are balanced, it can partially solve the class imbalance problem. $W_b^{(k)}$ is a mechanism to penalise those samples staying too long at $D_b^{(k)}$ to avoid over-fitting. Combining these two factors, BWS provides an effective solution to the dynamic class imbalance problem.

Regarding implementation, as shown in Algorithm 4.1, in a general consecutive batch learning framework, if RDCAL==True and BWS==True, Algorithm 4.1 will go to Algorithm 4.3. For the k-th time slice, the inputs of BWS include existing training set $D_t^{(k)}$ and sample weight $W^{(k)}$, the class balance size N_b and the time decay factor α.

To start with, if k equals 1, no historical balanced dataset and sample weight are available to work as old data. Therefore, both D_{old} and W_{old} are initialized as $\varnothing$, as shown in line 2 of Algorithm 4.3. Otherwise, the ${D_b}^{(k)}$ will act as the D_{old} for the next time slice, and ${W_b}^{(k)}$ will decay in a rate α and then work as W_{old} for the next time slice, as shown in line 15 of Algorithm 4.3.

For each time slice k, as shown in line 4 of Algorithm 4.3, we first concatenate the old data and current existing data to generate the initial balanced dataset ${D_b}^{(k)}$ and the corresponding sample weight ${W_b}^{(k)}$. Then, check if the sample size of each class in ${D_b}^{(k)}$ is bigger than the pre-set class balance size N_b. If yes, only keep the latest N_b samples and their balanced sample weights for each class, as shown in lines 5-14 in Algorithm 4.3.

Finally, the ${D_b}^{(k)}$ and ${W_b}^{(k)}$ is returned to Algorithm 4.1 and worked as the new training set $D_t^{(k)}$ and sample weight $W^{(k)}$ to update the classifier, as shown in lines 12-13 in Algorithm 4.1.

Algorithm 4.3 Balanced Window Strategy

Input: $D_t^{(k)}$; $W^{(k)}$; N_b; α.

Output: $D_b^{(k)}$; $W_b^{(k)}$.

if k=1 **then**
    $D_{old} = \varnothing$; $W_{old} = \varnothing$
end if
$D_b^{(k)}$ = concatenate(D_{old}, $D_t^{(k)}$); $W_b^{(k)}$ = concatenate(W_{old}, $W^{(k)}$)
get the indexes of all negative samples $idx0$
if len($idx0 > N_b$) **then**
    $idx0 = idx0$[-N_b : -1]
end if
get the indexes of all positive samples $idx1$
if len($idx1 > N_b$) **then**
    $idx1 = idx1$[-N_b : -1]
end if
$idx = idx0 \cup idx1$
$D_b^{(k)} = D_b^{(k)}[idx]$; $W_b^{(k)} = W_b^{(k)}[idx]$
$D_{old} = D_b^{(k)}$; $W_{old} = \alpha * W_b^{(k)}$
return $D_b^{(k)}$; $W_b^{(k)}$

4.4 Experimental Study

In this part, we validate the performance of RDCAL to learn the real-time dynamic patterns on a real-world vulnerability dataset. First, we set the experiments in Section 4.4.1. Then, we compare the performance of four classifiers, namely, DensNN, HoeffdingTree, SVM and LR, when applying and without applying RDCAL in Section 4.4.2, to verify if RDCAL is classifier-agnostic. Furthermore, we compared the performance of RDCAL and other five drift adaptation algorithms on the task of exploitability prediction in Section 4.4.3. Finally, we analyse the effects of hyperparameters of RDCAL in Section 4.4.4.

4.4.1 Experimental Setting

4.4.1.1 Dataset

Data source We validate RDCAL on a real-world dataset containing 140,758 vulnerabilities disclosed between 1988 and 2020. 23,413 of them have found corresponding exploits in ExploitDB, recognized as positive sample. Specifically, the National Vulnerability Database works as the consecutive raw input data stream $\mathcal{I}$, while ExploitDB provides the consecutive raw output data stream $\mathcal{O}$. The CVE-ID works as the SID to integrate NVD and EDB.

Data stream trend Fig. 4.2 shows the monthly number of disclosed vulnerabilities and exploits from 1988 to 2020, demonstrating the number of disclosed vulnerabilities are soaring and much more than available exploits in recent years. Therefore, accurate exploitability prediction is of importance to improve the remediation efficiency through filtering out low-risk vulnerabilities.

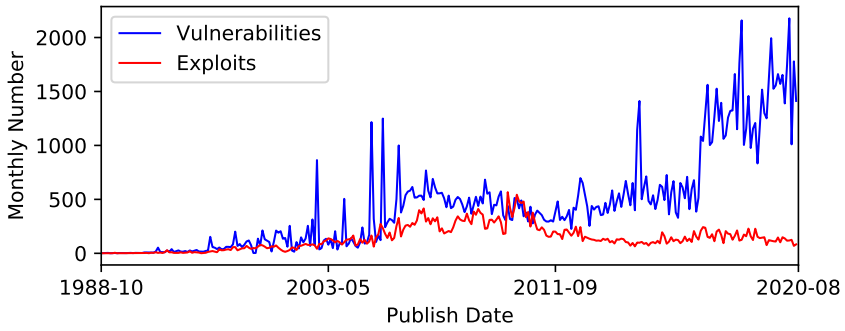


Fig. 4.2. Monthly number of disclosed vulnerabilities and exploits from 1988 to 2020.

Dynamic class imbalance status When dealing with the collected vulnerabilities and exploits as a real-time data stream, the real-time dynamic class proportion of the current label and rectified label is shown in Fig. 4.3. The dynamic class proportion is calculated by the Sliding Window Imbalance Factor Technique, proposed by [41], setting the window size $z = 1000$. The current label

y_c of each vulnerability is obtained following the formula (4.2), where Na sets to 1. The rectified label y_r of each vulnerability is obtained following Algorithm 4.3, where Na is also set to 1.

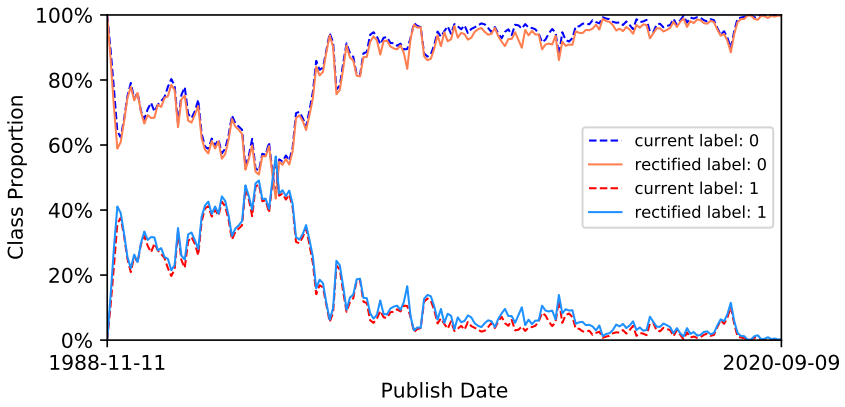


Fig. 4.3. Real-time dynamic class proportion status comparison of current label and rectified label from 1988 to 2020.

On one hand, Fig. 4.3 shows the dynamic changing trends of class imbalance status. At first, the proportion of class 0 was higher than class 1, and then decreased moderately to about 40%, which is lower than class 1. But, it went up to 80% shortly and then stabilized and in the interval of [80%, 100%]. On the other hand, Fig. 4.3 illustrates only a small portion of vulnerabilities suffers class drift problem. Therefore the proportion of rectified label 1 drawn in blue line is only a bit higher than the current label 1 drawn in red dashed line.

Class rectification To visualise the real-time class rectified vulnerabilities, we draw Fig. 4.4. Set $N_a = 1$, and once a vulnerability in $\mathcal{J}$ was disclosed, we will search for the corresponding exploits published in the same current time slice (the time period between current vulnerability and last vulnerability disclosed) in $\mathcal{O}$. These corresponding exploits will be used to rectify all existing vulnerabilities. The horizontal axis in Fig. 4.4 is the exploit publish date and is also the class rectification date. The vertical axis is the publish date of these rectified vulnerabilities.

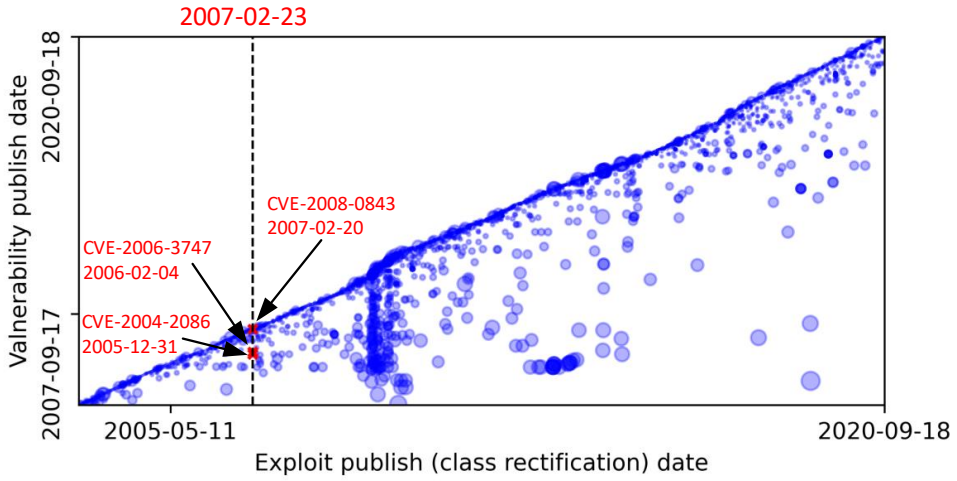


Fig. 4.4. Real-time class rectification results.

Taking the date 2007-02-23 as an example. A vulnerability CVE-2007-1083 was published on that date. To obtain the current label, we check all the exploits published on 2007-02-23, E_{list} = ['EXPLOIT-DB:25452', 'EXPLOIT-DB:3362', 'EXPLOIT-DB:3363', 'EXPLOIT-DB:3364', 'EXPLOIT-DB:3365', 'EXPLOIT-DB:3366', 'EXPLOIT-DB:3367', 'EXPLOIT-DB:29641', 'EXPLOIT-DB:29642', 'EXPLOIT-DB:29640', 'EXPLOIT-DB:29643']. Execute $f_{SID}(E_{list})$, all exploited vulnerabilities are found, V_{list}= ['CVE-2005-4832', 'CVE-2006-5276', 'CVE-2006-0549', 'CVE-2005-4832', 'CVE-2007-1133', 'CVE-2007-1130', 'CVE-2007-1131', 'CVE-2007-1126', 'CVE-2007-1124', 'CVE-2007-1127', 'CVE-2007-1125']. Since CVE-2007-1083 is not in the list, we can set its current label as unexploitable. Check the exploitability of vulnerabilities in V_{list}, we find that only three of them, namely CVE-2005-4832, CVE-2006-0549 and CVE-2006-5276 are labelled as unexploitable. Therefore, their class labels are rectified from unexploitable to exploitable on 2007-02-23. Specifically, we mark them with red '×' markers and also annotate their CVE-ID and publish date in Fig. 4.4.

4.4.1.2 Evaluation Metrics.

As exploitability prediction is binary-classification, we adopt four widely used evaluation metrics, namely, Accuracy, Precision, Recall and F1 score as evaluation

metrics. Furthermore, as we deal with stream data, to further evaluate algorithms over different time slices, the geometric mean (G-mean) is also calculated as an evaluation metric, following [41]. The definition of G-mean is shown in (4.4),

$$\text{G-mean}(x_1, x_2, \cdots, x_n) = \sqrt[n]{x_1 \times x_2 \times \cdots \times x_n} \tag{4.4}$$

where x_1, x_2, $\cdots$, x_n are the n elements to calculate the G-mean of them [41].

4.4.1.3 Feature Extraction and Selection

We extract features from raw input database NVD. Both vulnerability description and CVSS metrics are available. On one hand, we follow [7] and apply a fine-tuned BERT model to extract semantic features from vulnerability description. On the other hand, we select the identical CVSS V2.0 metrics with [41] as tabular features, applying one-hot encoding to transfer categorical features into one-hot numeric arrays. Finally, to reduce the computational complexity, 10 features from each side are selected via Principal Component Analysis (PCA) to concatenate a 20-dimensional feature set for exploitability prediction.

4.4.2 RDCAL Versus Vanilla Learning

To verify the effectiveness of RDCAL, we compare the performance of four classifiers, namely, DensNN, HoeffdingTree, SVM and LR, when using RDCAL and without using RDCAL. Specifically, DensNN is a fully-connected Neural Network with a 10-node-hidden layer, implemented with Keras [98]; HoeffdingTree is an incremental decision tree induction algorithm, implemented with a python package, scikit-multiflow [99]; SVM and LR are two traditional machine learning classifiers, implemented with scikit-learn [100]. The parameters for these classifiers keep default setting. When applying RDCAL, the hyperparameters are set as N_a=200, N_b=200 and α=0.4.

Table 4.1 summarises the experiment results, where classifier with a '_Vanilla' means using Algorithm 4.1 without RDCAL, while classifier with a '_RDCAL' means using Algorithm 4.1 with RDCAL. All classifiers in Table 4.1 are pre-trained with the first 100 samples, and then are evaluated in an interleaved-test-then-train evaluation mode.

Table 4.1: Overall performance comparison between Vanilla and RDCAL strategy

Classifier	Accuracy	Precision	Recall	F1 Score	G-mean	△G-mean
DenseNN_Vanilla	89.53%	84.21%	74.03%	78.70%	81.41%	0
DenseNN_RDCAL	90.31%	81.01%	82.88%	81.92%	83.95%	3.12%
HoeffdingTree_Vanilla	89.73%	**84.76**%	74.32%	79.15%	81.78%	0
HoeffdingTree_RDCAL	**90.59**%	81.61%	**83.25%**	**82.37**%	**84.37%**	3.16%
SVM_Vanilla	88.61%	83.03%	70.88%	76.41%	79.45%	0
SVM_RDCAL	89.73%	79.06%	82.88%	80.92%	83.05%	**4.53%**
LR_Vanilla	88.10%	80.99%	71.22%	75.77%	78.77%	0
LR_RDCAL	88.79%	76.49%	82.11%	79.19%	81.52%	3.49%

Values in columns Accuracy, Precision, Recall and F1 Score are the geometric mean of these metrics over all time-slice. Columns G-mean represents the G-mean of these four metrics. The last column △G-mean is calculated by (4.5).

$$\triangle\text{G-mean} = \frac{\text{G-mean(C\_RDCAL)} - \text{G-mean(C\_Vanilla)}}{\text{G-mean(C\_Vanilla)}} \times 100\% \tag{4.5}$$

Where 'C_Vanilla ' is the Vanilla learning version with the classifier C and 'C_RDCAL ' is the RDCAL learning version.

As shown in Table 4.1 and Fig. 4.5, RDCAL can improve the overall performance of all these four classifiers for more than 3%. Especially, for SVM, RDCAL learning strategy makes a significant improvement of 4.53%. Therefore, RDCAL is classifier-agnostic for improving the performance of concept drift learning with class drift problem and dynamic class imbalance problem.

Table 4.1 gives more details. For all these four classifiers, RDCAL can increase the Accuracy, Recall and F1 Score, but causes a decrease in the Precision. For the exploitability prediction problem, the positive samples are more valuable than negative samples. In other words, Recall is a more important metric than Precision. Therefore, the reduction in Precision caused by the promotion of Recall is acceptable. By comparing the G-mean of these four algorithms, we can see that HoeffdingTree achieves the best performance among both the Vanilla

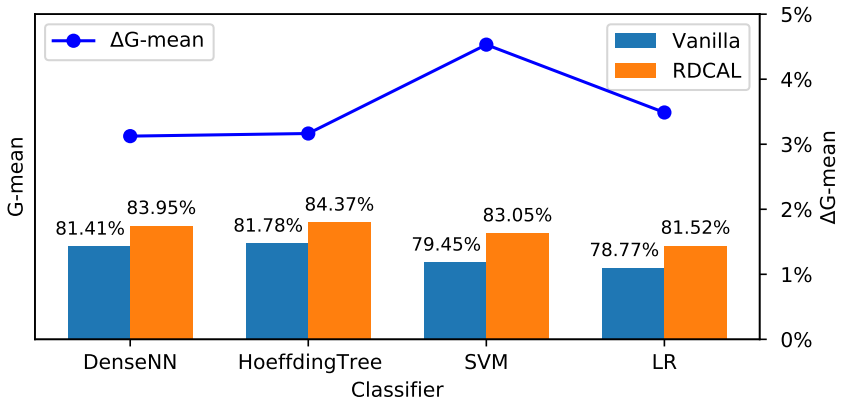


Fig. 4.5. Overall G-mean and △G-mean comparison between Vanilla and RDCAL learning strategy over four different classifiers

version classifiers and the RDCAL version classifiers. DenseNN takes the second place. LR, limited by its simplicity, performs worst.

Fig. 4.6 presents the corresponding real-time performance of these RDCAL version classifiers. The publish date starts from 1995-11-01 instead of from 1988, because the first 100 samples are used to pre-train the classifiers and thus are not reported. An interesting result reflected in Fig. 4.6 is that, in the early years, HoeffdingTree had an obvious advantage over DenseNN. However, in recent years, DenseNN gradually catches up and passes HoeffdingTree. A reasonable guess is that with the accumulation of learning data, the neural network algorithm gradually shows its advantages on learning complex data patterns. Another thing is that the performance of Precision, Recall and F1 Score have a similar fluctuating trend with label 1 in Fig. 4.3.

4.4.3 RDCAL Versus Other Drift Adaptive algorithms

In this section, we compared the performance of RDCAL with five other concept drift adaptation algorithms on the task of exploitability prediction. The experimental setting for all involved algorithms is specified below.

- RDCAL is the proposed consecutive batch learning with RDCAL strategy.

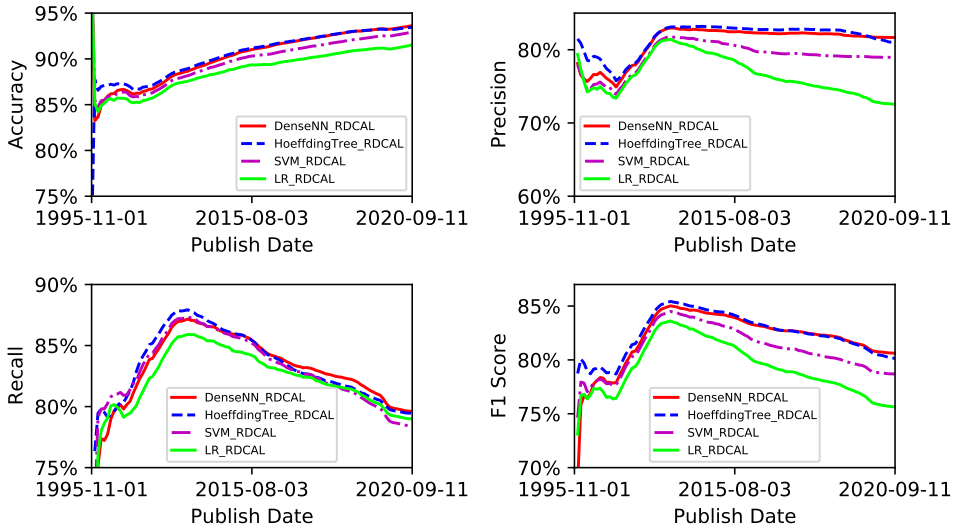


Fig. 4.6. Real-time performance comparison between four different classifiers with RDCAL learning strategy

The classifier used in this section is DenseNN and hyperparameters are set as N_a=200, N_b=500 and α=0.9.

- SAMKNN is the Self Adjusting Memory [93] model which builds an ensemble with models targeting current or former concepts.
- DWM is the dynamic weighted majority algorithm [96], which keeps a dynamic online weighted learner ensemble by operations like training, weighting, removing and adding base learners to cope with concept drift.
- HTA is the HoeffdingTree Classifier [101] employing ADWIN [85] to detect concept drift and bootstrapping strategy to get better performance.
- LPPNSE is the Learn++.NSE ensemble classifier [97], which is an incremental learning algorithm for all kinds of concept drift, including addition or deletion of concept classes.
- VFDRC is the Very Fast Decision Rules classifier [94], which is an incremental rule learning classifier to adapt with concept drift.

All of the above-mentioned adaptive algorithms except DenseNN are implemented using the python package, scikit-multiflow [99] in this chapter.

The average results of 10 times of independent experiments are shown in Table 4.2 and Fig. 4.7. Regarding the overall G-mean, RDCAL performs best at 83.20±0.05%. The second place is HTA at 82.16±0.46%. Next are SAMKNN and DWM, achieving 81.35±0.07% and 81.05±0.09%. Both LPPNSE and VFDRC have poor performance on this task, only obtaining around 74% of G-mean. As for single metric, Table 4.2 shows that RDCAL achieves the best Recall at 86.05±0.10% and best F1 Score at 81.11±0.05% with a large margin above others. However, HTA achieves the best Accuracy with a small advantage over others. It also achieves the best Precision at 81.52±0.59%.

Table 4.2: Overall performance comparison between RDCAL and other five drift adaptation algorithms

Algorithms	Accuracy	Precision	Recall	F1 Score	G-mean
RDCAL	89.48±0.03%	76.72±0.06%	**86.05±0.10%**	**81.11±0.05%**	**83.20±0.05%**
SAMKNN	89.19±0.04%	81.04±0.06%	76.83±0.11%	78.86±0.08%	81.35±0.07%
DWM	88.22±0.12%	75.30±0.24%	82.52±0.08%	78.73±0.09%	81.05±0.09%
HTA	**89.57±0.19%**	**81.52±0.59%**	78.26±1.18%	79.74±0.61%	82.16±0.46%
LPPNSE	84.73±0.12%	70.77±0.34%	71.20±0.16%	70.98±0.18%	74.19±0.17%
VFDRC	84.38±0.75%	69.63±2.08%	72.51±2.12%	70.98±1.09%	74.14±0.99%

Since HTA is an improvement based on HoeffdingTree, we can compare the performance of HTA with HoeffdingTree_Vanilla and HoeffdingTree_RDCAL in Table 4.1. The overall G-mean of HoeffdingTree_Vanilla is 81.78%, lower than HTA at 82.16%, showing the effectiveness of HTA in improving the performance of HoeffdingTree classifier. However, compared with the G-mean of HoeffdingTree_RDCAL at 84.37%, it is obvious that RDCAL has a much better effect on improving the performance of HoeffdingTree.

