A graph-based vulnerability detection method

Yevheniy Sierhieiev[1,*,†], Vadym Paiuk[1,†], Anatoliy Sachenko[2,†], Andrii Nicheporuk[1,†], Andrzej Kwiecien[3,†]

[1] *Khmelnytskyi National University, Institutska str., 11, Khmelnytskyi, 29016, Ukraine*
[2] *West Ukrainian National University, 11 Lvivska Str., Ternopil 46009, Ukraine*
[3] *Silesian University of Technology, Akademicka str., 2A, Gliwice, Poland*

Abstract

This paper presents a novel Graph-based Vulnerability Detection Method (GBVDM) designed to address the growing challenges in software security by leveraging graph theory to enhance the detection of vulnerabilities within software systems. This method constructs a dynamic dependency graph that captures the complex interactions between software components, enabling the identification of vulnerabilities through sophisticated analysis of these interactions. The GBVDM utilizes depth-first and breadth-first search algorithms to explore potential vulnerabilities, integrating machine learning techniques to improve the prediction and identification of new and existing security threats.

Through an in-depth examination of various vulnerability types, including SQL injections, Cross-Site Scripting (XSS), and buffer overflows, the paper evaluates the effectiveness of GBVDM against traditional vulnerability detection methods. The method's efficacy is demonstrated through experimental validation on software designed for Windows 10, highlighting its superior ability to detect vulnerabilities with higher accuracy and a lower rate of false positives compared to Static Application Security Testing (SAST) methods.

The GBVDM represents a significant advancement in cybersecurity, offering a comprehensive and efficient approach to vulnerability detection. This method identifies known vulnerabilities and predicts potential future threats, providing a valuable tool for developers and security analysts in the ongoing effort to secure software systems against increasingly sophisticated cyber attacks..

Keywords

vulnerabilies, cyber security, threat detection, cyber defense, xss, vulnerable detection, GBVDM

1. Introduction

Information technology has become an integral part of our lives in today's digital world. Therefore, it is crucial to ensure the security of software and the reliability of computer systems. The IT sector plays a significant role, not only in our daily lives but also in the functioning of critical infrastructure. As a result, data protection has become an essential issue. With the advancement of computer technology, cyber attackers have also become more sophisticated. This has led to researchers and developers exploring new methods of combating cyber threats.

The topic of detecting vulnerabilities in software is becoming increasingly important due to the rapid development of cyber threats. The need for both reactive actions to neutralize cyber-attacks and preventive measures to prevent them is crucial. Therefore, it's essential to

IntelITSIS'2024: 5th International Workshop on Intelligent Information Technologies and Systems of Information Security, March 28, 2024, Khmelnytskyi, Ukraine

* Corresponding author.
† These authors contributed equally.

ysierhieiev@gmail.com (Ye. Sierhieiev); vadympaiuk@gmail.com (V. Paiuk); as@wunu.edu.ua(A. Sachenko); andrey.nicheporuk@gmail.com (A. Nicheporuk); andrzej.kwiecien@polsl.pl (A. Kwiecien);
0009-0008-9877-9863 (Ye. Sierhieiev); 0000-0002-7253-893X (V. Paiuk); 0000-0002-0907-3682 (A. Sachenko); 0000-0002-7230-9475 (A. Nicheporuk); 0000-0003-1447-3303 (A. Kwiecien)

develop comprehensive methods that combine different approaches and technologies to ensure maximum protection efficiency. Research in this area can contribute greatly to the development of the industry, identifying new niches for the application of advanced technologies. For example, developing specialized tools to detect vulnerabilities in specific types of software or when using specific technologies can provide valuable solutions for protecting mission-critical systems.

The work aims to create a new method of detecting vulnerabilities, which will be more effective in detecting vulnerabilities in large software against the background of existing solutions.

The discovery of software vulnerabilities has a long history, dating back to the early days of computer systems. As protecting information has always been paramount, discovering these vulnerabilities initially relied on manual code analysis and security audits. However, these methods were time-consuming and required a deep understanding of the program's internal logic. Over time, it became clear that automated methods were necessary to protect against a wide range of threats. Current researchers, such as that of Valdés-Rodríguez et al., focus on integrating security practices into agile software development [1]. This emphasizes the importance of adapting to changing security requirements and developing a methodology that allows even inexperienced developers to create more secure applications. This approach signifies a change in emphasis from conventional techniques to more adaptable and flexible strategies as a part of the speedy advancement of technology and software. This is an essential step in guaranteeing reliability and security in the current digital world.