Fig. 4.7 shows the real-time performance on these four metrics. Consistent with Table 4.2, RDCAL, SAMKNN, DWM achieve equivalent best performance on Accuracy; HTA and SAMKNN are the best on Precision; RDCAL alone achieves the best Recall with an overwhelming advantage; RDCAL and HAT

achieve equivalent good performance on F1 score, followed by SAMKNN and DWM. Although, in recent years, the proportion of class 1 goes down sharply to about 10%, we can see that the Recall of RDCAL is still quite stable, compared with other algorithms.

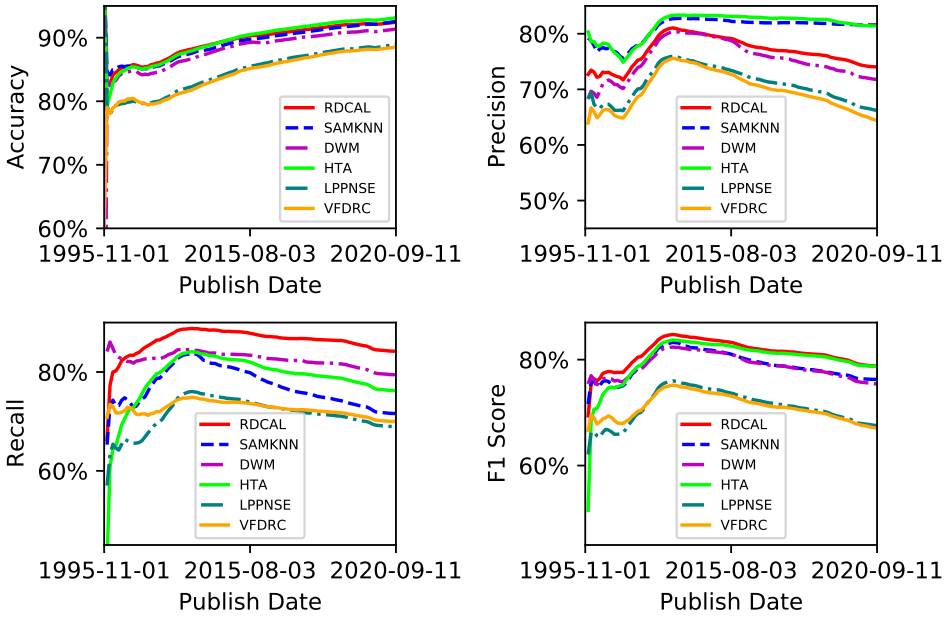


Fig. 4.7. Real-time performance comparison between RDCAL and other five drift adaptation algorithms

Both Table 4.2 and Fig. 4.7 demonstrate that RDCAL is the best concept adaptation algorithm in this exploitability prediction task. RDCAL only adopt a single classifier during the whole online learning process. However, its performance is even better than ensemble algorithms with multiple classifiers, such as SAMKNN, DWM and LPPNSE.

4.4.4 Hyperparameter Influence Analysis

The parameters of classifiers can be learnt from data automatically. However, the hyperparameters of RDCAL, namely, N_a, N_b and α, should be elaborately adjusted. As shown in Table 4.1, DenseNN_RDCAL achieves G-mean of 83.95%

by setting N_a=200, N_b=200 and α=0.4. However, in Table 4.2, with the same DenseNN classifier, RDCAL only achieves 83.20% on G-mean, when setting N_a=200, N_b=500 and α=0.9. Therefore, in this section, we discuss how these three hyperparameters affect the performance of RDCAL. Furthermore, since RDCAL consists of two optional learning strategies, CRS and BWS, we also discuss the effect of them separately.

Therefore, we study CRS and BWS separately under different settings of N_a, N_b and α in the following subsections. All experiments adopt the same consecutive batch learning framework described in Algorithm 4.1, employing an identical DenseNN as the classifier. Baseline is Algorithm 4.1 with neither CRS nor BWS, setting the consecutive batch size N_a to 200.

4.4.4.1 Class Rectification Strategy and N_a

To discuss the effect of CRS and N_a, we conduct a series of experiments to learn the performance of CRS when N_a traverses in [50, 100, 200, 500, 1000].

The the real-time performance of different settings is shown in Fig. 4.8. We can see that all settings adopting CRS have quite similar performance. Although Baseline has a higher Precision, solutions with CRS are much better in terms of Accuracy, Recall and F1 Score, regardless the value of N_a. Therefore, CRS alone is useful in improving the performance of concept drift learning.

Table 4.3 shows the overall performance comparison. When adopting CRS, N_a=200 achieves the best Accuracy, Recall, F1 Score and the overall G-mean. All other N_a settings achieve over 2% improvement in G-mean than Baseline.

Table 4.3: Overall performance comparison of CRS with different N_a

N_a	Accuracy	Precision	Recall	F1 Score	G-mean	ΔG-mean
Baseline	89.56±0.22%	**84.02±0.24%**	74.52±0.90%	78.89±0.51%	81.56±0.44%	0
50	90.19±0.05%	83.06±0.07%	79.11±0.18%	80.99±0.11%	83.23±0.10%	2.12%
100	90.18±0.02%	83.01±0.10%	79.18±0.15%	80.99±0.05%	83.24±0.04%	2.13%
200	**90.21±0.03%**	82.96±0.05%	**79.40±0.15%**	**81.08±0.08%**	**83.31±0.07%**	**2.21%**
500	90.15±0.03%	82.81±0.08%	79.32±0.13%	80.97±0.06%	83.21±0.05%	2.09%
1000	90.12±0.02%	82.68±0.06%	79.36±0.11%	80.91±0.05%	83.17±0.04%	2.04%

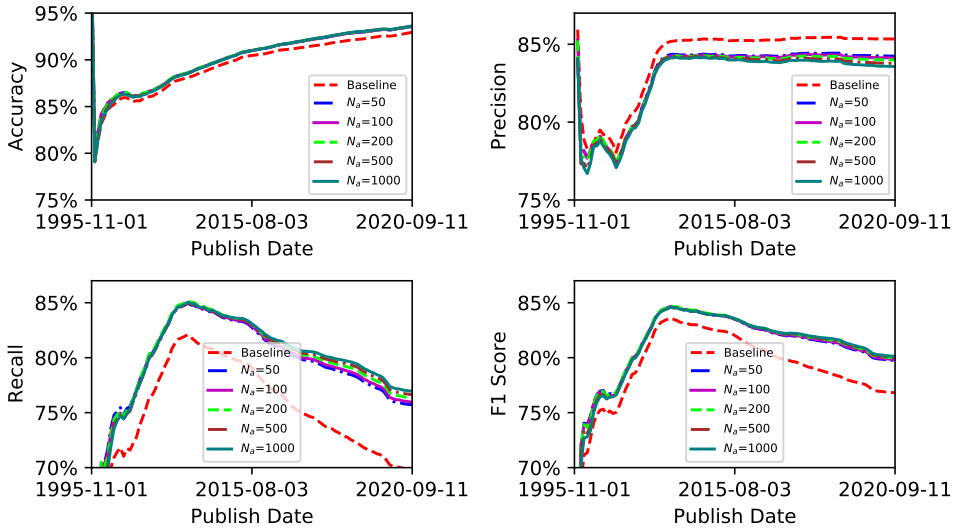


Fig. 4.8. Real-time performance comparison of CRS with different N_a

Fig. 4.9 shows the G-mean and ΔG-mean of CRS with different N_a. Obviously, the best performance is achieved when N_a=200, where G-mean is 83.31±0.07% and ΔG-mean is 2.21%.

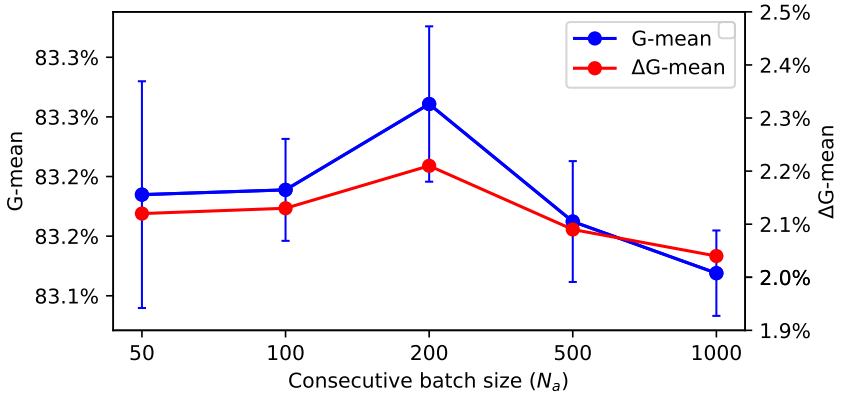


Fig. 4.9. Performance improvement comparison of CRS with N_a

4.4.4.2 Balanced Window Strategy and N_b

To discuss the effect of BWS and N_b, α is fixed to 1. Experiments are designed to compare the performance of BWS when N_b traverses in [50, 100, 200, 500, 1000].

Fig. 4.10 shows the real-time performance comparison of BWS with different selections of N_b. Baseline wins the best Accuracy and Precision, but gets the worst Recall. As for the F1 Score, Baseline is also among the best. Different settings distinguished each other at the beginning, and then the gaps were gradually narrowing down with the two classes became much more balanced. However, in recent years, the gaps began to widen due to the severe class imbalance.

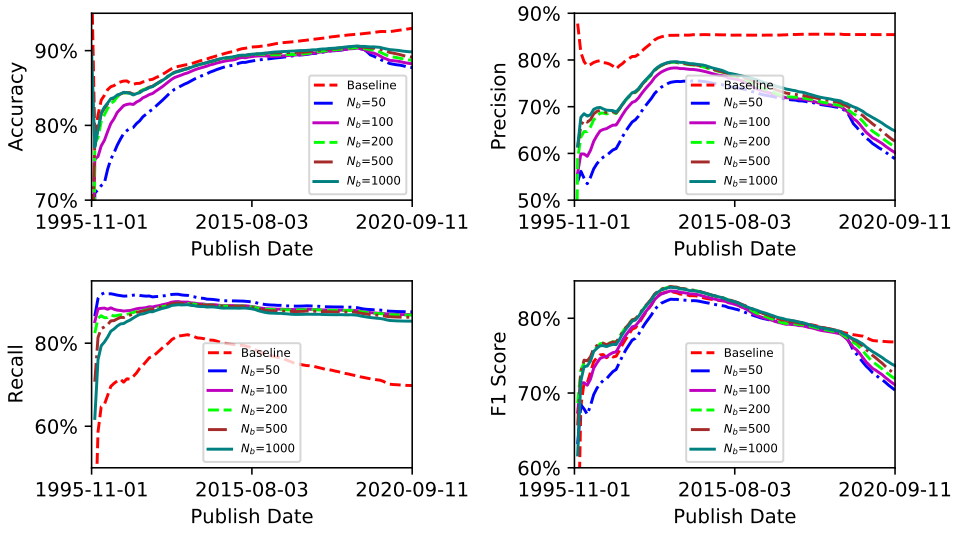


Fig. 4.10. Real-time performance comparison of BWS with different N_b ($\alpha = 1$)

Table 4.4 shows the overall performance comparison. N_b=500 achieves the best F1 Score at 79.33±0.09% and the overall G-mean at 81.64±0.08%. Baseline performs the best on Accuracy and Precision, while N_b=50 achieves the best Recall. With respect to ΔG-mean, only N_b=500 and N_b=1000 have a weak advantage over Baseline. Considering their extraordinary performance on Recall, the reason should be the over-fitting to minority class. BWS will keep the minority class samples in the balanced window more than one time slice, therefore they will be used to update the classifier more than once. Without a time decay weight, it is very likely to result in over-fitting.

Table 4.4: Overall performance comparison of BWS with different N_b ($\alpha = 1$)

N_b	Accuracy	Precision	Recall	F1 Score	G-mean	ΔG-mean
Baseline	**89.58±0.11%**	**84.17±0.13%**	74.32±0.42%	78.83±0.24%	81.52±0.21%	0
50	86.00±0.80%	68.44±1.76%	**89.73±1.55%**	77.50±0.47%	79.97±0.47%	-1.90%
100	87.30±0.12%	70.70±0.23%	88.32±0.25%	78.48±0.07%	80.87±0.06%	-0.80%
200	87.89±0.07%	72.02±0.16%	87.97±0.08%	79.16±0.10%	81.48±0.09%	-0.05%
500	88.15±0.11%	72.64±0.26%	87.44±0.23%	**79.33±0.09%**	**81.64±0.08%**	**0.14%**
1000	88.26±0.14%	73.21±0.27%	86.51±0.13%	79.29±0.13%	81.59±0.13%	0.09%

Fig. 4.11 shows the G-mean and ΔG-mean of BWS with different N_b. The best performance is achieved when N_b=500, which is only 0.14% better than Baseline. Therefore, it is vital to set an appropriate time decay factor to weaken the influence of old data when using BWS.

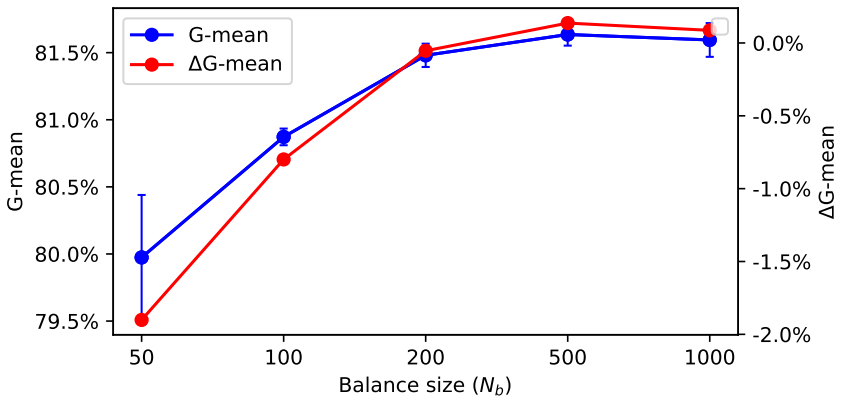


Fig. 4.11. Performance improvement comparison of BWS with different N_b ($\alpha = 1$)

4.4.4.3 Balanced Window Strategy and α

Similarly, to discuss the effect of BWS and α, N_b is fixed to 500. Experiments are designed to compare the performance of BWS when α traverses in [0.3, 0.5, 0.7, 0.8, 0.9, 1].

Fig. 4.12 shows the real-time performance comparison of BWS with different selections of α. In terms of Accuracy, all settings except for α=1 achieves equivalent results. For Precision and Recall, the results are quite different. Generally speaking, settings achieving good precision usually have poor performance on Recall and vice versa. Performance on F1 Score is highly related to the class proportion. For example, when class 1 and class 0 are almost half to half, all settings have good results, while when the two classes are highly imbalanced, different settings can make a big difference.

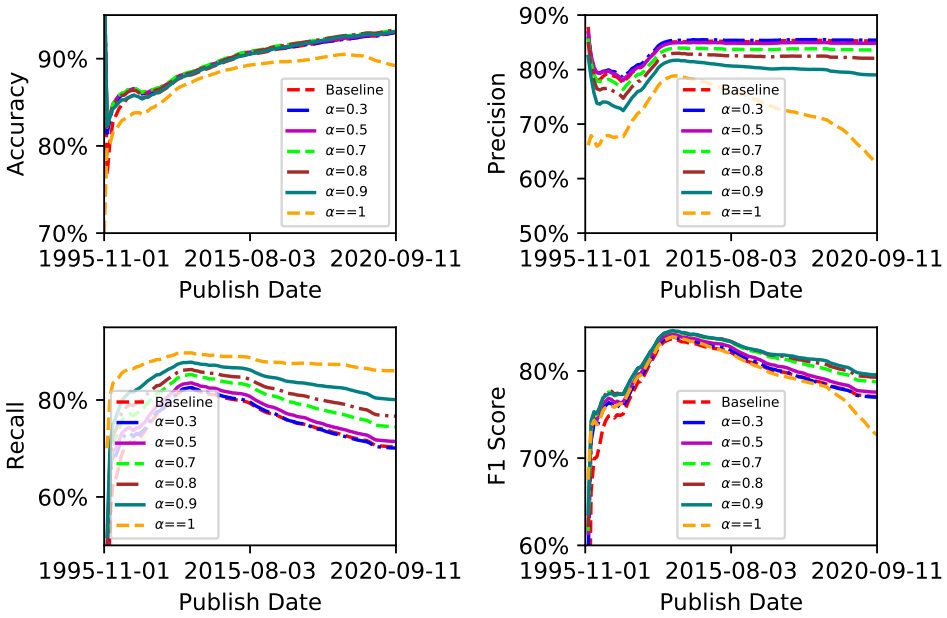


Fig. 4.12. Real-time performance comparison of BWS with different α (N_b=500)

Table 4.5 shows the overall performance comparison. We can see that the best overall G-mean is achieved by α=0.9, which is 2.19% better than Baseline. Followed by α=0.8 and 0.7. The worst is α=1, which means that BWS only keeps a balanced window but not weakens sample weights overtime. As we analysed before, in this case, the performance can decrease because of over-fitting to the minority class. α=0.3 only gets limited improvement due to the too fast time decay of minority samples in the balanced window.

Table 4.5: Overall performance comparison of BWS with different α (N_b=500)

α	Accuracy	Precision	Recall	F1 Score	G-mean	ΔG-mean
Baseline	89.54±0.12%	83.91±0.50%	74.46±0.35%	78.74±0.15%	81.47±0.14%	0
0.3	89.82±0.05%	**84.23±0.09%**	75.37±0.20%	79.50±0.14%	82.06±0.11%	0.72%
0.5	89.95±0.02%	83.76±0.08%	76.64±0.08%	79.99±0.03%	82.44±0.02%	1.19%
0.7	**90.09±0.03%**	82.57±0.06%	79.14±0.05%	80.77±0.05%	83.04±0.04%	1.93%
0.8	90.01±0.03%	81.28±0.07%	80.82±0.09%	81.00±0.06%	83.19±0.06%	2.11%
0.9	89.78±0.03%	79.15±0.06%	83.32±0.10%	81.14±0.05%	**83.25±0.05%**	**2.19%**
1	87.98±0.23%	72.28±0.61%	**87.45±0.42%**	79.11±0.19%	81.44±0.18%	-0.03%

Fig. 4.13 shows the G-mean and ΔG-mean of BWS with different values of α. The best performance is achieved with α=0.9, which is 2.19% better than the Baseline.

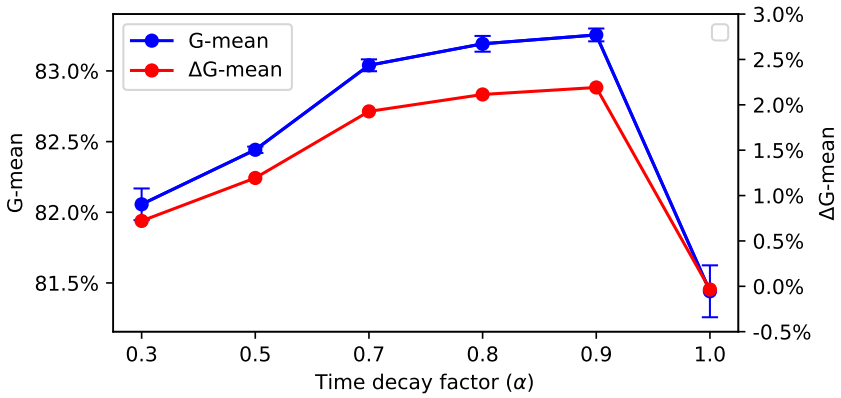


Fig. 4.13. Performance improvement comparison of BWS with different α (N_b=500)

According to Tables 4.3, 4.4 and 4.5, the best N_a is 200, the best N_b is 500 and the best α is 0.9. However, these results are obtained by ablation studies, which means only one factor is applied to the consecutive batch learning framework each time. When applying CRS and BWS at the same time, the best choice for these three hyperparameters should be tested simultaneously.

4.4.5 Discussion Remarks

The effectiveness of RDCAL is validated by three series of experiments. First, the experiments in Section 4.4.2 demonstrate that RDCAL can improve the performance of four different classifiers in a consecutive batch learning scenario by more than 3%. Then, the experiments in Section 4.4.3 show that RDCAL performs the best among six concept drift adaptation algorithms. Finally, we discuss the influence of hyperparameters on RDCAL and demonstrate the effectiveness of CRS and BWS separately. Therefore, RDCAL is indeed classifier-agnostic and a state-of-the-art concept drift learning algorithm in dealing with the online exploitability prediction problem.

4.5 Conclusion

This chapter proposes a novel real-time dynamic concept adaptive learning algorithm under a consecutive batch learning setting. Specifically, RDCAL consists of two strategies, namely, the Class Rectification Strategy (CRS) and the Balanced Window Strategy (BWS). CRS is designed to handle the actual drift in sample labels and BWS is a strategy to deal with the dynamic class imbalance problem.

Comprehensive experiments show that RDCAL can significantly improve the performance of a wide range of classifiers, including neural networks, SVM, decision trees and logistic regression in exploitability prediction. Furthermore, RDCAL achieves state-of-the-art performance on a real-world dataset containing 140,758 vulnerabilities, compared with the other five adaptive data stream learning algorithms. However, it is worth noting that the effectiveness of concept drift adaptation algorithms, including RDCAL, strongly depends on the characteristics of the data.

Chapter 5

Vulnerability Exploitation Time Prediction

ExBERT discussed in Chapter 3, and RDCAL discussed in Chapter 4 are solutions for offline and online vulnerability prediction, respectively. Both belong to binary classification problems, predicting whether a vulnerability will be exploited or not. To suit real-world data stream applications and provide more fine-grained results for vulnerability evaluation, this chapter specifically investigates the exploitation time prediction problem and formulates it as an online imbalanced multiclass classification problem. Specifically, the consecutive batch learning framework used in Chapter 4 has been generalized into a multiclass online learning setting, and an Adaptive Sliding Window Weighted Learning (ASWWL) algorithm within the generalized consecutive batch learning framework has been proposed to tackle the dynamic multiclass imbalance problem existing in many industrial applications, including exploitation time prediction in this chapter. Furthermore, a Sliding Window Imbalance Factor (SWIF) is proposed in this chapter as the index of measuring the dynamic imbalanced status of each class.

The rest of this chapter is organised as follows. Section 5.1 briefly introduces the background and contributions of this chapter, followed by related works on learning strategy and multiclass imbalanced learning in Section 5.2. Then a detailed description of SWIF, consecutive batch learning and the corresponding

framework for exploitation time prediction, as well as the ASWWL algorithm, are presented in Section 5.3. The datasets, evaluation metrics and experiment results are presented inSection 5.4. Finally, Section 5.5 concludes this chapter with a discussion on limitations.

5.1 Introduction

Exploitation time is an essential factor for vulnerability assessment in cybersecurity management. A substantial number of previous works have tried to develop more accurate and reliable exploitability prediction models. For example, the work in [11] proposed a neural language model-based approach, named DarkEmbed, to predict whether vulnerabilities will be exploited or not. The work in [50] investigated the effectiveness of different features, including common words from vulnerability descriptions, external references and vendor products, CVSS scores and categorical attributes, and Common Weakness Enumeration (CWE) numbers in predicting the exploitability of vulnerabilities. Jacobs, Romanosky et al. proposed an open, data-driven framework, called the Exploit Prediction Scoring System (EPSS), to estimate the probability of a vulnerability being exploited within the first twelve months after disclosure [51]. The work in [7] employed transfer learning to extract paragraph-level embeddings from vulnerabilities and built a high-performance exploitation predictive model.

However, to the best of our knowledge, no previous research has specifically investigated the exploitation time prediction problem, except for a few papers on Zero-Day Exploit detection [102], which is also a binary classification problem. Therefore, these works are incapable of predicting exploitation time in finer granularity. Furthermore, previous studies handled historical data in a batch learning manner, ignoring the temporal characteristics and possible implicit concept drifts in the data. Previous studies also overlooked another challenging problem, dynamic imbalanced data, by adopting batch learning.

To suit the real-world situation and provide more fine-grained results for vulnerability evaluation, this chapter treats the exploitation time prediction problem as a multiclass imbalanced online learning problem. An integrated consecutive

batch learning framework is proposed to predict the exploitation time in finer granularity. Instead of dealing with data using batch learning, consecutive batch learning is adopted as the online learning strategy to enable the classifier to capture the real-time temporal concepts. Furthermore, an ASWWL algorithm is designed to tackle the dynamic class imbalance problem caused by online learning to achieve better performance.

To summarize, the main contributions are as follows:

(1) An integrated consecutive batch learning framework is proposed to predict the exploitation time of vulnerabilities, which is a generalization of the online learning framework used in Chapter 3. The proposed framework handles historical data in a consecutive batch learning manner, enabling the classifier to capture the temporal features and possible concept drifts caused by the evolving system and user behaviours. Compared with previous studies, this chapter predicts the exploitation time as a multiclass classification problem instead of a binary classification problem to provide more fine-grained results for decision-makers.

(2) A general learning algorithm, ASWWL, is proposed to deal with the dynamic multiclass imbalance problem, which is pervasive in real-world applications. The effectiveness of ASWWL in improving the performance of the minority classes has been demonstrated by a series of experiments conducted on a real-world dataset.

(3) The proposed integrated consecutive batch learning framework with the ASWWL algorithm achieves the most robust and state-of-the-art performance on the multiclass exploitation time prediction task, compared with the other five consecutive batch learning algorithms on the same dataset.

5.2 Related Work

5.2.1 Learning Strategy

According to the availability of data and the way of feeding data to train and test classifiers, there are two learning algorithms, i.e. batch learning and stream learning. Batch learning is used for applications in which all data is available.

In this case, classifiers are usually evaluated in a hold-out manner, which means that data is randomly divided into a training set and a test set. Classifiers are evaluated in a separated test set. The data imbalanced status is determined and static in batch learning because all data is available.

By contrast, for stream learning, data appears over time and classifiers are incrementally trained when new data is available. In this case, classifiers are often evaluated on the newly arrived data before using it to train the classifiers[92]. Stream learning inherently faces more challenges than batch learning. For example, concept drifts and dynamic class imbalance are two major problems that need to be addressed.

Stream learning has been studied for a long time [103], among them ensemble learning is the most popular strategy to boost performance. Back in 2001, a Concept-adapting Very Fast Decision Tree (CVFDT) was proposed to learn time-changing concepts by keeping a decision tree consistent with a window of examples, based on a batch learning version of Very Fast Decision Tree (VFDT) [101]. In 2007, Kolter, J. Z. and Maloof, A. M. proposed a Dynamic Weighted Majority (DWM) ensemble algorithm, which dynamically trained multiple learners and weighted them according to their performance to get a global ensemble result [96]. The authors concluded that DWM outperformed algorithms that employ incrementally learning with a single learner, train all previously observed examples, or employ an unweighted, fixed-size ensemble of learners[96]. Paper [91] proposed a Hoeffding Tree Adaptive (HTA) algorithm to adaptively learn concept drifts over time, by maintaining a Hoeffding Window Tree and a Hoeffding Adaptive Tree at the same time in 2009. To handle the concept drift in nonstationary environments (NSEs), paper [97] introduced an ensemble Learn^{++}.NSE (LPPNSE) algorithm in 2011, by employing a dynamically weighted majority voting mechanism. Kosina, P. and Gama, J developed a Very Fast Decision Rules Classifier (VFDRC)[94] and the adaptive extension (AVFDRC) to detect and adapt changes in data distributions in 2015. Paper [93] proposed a Self Adjusting Memory K Nearest Neighbor (SAMKNN) model for heterogeneous concept drift, inspired by biological memory models in 2016. An open-source framework, named Scikit-Multiflow [92] was released in 2018, providing the implementation of multiple steam data generators, stream learning algorithms, evaluators and transformers

[104]. In 2019, Anwar, M. M. et al. proposed an index-based algorithm to discover and track the evolution of user groups drove by time-sensitive activities in social networks [105].

5.2.2 Multiclass Imbalanced Learning

This chapter focuses on the task of multiclass imbalanced online learning problem. Let $x^{(t)} \in \mathbb{R}^n$ is a vector in n dimensional feature space $\mathcal{X}$ observed at time step t. $y^{(t)}$ is the corresponding label and $y^{(t)} \in \{c_{[1]}, c_{[2]}, \cdots, c_{[n_c]}\} (n_c \geqslant 3)$, where $c_{[1]}$, $c_{[2]}$, $\cdots$, $c_{[n_c]}$ are the n_c class labels that have appeared so far. We call $x^{(t)}$ an instance of a data stream at time step t and a pair $(x^{(t)}, y^{(t)})$ a labelled instance.

In fact, most data streams suffer from dynamic class imbalance, which severely damages the performance of classifiers. To relieve the negative impact, the most important thing is to identify the class imbalance status and describe the severity of different multiclass imbalance levels.

Paper [106] defined an Imbalance Factor (IF) $w^{(t)}=\{w_{[1]}^{(t)}, w_{[2]}^{(t)}, \cdots, w_{[N]}^{(t)}\}$ to indicate the overall class proportions in time step t, which has the capability of keeping track of the real time imbalanced statuses of all classes. Specifically, the data proportion of class $c_{[k]}$, denoted as $w_{[k]}^{(t)}$, is updated by (5.1).

$$w_{[k]}^{(t)} = \frac{(t-1) * w_{[k]}^{(t-1)} + [(x^{(t)}, c_{[k]})]}{t}, (k = 1, 2, \cdots, n_c \text{ and } t \geq 1) \tag{5.1}$$

where $w_{[k]}^{(0)} = 0$, $[(x^{(t)}, c_{[k]})]=1$ if the true label of $x^{(t)}$ equals to $c_{[k]}$, otherwise 0. Equation (5.1) actually calculates the global Imbalance Factor over all previously encountered examples until time step t. As t increases, $w_{[k]}^{(t)}$ would be less sensitive to the latest imbalanced status. However, the imbalanced status in exploitation time prediction scenario, as well as many other data streams, often changes dynamically and irregularly. Therefore, Equation (5.1) is insensitive to reflect the new imbalanced status, due to the great influence of old data.

An improvement of equation (1) in weakening the influence of older data is introduced in paper [107]. As shown in formula (5.2), $w_{[k]}^{(t)}$ is updated along with a time decay factor θ.

$$w_{[k]}^{(t)} = \theta w_{[k]}^{(t-1)} + (1-\theta)[(x^{(t)}, c_{[k]})], (k = 1, 2, \cdots, n_c \text{ and } t \geq 1) \tag{5.2}$$

Time decay Imbalance Factor calculated by (5.2) assumes that all classes decay at the same rate θ and finally the first item in equation (5.2) will gradually converge to 0 over time. However, this assumption can hardly hold true in real-world applications, especially for data steam situations like the exploitation time prediction problem.

The Imbalance Factor is the basis of further processing of the multiclass imbalance problem. For example, after obtaining $w_{[k]}^{(t)}$, a Multiclass Oversampling (MO) method and a Multiclass Undersampling (MU) method were used to handle the multiclass imbalanced online learning problem in [107]. Both MO and MU resample data by training the current labelled instance ($x^{(t)}$, $y^{(t)}$) K times when updating classifier from $f^{(t)}$ to $f^{(t+1)}$, where K obeys a Poisson Distribution, namely, $K \sim Poisson(\lambda)$ and λ is defined by (5.3).

$$\lambda = \begin{cases} w_{\max}^{(t)} / w_{[k]}^{(t)}, (k = 1, 2, \cdots, n_c \text{ and } t \geq 1), \text{ for MO method} \\ w_{\min}^{(t)} / w_{[k]}^{(t)}, (k = 1, 2, \cdots, n_c \text{ and } t \geq 1), \text{ for MU method} \end{cases} \tag{5.3}$$

where $w_{\min}^{(t)}$=$\min_{k=1}^{N} w_{[k]}^{(t)}$ is the minimum class proportion at time step t and $w_{\max}^{(t)}$=$\max_{k=1}^{N} w_{[k]}^{(t)}$ is the maximum class proportion. Obviously, $\lambda \geq 1$ for the MO method. Therefore, according to the properties of Poisson Distribution, the MO algorithm can increase the training epochs for the minority samples, equivalent to oversample instances from the minority class. Similarly, $\lambda \leq 1$ for the MU method and the chance of learning majority classes will be reduced.

Based on those previous works, a new Sliding Window Imbalance Factor (SWIF) is proposed to indicate the real-time multiclass imbalance status in Section 5.3.1. Furthermore, the Adaptive Sliding Window Weighted Learning algorithm is proposed to boost the performance of multiclass-imbalanced online learning in section 5.3.4.

5.3 Methodology

5.3.1 Sliding Window Imbalance Factor

To overcome the drawbacks of IF calculated by (5.1) and (5.2), Sliding Window Imbalance Factor (SWIF) is designed to dynamically reflect the latest multiclass proportions in a data stream.

The SWIF of each class $w^{(t)}$ =$[w_{[1]}^{(t)}, w_{[2]}^{(t)}, \cdots, w_{[c_n]}^{(t)}]$ is designed on the basis of Equation (5.1) and (5.2), where $w_{[k]}^{(t)}$ (k=$\{1, 2, \cdots, n_c\}$) represents the imbalance factor of class $c_{[k]}$. Instead of calculating the overall imbalance factor so far like Equation (5.1) or setting a fixed time decay factor θ for all classes and time steps like Equation (5.2), SWIF calculates the imbalance factor of class $c_{[k]}$ with Equation (5.4).

$$w_{[k]}^{(t)} = \begin{cases} \frac{(t-1)*w_{[k]}^{(t-1)}+[(x^{(t)}, c_{[k]})]}{t}, \ (k = 1, 2, \cdots, n_c \text{ and } 1 \leq t < z) \\ \frac{z*w_{[k]}^{(t-1)}-[(x^{(t-z)}, \ c_{[k]})]+[(x^{(t)}, \ c_{[k]})]}{z}, \ (k = 1, 2, \cdots, n_c \text{ and } t \geqslant z) \end{cases} \tag{5.4}$$

where z ($z \geq n_c$) is the number of samples within a sliding window, which can be fixed during the whole online learning process or it can also be adaptively adjusted according to some hyper-parameter optimisation strategies, such as Grid Search, Random Search and Bayesian Optimisation. It is worth noting that, when $1 \leq t < z$, SWIF is identical with Equation (5.1).

5.3.2 Consecutive Batch Learning

Consecutive batch learning is a special case of data stream learning, where stream data is handled in consecutive batches. Let $S = \{D^{(0)}, D^{(1)}, \cdots, D^{(k)}, \cdots\}$ be a chronological data stream, where $D^{(k)}$=$\{X^{(k)}, Y^{(k)}\}$ ($k \geq 1$) is the k-th batch of observed samples or instances. $X^{(k)}$=$\{x_1^{(k)}, x_2^{(k)}, \cdots, x_{n_b}^{(k)}\}$, where n_b is the total number of samples in the k-th data batch and $x_i^{(k)} \in \mathbb{R}^n$ (i= 1, 2, $\cdots$, n_b) is the feature vector of the i-th sample in the k-th batch. $Y^{(k)}$=$\{y_1^{(k)}, y_2^{(k)}, \cdots, y_{n_b}^{(k)}\}$ is the corresponding labels for batch k.

Let $f^{(1)}(\cdot), \cdots, f^{(k)}(\cdot), \cdots$ denote the consecutive classifier statuses which are incrementally trained from the consecutive data batches. For consecutive batch learning, at each time step k ($k \geq 1$), $f^{(k)}(\cdot)$ will be used to predict the labels of arrived data $X^{(k)}$ for the usage of downstream applications, when the corresponding true labels $Y^{(k)}$ are unavailable. The predicted labels $\hat{Y}^{(k)}$ is calculated as (5.5).

$$\hat{Y}^{(k)} = f^{(k)}(X^{(k)}), (k \geq 1). \tag{5.5}$$

After the ground truth $Y^{(k)}$ for the k-th data batch is available, the labelled data $D^{(k)}$=$\{X^{(k)}, Y^{(k)}\}$ will be used to fit $f^{(k)}(\cdot)$ to get the next classifier status $f^{(k+1)}(\cdot)$.

5.3.3 An Integrated Framework for Exploitation Time Prediction

The work flow of the proposed integrated consecutive batch learning framework for exploitation time prediction problem is illustrated in Fig. 5.1. Like any other consecutive batch learning, there are two major stages for each data batch, namely, prediction and classifier update. The notations in Fig. 5.1 are consistent with the definitions in Section 5.3.2. We will first explain all of the components displayed in Fig. 5.1 below.

Descriptions are a collection of vulnerability descriptions in the k-th batch. Specifically, a description is a brief paragraph containing information like the affected products, the required privilege, the possible attack paths of a vulnerability. Descriptions are usually disclosed with vulnerabilities by some publicly available database, such as the National Vulnerability Database (NVD) [108] and the Common Vulnerabilities and Exposures database(CVE) [34]. Descriptions should be further treated before being fed to a classifier.

Pre-trained BERT is a released Bidirectional Encoder Representations from Transformers (BERT) model [64]. BERT has proven to be an excellent Natural Language Processing (NLP) model which can extract semantic features from both words and sentences. In the proposed framework, we adopt a pre-trained BERT

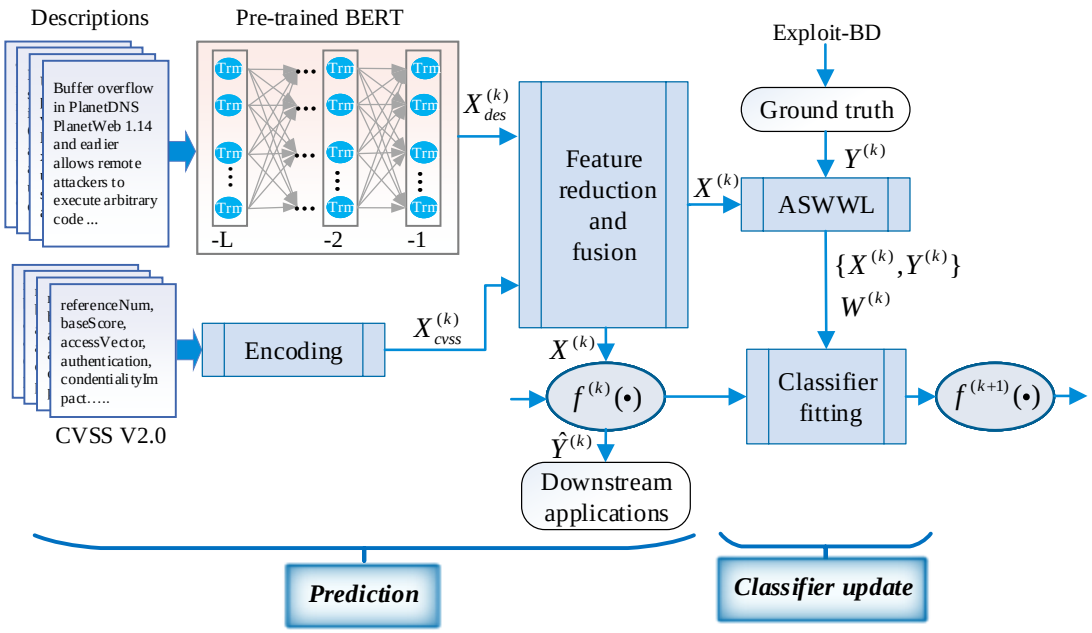


Fig. 5.1. Workflow of the proposed integrated framework ($k \geq 1$).

model to extract sentence-level semantic feature $X_{des}^{(k)}$ from Descriptions instead of training a BERT model from scratch, considering the limited scale of vulnerability description corpus.

CVSS V2.0 represents a set of attributes selected from the 2.0 version of CVSS metrics. Some of the attributes are not numerical. For example, the value range of the attribute access complexity is {High, Medium, Low}.

Encoding is the method to encode all CVSS V2.0 attributes to numerical features $X_{cvss}^{(k)}$.

Feature reduction and fusion involves a feature reduction process and a feature fusion process. In the proposed framework, we first separately reduce the dimension of $X_{des}^{(k)}$ and $X_{cvss}^{(k)}$ to an acceptable value by applying feature reduction algorithms. Then, fuse the results to form the feature $X^{(k)}$ of the k-th batch via concatenating, averaging, ranking or other fusion strategies.

Downstream applications are the possible industry applications relying on the predicted labels $\hat{Y}^{(k)}$, such as the cybersecurity management system, which will use the predicted labels to make decisions on patching the corresponding

vulnerability or not.

Exploit-DB [**38**] is an open-source exploit database, working as the data source of ground truth in our proposed framework.

Ground truth is the class labels corresponding to $X^{(k)}$, which can be encoded into $Y^{(k)}$, by one hot encoding, label encoding, custom binary encoding or other methods.

ASWWL is the proposed Adaptive Sliding Window Weighted Learning algorithm. Instead of weighting different learners, we weigh each sample based on real-time imbalanced status. The detailed description of ASWWL will be presented in Section 5.3.4.

Classifier fitting is the process of fitting the classifier $f^{(k)}(\cdot)$ to the k-th data batch $\{X^{(k)}, Y^{(k)}\}$ with weight $W^{(k)}$ based on a predefined cost function.

As shown in Algorithm 5.1, the inputs of the proposed framework include data sources like descriptions, CVSS V2.0 and Exploit-DB, a pre-trained BERT model, an initialized classifier $f^{(1)}(\cdot)$ and the size of data batch n_b. As time passes, the framework will generate a series of intermediate results, namely, the sequence of features extracted from descriptions $\mathcal{X}_{des}$= $\{X_{des}^{(1)}, \cdots, X_{des}^{(k)}, \cdots\}$, features extracted from descriptions CVSS 2.0 $\mathcal{X}_{cvss}$=$\{X_{cvss}^{(1)}, \cdots, X_{cvss}^{(k)}, \cdots\}$, fused features $\mathcal{X}$=$\{X^{(1)}, \cdots, X^{(k)}, \cdots\}$, true labels $\mathcal{Y}$=$\{Y^{(1)}, \cdots, Y^{(k)}, \cdots\}$ and classifier statuses $f^{(2)}(\cdot), \cdots, f^{(k+1)}(\cdot), \cdots$. The final valuable output for downstream applications is the sequence of predicted labels $\widehat{\mathcal{Y}}$=$\{\widehat{Y}^{(1)}, \cdots, \widehat{Y}^{(k)}, \cdots\}$.

Algorithm 5.1 An integrated framework for exploitation time prediction

Input: Descriptions; CVSS V2.0; Exploit-DB; Pre-trained BERT; $f^{(1)}(\cdot)$, n_b.

Output: $\widehat{\mathcal{Y}}$=$\{\widehat{Y}^{(1)}, \cdots, \widehat{Y}^{(k)}, \cdots\}$.

1: **for** the k-th batch ($k \geq 1$) **do**
2: Feed the k-th batch of Descriptions to a pre-trained BERT model to extract features $X_{des}^{(k)}$.
3: Encode the k-th batch of CVSS V2.0 attributes to numerous features $X_{cvss}^{(k)}$.
4: Apply feature reduction algorithm to $X_{des}^{(k)}$ and $X_{cvss}^{(k)}$ separately and fuse the results to get the fused feature $X^{(k)}$.
5: Predict the labels $\hat{Y}^{(k)}$ of $X^{(k)}$ for downstream applications by Equation (5.5).
6: From Exploit-DB data find out the ground truth and encode it to $Y^{(k)}$.
7: $D^{(k)}$=$\{X^{(k)}, Y^{(k)}\}$.
8: **if** ASWWL == True **then**
9: Run Algorithm 5.2 and get the returned $W^{(k)}$
10: **else**
11: $W^{(k)}$=None
12: **end if**
13: Update $f^{(k)}$ to $f^{(k+1)}$ based on $D^{(k)}$ and $W^{(k)}$
14: **end for**

For the k-th batch ($k \geq 1$), steps 2-5 specify the implementation of the prediction stage shown in Fig. 5.1, while steps 6-13 are the implementation of the classifier update stage. ASWWL is an optional strategy for improving performance, as shown in steps 8-12. The framework without ASWWL is a baseline version and the framework with ASWWL is an improved version, providing adaptive weights for each single sample in $D^{(k)}$.