In the field of cyber security, identifying vulnerabilities in software is a key aspect of protecting information systems from growing cyber threats. Current research in this area opens up new opportunities for improving methods of analysis and identification of potential weak points.

Vulnerability refers to weak points in protection systems that can be exploited by threats due to mistakes, or imperfections in procedures, implementations, or projects. In simpler terms, vulnerabilities are any factors that make it possible for threats to succeed. Research shows that vulnerability is the primary cause of attacks. Threats are visible risks that can cause damage to a system. Vulnerable systems can be targeted by threats. Weaknesses in protection can be exploited by one or more threats, leading to unwanted incidents and unstable functioning of information system components.

The following types of activity risks are typical for innovative enterprises, which include companies in the IT sector:

- organizational (low qualification of project developers, delay in implementation of stages of its implementation);
- scientific and technical (wear and tear of technological equipment, lack of capacity reserves or typical design solutions);
- financial and economic (marketing, project financing risk, inflation, interest, tax and operational risks).

Weak points in information system protection can result from a variety of factors, ranging from employee negligence to malicious intruder actions. If a breach of information security occurs, the system will need to be restored, resulting in significant costs.

The presence of risk is the probability that certain undesirable events will occur, which can negatively affect the achievement of the goals of a particular business process. In particular, the functioning of the enterprise in the IT sector is related to innovative processes, development, and production of new products, works, and services. Innovative activity, striving for a competitive advantage forces the company to implement the latest achievements of science, new products, and technology, and a new system of labor and production management to maintain leading market positions, which is combined with numerous risks, the impact of which on the company's economic results is quite significant. In this regard, the timely, prompt, and correct assessment of the risks of a decrease or complete loss of information security is an urgent problem in the activities of any organization today. Information security, determining the level of security of the business environment, becomes an important aspect of general economic security in the activities of a modern company.

Today, when the number and complexity of cyberattacks are constantly growing, the issue of software security is becoming especially relevant. Malware capable of circumventing traditional defense mechanisms through the use of obfuscation and metamorphism[2] causes significant financial losses to companies and individual users. According to well-known antivirus companies, such as McAfee and ESET, worldwide losses from cybercrime are estimated at hundreds of billions of dollars annually, highlighting the critical need to develop and implement more effective protection methods and early detection of vulnerabilities. For example, a 2020 McAfee report indicates that cybercrime losses have reached about $1 trillion, which is about 1% of global GDP[3].

2. Classification of vulnerabilities in software

Various reasons can lead to vulnerabilities in software and digital services, including design errors, software flaws, weak passwords, incorrect configurations, and security attacks. It is crucial to identify these vulnerabilities to ensure the security of information systems. There are different categories of vulnerabilities that can be classified based on the type of modern information systems.

- Vulnerabilities in software (SQL injections, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Obtaining incorrect access, vulnerabilities in session management, code injection)
- Vulnerabilities at the network level (Insufficient access control, ARP poison, Man-in-the-Middle (MitM) attacks, Vulnerabilities in wireless networks, Denial of Service attacks (DoS), and Distributed Denial of Service (DDoS))
- Vulnerabilities at the level of the operating system (insufficient control of access to files and directories, vulnerabilities in services and utility programs, use of incorrect permission to execute programs)
- Vulnerabilities in cloud services (Insufficient data access control, data privacy issues in the cloud, attacks from cloud services, identity, and access management vulnerabilities)
- Social engineering (phishing, information leakage through social networks, attacks based on personnel information)
- Physical vulnerabilities (Unauthorized access to physical devices and server rooms, Decommissioning of equipment)
- Insufficient security of network devices (Vulnerabilities in routers and switches, insufficient control of access to network devices)
- Vulnerabilities in virtualization and containerization (Information leakage between virtual machines, insufficient control of access to containers)

Specific vulnerabilities depend on configuration, software used, and other factors. Ensuring system security requires constant monitoring and improvement of security measures.

Let's discuss these vulnerabilities in greater detail. For our method, we have identified only a few target vulnerabilities that we aim to detect:

1. SQL injection is a vulnerability that allows attackers to insert or "inject" arbitrary SQL code into a query executed by the database[4]. This can lead to unauthorized access to data, its deletion, modification, or execution of operations for which the attacker does not have permission. Common attacks: Obtaining a list of users or extracting data from other tables using login form injection and Deleting tables or databases by injecting malicious commands into text fields of web forms. Today, the methods of combating them are:

- Using parameterized queries (also known as prepared expressions), which effectively separates the SQL code from the input data, preventing it from being interpreted as part of an SQL command.
- Input data sanitization: Validating and sanitizing input data to remove or escape potentially harmful characters.
- Using ORM (Object-Relational Mapping) libraries: These libraries often automatically apply parameterized queries and other secure data access practices.