5.3.4 Adaptive Sliding Window Weighted Learning

Adaptive Sliding Window Weighted Learning is designed for handling dynamic multiclass imbalance problem existing in data stream. Sliding window is a widely-used strategy in multiple areas [91, 101, 109], such as signal processing and ensemble learning. We employ this idea to adaptively measure the current class

imbalanced status and then calculate weights $W^{(k)}$=$\{w_1^{(k)}, w_2^{(k)}, \cdots, w_{n_b}^{(k)}\}$ for the k-th data batch.

Specifically, let $\mathbf{Y}_e^{(k)}$ denote the list of all existing sample labels in chronological order, and z ($z \geq n_c$) is the sliding window size, where n_c is the number of unique labels in the data stream. The first step of ASWWL is to find out the labels of the latest z samples, denoted as Y_{sw}. The second step is counting the number of samples N_c (c=1, 2, $\cdots$, n_c) belonging to each class within Y_{sw}. Finally, calculate the weight $w_i^{(k)}$ of the i-th sample in the k-th batch by formula (5.6).

$$w_i^{(k)} = \frac{N_{max}}{N}, (i = 1, 2, \cdots, n_b). \tag{5.6}$$

where $N_{max} = \max_{c=1}^{n_c} N_c$ and N is the number of samples belonging to the same class of sample $\{x_i^{(k)}, y_i^{(k)}\}$. Obviously, $w_i^{(k)} \geq 1$.

The implementation of ASWWL is illustrated in Algorithm 5.2. The input includes all existing true labels $\mathcal{Y}_e^{(k)}$=$\{Y^{(1)}, \cdots, Y^{(k)}\}$, the current batch number k, batch size n_b and a pre-set sliding window size z. The output is the weight for batch k. Step 1 initializes the weight for each sample in batch k to 1. Step 2 expands all existing label collections in $\mathcal{Y}_e^{(k)}$ into a list $\mathbf{Y}_e^{(k)}$. Steps 3-19 calculate the weight of each sample using a loop. Among them, steps 4-9 is to find out the labels of the latest samples within the sliding window Y_{sw}; steps 10-16 count the number of samples of each class in Y_{sw}; and steps 17-18 calculates the weight for the i-th sample $w_i^{(k)}$.

Algorithm 5.2 Adaptive Sliding Window Weighted Learning

Input: $\mathcal{Y}_e^{(k)}$=$\{Y^{(1)}, \cdots, Y^{(k)}\}$, k, n_b, z.

Output: $W^{(k)}$.

$W^{(k)}$=ones(n_b,1)
$\mathbf{Y}_e^{(k)}$= [y for y in $\mathcal{Y}_e^{(k)}$]
for i in range(1,n_b) **do**
    idx=$(k-1) * n_b + i$
    if idx-$z \geq$0 **then**
        Y_{sw}=$\mathbf{Y}_e^{(k)}$[inx-z:idx]
    else
        Y_{sw}=$\mathbf{Y}_e^{(k)}$[0:idx]
    end if
    C = unique(Y_{sw}), n_c=len(C).
    for c in range(n_c) **do**
        Count N_c (the number of samples belonging to $C[c]$ in Y_{sw}).
        if $Y_i^{(k)}$==$C[c]$ **then**
            $N = N_c$
        end if
    end for
    $N_{max} = \max_{c=1}^{n_c} N_c$
    $w_i^{(k)}$=N_{max}/N
end for
return $W^{(k)} = \{w_1^{(k)}, w_2^{(k)}, \cdots, w_{n_b}^{(k)}\}$

5.4 Experiments

In this section, we report the experiment results of exploitation time prediction. Section 5.4.1 presents the dataset and experimental setting of this work, followed by the introduction of evaluation metrics in Section 5.4.3. Section 5.4.4 gives the performance comparison of the baseline version and the ASWWL version of the proposed framework, adopting three classifiers, namely, Neural Networks (NN), Hoeffding Tree (HT) and Naive Bayes (NB). Section 5.4.5 provides the

performance comparison of our method with five other data stream algorithms.

5.4.1 Dataset and Experimental Setting

The vulnerability data including descriptions and CVSS 2.0 attributes is collected from NVD and the exploit data comes from Exploit-DB. In this chapter, when a proof-of-concept exploit exists in Exploit-DB, the corresponding vulnerability is identified as exploited. Vulnerabilities and exploits are integrated by CVE-IDs. The dataset used in this work contains 23,413 exploited vulnerabilities collected between 1990 to 2020.

A vulnerability's exploitation time is the time interval between the publication date of vulnerability and the earliest publication date of its corresponding exploits in days. When a vulnerability is exploited after being published, its exploitation time is positive and the class label is marked as Pos. Similarly, when a vulnerability is exploited before being published, the exploitation time is negative and the class label is marked as Neg. When the vulnerability and its earliest exploit are published on the same day, the exploitation time equals to zero, and the corresponding class label is ZeroDay.

The total number of vulnerabilities in the collected dataset for classes Pos, Neg and ZeroDay are 3,971 (16.96%), 18,127 (77.42%) and 1,315 (5.62%) respectively. Fig. 5.2 shows the exploitation time distribution of all collected 23,413 exploitable vulnerabilities. All vulnerabilities shown in Fig. 5.2 will be used as the input data stream to the proposed online learning framework.

To have an intuition on the multiclass imbalanced dynamics of the collected dataset, we draw the overall imbalance factor calculated by (5.1) and the SWIF calculated by Equation (5.4) in Fig. 5.3. Although subplots (a) and (b) share the same trend, (a) is too smooth to reflect the latest imbalanced dynamics. For example, the subplot (b) of Fig. 5.3 shows that in recent years the ratio of Pos vulnerabilities increased dramatically and sometimes the proportion is even over 0.6. However, due to the great influence of history data, the overall imbalance factor shown in subplot (a) can only see a very smooth increase.

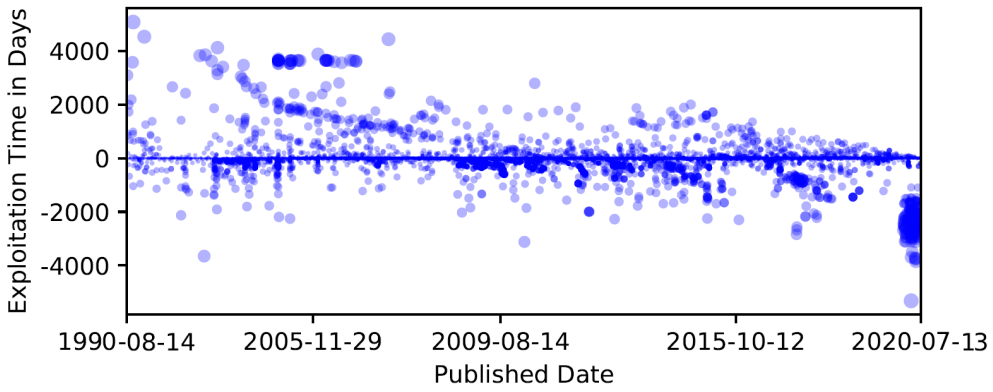


Fig. 5.2. Exploiltation time distributions of vulnerabilities from 1990 to 2020

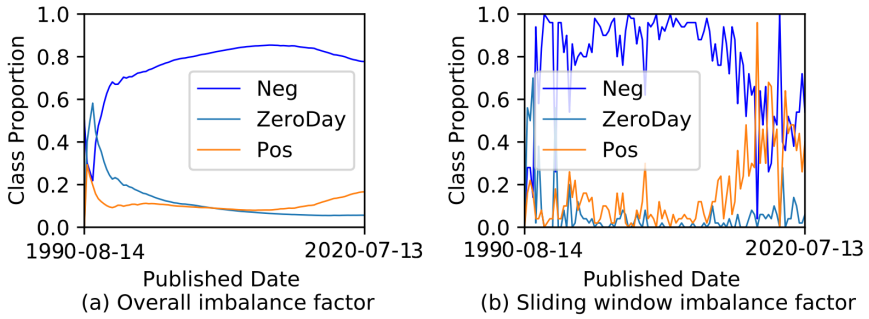


Fig. 5.3. Dynamical multiclass imbalanced statuses over time: (a) is calculated by Equation (5.1); (b) is calculated by Equation (5.4) when z=50.

5.4.2 Feature Extraction, Reduction and Fusion

Recently developed machine learning and deep learning algorithms are powerful to extract features from unstructured raw data [15, 110, 111]. Previous studies usually extract features from CVSS metrics or vulnerability descriptions for vulnerability assessment or exploitability prediction [11, 55, 111]. In this chapter, we extract features from both sides to maximise the representativeness of extracted features.

Specifically, the extracted feature attributes from CVSS metrics and their value types are listed in Table 5.1. The adopted feature attributes all come from CVSS version V2.0, instead of version V3.0, because V2.0 is available for almost all published vulnerabilities, while V3.0 is only available for vulnerabilities pub-

lished after 2015 in most cases. These attributes are encoded by one hot encoding method. Furthermore, the data batch size n_b is set to 100 for all experiments in this work. Therefore, after encoding, the features extracted from CVSS V2.0 $X_{cvss}^{(k)} \in \mathbb{R}^{30*100}$.

Table 5.1: Selected CVSS V2.0 attributes and their value types

Value types	CVSS 2.0 attributes
Numerical	Number of References *, Base Score, Impact Score, Exploitability Score
Categorical	Access Vector, Access Complexity, Authentication, Confidentiality Impact, Integrity Impact, Availability Impact, Severity
Boolean	User Interaction Required, Obtain User Privilege, acInsufInfo, Obtain Other Privilege

* This attribute does not belong to CVSS V2.0 metrics. It is the total number of reference information of a vulnerability listed in NVD dataset.

Meanwhile, a pre-trained Bidirectional Encoder Representations from Transformers (BERT) [64] model is employed to extract sentence-level semantic features from vulnerability descriptions. The implementation and pre-trained BERT model are available at GitHub [66]. We use the token embedding of [CLS] at the last Transformer Layer of BERT model as the final feature representation of a vulnerability's description, which is a vector in a 768 dimensional feature space. Considering that the data batch size n_b is set to 100, $X_{des}^{(k)} \in \mathbb{R}^{768*100}$.

The feature reduction method adopted in this work is Principal Component Analysis (PCA) and the feature fusion method is concatenation. We extract 10 features from $X_{des}^{(k)}$ and $X_{cvss}^{(k)}$ respectively and then concatenate them to get a $X^{(k)} \in \mathbb{R}^{20*100}$. The data batch size n_b is set to 100 for all experiments in this work.

5.4.3 Evaluation Metrics

For classification problems, Accuracy, the number of correct predictions divided by all predictions made, is the most important evaluation metrics. However, for

tasks with class imbalance, Accuracy alone can be misleading. Therefore, Precision and Recall are used as a supplement to show the exactness and completeness of a classifier. Furthermore, F1 Score is usually considered as the final measure to decide which classifier is better, because it conveys balances between Precision and Recall.

As exploitation time prediction is an imbalanced multiclass classification problem, we adopt Accuracy, Precision, Recall and F1 score to measure the performance of a classifier on each single class. Furthermore, we employ two average strategies to show the overall performance on all classes, namely, Macro and Micro.

Macro is an average strategy that calculates all metrics for each class separately, and then find their unweighted mean.

Micro is an average strategy that calculates all metrics globally on all classes by counting the total true positives, false positives, true negatives and false negatives.

5.4.4 Comparison Between ASWWL and the Baseline

As described in Section 5.3.3, the proposed framework has a baseline version and an ASWWL version. To verify the effectiveness of the proposed ASWWL algorithm, we perform comparison studies on three different classifiers, namely, Neural Networks (NN), Hoeffding Tree (HT) and Naive Bayes (NB). The hyperparameters are set as follows, batch step n_b=100, sliding window size z=3 for classifiers NN and NB and z=5 for classifier HT.

In this subsection, we present the performance comparison on each single class separately in Figs. 5.4, 5.5, 5.6, and Tables 5.2, 5.3, 5.4, followed by the Macro and Micro average performance on all classes shown in Table 5.5.

5.4.4.1 Classification Results of Different Classifiers on the Majority Class Neg

Fig. 5.4 shows the real-time classification result on class Neg. The red line, IF, in subplot (a) is the overall Imbalance Factor of class Neg, calculated by Equation

(5.2), while the blue line, SWIF, in subplot (a) represents the Sliding Window Imbalance Factor calculated by Equation (5.4), when z=50. Obviously, SWIF can reflect the latest imbalanced status, while IF is much less sensitive than SWIF due to the influence of history samples.

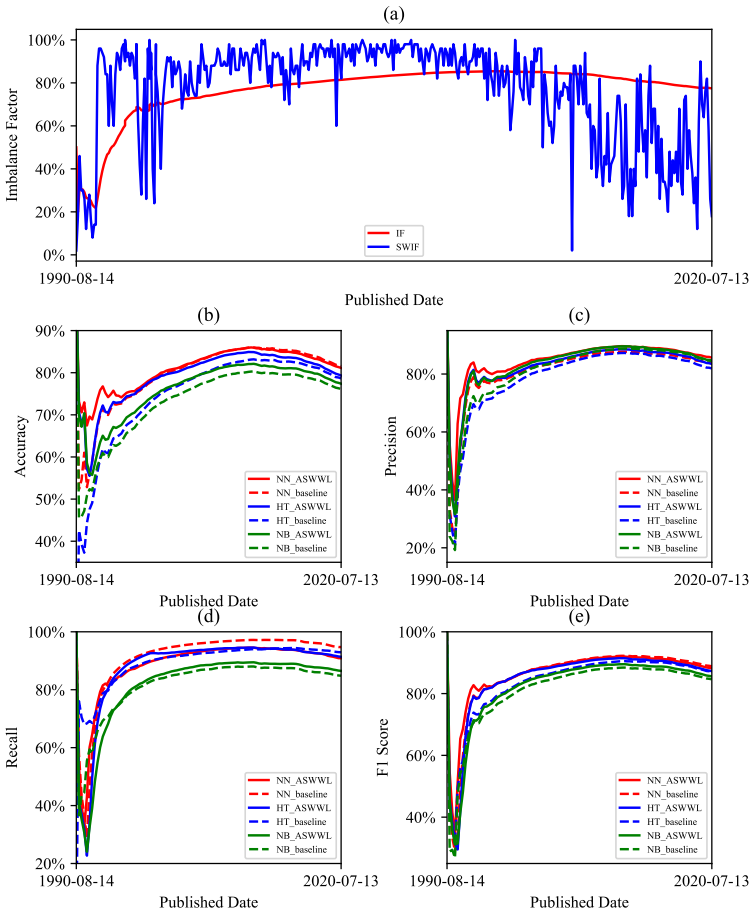


Fig. 5.4. Real-time classification results of different classifiers on class Neg.

Subplots (b), (c), (d), (e) present four metrics respectively. Generally speaking, all these metrics fluctuate in the same trend as the Imbalance Factor of class Neg. Classifier NN, in red color, performs best, followed by HF in blue and NB in green. Furthermore, the ASWWL version performs better than the baseline version for all three classifiers most times.

Table 5.2 shows the quantitative results evaluated on all existing samples in a prequential-evaluation manner. We emphasise the best of each metric and each

classifier in bold. Overall, NN_baseline achieves the best Accuracy, Recall and F1 Score, while NN_ASWWL performs the best in Precision. NN_ASWWL gets a slightly worse F1 Score than NN_baseline. However, the ASWWL version works better than the baseline version when adopting HT or NB as the classifier of the proposed framework.

Table 5.2: Overall classification results of different classifiers on all existing samples of class Neg.

Algorithms	Accuracy	Precision	Recall	F1 Score
NN_ASWWL	81.13%	**85.64%**	90.88%	88.18%
NN_baseline	**81.57%**	83.65%	**94.71%**	**88.84%**
HT_ASWWL	**79.33%**	**83.50%**	91.37%	**87.26%**
HT_baseline	78.65%	81.93%	**92.93%**	87.09%
NB_ASWWL	**77.40%**	**84.66%**	**86.51%**	**85.57%**
NB_baseline	76.14%	84.43%	84.83%	84.63%

Generally speaking, the difference between the ASWWL version and the baseline version on majority class Neg is non-significant and both versions can achieve quite good performance.

5.4.4.2 Classification Results of Different Classifiers on the Minority Class—ZeroDay

Similarly, Fig. 5.5 shows the real-time classification results on one of the minority classes—ZeroDay. The subplot (a) shows the IF in red and SWIF in blue. Subplots (c), (d), (e) display the same fluctuation trend with (a).

For Accuracy in subplot (b), classifier NN and HT obtain an equivalent performance, better than classifier NB. There is little difference between the ASWWL version and the baseline version on Accuracy. However, the ASWWL version of all three classifiers perform significantly better than the baseline version on Precision, Recall and F1 Score, which verify the effectiveness of ASWWL in boosting the performance on the minority classes.

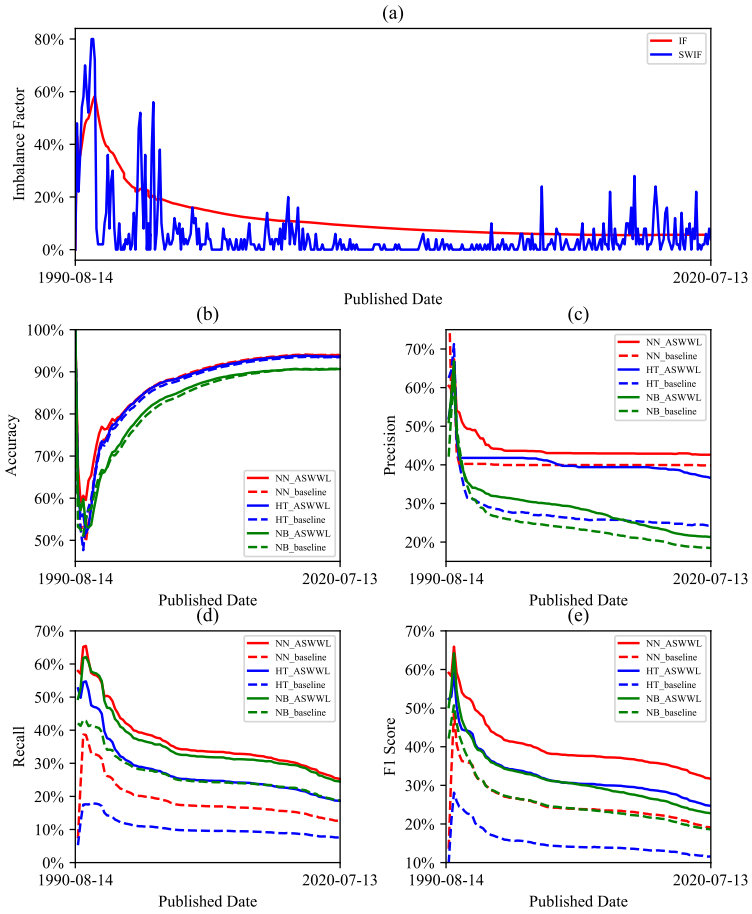


Fig. 5.5. Real-time classification results of different classifiers on class ZeroDay.

Table 5.3 is the corresponding quantitative classification results evaluated on all existing samples. Obviously, the ASWWL version has an overwhelming advantage than the baseline version on Precision, Recall and F1 Score while keeping an equivalent performance on Accuracy. Classifier NN is also the best classifier among these three classifiers.

5.4.4.3 Classification Results of Different Classifiers on the Minority Class—Pos

Class Pos is another minority class in exploitation time prediction problem. The real-time classification results on class Pos are shown in Fig. 5.6. Subplot (a)

Table 5.3: Overall classification results of different classifiers on all existing samples of class ZeroDay.

Algorithms	Accuracy	Precision	Recall	F1 Score
NN_ASWWL	93.89%	**42.62%**	**25.25%**	**31.71%**
NN_baseline	**94.02%**	39.86%	12.55%	19.09%
HT_ASWWL	**93.61%**	**36.57%**	**18.63%**	**24.69%**
HT_baseline	93.48%	24.33%	7.60%	11.59%
NB_ASWWL	90.67%	**21.30%**	**24.49%**	**22.78%**
NB_baseline	**90.77%**	18.45%	18.78%	18.61%

shows that in the beginning years, class Pos accounted for about 35%, and then dropped to less than 20% very quickly. In recent years, its SWIF increased to a level of more than 60%, leading to a remarkable increase in the performance of Precision, Recall and F1 Score, as shown in subplot (c), (d), (e). Classifier NN also performs best in most times. Classifier NB performs much better than HT on Recall and F1 Score for class Pos.

Table 5.4 presents the overall classification result on all existing samples of class Pos. We bold the best performance on each classifier and each metric. The results in Table 5.4 show that both baseline and ASWWL may achieve the best on some metrics. However, considering F1 Score is a balanced metrics reflecting both precision and recall, we conclude that the ASWWL version performs much better than the baseline version, when adopting NN and HT as the classifier. The baseline version performs slightly better than the ASWWL version when choosing NB as the classifier.

5.4.4.4 Average Classification Results of Different Classifiers on Multiple Classes

According to the description above, generally speaking, the ASWWL version performs better than the baseline version on F1 Score, especially for these two minority classes, no matter adopting which classifier. Two exceptions are NN_ASWWL on the class Neg and NB_ASWWL on the class Pos.

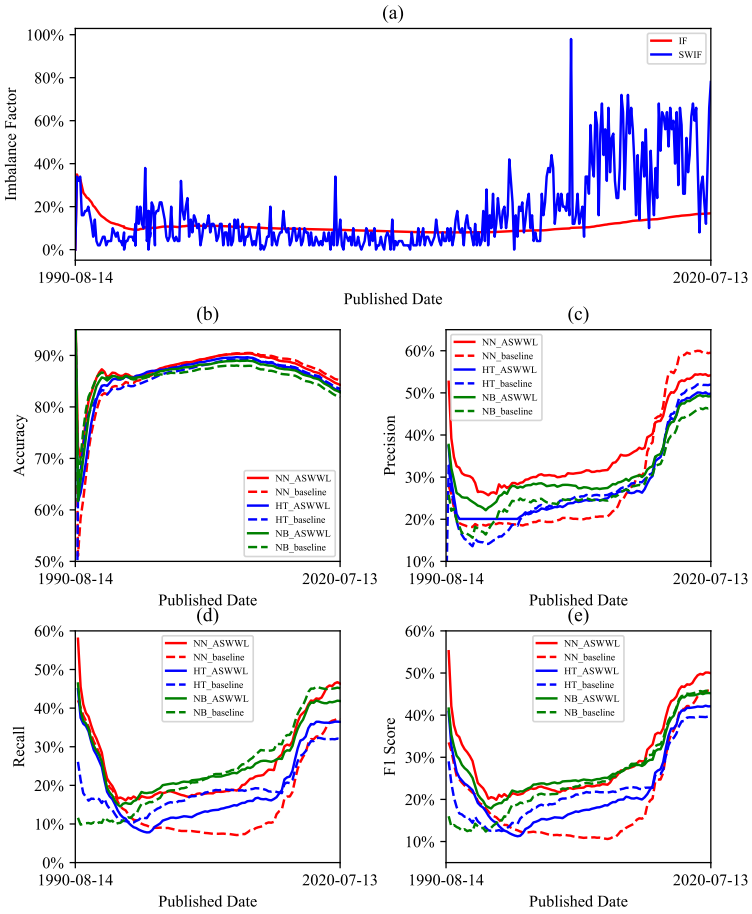


Fig. 5.6. Real-time classification results of different classifiers on class pos.