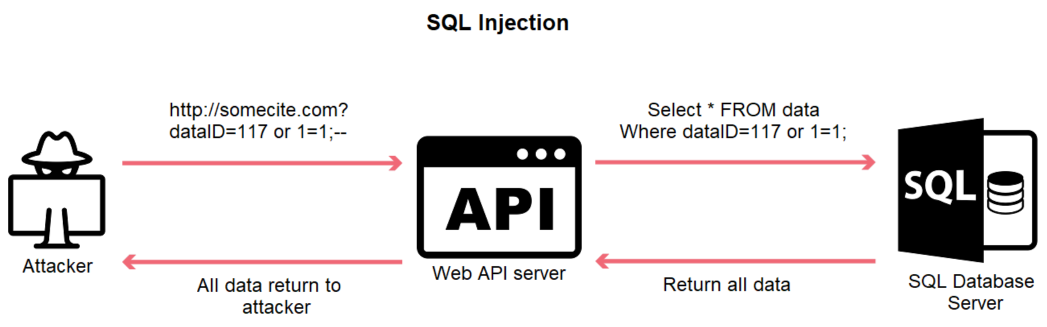

Figure 1: SQL Injection example.

2. Cross-Site Scripting (XSS) is a vulnerability that occurs when an attacker can inject a malicious script into a web page, which is then executed by other users. This can happen if a web application accepts input (for example, through URL parameters, forms, or cookies) and displays it improperly on the page without escaping or sanitizing it. XSS can be used to steal cookies, and session tokens, intercept personal data, and change the appearance of a web page[5]. The most common attack variant: injection through the URL parameter <script>alert('XSS');</script>, which displays a pop-up window for each user visiting the compromised page, or injection of JavaScript code that steals the user's cookies, through comments on the forum or in reviews. The following methods are used for prevention:

- Output Escaping: Ensure that any input-output to the page is properly escaped so that it is output as plain text and not as executable code.

- Using Content Security Policy (CSP): CSP allows websites to determine from which sources scripts can be executed, which helps prevent malicious scripts from being executed.
- Inbound Sanitation: Proactively inspect and sanitize inbound data to remove or block malicious code before it is processed or output.

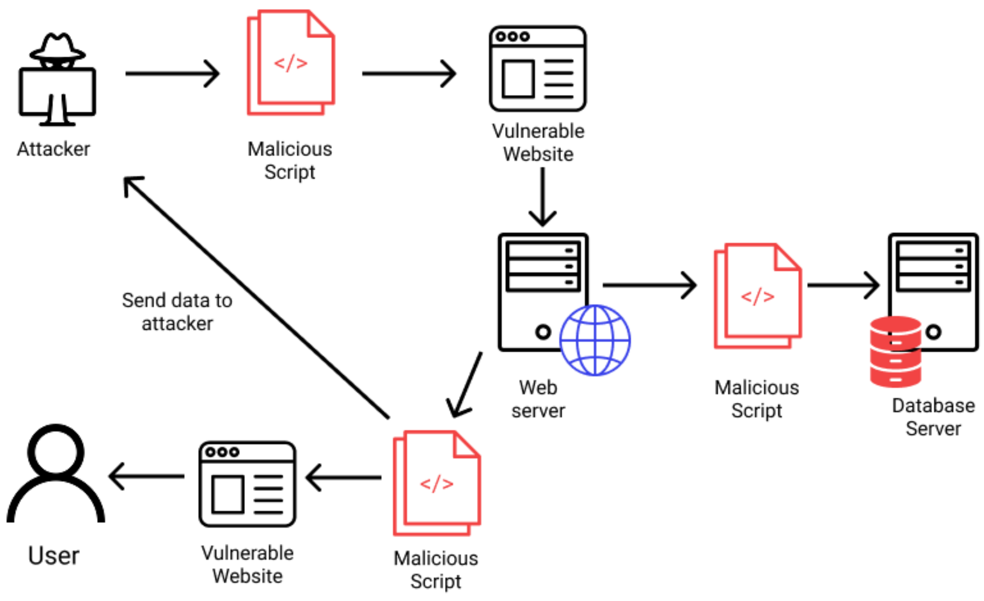

Figure 2: Cross-Site Scripting example.

3. Buffer overflow is a vulnerability that occurs when a program writes data to a buffer outside its limits. This can overwrite adjacent data or executable code, allowing attackers to execute malicious code[6].
4. Unsafe Deserialization - is a vulnerability that occurs when untrusted data is deserialized without proper validation or processing. This can lead to malicious code execution, application logic attacks, or data leakage[7].
5. Use of components with known vulnerabilities[8]: – Many programs include third-party components that may contain vulnerabilities. Using such components without updating to secure versions puts the security of the entire system at risk.
6. Insecure session management is a vulnerability in session management that can allow attackers to intercept or spoof session IDs, thereby gaining access to someone else's accounts[9].
7. Security Misconfiguration – Improper security settings can include leaving default settings, misconfiguring security headers, or weak password policies.

The study conducted by scholars Zhou, X., Verma, R.M.[10] highlights the significance of quantitative analysis in managing and evaluating software system reliability. The authors suggest the application of quantitative techniques to predict the number of vulnerabilities that may be discovered in the future, based on data on vulnerabilities found in some popular operating systems. This research helps to understand how managing and detecting software vulnerabilities can be improved, leading to better protection against potential threats. This approach stresses the importance of analytical methods in vulnerability detection and prediction, which can lead to the development of more effective cybersecurity strategies. The use of quantitative analysis enables a deeper comprehension of the dynamics of vulnerability detection, as well as the development of methods to minimize risks for software systems.

The increasing complexity of the software, on the one hand, contributes to the expansion of its functionality, but on the other hand, it increases the number of potential vulnerabilities. These vulnerabilities can be used by attackers to carry out cyberattacks that threaten the security of users and organizations. Cyber-attacks can take various forms, from stealing confidential information to providing unauthorized access to systems [11,12]. Such a situation requires careful analysis of existing protection methods and the development of new approaches capable of resisting current and potential threats.

3. Related works

Cybersecurity research plays a key role in the continuous development of vulnerability detection techniques. This helps identify and eliminate potential threats before they can cause harm. Innovative approaches, such as the use of machine learning, deep code analysis, and the extension of fuzzing capabilities, open new horizons for increasing the efficiency and accuracy of vulnerability detection.

Let's consider methods that can be used individually or in combination to ensure full detection of vulnerabilities in software and digital services.

Port and service scanning:

- Using tools to scan ports and identify active services on the system. This helps identify potentially vulnerable services.

Analysis of software vulnerabilities:

- Using automated tools to find vulnerabilities in software. This may include web application scanning, source code analysis, and other techniques.

Penetration testing:

- Ethical use of penetration testing to identify vulnerabilities in software. This allows security professionals to emulate attacks and identify system vulnerabilities.

Monitoring event logs:

- Analyze event logs and system logs to detect unusual activity or unauthorized access attempts.

Traffic analysis:

- Using tools to monitor traffic and detect anomalies, such as bounce-based attacks or large requests.

Updates and Patches:

- Systematic updating of software and application of patches to eliminate vulnerabilities that were discovered and resolved by developers.

Configuration Audit:

- Checking system and application configuration for weaknesses in settings that could create potential vulnerabilities.

Social engineering:

- Identify human-related vulnerabilities through social engineering analysis and detection of phishing opportunities.

Special attention in modern research is paid to machine learning methods, which open up new perspectives in detecting and countering cyberattacks. The prospects for the use of machine learning and artificial intelligence in the field of vulnerability detection look optimistic, offering a path to more automated, accurate, and adaptive cybersecurity systems. The development of these technologies requires a constant dialogue between researchers, practitioners, and software developers, as well as a focus on the ethical aspects of the use of artificial intelligence in vulnerability detection. Further innovations and research in this area promise not only to improve the ability to resist current and future cyber threats but also to open new horizons for the protection of the digital world [13, 14].

According to scientist Khraisat A., detecting cyber intrusions accurately is a challenging task due to the continuous advancement of cyberattack methods [15]. The author highlights the importance of adjusting intrusion detection systems to evolving threats, which involves the incorporation of advanced technologies, especially machine learning, to enhance the precision and swiftness of response to cyber-attacks. Machine learning includes the process of extracting knowledge from large data sets. It uses a set of rules, methods, or complex "transformation functions" that can be applied to discover interesting patterns in data or to recognize or predict behaviour. Techniques such as clustering, neural networks, associative rules, decision trees, genetic algorithms, and nearest-neighbour methods have been applied to analyze intrusion data. Machine learning techniques can sometimes produce a high number of false positives or have low accuracy because of difficulties in updating information about new attacks. This issue becomes even more critical when attempting to detect zero-day attacks that have no previously known signatures or behaviour patterns. Some approaches, particularly ensemble methods, may require significant computing resources and time to train on large datasets, making them unsuitable for real-time use [12]. However, machine learning can help analyze large volumes of data and identify complex dependencies between program characteristics, which can ultimately lead to more effective detection of vulnerabilities [16].