We list the Macro and Micro average performance over all three classes in Table 5.5. We can see that the ASWWL version achieves a much better Macro average F1 Score than the baseline version on all three classifiers. Among them, NN_ASWWL achieves the best Macro F1 Score on 56.62%, 5.42% higher than NN_baseline and 5.27% higher than the second place, HT_ASWWL. For Micro F1 Score, the ASWWL version with a HT or NB classifier also performs better than the baseline version over 1%. NN_baseline is slightly better than NN_ASWWL.

To conclude, ASWWL is effective to boost the classification performance of the minority class, by giving larger weights to minority class samples adaptively.

Table 5.4: Overall classification results of different classifiers on all existing samples of class Pos.

Algorithms	Accuracy	Precision	Recall	F1 Score
NN_ASWWL	84.29%	54.19%	**46.35%**	**49.97%**
NN_baseline	**85.09%**	**59.56%**	37.05%	45.69%
HT_ASWWL	83.04%	49.83%	**36.45%**	**42.10%**
HT_baseline	**83.50%**	**52.00%**	31.90%	39.54%
NB_ASWWL	**82.85%**	**49.20%**	41.83%	45.22%
NB_baseline	81.91%	46.43%	**45.14%**	**45.77%**

5.4.5 Comparison With Other Data Stream Learning Algorithms

The proposed framework is designed to predict the exploitation time in a data stream context. To verify its performance, we compared the ASWWL version of the proposed framework with a NN classifier (our method) with five data stream learning algorithms introduced in Section 5.2.1, namely, SAMKNN, DWN, HTA, LPPNSE and VFDRC on the same dataset. We first list the results on each class separately and then give the average performance on these classes.

5.4.5.1 Classification Results of Different Data Stream Learning Algorithms on the Majority Class Neg

The realtime classification performance comparison on class Neg is shown in Fig. 5.7. Subplot (a) is the realtime IF and SWIF of class Neg. Generally speaking, SAMKNN, HTA and our method achieve better performance than the rest on all four metrics. The detailed results are summarized in Table 5.6.

As shown in Table 5.6, our method gets an 88.18% F1 Score, which is at the equivalent level of the best, 88.45%, achieved by SAMKNN. HTA also gets quite a high F1 Score at 88.11% compared with the rest. DWD performs the worst on class Neg, with an 82.86% F1 Score. our method also achieves almost the same

Table 5.5: Average classification results of different classifiers on multiple classes.

Average	Algorithms	Accuracy	Precision	Recall	F1 Score
Macro	NN_ASWWL	79.66%	60.82%	54.16%	**56.62%**
	NN_baseline	**80.34%**	61.02%	48.11%	51.20%
	HT_ASWWL	**77.99%**	56.63%	48.82%	**51.35%**
	HT_baseline	77.81%	52.75%	44.15%	46.07%
	NB_ASWWL	**75.46%**	51.72%	50.94%	**51.19%**
	NB_baseline	74.41%	49.77%	49.58%	49.67%
Micro	NN_ASWWL	79.66%	79.66%	79.66%	79.66%
	NN_baseline	**80.34%**	80.34%	80.34%	**80.34%**
	HT_ASWWL	**77.99%**	77.99%	77.99%	**77.99%**
	HT_baseline	77.81%	77.81%	77.81%	77.81%
	NB_ASWWL	**75.46%**	75.46%	75.46%	**75.46%**
	NB_baseline	74.41%	74.41%	74.41%	74.41%

Accuracy as SAMKNN. DWM performs best on Precision with 97.55% and HTA on Recall with 93.57%.

5.4.5.2 Classification Results of Different Data Stream Learning Algorithms on the Minority Class—ZeroDay

The real-time performance on one of the minority classes ZeroDay is presented in Fig. 5.8. Subplots (a) to (e) demonstrate the real-time imbalance status and four classification metrics on class ZeroDay. SAMKNN presented by the blue dotted lines achieves the best Accuracy, Precision and F1 score, followed by our method in red solid lines. As for Recall, the HTA in green solid line achieves the best, followed by SAMKNN, our method and VFDRC. DWM performs the worst on class ZeroDay.

Table 5.7 gives the overall classification result on all existing samples of class ZeroDay. SAMKNN also performs the best on Accuaray, Precision and F1 Score. HTA achieves the best Recall. our method is the runner-up on all metrics. HTA is also in the third place of the overall performance measured by F1 Score. The worst

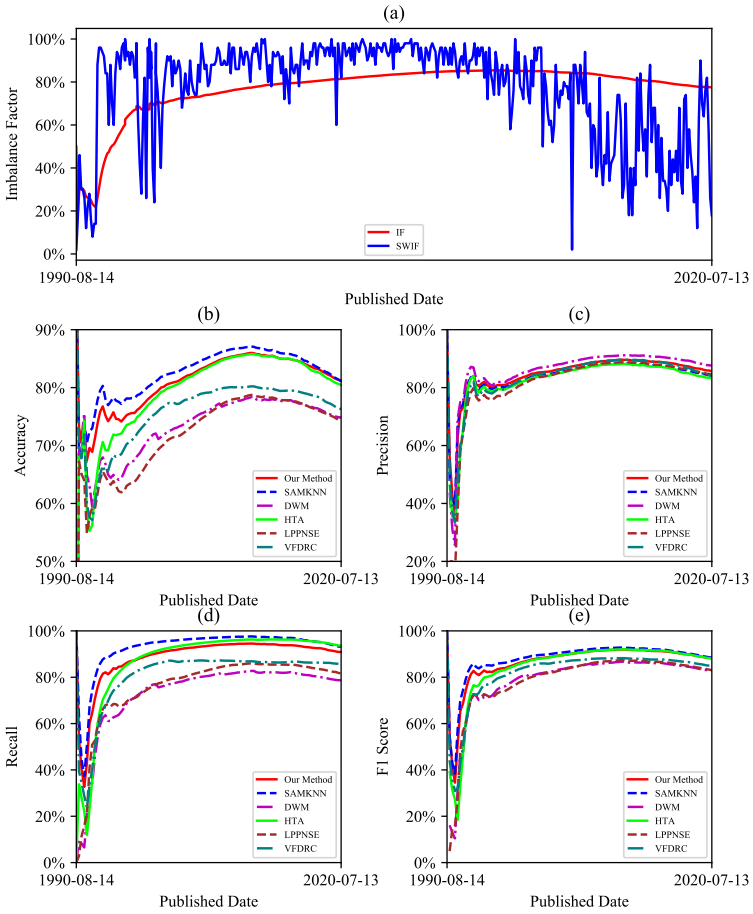


Fig. 5.7. Real-time classification results of different data stream learning algorithms on class Neg.

F1 Score belongs to LPPNSE with 19.03%, which is also quite good, considering that class ZeroDay only accounts for 5.62%.

5.4.5.3 Classification Results of Different Data Stream Learning Algorithms on the Minority Class—Pos

Finally, we present the real-time classification result on class Pos in Fig 5.9. HTA, SAMKNN and our methods are at the first level in regard to Accuracy and Precision. However, both HTA and SAMKNN have very poor performance on Recall and F1 Score. our method, along with LPPNSE and VFDRC, achieves

Table 5.6: Overall classification results of different data stream learning algorithms on all existing samples of class Neg.

Algorithms	Accuracy	Precision	Recall	F1 Score
Our Method	81.13%	85.64%	90.88%	88.18%
SAMKNN	**81.15%**	84.17%	93.19%	**88.45%**
DWM	74.80%	**87.55%**	78.65%	82.86%
HTA	80.44%	83.26%	**93.57%**	88.11%
LPPNSE	74.19%	84.50%	81.66%	83.06%
VFDRC	76.28%	83.98%	85.73%	84.85%

Table 5.7: Overall classification results of different data stream learning algorithms on all existing samples of class ZeroDay.

Algorithms	Accuracy	Precision	Recall	F1 Score
Our Method	93.89%	42.62%	25.25%	31.71%
SAMKNN	**94.32%**	**48.96%**	24.94%	**33.05%**
DWM	92.91%	31.89%	22.97%	26.70%
HTA	93.13%	35.13%	**26.24%**	30.04%
LPPNSE	89.71%	17.05%	21.52%	19.03%
VFDRC	90.90%	22.54%	25.40%	23.88%

the best F1 Score most of the times.

Table 5.8 displays the overall classification result on all existing samples of class Pos. HTA achieves the best Accuracy and Precision and DWM performs best on Recall. Taking F1 Score as the overall performance of an algorithm, our method obtains the best at 49.97%, 5.69% higher than the second one, 45.52% achieved by DWM.

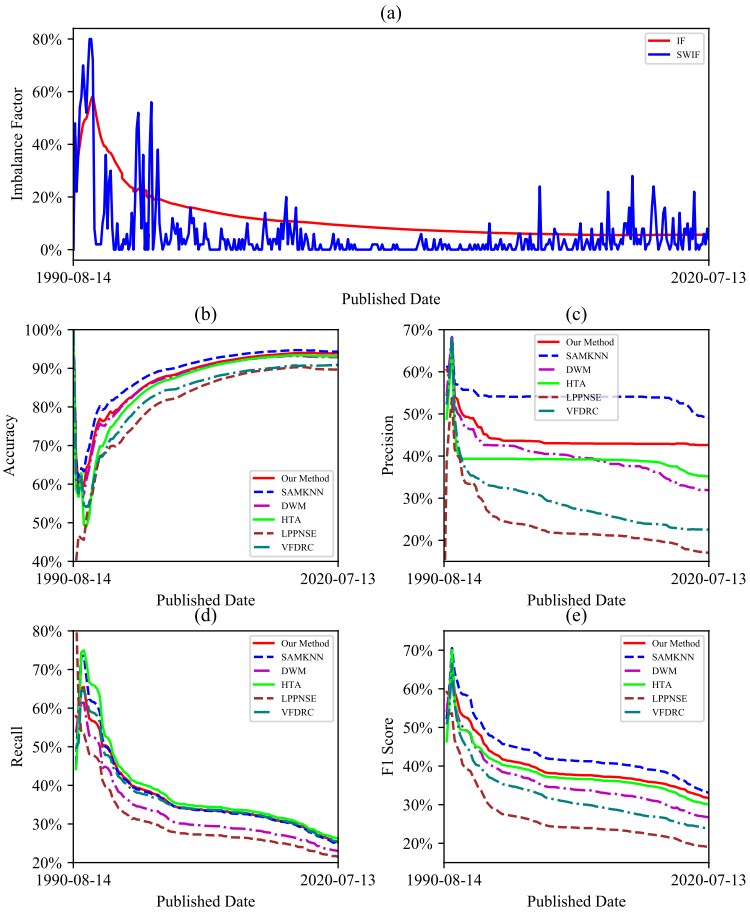


Fig. 5.8. Real-time classification results of different data stream learning algorithms on class ZeroDay.

5.4.5.4 Average Classification Results of Different Data Stream Learning Algorithms on Multiple Classes

As discussed above, not a single algorithm achieves the best on all classes or all metrics. We further discuss the performance of the afore-mentioned algorithms by calculating their Macro and Micro average classification performance. As shown in Table 5.9, our method achieves the best Macro F1 Score with 56.62%, which is 1.36% higher than the second place, SAMKNN, and 3.73% higher than the third place (HTA).

With respect to the Micro F1 Score, SAMKNN achieves the best with 79.85,

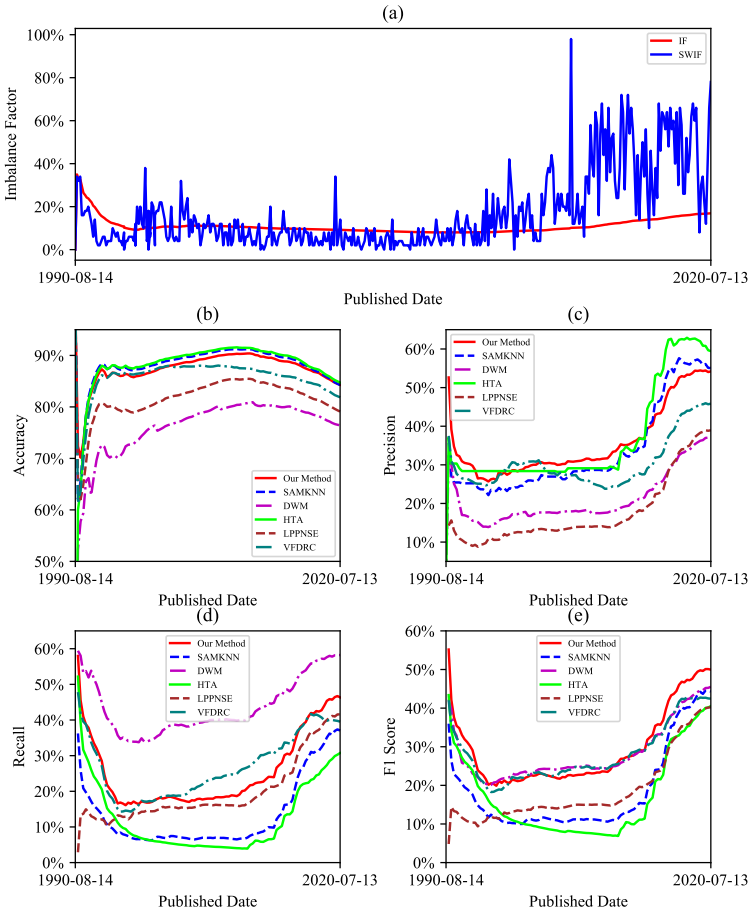


Fig. 5.9. Real-time classification results of different data stream learning algorithms on class Pos.

which is only 0.19% better than our method and 0.70% higher than HTA, the third best algorithm.

The results show that our method is among the best of all these six algorithms. Our method is also the most robust algorithm, which can always achieve the best or the equivalent best performance on different metrics and classes.

Table 5.8: Overall classification results of different data stream learning algorithms on all existing samples of class Pos.

Algorithms	Accuracy	Precision	Recall	F1 Score
Our Method	84.29%	54.19%	46.35%	**49.97%**
SAMKNN	84.24%	55.07%	37.03%	44.28%
DWM	76.42%	37.37%	**58.22%**	45.52%
HTA	**84.74%**	**59.45%**	30.74%	40.53%
LPPNSE	79.09%	38.96%	41.58%	40.22%
VFDRC	81.87%	45.84%	39.53%	42.45%

Table 5.9: Average classification results of different data stream learning algorithms on multiple classes.

Average	Algorithms	Accuracy	Precision	Recall	F1 Score
Macro	Our Method	**79.66%**	**60.82%**	**54.16%**	**56.62%**
	SAMKNN	**79.85%**	**62.73%**	51.72%	55.26%
	DWM	72.07%	52.27%	53.28%	51.70%
	HTA	79.15%	59.28%	50.18%	**52.89%**
	LPPNSE	71.50%	46.84%	48.25%	47.44%
	VFDRC	74.53%	50.79%	50.22%	50.39%
Micro	Our Method	**79.66%**	**79.66%**	**79.66%**	**79.66%**
	SAMKNN	**79.85%**	**79.85%**	**79.85%**	**79.85%**
	DWM	72.07%	72.07%	72.07%	72.07%
	HTA	79.15%	79.15%	79.15%	**79.15%**
	LPPNSE	71.50%	71.50%	71.50%	71.50%
	VFDRC	74.53%	74.53%	74.53%	74.53%

5.5 Conclusions

Vulnerability exploitation time prediction is of importance for vulnerability assessment and cybersecurity management. To provide results with finer granularity and adopt a real-world online learning situation, we treat this task as a

data stream classification problem. This chapter proposes an integrated consecutive batch learning framework to predict the exploitation time of vulnerabilities. Furthermore, a SWIF index is designed to indicate the real-time dynamic class imbalance status. We further propose an ASWWL algorithm to handle the existing dynamic multiclass imbalance problem in data stream scenarios. Experiments conducted on real-world vulnerabilities collected between 1990 and 2020 show that the ASWWL algorithm is effective in boosting the classification performance of the minority classes without compromising the performance of the majority class. We also compare our method with the other five data stream learning algorithms. The results show that our method is the most robust algorithm on different metrics and classes. The performance of our method is also among the best of all these six algorithms.

Chapter 6

Vulnerability Co-exploitation Behaviour Discovery

Complex graph connections exist between vulnerabilities and exploits. However, previous studies and tools on vulnerability assessment are often designed to assess a single vulnerability separately, based on its text description information, affected products, Common Vulnerability Scoring System (CVSS) metric scores and other available attributes [7, 21, 42, 45]. Leveraging the latest advances in graph driven intelligence, this chapter explores the vulnerability risk evaluation by analysing the co-exploitation behaviours between vulnerabilities.

Co-exploitation behaviour, referring to multiple software vulnerabilities being exploited jointly by one or more exploits, brings enormous challenges to the prevention and remediation of cyberattacks. This chapter formulates software vulnerability co-exploitation behaviour discovery as a link prediction problem between vulnerability entities within a cybersecurity domain-specific knowledge graph. To boost performance and enhance the explainability of link prediction, a Modality-Aware Graph Convolutional Network (MAGC) module is proposed to embed multimodality entity attributes and topological graph connectivity features into a unified lower-dimensional feature space. Further, a Graph Knowledge Transfer Learning (GKTL) strategy is designed to transfer knowledge between subgraphs extracted from the same knowledge graph. The proposed strategies

are demonstrated by the experiments conducted on a real-world cybersecurity knowledge graph consisting of co-exploitation incidents between 1995 to 2021.

The rest of this chapter is organised as follows. Section 6.1 briefly introduces the co-exploitation discovery problem and the contributions of this chapter. Section 6.2 presents the related works on graph embedding and representation. Section 6.3 specifies the principal and implementation details of the proposed MAGCN and GKTL algorithms. Section 6.4 presents the experimental data and the results of co-exploitation behaviour discovery. A conclusion is presented in the last section.

6.1 Introduction

6.1.1 Motivations

Software vulnerabilities, also known as bugs or weaknesses in software, pose a significant threat to modern information systems' data security and privacy protection. To make matters worse, the explosive growth in the number of software vulnerabilities leaves vendors overwhelmed, and it is difficult for vendors to provide patches and remedies on time. Vulnerability evaluation and assessment is the fundamental and systematic review process to identify the severity levels of existing vulnerabilities in cybersecurity [112, 113]. It is essential for information systems to find out potential security weaknesses and thus to recommend remediation or mitigation actions timely [42, 114, 115]. Previous studies and tools on vulnerability assessment are often designed to assess individual vulnerability separately based on its text description information, affected products, Common Vulnerability Scoring System (CVSS) metric scores, and other available attributes [7, 21, 45]. Consequently, they fail to analyse the inner relationships between different vulnerabilities.

In-depth analysis of security incidents shows that more and more exploits, also known as malware or malicious software, tend to attack multiple vulnerabilities jointly. With the explosive growth trend in the number of software vulnerabilities, co-exploitation behaviours will also increase dramatically in future. An

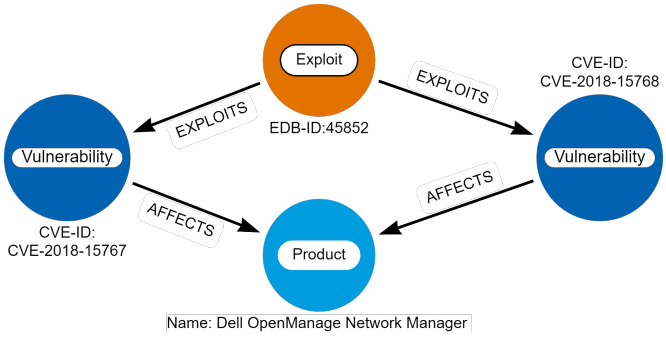


Fig. 6.1. Examples of real world co-exploitation behaviours.

example of co-exploitation is shown on the official web page [116] of Dell Technologies. Multiple vulnerabilities are found in Dell OpenManage Network Manager (OMNM), identified as CVE-2018-15767 [117] and CVE-2018-15768 [118]. The first vulnerability can make malicious users get root privileges to run arbitrary commands on OMNM. If the first vulnerability is exploited, attackers can leverage the escalated privileges to read/write files stored on the server filesystem through the second vulnerability. A proof-of-concept exploit, identified as EDB-ID: 48582, can co-exploit these two vulnerabilities and is available in the Exploit DataBase (ExploitDB) [119]. Co-exploitation behaviour brings enormous challenges to the prevention and remediation of cyber-attacks. It takes Dell Technologies over 8.5 months to release a new version of OMNM to patch these two vulnerabilities from February 16, 2018, until November 2, 2018.

We illustrate this example in a knowledge graph, as shown in Fig. 6.1. There are three types of entities with labels, i.e., 'Exploit' in orange, 'Vulnerability' in navy and 'Product' in blue accordingly. The 'Exploit' entity identified by EDB-ID: 45852 [119] is the proof-of-concept exploit to co-exploit the two vulnerabilities found in Dell OMNM, namely, CVE-2018-15767 [117] and CVE-2018-15768 [118], respectively. Therefore, 'EXPLOITS' relationships exist between them. Furthermore, both vulnerabilities have an 'AFFECTS' relationship with the 'Product' entity, which has a 'Name: Dell OpenManage Network Manager' property. To conclude, Fig. 6.1 demonstrates that the exploit EDB-ID: 45852 can attack Dell OMNM by co-exploiting CVE-2018-15767 and CVE-2018-15768.