However, according to studies, machine learning is not a panacea and requires further improvements, in particular in terms of adapting to new types of threats and reducing the number of false positives [17]. Despite the significant potential, the application of machine learning in cyber security is accompanied by certain challenges. One of the main ones is ensuring the quality and relevance of data for model training. Incorrectly selected or outdated data can lead to detection errors, increasing the risk of missing real threats or generating false alarms. Additionally, the complexity and "black box" nature of some machine learning algorithms complicate the interpretation of results and their integration into existing security systems [12, 15].

Automation of the vulnerability detection process has evolved with the advent of static code analysis tools that allow the identification of potential problems without the need to execute programs. These tools analyse code for known error patterns and unsafe programming practices, offering developers recommendations for their elimination [18].

Static analysis is a technique that helps to identify potential vulnerabilities in software before it is executed in a user environment. It involves analysing the code of the program to detect weaknesses and flaws that can be exploited by attackers. While static analysis is a powerful tool, it may not be able to detect all types of vulnerabilities, such as those that require code execution to be triggered, like some runtime vulnerabilities. Nonetheless, it remains an important method for improving the security and reliability of software. False positives are a common problem with static analysers. They can wrongly identify correct code as potentially vulnerable. Moreover, these analysers require a deep understanding of a program's source code and its dependencies, making them challenging for non-experts to use [12]. Meanwhile, dynamic analysis is being developed as an alternative approach. Unlike static analysis, dynamic analysis requires the execution of a program in a controlled environment to detect errors that occur only during runtime [19].

One of the turning points in the detection of vulnerabilities was the introduction of fuzzing, a method that involves the generation of a large number of unpredictable input data for a program to detect processing errors. A study by scientists Huang, Y., Wang, Z., Ou, H., Chi, Y. (2022) demonstrated the high effectiveness of fuzzing in the context of mobile office software on the Android platform, using multi-method approaches to generate test cases and analyse the results [20]. Fuzzing is a security detection technique that injects invalid or random data into a program and outputs behaviour that is not expected, thereby identifying program errors and potential vulnerabilities. The main idea of fuzzing is to generate data, where tests are carried out for failure of the source program, and also to choose the right tools to monitor the process. Fuzzing can generate a large number of false positives because it is not always possible to determine with certainty whether the detected program behavior is a valid vulnerability. This method requires significant resources to generate and process test cases, especially when using large-scale tests with a large amount of input data, and may not detect vulnerabilities that are activated only under very specific conditions or require complex user interaction [12]. Fuzzing techniques continue to evolve, including the development of intelligent systems capable of adapting data generation strategies to improve detection performance. The proposed fuzzing-based vulnerability detection system demonstrates its viability and can provide support to developers to improve software security.

Let's review and analyse Web Application Scanners used to test web applications for vulnerabilities through penetration testing. These tools analyse a web application by generating malicious input and evaluating the application's response. However, automated web application scanners may miss vulnerabilities that require manual inspection or understanding of the context of the application's business logic. Additionally, they may generate false positive results due to insufficient understanding of the specifics of the application, which can lead to consuming resources for checking "vulnerabilities" unnecessarily. Moreover, web scanners can have limitations in evaluating complex web applications that use modern frameworks and dynamic content.

The scientific community, combining the efforts of researchers, practitioners, and developers, contributes to the creation of knowledge that can be applied to strengthen the security of information systems at all levels. This, in turn, supports sustainable growth and innovation in IT, providing robust protection against ever-evolving cyber threats.

Despite significant progress in the development of vulnerability detection methods, current approaches still face challenges that limit their effectiveness. Static analysis, for example, often leads to false positive results, which complicates the interpretation of its findings and requires further improvement to increase accuracy [12]. Dynamic analysis, on the other hand, may not detect vulnerabilities that only appear under specific conditions or with certain inputs, indicating the need for more flexible and adaptive solutions.

A review of current research in the field of vulnerability detection indicates a constant search for a balance between detection efficiency, accuracy of results, and requirements for computing resources. The development of integrated approaches combining various methods and technologies, including machine learning and automated solutions, opens up new perspectives for increasing the efficiency of the processes of detection and analysis of vulnerabilities in software [21, 22].