Co-exploitation behaviour can reveal intricate interrelationships between different vulnerabilities. Taking co-exploitation behaviours into account can provide a broader and deeper perspective for vulnerability assessment than considering a single vulnerability itself. Furthermore, co-exploitation behaviour discovery can help the vendors remedy multiple interrelated vulnerabilities in one update or solution. It can not only save money and time for vendors but also improve user experience by less bothering them with vulnerability alerts and remedy releases. However, traditional deep learning algorithms are not good at analysing the relationships between different entities because of the inherent deficiency of their implementation process. To the best of our knowledge, no related works have been trying to predict the possible co-exploitation behaviours between different vulnerabilities. To fill this gap, this chapter is trying to explore a feasible way to predict if two vulnerabilities will be co-exploited in the future, leveraging the knowledge graph's power.

6.1.2 Challenges

Knowledge graph (KG), as a new data structure containing not only the topological connectivity information between nodes but also the non-topological attributes and features of nodes, has emerged as the primary tool for knowledge representation and reasoning across the areas of semantic webs, finance industry, social networks, e-commerce recommendations, protein interaction analysis et al.. Among KG-empowered applications, link prediction has been attracting increased attention in academia and industry.

The co-exploitation behaviour discovery problem can be formulated as a link prediction problem between different 'Vulnerability' entities within a cybersecurity domain-specific KG. Specifically, we can construct a co-exploitation graph by adding virtual 'CO-EXPLOITATION' edges between two 'Vulnerability' entities exploited by the same 'Exploit' entity. For example, Fig. 6.1 could be refactored as Fig. 6.2 in this way. The corresponding 'Exploit' entity EDB-ID: 45822 is saved as a property of the added 'CO-EXPLOITATION' relationship. As a result, the problem of predicting if co-exploitation behaviour exists between two vulnerabili-

Fig. 6.2. Transformed co-exploitation graph.

ties is transformed into predicting if a 'CO-EXPLOITATION' relationship exists between two 'Vulnerability' entities in a graph.

As a link prediction task, based on the cybersecurity domain knowledge, the main challenges for co-exploitation behaviour discovery are identified as below.

(1) Multi-modality problem. A huge volume of multimodal information, including videos, photos, speeches, reports, code snippets and texts, can be involved in vulnerability co-exploitation behaviour prediction. Diverse information comes from credible vendors, cybersecurity experts, technical posts, and social media. How to properly fuse, aggregate and embed multi-modality information is a key problem to be solved.

(2) Graph sparsity problem. As a subgraph extracted from a cybersecurity knowledge graph, the co-exploitation graph is a very sparse one. According to [41], only about 17.60% of vulnerabilities published between 1999 to 2020 have corresponding exploits. Based on our further investigation, the ratio of vulnerabilities involving co-exploitation behaviour is less than 5% among all vulnerabilities. Therefore, how to boost the link prediction performance is another challenge when considering the sparsity of the co-exploitation graph.

(3) Time difference problem. Because of the diversity of data sources and the possible conflicting interests between different parties, disclosure time difference often exists between different information sources on the same vulnerability. For example, for a vulnerability, usually, the verified vendor description is available earlier than the CVSS scores and its exploitation information. The time difference problem should also be considered when designing the link prediction algorithms within a cybersecurity knowledge graph.

6.1.3 Contributions

As the first exploration of co-exploitation behaviour discovery in cybersecurity, the main contributions of this chapter are threefold.

(1) We propose a Modality-Aware Graph Convolutional Networks (MAGCN) for graph knowledge embedding and representation in multi-modality scenarios. Instead of concentrating on multi-modality features before inputting a Graph Neural Networks (GNN) model, MAGCN treats each modality separately with flexible message passing, aggregation and update functions within a traditional GNN forward propagation process. Therefore, the intermediate embedding results of each modality are detachable and available for use, which increases the transparency and explainability of the contribution of each modality. MAGCN can work as a general GNN module combined with other GNN modules to form more complex deep learning models.

(2) We design a Graph Knowledge Transfer Learning (GKTL) strategy to tackle sparsity and time difference problems. Firstly, we extract a target graph and a source graph from a knowledge graph following some restrictions to ensure that the source graph is much denser than the target graph and no time difference exists in the extracted sub-graphs. Then, train and learn graph knowledge from the source graph with a self-supervised link prediction task. Finally, transfer the learnt graph knowledge to the target graph and finalise the target link prediction task. GKTL provides a general solution for link prediction tasks' sparsity and time difference problems.

(3) We explore using link prediction algorithms to discover co-exploitation behaviours empowered by a cybersecurity domain-specific knowledge graph. The effectiveness of the proposed MAGCN and GKTL algorithms are verified on a real-world KG containing co-exploitation instances between 1995 to 2021.

The rest of this chapter is organised as follows. Section 6.2 introduces related works. Section 6.3 specifies the proposed MAGCN and GKTL algorithms. Section 6.4 presents experimental data and results of co-exploitation behaviour discovery. Conclusion and future works are discussed in the last section.

6.2 Related Works

Graph embedding and representation is the foundation of most graph-based applications. Traditional node topological feature embedding methods in graph theory include Page Rank [120], Article Rank [121], Betweenness Centrality [122], Harmonic Centrality [123] and so forth. These algorithms are widely used to measure the importance of nodes based on graph connectivity. Inspired by recent deep learning advances, such as the skip-gram and word2vec models, many research works also embed the nodes connectivity information with deep learning techniques. Among them, Deepwalk [124] and node2vec [125] are two well-known pioneer graph embedding methods. Other works embed the graph connectivities from a matrix factorization perspective, such as SocDim, NEU, HOPE and GraRep [126].

However, the aforementioned methods are all focused on the structural features and ignore the nodes attributes. To leverage the features from both non-topological and topological aspects, a series of algorithms based on GNN have been proposed. These algorithms learn a function to generate a node's embedding by aggregating the non-topological features from itself, its neighbour nodes and their relationships. The learnt GNN can be generalized to unseen nodes or graphs. Below, we present some representative GNN examples, namely, Graph Convolutional Networks (GCN) [127], GraphSAGE [128] and Graph Attention Networks (GAT) [129].

Let a graph $\mathcal{G}$ be represented by a triplet ($\mathcal{V}$, A, X), where $\mathcal{V}$ is the vertex set, A is the adjacency matrix indicating edges between nodes and $X \in \mathbb{R}^{n \times |\mathcal{V}|}$ is the feature matrix corresponding to $\mathcal{V}$, n is the node's attribute dimension and $|\mathcal{V}|$ is the total number of nodes in $\mathcal{V}$. X could be extracted from multi-modality sources. For example, node feature sources in social networks may include users' profiles, images, and posts.

Generalizing the idea of convolutional networks beyond simple pixel lattices, GCN transforms and combines the information from neighbours of a node to compute its embedding. The primary forward propagation process is formulated

as (6.1).

$$h_v^{(l)} = \sigma\left(\sum_{u\in\mathcal{N}(v)} \frac{1}{c_{uv}} h_u^{(l-1)} W^{(l)} + b^{(l)}\right), (l \geq 1), \tag{6.1}$$

where $h_v^{(l)}$ is the embedding of node v; l is the current depth of the GCN layer; $\mathcal{N}(v)$ denotes the set of neighbour nodes of node v; $c_{uv} = \sqrt{|\mathcal{N}(u)|}\sqrt{|\mathcal{N}(v)|}$ is a normalization constant based on graph structure working as the weight of massage from node $u \in \mathcal{N}(v)$ to v; $|\cdot|$ denotes the number of distinct nodes in the specified set; σ is the activation function; $W^{(l)}$, $b^{(l)}$ are the node-wise learnable parameters, shared by all nodes in V. The initialization embedding of node v at layer $l = 0$ is its original node features, i.e., $h_v^{(0)} = x_v$.

To provide more flexible aggregation strategies when dealing with massages from $\mathcal{N}(v)$ and take the embedding of node v itself into consideration, the authors [128] proposed a GraphSAGE GNN module, as shown in (6.2) and (6.3).

$$h_{\mathcal{N}(v)}^{(l)} = \mathrm{AGG}\left(\{h_u^{l-1}, \forall u \in \mathcal{N}(v)\}\right), \tag{6.2}$$

where $h_{\mathcal{N}(v)}^{(l)}$ is the aggregation of massages from $\mathcal{N}(v)$ at layer l, 'AGG' denotes the aggregation operator, which can be flexibly designed as 'mean', 'average', 'max', 'min' or more complicated functions, such as MLP and LSTM layers.

$$h_v^{(l)} = \sigma\left(\left(h_v^{(l-1)} \parallel h_{\mathcal{N}(v)}^{(l)}\right) W^{(l)} + b^{(l)}\right), \tag{6.3}$$

where $\parallel$ means the concatenation operator and $h_v^{(l-1)}$ is the embedding of node v at layer $l-1$.

To automatically learn the weights of neighbour nodes when aggregating messages, GAT was proposed to apply an attention strategy to GCN. The main implementation steps of GAT are broken down as (6.4)-(6.7), [129].

$$z_v^{(l-1)} = W^{(l-1)} h_v^{(l-1)}, \tag{6.4}$$

where $z_v^{(l-1)}$ is the linear transformation for node embedding $h_v^{(l-1)}$ and $W^{(l-1)}$ is the corresponding learnable node-wise shared weight matrix.

$$e_{vu}^{(l-1)} = \mathrm{LeakyReLU}\left(a^{(l-1)^T}(z_v^{(l-1)} \parallel z_u^{(l-1)})\right), \tag{6.5}$$

where $e_{vu}^{(l-1)}$ is the attention score between two nodes v and u, activated by a LeakyReLU function; $a^{(l-1)}$ is a learnable weight vector for the concatenation of $z_v^{(l-1)}$ and $z_u^{(l-1)}$.

$$\alpha_{vu}^{(l-1)} = \frac{\exp(e_{vu}^{(l-1)})}{\sum_{k \in \mathcal{N}(v)} \exp(e_{vk}^{(l-1)})}, \tag{6.6}$$

where $\alpha_{vu}^{(l-1)}$ is the normalized attention scores calculated by applying a softmax function on the pair-wise attention scores between node v and its neighbours.

$$h_v^{(l)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l-1)} z_u^{(l-1)}\right). \tag{6.7}$$

Equation (6.7) shows the final aggregation process, where $\alpha_{vu}^{(l-1)}$ works as the weight to show the importance of the corresponding relationship between v and u.

In addition to the attention mechanisms, many other modern deep learning techniques can also be incorporated into a GNN module, such as batch normalization, dropout, gating and skip connections. Our proposed MAGCN belongs to a variation of GCN, focusing on multi-modality scenarios.

6.3 Methodology

6.3.1 Modality-Aware Graph Convolutional Networks

MAGCN is tailored for node embedding of monopartite graphs with multiple-modality node features. Typical feature modalities include visual, acoustic, textual, tabular, etc. A schematic illustration of MAGCN is shown in Fig. 6.3, along with the elaboration of our design and implementation presented in the following sections.

6.3.1.1 Problem Setting

Let a monopartite graph $\mathcal{G}$ be represented by a triplet ($\mathcal{V}$, A, X), where $\mathcal{V}$ is the vertex set; A is the adjacency matrix and $X = [X_1 \parallel X_2 \parallel \cdots \parallel X_m]$ is the

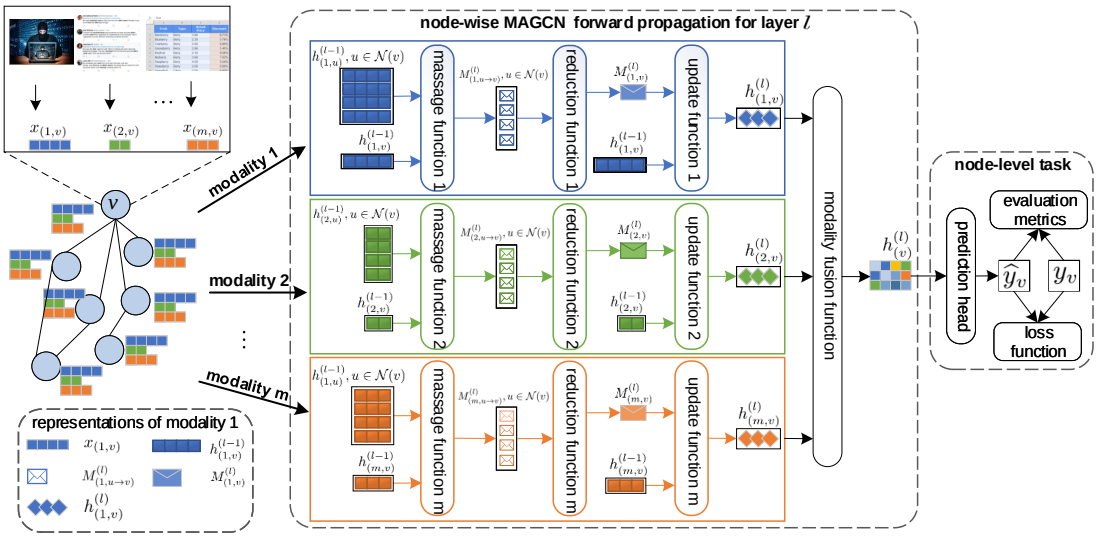


Fig. 6.3. Schematic illustration of the proposed MAGCN module.

concatenated graph feature matrix. $X_i \in \mathbb{R}^{n_i \times |\mathcal{V}|}$ $(i \in \{1, 2, \cdots, m)\}$ is the feature matrix of the i-th modality; n_i is the corresponding feature dimension; $|\mathcal{V}|$ is the total number of nodes in $\mathcal{V}$ and m is the total number of feature modalities. For an individual node $v \in \mathcal{V}$, the i-th modality feature vector is denoted as $x_{(i,v)}$, which is the i-th vector in feature matrix X_i. Fig. 6.3 demonstrates a typical node v and its multi-modal features $x_{(1,v)}$, $x_{(2,v)}$, $\cdots$, $x_{(m,v)}$ in the upper left corner.

6.3.1.2 Modality-Aware Node Embedding

As shown in Fig. 6.3, the main idea of MAGCN is to set m channels to deal with m modalities separately. For a specific channel $i \in \{1, 2, \cdots, m\}$, the main forward propagation processes of layer l $(l \geq 1)$ include an edge-wise message passing, a node-wise message reduction and a node-wise embedding update.

The edge-wise message passing process for the i-th modality in layer l $(l \geq 1)$ can be formulated as (6.8)-(6.10), where $z_{(i,v)}^{(l-1)}$ is a linear transformation; W_i^{l-1} is the node-wise shared trainable parameters; $h_{(i,v)}^{(l-1)}$ is the i-th modality embedding of layer $l-1$ for node v, which is initialized as (6.9). In Equation (6.10), $M_{(i,u\to v)}^{(l)}$ is the massage passed from $u \in \mathcal{N}(v)$ to v through edge $u \to v$; $\phi_i^{(l)}$ is the applied

message function, which could be 'add', 'sub', 'mul', 'div', dot product or other vector to vector operations.

$$z_{(i,v)}^{(l-1)} = W_i^{(l-1)} h_{(i,v)}^{(l-1)}, (v \in \mathcal{V}). \tag{6.8}$$

$$h_{(i,v)}^{(0)} = x_{(i,v)}, (i \in \{1, 2, \cdots, m\}, v \in \mathcal{V}). \tag{6.9}$$

$$\begin{aligned} &M_{(i,u\to v)}^{(l)} = \phi_i^{(l)} \left(z_{(i,v)}^{(l-1)}, z_{(i,u)}^{(l-1)} \right), \\ &(v \in \mathcal{V}, u \in \mathcal{N}(v), l \geq 1, i \in \{1, 2, \cdots, m\}). \end{aligned} \tag{6.10}$$

The next is the node-wise message reduction process, formulated as (6.11), where $\rho_i^{(l)}$ is the applied reduce function to aggregate all messages from $\mathcal{N}(v)$ into a final message $M_{(i,v)}^{(l)}$. General reduce functions include 'sum', 'max', 'min', 'mean', etc.

$$M_{(i,v)}^{(l)} = \rho_i^{(l)} \left(\left\{ M_{(i,u\to v)}^{(l)}, \forall u \in \mathcal{N}(v) \right\} \right), (v \in \mathcal{V}). \tag{6.11}$$

Sequentially, MAGCN will update node-wise embeddings by aggregating $M_{(i,v)}^{(l)}$ and $h_{(i,v)}^{(l-1)}$ with an update function $\psi_i^{(l)}$, as shown in Equation (6.12), where $h_{(i,v)}^{(l)}$ is the embedding of the i-th modality of node v at layer l. Θ_i^l is the node-wise shared trainable parameter matrix of $\psi_i^{(l)}$.

$$h_{(i,v)}^{(l)} = \psi_i^{(l)} \left(\Theta_i^l, h_{(i,v)}^{(l-1)}, M_{(i,v)}^{(l)} \right), (v \in \mathcal{V}). \tag{6.12}$$

Fig. 6.3 demonstrates the modality-aware node embedding process of modality 1 with blue, modality 2 with green and modality m with orange. In the lower-left corner, taking the modality 1 as an example, we list a legend of different symbols and representations.

6.3.1.3 Modality Fusion, Module Stacking and Training

After the node-wise modality-aware embedding, we get the node embeddings $h_{(i,v)}^{(l)}(i \in \{1, 2, \cdots, m), v \in \mathcal{V}\}$ from all modalities, which could be used to stack more MAGCN layers. However, when stacking with other GNN layers (i.e., GCN, GAT) after a MAGCN layer, we need to fuse the node embeddings of v from all modalities using a modality fusion function $f^{(l)}$, as shown in Equation (6.13). $h_{(v)}^{(l)}$ is the fused overall embedding and $\Lambda^{(l)}$ is the node-wise shared trainable parameter matrix of $f^{(l)}$. Apart from being used to stack with other GNN layers, the

overall node embedding $h_{(v)}^{(l)}$ also works as the input of the subsequent prediction head layer to deal with node-/edge-/graph-level tasks. The right part of Fig. 6.3 illustrates an example of a node-level task.

$$h_{(v)}^{(l)} = f^{(l)}\left(\Lambda^{(l)}, \{h_{(i,v)}^{(l)}, \forall i \in \{1, 2, \cdots, m\}\}\right). \tag{6.13}$$

The node-wise learnable parameters Wi^l, Θ_i^l and $\Lambda^{(l)}$ of the MAGCN model will be jointly trained with node-/edge-/graph-level tasks in an end-to-end fashion. Specifically, for node-level tasks, the prediction result could be calculated as (6.14), where the operator $\text{Head}_{\text{node}}$ could be a simple linear transformation or other more complicated functions.

$$\widehat{y}_v = \text{Head}_{\text{node}}\left(h_v^{(l)}\right), (v \in \mathcal{V}). \tag{6.14}$$

For edge-level tasks, including link prediction, the prediction would be made using pairs of node embeddings, as shown in (6.15), where u and v are the end nodes of the edge $u \to v$. Edge-level prediction head could be arbitrary function with two vector inputs, for example, $\text{Head}_{\text{edge}}\left(h_u^{(l)}, h_v^{(l)}\right) = \left(h_u^{(l)}\right)^T h_v^{(l)}$.

$$\widehat{y}_{uv} = \text{Head}_{\text{edge}}\left(h_u^{(l)}, h_v^{(l)}\right), (u, v \in \mathcal{V}). \tag{6.15}$$

Similarly, a graph-level task makes predictions using all node embeddings in a graph, as shown in (6.16). The simplest way for $\text{Head}_{\text{graph}}$ is pooling operations, such as 'max', 'min' and 'mean'.

$$\widehat{y}_{\mathcal{G}} = \text{Head}_{\text{graph}}\left(\left\{h_v^{(l)}, \forall v \in \mathcal{V}\right\}\right) \tag{6.16}$$

Comparing the results of the prediction head layer with the corresponding labels, a Mean Squared Error (MSE) loss function for regression tasks or a cross-entropy (CE) loss function for classification tasks could be defined and used to train the parameters of MAGCN model with advanced optimization algorithms, such as Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam). The performance of a MAGCN model can also be evaluated based on the results of the prediction head and the corresponding labels.

To conclude, compared with other GNN modules, MAGCN is always modality-aware in the whole massage passing, reduction and aggregation process. Therefore, users can flexibly design the massage function, reduction function and update function based on the characteristics of the modality. Furthermore, MAGCN enhances the explainability of link prediction by improving the transparency of the intermediate embedding results of each modality and providing the possibility of interpreting the contribution of each modality to the final prediction results.

6.3.2 Graph Knowledge Transfer Learning

6.3.2.1 Problem Setting

Inspired by the transfer learning strategy in Deep Learning (DL), GKTL aims to improve the performance of the predictive tasks on the target graph by using the knowledge learnt from the source graph. When the target graph is incomplete or too sparse to learn enough knowledge from itself, GKTL provides an option to transfer graph knowledge from another more dense and complete source graph.

The target graph is determined by the application tasks. For example, for co-exploitation discovery, the co-exploitation graph with 'Vulnerability' entities and 'CO_EXPLOITATION' relationships is the target graph. There are some restrictions when selecting the corresponding source graph. (1) The source and target graphs are sub-graphs extracted from the same KG, but the source graph is much denser than the target graph. (2) Both the source graph and target graph are monopartite graphs. Their entity set and the corresponding entity property matrix are identical, but their relationship types are different.

Let the target graph be present as $\mathcal{G}_t = (\mathcal{V}, A_t, X, r_t)$, where $\mathcal{V}$ and X are the entity set and its corresponding feature matrix; A_t is the adjacency matrix, which is not full rank when $\mathcal{G}_t$ is unconnected; r_t is the relationship type. Similarly, the source graph is represented as $\mathcal{G}_s = (\mathcal{V}, A_s, X, r_s)$, where $\mathcal{V}$ and X are exactly the same with the target graphs; A_s is the adjacency matrix and r_s is the relationship type of $\mathcal{G}_s$.

Fig. 6.4 shows the schematic illustration of GKTL. A source graph and a target graph are extracted from the same knowledge graph. They share the

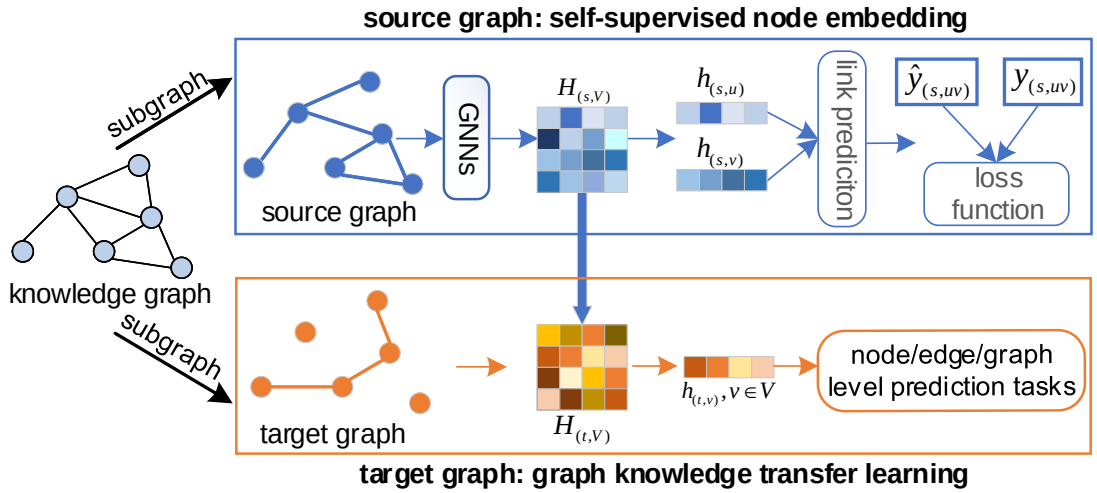


Fig. 6.4. Schematic illustration of GKTL.

same $\mathcal{V}$ and X but have different relationship types, differentiated with blue and orange. Firstly, the source graph learns node embeddings $H_{(s,\mathcal{V})}$ with arbitrary GNNs and a self-supervised link prediction task. Then, the target graph transfers the learnt $H_{(s,\mathcal{V})}$ as its own node embedding $H_{(t,\mathcal{V})}$, followed by node-/edge-/graph-level prediction tasks. The detailed implementation is presented in the following subsections.

6.3.2.2 Source Graph: Self-supervised Node Embedding

Both topological and non-topological knowledge of the entity set $\mathcal{V}$ could be learnt from the source graph. The source graph learns node embedding with a self-supervised link prediction task to ensure no additional information is needed except the source graph itself. Concretely, for $\mathcal{G}_s = (\mathcal{V}, A_s, X, r_s)$, we input the graph into a GNN-based model (GNNs), followed by an edge-level link-prediction head to learn the node embedding of the graph. The GNNs consist of one or more GNN layers, such as GCN, GraphSAGE, GAT and MAGCN. The output node embeddings of $\mathcal{V}$ could be expressed as (6.17).

$$H_{(s,\mathcal{V})} = f_{\text{GNNs}}(A_s, X, \mathcal{P}) \tag{6.17}$$

where f_{GNNs} represents the forward propagation mapping function of the applied GNNs, $\mathcal{P}$ is the involved trainable parameters.

For $\forall\ v \in \mathcal{V}$, its node embedding $h_{(s,v)}$ could be found in $H_{(s,\mathcal{V})}$ with its index in $\mathcal{V}$. Correspondingly, the link prediction result of edge $u \to v$ with nodes u and v is shown as (6.18).

$$\widehat{y}_{(s,uv)} = \text{Head}_{\text{edge}}\left(h_{(s,u)}, h_{(s,v)}\right), (u, v \in \mathcal{V}), \tag{6.18}$$

where $\text{Head}_{\text{edge}}$ could be any edge mapping function.

To train the parameters $\mathcal{P}$ of GNNs, the self-supervised node embedding process is implemented as follows. We randomly split all the edges in the source graph into a training set and a validation set as positive samples ($y_{(s,uv)=1}$). Then randomly sample some non-existing edges in the source graph as the corresponding negative samples ($y_{(s,uv)=0}$). Finally, the CE loss function is applied to train the GNNs parameters $\mathcal{P}$ on the training set, and the hyper-parameters of GNNs are optimized on the validation set.

6.3.2.3 Target Graph: Graph Knowledge Transfer Learning

The target graph transfers the graph knowledge learnt from the source graph as its node embeddings directly, as shown in (6.19).

$$H_{(t,\mathcal{V})} = H_{(s,\mathcal{V})} \tag{6.19}$$

where $H_{(s,\mathcal{V})}$ is the node embedding matrix of $\mathcal{V}$ learnt from source graph. The logic behind equation (6.19) is that the node set $\mathcal{V}$ in the target graph is exactly the same as the source graph. The node set $\mathcal{V}$ of both the source graph and the target graph comes from the same KG and presents the same group of entities in a physical world. Therefore, the entity embeddings are transferable between the source and target graphs.

GKTL can tackle the sparsity problem of the target graph by learning graph knowledge from a much dense source graph. The time difference problem can be eliminated by deliberately selecting the source graph.

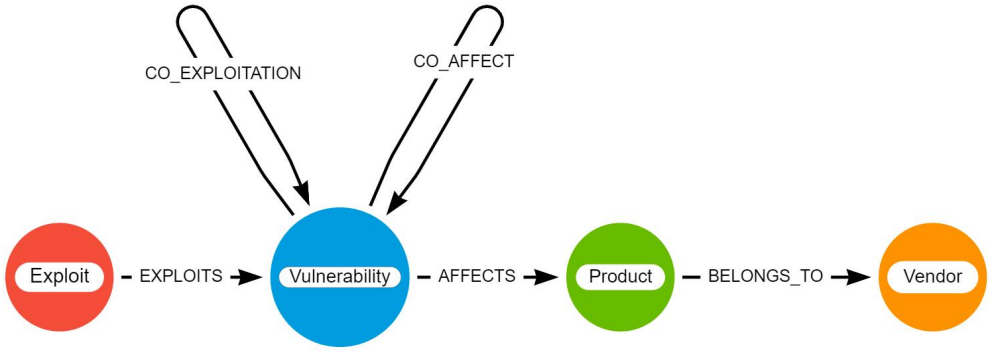


Fig. 6.5. Visualized schema of the cybersecurity knowledge graph.

6.4 Experiments: Co-exploitation Behaviour Discovery

6.4.1 Dataset Introduction

We built a KG using vulnerabilities and exploits information between 1995 to 2021 collected from CVE Details[1], National Vulnerability Database (NVD)[2] and the exploit database[3]. Fig. 6.5 shows the visualized schema of this KG. The KG is implemented in Neo4j [4] graph database platform. Python is the main programming language, and Cypher is also used to manipulate KG. We adopt 'py2neo' [5] to connect the Neo4j graph database with Python applications.

For each 'Vulnerability' entity, two modality features are involved. One is the textual modality, which contains the description of a vulnerability given by cyber-experts when published. Following the same preprocessing process of [40, 41], we apply a pre-trained BERT model to embed the textual modality into a 768-dimensional feature then reduce the dimension to 20 with principal component analysis. Another is the tabular modality, consisting of 8 numerical features, 10 boolean features and 16 category features.

[1]https://www.cvedetails.com/
[2]https://nvd.nist.gov/
[3]https://www.exploit-db.com/
[4]https://neo4j.com/
[5]https://py2neo.org/2021.1/

Table 6.1: Basic statistical information of the dataset

Stats.	Original knowledge graph	Source graph	Target graph
Entity types	Exploit, Vulnerablity, Product, Vendor	Vulnerablity	Vulnerablity
Relationship types	EXPLOITS, AFFECTS, BELONGS_TO, CO_EXPLOITATION, CO_AFFECT	CO_AFFECT	CO_EXPLOITATION
No. of nodes	256,971	6,090	6,090
No. of edges	288,956	30,416	6,880
No. of isolated nodes	0	712	1,833

For co-exploitation behaviour discovery, the co-exploitation subgraph is the target graph. We randomly add some isolated 'Vulnerability' nodes into the target graph to simulate the real-world situation more realistically. Following the two restrictions listed in Section 6.3.2.1, we set the much dense co-affect subgraph as the source graph. Table 6.1 shows the basic statistical information of the original knowledge graph and extracted subgraphs.

information are disclosed chronologically. To avoid information leakage caused by using future co-exploitation events to deduce past events, we split the edges in the target graph by the co-exploitation time. Fig. 6.6 shows the yearly distribution of the number of co-exploitation events. Accordingly, we set co-exploitation behaviour before 2015 as the positive samples in the training set, which contains 5349 'CO_EXPLOITATION' edges in total. The rest 1531 co-exploitation events that happened in 2015 or later are included in the test set as the positive samples. Negative samples are randomly sampled from the entities set. For the source graph splitting, the guideline is to make sure the 'Vulnerability' entity set is always the same as the target graph.

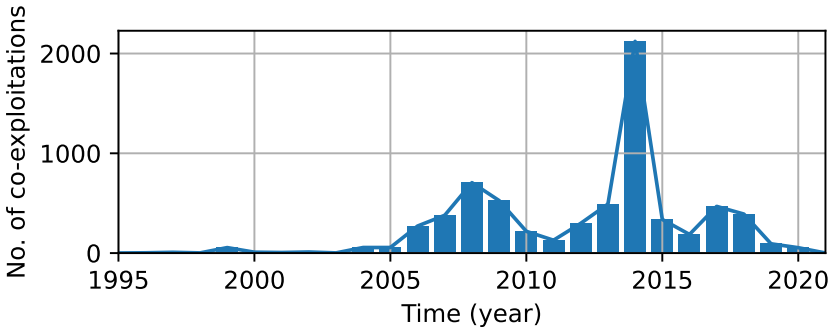


Fig. 6.6. The yearly distribution of number of co-exploitation behaviours

6.4.2 Co-exploitation Discovery Performance

To verify co-exploitation discovery performance when applying MAGCN and GKTL, we first train the source graph via the 'CO_AFFECT' edge prediction task to get the embeddings of the entire 'Vulnerability' entity set. The parameters of adopted GNNs in the source graph are learnt from the source graph training set. Since the 'CO_AFFECT' edge information is available once the corresponding 'Vulnerability' entity is added in the cybersecurity KG, the learnt GNNs can be generalized to the entire 'Vulnerability' entity set of the source graph. Therefore, the entire 'Vulnerability' entity embeddings $H_{(s,\mathcal{V})}$ could be calculated with Equation (6.17). Then, take $H_{(s,\mathcal{V})}$ as the node embeddings of the target graph $H_{(t,\mathcal{V})}$ and add a binary classifier as the prediction head to predict the edge-level co-exploitation discovery task, as formulated in Equation (6.18).

To demonstrate the performance of the proposed MAGCN module, we separately apply MAGCN, GCN [127], GraphSAGE [128], EdgeGCN [130] and GINGCN [131] as the GNNs layer adopted in the source graph self-supervised node embedding process. Pytorch and Deel Graph Library (DGL) [1] are used for implementation. In the process of graph knowledge transfer learning, we apply four widely used classifiers as the edge-level prediction head, namely, multilayer perceptron (MLP), random forest (RF), Support Vector Machine (SVM) and logistic regression (LR). These classifiers are implemented with the sklearn library. In this chapter, a 2-layer MAGCN is adopted, within which 'max' and 'mean'

[1] https://www.dgl.ai/

are adopted massage functions; 'e_mul_u' and 'u_sub_v' are adopted reduction functions; concatenation function is adopted as the reduce function and fusion function.

Table 6.2 shows the final co-exploitation discovery results on the test set of the target graph. To reflect performance on both positive and negative samples, we take the F1 score as the overall performance. Table 6.2 shows the proposed MAGCN achieves the best F1 score with all classifiers. The last column, ΔF1 shows the percentage improvement of MAGCN on the F1 score compared with the corresponding GNN module, ranging from 0.98% to 19.13%.

6.4.3 GKTL Performance Analysis

Table 6.3 shows the performance comparison with or without GKTL as indicated by the first column. When GKTL is not applied, the 'Vulnerability' entity embedding matrix $H_{(t,\mathcal{V})}$ is learnt from the sparse target graph itself. The edge-level prediction head used in Table 6.3 is MLP. The last column, ΔF1 shows the percentage improvement on the F1 score when applying GKTL. Table 6.3 shows GKTL is effective for all GNN modules. Furthermore, when without applying the GKTL strategy, MAGCN also achieves the best F1 score at 74.60% compared with other GNN modules.

6.4.4 Discussion

We briefly discuss how the proposed MAGCN and GKTL tackle the challenges mentioned in Section 6.1.

(i) MAGCN is proposed to solve the multi-modality problem. Table 6.2 shows MAGCN is superior to other GNN modules. The fundamental reason is that other GNN modules concatenate different modalities before inputting them to the GNN model and treat the concatenated features as the same modality. However, MAGCN inputs different modalities into different channels and therefore can flexibly choose the most suitable massage function, reduce function and update function for each modality. Consequently, MAGCN needs to do more work

Table 6.2: Co-exploitation discovery performance

Classifier	GNNs	Acc(%)	Pre(%)	Recall (%)	F1(%)	ΔF1
MLP	**MAGCN**	81.58	83.32	81.58	**81.34**	
MLP	GCN	79.07	79.67	79.07	78.96	↑3.01%
MLP	GraphSAGE	79.95	81.98	79.95	79.62	↑2.16%
MLP	EdgeGCN	78.31	80.26	78.31	77.96	↑4.34%
MLP	GINGCN	80.60	80.95	80.60	80.55	↑0.98%
RF	**MAGCN**	80.05	83.65	80.05	**79.50**	
RF	GCN	77.43	80.47	77.43	76.86	↑3.44%
RF	GraphSAGE	76.42	80.37	76.42	75.63	↑5.12%
RF	EdgeGCN	78.31	81.58	78.31	77.74	↑2.26%
RF	GINGCN	79.23	81.08	79.23	78.91	↑0.75%
SVM	**MAGCN**	78.22	78.22	78.22	**78.22**	
SVM	GCN	73.12	73.24	73.12	73.09	↑7.02%
SVM	GraphSAGE	75.83	76.39	75.83	75.71	↑3.32%
SVM	EdgeGCN	68.35	69.36	68.35	67.94	↑15.13%
SVM	GINGCN	73.87	73.91	73.87	73.86	↑5.90%
LR	**MAGCN**	77.56	78.72	77.56	**77.34**	
LR	GCN	65.35	66.11	65.35	64.93	↑19.13%
LR	GraphSAGE	73.97	76.20	73.97	73.40	↑5.37%
LR	EdgeGCN	68.81	70.50	68.81	68.16	↑13.47%
LR	GINGCN	72.24	73.59	72.24	71.84	↑7.66%

on selecting those essential functions. MAGCN treats these functions as hyperparameters, which can be selected based on expert experience, conventional practice or general hyperparameter optimization algorithms, such as grid search and random.

(ii) GKTL is proposed to solve the graph sparsity problem. The prediction task-related target graphs are usually relatively sparse because of their incompleteness and unavailability. However, we can learn from other more dense source graphs as long as the source and target graphs share the same entity set. As shown in Table 6.1, it is evident that the source graph (with 30,416 edges and 712 iso-

Table 6.3: GKTL influence on co-exploitation discovery performance

GKTL	GNNs	Acc(%)	Pre(%)	Recall (%)	F1(%)	ΔF1
N	MAGCN	75.44	79.34	75.44	74.60	
Y	MAGCN	81.58	83.32	81.58	81.34	↑9.03%
N	GCN	59.01	59.34	59.01	58.65	
Y	GCN	79.07	79.67	79.07	78.96	↑34.62%
N	GraphSAGE	50.65	65.79	50.65	35.10	
Y	GraphSAGE	79.95	81.98	79.95	79.62	↑126.84%
N	EdgeGCN	70.87	72.58	70.87	70.30	
Y	EdgeGCN	78.31	80.26	78.31	77.96	↑10.90%
N	GINGCN	65.97	68.40	65.97	64.81	
Y	GINGCN	80.60	80.95	80.60	80.55	↑24.29%

lated nodes) is about 4.42 times denser than the target graph (with 6,880 edges and 1,833 isolated nodes). It's straightforward that more knowledge could be learnt from more dense graphs. Therefore, GKTL can improve the performance of the target graph, no matter what kind of GNN module is applied.

(iii) GKTL can also tackle the time difference problem by extracting the information with the same available time into the same sub-graph. For example, the 'AFFECTS' and 'CO_AFFECT' information is available when a vulnerability is officially published. However, the 'EXPLOITS' and 'CO_EXPLOITATION' information may be known several months or years later after a vulnerability is published. Therefore, a time difference exists between the co-affect information and the co-exploitation information for the same vulnerability. GKTL avoids dealing with the graphs existing time differences by extracting and learning from time-consistent sub-graphs.

6.5 Conclusion

Accurate and timely co-exploitation discovery is of importance for cybersecurity experts to effectively identify and remediate sophisticated cyber attacks with

chained vulnerabilities. This chapter formulates the co-exploitation discovery as a link prediction problem in KG. To solve the existing challenges and boost the performance of co-exploitation discovery, this chapter proposed a general MAGCN module for graph node embedding and representation and a GKTL strategy for graph knowledge transfer learning. The effectiveness of both MAGCN and GKTL are verified on a real-world cybersecurity knowledge graph.

Chapter 7

Conclusion

This chapter summarizes the main work of the thesis in Section 7.1 and points out some promising research topics for future work in Section 7.2.

7.1 Summary

Rigorous vulnerability evaluation and assessment empowers organisations to make informed and knowledge-powered risk management decisions. The research of this thesis focuses on establishing state-of-the-art vulnerability exploitability prediction models and risk assessment frameworks, based on the latest deep learning and knowledge graph techniques. Specifically, three tasks and the corresponding solutions are presented in this thesis.

7.1.1 Vulnerability Exploitability Prediction

The exploitability of a vulnerability indicates if a vulnerability will be exploited or not. Considering the inaccuracy of the CVSS Exploitability score calculated by Equation (2.1) and the poor performance of traditional machine learning algorithm-based exploitability prediction models, the first research aim of this thesis is to develop a high-performance exploitability prediction strategy.

Chapter 3 details the effort of improving exploitability prediction performance under an offline setting. Specifically, a binary classification framework named ExBERT was proposed to predict the exploitability of vulnerabilities based on their descriptions only. ExBERT applies the transfer learning technique and fine-tunes a widely used pre-trained NLP model, BERT, on the corpus consisting of vulnerability descriptions to extract domain-specific semantic features. The extracted semantic features are fed into a pooling layer and an LSTM classification layer for the final decision-making. The experiments showed that ExBERT achieved state-of-the-art performance on a real-world dataset.

Chapter 4 is an extension work on exploitability prediction. Compared with Chapter 3, this chapter focuses on the concept drift and dynamic class imbalance problem existing in the exploitability prediction problem under a real-time online learning setting and proposed an improved consecutive batch learning framework named RDCAL. Instead of merely extracting semantic features from vulnerability description, RDCAL also extracted tabular features from relational databases like NVD. The experiment results conducted on real-world vulnerabilities collected between 1988 and 2020 show that RDCAL is effective in improving the online exploitability prediction performance of a variety of classifiers, including neural networks, SVM, HoeffdingTree and logistic regression by over 3%.

7.1.2 Vulnerability Exploitation Time Prediction

To provide more fine-grained results for vulnerability evaluation, this thesis further formulates exploitation time prediction as a multiclass classification problem.

Chapter 5 explored the vulnerability exploitation time prediction problem. A generalized consecutive batch learning framework was adopted to predict the probable exploitation time period of vulnerabilities. Within the framework, SWIF was designed as an index to reflex the real-time multiclass imbalanced status, and an ASWWL algorithm was proposed to tackle the general dynamic multiclass imbalanced problems existing in many real-world applications, including the exploitation time prediction problem. The experiment results demonstrate that the ASWWL algorithm can significantly enhance the performance of the minority classes without compromising the performance of the majority class.

Furthermore, the consecutive batch learning framework with the ASWWL algorithm achieved the most robust and state-of-the-art performance compared with the other five consecutive batch learning algorithms.

7.1.3 Vulnerability Co-exploitation Behaviour Discovery

Complex connections exist between vulnerabilities and exploits. For example, a known exploit can take advantage of several different vulnerabilities. If the co-exploitation behaviour between different vulnerabilities can be identified, decision-makers can have a wider and deeper perspective for vulnerability assessment than considering a single vulnerability itself.

Chapter 6 formulated the vulnerability co-exploitation behaviour discovery problem as a link prediction problem. A MAGCN module was designed to solve the multimodality information fusion and feature extraction problem. Furthermore, the GKTL strategy was designed to solve the graph sparsity and time difference problem identified in the co-exploitation behaviour discovery problem. The performance of MAGCN and GKTL was demonstrated by the experiments conducted in a real-world cybersecurity-domain specific knowledge graph consisting of co-exploitation incidents between 1995 and 2021.

7.1.4 Contributions to Vulnerability Management

A typical vulnerability management process can be broken down into four sequential steps, i.e., identifying vulnerabilities, evaluating vulnerabilities, treating vulnerabilities and reporting vulnerabilities. The results of this thesis can help automate vulnerability management in the step of evaluating vulnerabilities. Specifically, the online exploitability prediction framework proposed in Chapter 4 can be deployed to evaluate the exploitability of identified vulnerabilities. Within the framework, ExBERT proposed in Chapter 3 can help improve the accuracy of exploitability prediction by extracting high-level semantic features from vulnerability descriptions. Once a vulnerability is predicted to be exploitable, the algorithms presented in Chapter 5 can be used to help infer the specific time

frame in which a vulnerability would be exploited. Then, the system can recommend the most urgent vulnerabilities to the next step, treating vulnerabilities. Furthermore, the algorithms proposed in Chapter 6 can prioritise vulnerabilities by identifying their possible ci-exploitation behaviours.

To summarise, the results of this thesis can be adopted to improve the decision making of evaluating vulnerabilities from the perspective of exploitability prediction, exploitation time prediction and co-exploitation behaviour discovery. A better vulnerability evaluation and assessment is the basis of allocating budget and resources effectively and efficiently and thus achieving better risk management and mitigation in modern information systems.

7.2 Future Work

The research works of this thesis provided some feasible solutions for vulnerability exploitability prediction and analysis. However, there are still many unsolved challenging problems. This section lists some promising directions for future work in improving vulnerability assessment and management.

(1) Combine the exploitability prediction result with other vulnerability assessment metrics to form a more comprehensive vulnerability risk evaluation model.