One of the main obstacles to the effectiveness of existing methods is their inability to quickly adapt to new and evolving vulnerabilities. Attackers are constantly improving their methods, using the latest technologies to bypass defense mechanisms. This requires detection methods to constantly update and expand knowledge bases to identify new threats. However, the updating process often lags behind the speed of development of cyber threats, which calls into question the effectiveness of these methods against the latest attacks [11].

Current methods of detecting vulnerabilities in software often face some challenges that limit their effectiveness. Static analysis, although it provides extensive code coverage, can generate a large number of false positives, which complicates the process of identifying true vulnerabilities. The problem of false-positive and false-negative results remains relevant for many detection methods. The high number of false positives in static analysis methods requires security professionals to spend a lot of effort to check each such case, which reduces their productivity and can distract from the detection of real vulnerabilities. False negatives, in turn, create an illusion of security, hiding existing vulnerabilities from analysts. On the other hand, dynamic analysis requires the program to be executed in a controlled environment, which may not reveal vulnerabilities that only manifest under certain conditions.

After considering the problems stated above, it is evident that the industry needs new and innovative approaches and methodologies to effectively adapt to the evolving landscape of cyber threats. This will ensure timely and accurate detection of vulnerabilities. Developing such methods requires a deep understanding of both the technical aspects of software and the dynamics of cyber threats, as well as the integration of cutting-edge research in machine learning, computational logic, and data analytics.

In addition, software security is a problem not only with the variety and complexity of attacks but also with the methods that attackers use to hide their actions. For example, the use of code obfuscation and metamorphism techniques makes it difficult to detect malicious programs and malicious code[2, 23].

Code obfuscation is the process of intentionally complicating a program's code to make it difficult to analyze, understand, and modify without changing its primary functionality. This may include replacing variable names with obscure symbols, changing the program's control flow, and other transformations that make the code less understandable [18, 19].

Metamorphism, on the other hand, refers to a technique that allows malicious code to change its structure at each execution without changing its underlying behavior. This makes it difficult to detect by anti-virus programs, as malware signatures are constantly changing[24-26].

Existing defenses against such threats include a variety of static and dynamic analysis tools, intrusion detection systems, and antivirus programs that use heuristic analysis, behavioral analysis, and signature patterns to identify potentially malicious activities. However, the effectiveness of these tools can be limited due to the use of the mentioned concealment techniques[26, 27].

4. A graph-based vulnerability detection method

After analyzing different software vulnerability detection methods, their advantages and disadvantages, we came up with a new method called the "Graph-based Vulnerability Detection Method". This method can be integrated with existing technologies discussed in the article and can provide a more in-depth analysis of dependencies in the program code. By doing so, it can identify potential vulnerabilities more effectively. We believe that this new method can help software developers classify and identify potential vulnerabilities, giving them a powerful tool to enhance the security of their products in the early stages of development.

The concept is to build a dynamic dependency graph for software. The vertices of the graph represent distinct elements of the program, such as functions, classes, modules, and global variables. The edges represent the connections between these elements, such as function calls, class inheritance, and module imports. By examining the graph, potential vulnerabilities can be recognized based on the manner in which the components interact with each other.

Its main application is for large software with many modules, classes, globals, and methods, which is what our method will cover. To use the method, you first need to use a static analysis tool to analyze the files to identify all of the above elements and the dependencies between them, getting the required input data.

The graph of dependencies can be represented as a directed graph G = (C, E), where C is a set of vertices representing program components, and E is a set of directed edges representing dependencies between these components.

Operations on the graph:

1. Definition of the degree of dependence (D): in-degree(|{e∈E|e=(c,k)}|) is the number of edges included in the vertex v (input dependencies). out-degree(|{e∈E|e=(c,k)}|) is the number of edges coming out of the vertex v (outgoing dependencies). c, k are vertices in the graph, representing program components. E is a set of edges that represent dependencies between components. This allows you to identify components with a high degree of dependency that may pose a higher risk. Equation 1.

$$D(c,k,e) = in - degree(|\{e \in E|e = (c,k)\}|) + out - degree(|\{e \in E|e = (c,k)\}|) \quad (1)$$

2. Path analysis: Using depth-first search (DFS) or breadth-first search (BFS) algorithms, all possible paths between two vertices can be identified. This allows you to detect unsafe data transmission paths that can lead to vulnerabilities.
3. Risk assessment of components: The risk of a component can be assessed based on its dependencies and importance in the context of the overall functionality of the application. Equation 2.

$$R(c) = \alpha * D(c,k,e) + \beta * I(c) \quad (2)$$

4. Where I(v) is the importance index of the vertex (for example, based on its functional role), and α and β are coefficients that determine the weight of the degree of dependence and the importance index, respectively.

Machine learning can be integrated to efficiently analyze the dependency graph and recognize patterns that may indicate potential vulnerabilities. Historical data about vulnerabilities and their contexts in software can be used as training data to enable the system to predict security risks for new or changed application components. This approach not only detects existing vulnerabilities but also predicts potential locations of new vulnerabilities, providing comprehensive protection against software security threats.

5. Experimental studies

We selected the software intended for the Windows 10 system, which comprises around 1000 components, including classes, modules, and functions, for testing purposes. During the software counting process, we identified 3,217 dependencies between the components. It has been found that the program has the potential to contain nearly 150 vulnerabilities.

To compare the results of the Static Application Security Testing (SAST) detection method with the Dynamic Analysis of the Dependency Graph, we conducted the testing, and the outcome is presented in Table 1.

Table 1
Results of experiments

Parameter	SATS	GBVDM
Vulnerabilities have been detected	112	141
False positives	20	12
No vulnerabilities detected	38	9
Analysis time (minutes)	32	44
Code coverage(%)	95%	98%
Complex dependencies are revealed	-	250

It can be observed that although DAGD takes more time for analysis (12 minutes more), its efficiency is higher by 19.44%. Our method also covers 3% more code, which can be critical for large and complex projects. Another significant advantage of DAGD is its ability to effectively identify complex dependencies that can lead to vulnerabilities, thus helping to better understand the application architecture and potential risks.

6. Conclusions

Thus, identifying vulnerabilities in software is a critically important aspect of cyber security. The development of the latest methods and approaches to the analysis and protection of software is key to countering modern cyber threats, which in turn will contribute to the creation of a safer cyberspace. Further research in this area will not only improve existing techniques for detecting vulnerabilities but also contribute to the development of new approaches and security standards that can provide reliable protection against modern cyber threats.

Improving cyber security requires an interdisciplinary approach that involves the application of knowledge from various fields of science and technology. This approach aims to develop complex solutions in the field of software protection. It is important to have a deep understanding of technology and an awareness of the social, economic, and psychological factors that impact cyber security.

Collaboration between government institutions, the private sector, and academic circles can hasten the development and implementation of innovative methods to protect against cyber threats. This can also help establish widely recognized standards and regulations in the field of cybersecurity. By utilizing advanced technologies such as artificial intelligence, machine learning, and big data analytics, we can enhance the efficiency of our systems for detecting and neutralizing cyber threats.

In this context, the proposed method we developed shows significant advantages over traditional approaches such as SAST. An experimental application of the GBVDM method on Windows 10 software with 1000 components showed that it can identify more vulnerabilities with fewer false positives and better code coverage. This confirms the importance of further development and integration of the latest analysis methods into standard cyber defense procedures to ensure more effective detection and remediation of vulnerabilities.

References

[1] Y. Valdés-Rodríguez, J. Hochstetter-Diez, J. Díaz-Arancibia, R. Cadena-Martínez, Towards the Integration of Security Practices in Agile Software Development. A Systematic Mapping Review. Appl. Sci. 13 (2023) 4578.

[2] O. Savenko, S. Lysenko, A. Nicheporuk, B. Savenko, Metamorphic Viruses' Detection Technique Based on the Equivalent Functional Block Search. CEUR-WS, 1844 (2017) 555–569.

[3] New McAfee Report Estimates Global Cybercrime Losses to Exceed $1 Trillion. 2020. URL: https://www.mcafee.com/en-gb/consumer-corporate/newsroom/press-releases/press-release.html?news_id=6859bd8c-9304-4147-bdab-32b35457e629&virus_k=98318.

[4] SQL injection. 2024. URL: https://portswigger.net/web-security/sql-injection

[5] Cross-Site Scripting. 2024. URL: https://portswigger.net/web-security/cross-site-scripting

[6] buffer overflow. 2021. URL: https://www.techtarget.com/searchsecurity/definition/buffer-overflow

[7] Insecure deserialization. 2024. URL: https://portswigger.net/web-security/deserialization

[8] What Is Components With Known Vulnerabilities? How To Mitigate The Risks Associated With The Usage Of Such Components? 2023. URL: https://prophaze.com/web-application-fireall/components-with-known-vulnerabilities/

[9] Session Management Cheat Sheet. 2024. URL: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[10] Zhou, X., Verma, R.M. (2023). Software Vulnerability Detection via Multimodal Deep Learning. In: Lenzini, G., Meng, W. (eds) Security and Trust Management. STM 2022. Lecture Notes in Computer Science, vol 13867. Springer, Cham. https://doi.org/10.1007/978-3-031-29504-1_5.