Although chapters 3 and 4 provided feasible solutions for both offline and online exploitability prediction, there is still no comprehensive vulnerability risk evaluation model. In addition to exploitability, the risk level of a vulnerability is affected by many other aspects, such as the affected number of devices and users, the affected business process and the cost comparison between exploitation and remediation. CVSS is an example of a comprehensive vulnerability risk evaluation model. However, its effectiveness is far from satisfactory. More accurate availability prediction is undoubtedly conducive to vulnerability risk assessment, but how to combine the exploitability prediction results with other influencing factors to form a realistic and effective vulnerability risk assessment model will be a challenge for a long time in the future.

(2) Explore the exploitation time prediction with much finer granularity.

Chapter 5 divided exploitation time into three classes, Neg, ZeroDay and Pos, based on the time differences between the date of a vulnerability being exploited and being published. Although the prediction results given in Chapter 5 are more detailed than exploitability prediction, a finer-granular exploitation time prediction would be more useful in practice, especially for Pos vulnerabilities. For example, the predicted exploitation time period can be yearly, monthly, weekly or even daily, which makes it a regression problem. The main challenge of finer granularity comes from data deficiencies and data imbalance within each granularity. With the increase in available vulnerabilities and exploit data and the development of unsupervised learning techniques, we may have good solutions for this problem in the future.

(3) Construct cybersecurity-domain specific knowledge graph and explore more knowledge graph powered vulnerability intelligence applications.

Chapter 6 is an example of leveraging knowledge graphs and graph theory to solve a specific problem in vulnerability assessment and risk evaluation. In Chapter 6, the major source of vulnerabilities and exploits comes from existing well-organised databases, such as NVD, EDB and CVE Details. However, there are scalable vulnerability raw data from multimodal information sources, such as social media, software vendors, technical forums. This information can be used to build a large-scale open-source cybersecurity-domain specific knowledge graph. Based on the constructed knowledge graph, more knowledge graph powered vulnerability intelligence applications can be implemented, including but not limited to subgraph matching to discover multi-stage and highly sophisticated cyberattack tactics and multi-hop question-and-answer systems, which can make highly specialised cyber knowledge more accessible.

List of References

[1] Y.-F. Ge, J. Cao, H. Wang, J. Yin, W.-J. Yu, Z.-H. Zhan, and J. Zhang, "A benefit-driven genetic algorithm for balancing privacy and utility in database fragmentation," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 771–776.

[2] Y.-F. Ge, M. Orlowska, J. Cao, H. Wang, and Y. Zhang, "Mdde: multitasking distributed differential evolution for privacy-preserving database fragmentation," *The VLDB Journal*, pp. 1–19, 2022.

[3] R. U. Rasool, U. Ashraf, K. Ahmed, H. Wang, W. Rafique, and Z. Anwar, "Cyberpulse: a machine learning based link flooding attack mitigation system for software defined networks," *IEEE Access*, vol. 7, pp. 34 885–34 899, 2019.

[4] L. Bilge and T. Dumitraş, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 833–844.

[5] Y.-F. Ge, J. Cao, H. Wang, Z. Chen, and Y. Zhang, "Set-based adaptive distributed differential evolution for anonymity-driven database fragmentation," *Data Science and Engineering*, vol. 6, no. 4, pp. 380–391, 2021.

[6] Y.-F. Ge, W.-J. Yu, J. Cao, H. Wang, Z.-H. Zhan, Y. Zhang, and J. Zhang, "Distributed memetic algorithm for outsourced database fragmentation," *IEEE Transactions on Cybernetics*, vol. 51, no. 10, pp. 4808–4821, 2020.

[7] J. Yin, M. Tang, J. Cao, and H. Wang, "Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description," *Knowledge-Based Systems*, p. 106529, 2020.

[8] J. Jacobs, S. Romanosky, I. Adjerid, and W. Baker, "Improving vulnerability remediation through better exploit prediction," *Journal of Cybersecurity*, vol. 6, no. 1, p. tyaa015, 2020.

[9] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 105–114.

[10] C. Sabottke, O. Suciu, and T. Dumitraș, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 1041–1056.

[11] N. Tavabi, P. Goyal, M. Almukaynizi, P. Shakarian, and K. Lerman, "Darkembed: Exploit prediction with neural language models," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018, pp. 7849–7854.

[12] S. Wang and X. Yao, "Multiclass imbalance problems: Analysis and potential solutions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 4, pp. 1119–1130, 2012.

[13] M. You, J. Yin, H. Wang, J. Cao, and Y. Miao, "A minority class boosted framework for adaptive access control decision-making," in *International Conference on Web Information Systems Engineering*. Springer, 2021, pp. 143–157.

[14] Y. Shen, T. Zhang, Y. Wang, H. Wang, and X. Jiang, "Microthings: A generic iot architecture for flexible data aggregation and scalable service cooperation," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 86–93, 2017.

[15] J. Yin, M. You, J. Cao, H. Wang, M. Tang, and Y.-F. Ge, "Data-driven hierarchical neural network modeling for high-pressure feedwater heater group," in *Australasian Database Conference*. Springer, 2020, pp. 225–233.

[16] J. Yin, J. Cao, S. Siuly, and H. Wang, "An integrated mci detection framework based on spectral-temporal analysis," *International Journal of Automation and Computing*, vol. 16, no. 6, pp. 786–799, 2019.

[17] W. Wang, W. Wang, and J. Yin, "A bilateral filtering based ringing elimination approach for motion-blurred restoration image," *Current Optics and Photonics*, vol. 4, no. 3, pp. 200–209, 2020.

[18] G. Hassan, N. El-Bendary, A. E. Hassanien, A. Fahmy, V. Snasel *et al.*, "Retinal blood vessel segmentation approach based on mathematical morphology," *Procedia Computer Science*, vol. 65, pp. 612–622, 2015.

[19] J. Yin, M. Tang, J. Cao, H. Wang, and M. You, "A real-time dynamic concept adaptive learning algorithm for exploitability prediction," *Neurocomputing*, vol. 472, pp. 252–265, 2022.

[20] C. Tang and J. Yin, "A localization algorithm of weighted maximum likelihood estimation for wireless sensor network," *Journal of Information &Computational Science*, vol. 8, no. 16, pp. 4293–4300, 2011.

[21] M. Tang, M. Alazab, and Y. Luo, "Big data for cybersecurity: Vulnerability disclosure trends and dependencies," *IEEE Transactions on Big Data*, vol. 5, no. 3, pp. 317–329, 2019.

[22] R. Anderson and T. Moore, "The economics of information security," *science*, vol. 314, no. 5799, pp. 610–613, 2006.

[23] S. Özkan. (2021) Cve details, the ultimate security vulnerability database. [Online]. Available: https://www.cvedetails.com/

[24] T. M. Corporation. (2021) About cve - terminology. [Online]. Available: https://cve.mitre.org/about/terminology.html

[25] C. Tang, Y. Cheng, and J. Yin, "An optimized algorithm of grid calibration in wsn node deployment based on the energy consumption distribution model," *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, vol. 9, no. 4, pp. 1035–1042, 2012.

[26] L. Rosencrance. (2017) vulnerability disclosure. [Online]. Available: https://searchsecurity.techtarget.com/definition/vulnerability-disclosure

[27] H. Wang, J. Cao, and Y. Zhang, "Building access control policy model for privacy preserving and testing policy conflicting problems," in *Access Control Management in Cloud Environments*. Springer, 2020, pp. 225–247.

[28] A. Younis, Y. K. Malaiya, and I. Ray, "Assessing vulnerability exploitability risk using software properties," *Software Quality Journal*, vol. 24, no. 1, pp. 159–202, 2016.

[29] Wikipedia. (2021) Exploit (computer security). [Online]. Available: https://en.wikipedia.org/wiki/Exploit_(computer_security)

[30] F. of Incident Response and S. Teams. (2021) Common vulnerability scoring system v3.1: Specification document. [Online]. Available: https://www.first.org/cvss/v3.1/specification-document

[31] S. Frei, D. Schatzmann, B. Plattner, and B. Trammell, "Modeling the security ecosystem-the dynamics of (in) security," in *Economics of Information Security and Privacy*. Springer, 2010, pp. 79–106.

[32] H. Wang and L. Sun, "Trust-involved access control in collaborative open social networks," in *2010 fourth international conference on network and system security*. IEEE, 2010, pp. 239–246.

[33] B. L. Bullough, A. K. Yanchenko, C. L. Smith, and J. R. Zipkin, "Predicting exploitation of disclosed software vulnerabilities using open-source data," in *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics*, 2017, pp. 45–53.

[34] T. M. Corporation. (2021) The mission of the cve program is to identify, define, and catalog publicly disclosed cybersecurity vulnerabilities. [Online]. Available: https://cve.mitre.org/

[35] CVE, "Cve - frequently asked questions," https://cve.mitre.org/about/faqs.html#cve_entry_descriptions_created., accessed Oct 18, 2019.

[36] N. I. of Standards and U. D. o. C. Technology. (2021) General information. [Online]. Available: https://nvd.nist.gov/general

[37] N. I. of Standards and U. D. o. C. Technology. (2021) Nvd data feeds. [Online]. Available: https://nvd.nist.gov/vuln/data-feeds

[38] O. Security. (2021) Exploit database. [Online]. Available: https://www.exploit-db.com/

[39] L. Allodi and F. Massacci, "Comparing vulnerability severity and exploits using case-control studies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 1, p. 1, 2014.

[40] J. Yin, M. Tang, J. Cao, H. Wang, M. You, and Y. Lin, "Vulnerability exploitation time prediction: an integrated framework for dynamic imbalanced learning," *World Wide Web*, pp. 1–23, 2021.

[41] J. Yin, M. Tang, J. Cao, H. Wang, M. You, and Y. Lin, "Adaptive online learning for vulnerability exploitation time prediction," in *Web Information Systems Engineering – WISE 2020*. Springer, 2020, pp. 252–266.

[42] M. Tang, J. Yin, M. Alazab, J. Cao, and Y. Luo, "Modeling of extreme vulnerability disclosure in smart city industrial environments," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4150–4158, 2020.

[43] S. Frei, M. May, U. Fiedler, and B. Plattner, "Large-scale vulnerability analysis," in *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, 2006, pp. 131–138.

[44] L. Allodi, M. Cremonini, F. Massacci, and W. Shim, "The effect of security education and expertise on security assessments: The case of software vulnerabilities," *arXiv preprint arXiv:1808.06547*, 2018.

[45] M. Alazab and M. Tang, *Deep Learning Applications for Cyber Security*. Springer, 2019.

[46] M. Schiffman, A. Wright, D. Ahmad, and G. Eschelbeck, "The common vulnerability scoring system," *National Infrastructure Advisory Council, Vulnerability Disclosure Working Group, Vulnerability Scoring Subgroup*, 2004.

[47] F. of Incident Response and S. Teams. (2021) Forum of incident response and security teams (first). [Online]. Available: https://www.cybersecurityintelligence.com/forum-of-incident-response-and-security-teams-first-5620.html

[48] Oracle. (2021) Use of common vulnerability scoring system (cvss) by oracle. [Online]. Available: https://www.oracle.com/technetwork/topics/security/cvssscoringsystem-091884.html

[49] Y. Roumani, J. K. Nwankpa, and Y. F. Roumani, "Time series modeling of vulnerabilities," *Computers & Security*, vol. 51, pp. 32–40, 2015.

[50] M. Edkrantz and A. Said, "Predicting cyber vulnerability exploits with machine learning." in *SCAI*, 2015, pp. 48–57.

[51] J. Jacobs, S. Romanosky, B. Edwards, M. Roytman, and I. Adjerid, "Exploit prediction scoring system (epss)," *arXiv preprint arXiv:1908.04856*, 2019.

[52] J. Ma, G. Zhang, and J. Lu, "A state-based knowledge representation approach for information logical inconsistency detection in warning systems," *Knowledge-Based Systems*, vol. 23, no. 2, pp. 125–131, 2010.

[53] F. Liu, X. Zhou, J. Cao, Z. Wang, H. Wang, and Y. Zhang, "Arrhythmias classification by integrating stacked bidirectional lstm and two-dimensional cnn," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2019, pp. 136–149.

[54] J. Lu, Z. Yan, J. Han, and G. Zhang, "Data-driven decision-making (d 3 m): Framework, methodology, and directions," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 3, no. 4, pp. 286–296, 2019.

[55] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 125–136.

[56] E. R. Russo, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities," *Journal of Systems and Software*, 2019.

[57] Y. Zhang, H. Chen, J. Lu, and G. Zhang, "Detecting and predicting the topic change of knowledge-based systems: A topic-based bibliometric analysis from 1991 to 2016," *Knowledge-Based Systems*, vol. 133, pp. 255–268, 2017.

[58] W. Gao, M. Peng, H. Wang, Y. Zhang, Q. Xie, and G. Tian, "Incorporating word embeddings into topic modeling of short text," *Knowledge and Information Systems*, vol. 61, no. 2, pp. 1123–1145, 2019.

[59] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, and P. Shakarian, "Proactive identification of exploits in the wild through vulnerability mentions online," in *2017 International Conference on Cyber Conflict (CyCon US)*. IEEE, 2017, pp. 82–88.

[60] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[61] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[62] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G. Zhang, "Transfer learning using computational intelligence: A survey," *Knowledge-Based Systems*, vol. 80, pp. 14–23, 2015.

[63] Y.-F. Ge, M. Orlowska, J. Cao, H. Wang, and Y. Zhang, "Knowledge transfer-based distributed differential evolution for dynamic database fragmentation," *Knowledge-Based Systems*, vol. 229, p. 107325, 2021.

[64] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[65] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[66] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. (2018) Bert github implementation. [Online]. Available: https://github.com/google-research/bert

[67] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 19–27.

[68] Y. Hao, L. Dong, F. Wei, and K. Xu, "Visualizing and understanding the effectiveness of bert," *arXiv preprint arXiv:1908.05620*, 2019.

[69] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[70] H. Jiang, R. Zhou, L. Zhang, H. Wang, and Y. Zhang, "Sentence level topic models for associated topics extraction," *World Wide Web*, vol. 22, no. 6, pp. 2545–2560, 2019.

[71] M. Peng, J. Zhu, H. Wang, X. Li, Y. Zhang, X. Zhang, and G. Tian, "Mining event-oriented topics in microblog stream with unsupervised multi-view hierarchical embedding," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 12, no. 3, pp. 1–26, 2018.

[72] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[73] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[74] F. Khalil, H. Wang, and J. Li, "Integrating markov model with clustering for predicting web page accesses," in *Proceeding of the 13th Australasian world wide web conference (AusWeb07)*. AusWeb, 2007, pp. 63–74.

[75] J. Huang, M. Peng, and H. Wang, "Topic detection from large scale of microblog stream with high utility pattern clustering," in *Proceedings of the 8th Workshop on Ph. D. Workshop in Information and Knowledge Management*, 2015, pp. 3–10.

[76] k.-t. François Chollet. (2021) Keras: the python deep learning api. [Online]. Available: https://keras.io/

[77] H. Wang, Y. Wang, T. Taleb, and X. Jiang, "Special issue on security and privacy in network computing," *World Wide Web*, vol. 23, no. 2, pp. 951–957, 2020.

[78] J. Ruohonen, "A look at the time delays in cvss vulnerability scoring," *Applied Computing and Informatics*, vol. 15, no. 2, pp. 129–135, 2019.

[79] Y. Dong, Y. Fu, L. Wang, Y. Chen, Y. Dong, and J. Li, "A sentiment analysis method of capsule network based on bilstm," *IEEE Access*, vol. 8, pp. 37 014–37 020, 2020.

[80] L. Wang, J. Ren, B. Xu, J. Li, W. Luo, and F. Xia, "Model: Motif-based deep feature learning for link prediction," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 2, pp. 503–516, 2020.

[81] S. Sun, N. Akhtar, H. Song, C. Zhang, J. Li, and A. Mian, "Benchmark data and method for real-time people counting in cluttered scenes using depth sensors," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3599–3612, 2019.

[82] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[83] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera, "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing*, vol. 239, pp. 39–57, 2017.

[84] M. Cejnek and I. Bukovsky, "Concept drift robust adaptive novelty detection for data streams," *Neurocomputing*, vol. 309, pp. 46–53, 2018.

[85] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.

[86] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine learning*, vol. 90, no. 3, pp. 317–346, 2013.

[87] C. Raab, M. Heusinger, and F.-M. Schleif, "Reactive soft prototype computing for concept drift streams," *Neurocomputing*, 2020.

[88] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Brazilian symposium on artificial intelligence*. Springer, 2004, pp. 286–295.

[89] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Fourth international workshop on knowledge discovery from data streams*, vol. 6, 2006, pp. 77–86.

[90] I. Frías-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2014.

[91] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *International Symposium on Intelligent Data Analysis*. Springer, 2009, pp. 249–260.

[92] S. multiflow developers. (2021) scikit-multiflow. [Online]. Available: https://github.com/scikit-multiflow/scikit-multiflow

[93] V. Losing, B. Hammer, and H. Wersing, "Knn classifier with self adjusting memory for heterogeneous concept drift," in *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 2016, pp. 291–300.

[94] P. Kosina and J. Gama, "Very fast decision rules for classification in data streams," *Data Mining and Knowledge Discovery*, vol. 29, no. 1, pp. 168–202, 2015.

[95] S. Ren, B. Liao, W. Zhu, Z. Li, W. Liu, and K. Li, "The gradual resampling ensemble for mining imbalanced data streams with concept drift," *Neurocomputing*, vol. 286, pp. 150–166, 2018.

[96] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, no. Dec, pp. 2755–2790, 2007.

[97] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[98] K. Team. (2021) Dense layer. [Online]. Available: https://keras.io/api/layers/core_layers/dense/

[99] S. multiflow developers. (2021) Api reference. [Online]. Available: https://scikit-multiflow.readthedocs.io/en/stable/api/api.html

[100] scikit-learn developers. (2021) Api reference. [Online]. Available: https://scikit-learn.org/stable/modules/classes.html

[101] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 97–106.

[102] A. AlEroud and G. Karabatis, "A contextual anomaly detection approach to discover zero-day attacks," in *2012 International Conference on Cyber Security*. IEEE, 2012, pp. 40–45.

[103] H. Li, Y. Wang, H. Wang, and B. Zhou, "Multi-window based ensemble learning for classification of imbalanced streaming data," *World Wide Web*, vol. 20, no. 6, pp. 1507–1525, 2017.

[104] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *Journal of Machine Learning Research*, vol. 19, no. 72, pp. 1–5, 2018. [Online]. Available: http://jmlr.org/papers/v19/18-251.html

[105] M. M. Anwar, C. Liu, and J. Li, "Discovering and tracking query oriented active online social groups in dynamic information network," *World Wide Web*, vol. 22, no. 4, pp. 1819–1854, 2019.

[106] S. Wang, L. L. Minku, and X. Yao, "A learning framework for online class imbalance learning," in *2013 IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL)*. IEEE, 2013, pp. 36–45.

[107] S. Wang, L. L. Minku, and X. Yao, "Dealing with multiple classes in online class imbalance learning." in *IJCAI*, 2016, pp. 2118–2124.

[108] N. I. of Standards and U. D. o. C. Technology. (2021) National vulnerability database. [Online]. Available: https://nvd.nist.gov/

[109] L. Chen, S. Shang, B. Yao, and K. Zheng, "Spatio-temporal top-k term search over sliding window," *World Wide Web*, vol. 22, no. 5, pp. 1953–1970, 2019.

[110] X. Wang, S. Wang, Y. Xin, Y. Yang, J. Li, and X. Wang, "Distributed pregel-based provenance-aware regular path query processing on rdf knowledge graphs," *World Wide Web*, pp. 1–32, 2019.

[111] Y. Yang, Z. Guan, J. Li, J. Huang, and W. Zhao, "Interpretable and efficient heterogeneous graph convolutional network," *arXiv preprint arXiv:2005.13183*, 2020.

[112] X. Ning and J. Jiang, "Design, analysis and implementation of a security assessment/enhancement platform for cyber-physical systems," *IEEE Transactions on Industrial Informatics*, 2021.

[113] K. Cheng, L. Wang, Y. Shen, H. Wang, Y. Wang, X. Jiang, and H. Zhong, "Secure k k-nn query on encrypted cloud data with multiple keys," *IEEE Transactions on Big Data*, vol. 7, no. 4, pp. 689–702, 2017.

[114] G. George and S. M. Thampi, "Combinatorial analysis for securing iot-assisted industry 4.0 applications from vulnerability-based attacks," *IEEE Transactions on Industrial Informatics*, 2020.

[115] P. Radoglou-Grammatikis, K. Robolos, P. Sarigiannidis, V. Argyriou, T. Lagkas, A. Sarigiannidis, S. K. Goudos, and S. Wan, "Modelling, detecting and mitigating threats against industrial healthcare systems: a combined sdn and reinforcement learning approach," *IEEE Transactions on Industrial Informatics*, 2021.

[116] D. Technologies. Dell OpenManage Network Manager Security Vulnerabilities. [Online]. Available: https://www.dell.com/support/kbdoc/en-au/000138987/dell-openmanage-network-manager-security-vulnerabilities

[117] C. Details. Vulnerability Details : CVE-2018-15767. [Online]. Available: https://www.cvedetails.com/cve/CVE-2018-15767/

[118] C. Details. Vulnerability Details : CVE-2018-15768. [Online]. Available: https://www.cvedetails.com/cve/CVE-2018-15768/

[119] KoreLogic. Dell OpenManage Network Manager 6.2.0.51 SP3 - Multiple Vulnerabilities. [Online]. Available: https://www.exploit-db.com/exploits/45852

[120] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[121] J. Li and P. Willett, "Articlerank: a pagerank-based alternative to numbers of citations for analysing citation networks," in *Aslib Proceedings*. Emerald Group Publishing Limited, 2009.

[122] U. Brandes and C. Pich, "Centrality estimation in large networks," *International Journal of Bifurcation and Chaos*, vol. 17, no. 07, pp. 2303–2318, 2007.

[123] M. Marchiori and V. Latora, "Harmony in the small-world," *Physica A: Statistical Mechanics and its Applications*, vol. 285, no. 3-4, pp. 539–546, 2000.

[124] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[125] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[126] X. Liu, T. Murata, K.-S. Kim, C. Kotarasu, and C. Zhuang, "A general view for network embedding as matrix factorization," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 375–383.

[127] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[128] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[129] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[130] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[131] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.