[11] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu and Z. Chen, SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. IEEE Transactions on Dependable and Secure Computing, 19 4 (2022) 2244-2258

[12] R. Amankwah, P. Kudjo, S. Yeboah, Evaluation of Software Vulnerability Detection Methods and Tools. A Review. International Journal of Computer Applications (2017) doi: 169. 22-27. 10.5120/ijca2017914750.

[13] T.Sultan, S. Hendaoui, Advancing Network Security: Enhancing Dynamic Vulnerability Detection in Secure and Insecure Programming through SDN-ML Hybrid Architecture, (2023) doi: 10.21203/rs.3.rs-3318480/v1.

[14] Xuan, Cogent Engineering, 10: 2221962 (2023) doi: 10.1080/23311916.2023.2221962.

[15] A. Khraisat, I. Gonda, P. Vamplew, J. Kamruzzaman, Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecur. 2:20 (2019). doi:10.1186/s42400-019-0038-7.

[16] Wan, B., Xu, C., & Koo, J. (2023). Exploring the Effectiveness of Web Crawlers in Detecting Security Vulnerabilities in Computer Software Applications. International Journal of Informatics and Information Systems, 6(2), 56-65. doi:https://doi.org/10.47738/ijiis.v6i2.158.

[17] A. Anwar, A. Khormali, J. Choi, H. Alasmary, Sung J. Choi, S. Salem, D. Nyang, D. Mohaisen , Measuring the Cost of Software Vulnerabilities, SESA EAI (2020) doi: 10.4108/eai.13-7-2018.164551.

[18] M. Chornobuk, V. Dubrovin, L. Deineha. Cybersecurity: Research on methods for detection DDoS attacks. Computer Systems and Information Technologies, 2023, (4), 6–9. https://doi.org/10.31891/csit-2023-4-1.

[19] L. Allodi, M. Cremonini, F. Massacci, Measuring the accuracy of software vulnerability assessments: experiments with students and professionals. Empir Software Eng 25, 1063–1094 (2020).

[20] Y. Huang, Z. Wang, H. Ou, Y. Chi, Fuzzing-Based Office Software Vulnerability Mining on Android Platform. In: Z. Qian, M. Jabbar, X. Li, (Eds) Proceeding of 2021 International Conference on Wireless Communications, Networking and Applications. Lecture Notes in Electrical Engineering. Springer, Singapore (2022). doi: 10.1007/978-981-19-2456-9_114.

[21] Z. Shen, S. Chen, A Survey of Automatic Software Vulnerability Detection, Program Repair, and Defect Prediction Techniques. Security and Communication Networks (2020) 8858010. doi: 10.1155/2020/8858010.

[22] A. Shukla, B. Katt, L. O. Nweke, P. K. Yeng, G. K. Weldehawaryat, System security assurance. A systematic literature review, Computer Science Review, 45 (2022) 100496, 1574-0137.

[23] O. Savenko, S. Lysenko, A. Nicheporuk, B. Savenko, Approach for the Unknown Metamorphic Virus Detection, Proceedings of the 8-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, September 21–23, Bucharest (Romania), 2017, pp. 71–76.

[24] O. Pomorova, O. Savenko, S. Lysenko, A. Nicheporuk, Metamorphic Viruses Detection Technique based on the the Modified Emulators. CEUR-WS 1614 (2016) 375-383.

[25] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, A.Nicheporuk, Technique for detection of bots which are using polymorphic code. Communications in Computer and Information Science,431 (2014) 265-276, 1865-0929.

[26] A. Sadeghi, N. Esfahani, S. Malek, Mining the Categorized Software Repositories to Improve the Analysis of Security Vulnerabilities. In: S. Gnesi, A. Rensink, (eds) Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science, 8411. Springer, Berlin, Heidelberg (2014). doi: 10.1007/978-3-642-54804-8_11.

[27] I. Kalouptsoglou, M. Siavvas, A. Ampatzoglou, D. Kehagias, A. Chatzigeorgiou, Software vulnerability prediction. A systematic mapping study, Information and Software Technology, 164 (2023) 107303, 0950-5849.