# 1. Introduction

In this chapter, some fundamental concepts and knowledge about the plugin development in current software industry will be illustrated and the reason why it is a popular topic the modern software development area will also be demonstrated in Section 1.1. We will also discuss the security issues accompanying with the emergency of plugins in Section 1.2. At last, we will indicate the outline of this thesis in Section 1.3

## 1.1 Popular Fields: Plugin Development in Modern Software Ecosystems

In modern software ecosystems, plugin development has become an essential part of software engineering due to its flexibility, extensibility, and cost-effectiveness, particularly for popular platforms such as IntelliJ and VScode[1]. Plugins are software components that extend the functionality of existing software by adding new features, capabilities, or services to enhance the user experience which allows developers to customize existing software systems or create new functionalities without altering the core codebase of the system[2]. Plugins can be developed by third-party developers or even end-users, making them a critical element of the software development process. The concept of plugins is not proposed recently; it has been around for several decades. However, with the advent of modern software ecosystems, the use of plugins has become more prevalent than ever[3]. Modern software ecosystems are complex and dynamic, and they require a high degree of flexibility to adapt to changing user needs. Plugins provide this flexibility by allowing developers to add new features to an application without having to modify the core functionality of the application.

In recent years, plugin development has become an integral part of many popular software ecosystems, including web browsers, content management systems, and integrated development environments[4]. For example, web browsers such as Google Chrome and Mozilla Firefox support a wide range of plugins that allow users to customize their browsing experience and add new functionality to the browser.[5] Similarly, content management systems like WordPress and Drupal support a vast number of plugins that can be used to extend the functionality of the core system.[6]

The popularity of plugin development can be attributed to several factors. First, plugins allow developers to build on top of existing software applications, which can significantly reduce development time and effort. Second, plugins can be used to add new functionality to an application without requiring changes to the core codebase, which can make it easier to maintain the application over time. Finally, plugins can be developed by third-party developers, which can help to create a vibrant ecosystem of plugins around a particular software application.

## 1.2 Challenges in plugin development

Despite its many benefits, plugin development is not without its challenges. One of the most significant challenges facing plugin developers is security. Plugins can introduce security vulnerabilities into an application if not developed properly, which can put the application and its users at risk. For example, a malicious plugin can be used to steal sensitive user data, compromise the security of the application, or even take control of the user's system.[7]

One of the primary security challenges associated with plugin development is ensuring the integrity and authenticity of plugins. Since plugins are developed by third-party developers, there is a risk that they may contain malicious code or vulnerabilities that could be exploited by attackers. In addition, the development process of plugins may involve the use of third-party libraries or frameworks, which may also have security issues that could be exploited by attackers.[8] Therefore, it is crucial to ensure that plugins are developed using secure coding practices and undergo rigorous testing to detect and fix any vulnerabilities.

Another significant security challenge in plugin development is the need to manage access control. Plugins can be granted varying levels of access to system resources, and any vulnerabilities in these access control mechanisms can be exploited by attackers to gain unauthorized access.[9] Therefore, it is essential to implement proper access control measures to restrict plugin access to only the necessary resources and functions.

Moreover, plugins may have dependencies on other plugins or libraries, which can Introduce security risks. These dependencies may contain vulnerabilities that could be exploited by attackers.[10] As a result, it is essential to monitor and update plugins and their dependencies regularly to ensure they are secure and free of vulnerabilities.

The following section outlines the research objectives and goals of this thesis, which aims to contribute to the existing literature by proposing a comprehensive framework for the secure development of plugins.

## 1.3 Thesis statement and outline

In spite of plugin development bringing numerous benefits, there are significant security challenges associated with it. Access control vulnerabilities are a critical security issue in plugin development that can result in serious consequences, such as data breaches and system compromises. Therefore, the primary objective of this thesis is to evaluate the effectiveness of language-based security mechanisms in preventing access control vulnerabilities in plugin development by exploring the security challenges that arise from unexpected access to files, network, and other parts of the application. We aim to investigate the current state of security in plugin development, the existing solutions, and their limitations, and also investigate capability-based module systems' potential to address these vulnerabilities.

# 2. Background

In this chapter, we will firstly demonstrate the background of developing plugins through different languages and elaborate the relationships among them as well as the history of plugins development in Section 2.1. After that, we will introduce one of the most important aspects: IDE which is the basis of our experiments in Section 2.2. Finally, we will discuss the access control vulnerabilities in plugin development including the principles and consequences of them in Section 2.3.

## 2.1 Background of programming languages and plugin development

Plugins are an essential component in modern software ecosystems, allowing developers to extend the functionality of existing software. The development of plugins has been facilitated by the availability of a range of programming languages, each with its own strengths and weaknesses.

One of the most popular programming languages for developing plugins is Java, due to its platform independence, object-oriented design, and widespread adoption. Java plugins can be used across multiple platforms and operating systems, and can be easily distributed to end-users[11]. The Gradle provides a standard framework for developing Java plugins, enabling developers to create reusable and interoperable components.[12]

Other programming languages that are commonly used for plugin development include C++, Python, and JavaScript. C++ is popular for developing plugins that require high performance, such as those used in video games or graphics-intensive applications. Python is commonly used for scripting plugins, due to its ease of use and flexibility. JavaScript is widely used for web-based plugins, such as those used in web browsers.[13]

The history of plugin development dates back to the early days of software development. One of the first instances of plugin development can be traced back to the Emacs editor in the 1970s.[14] Emacs provided a framework for users to extend the editor's functionality through the use of Lisp scripts.

In the 1990s, plugins became more widespread with the advent of web browsers. Browsers such as Netscape Navigator and Internet Explorer allowed developers to create plugins that extended the browser's functionality, such as the Adobe Flash plugin for playing multimedia content.[15]

Today, plugins are used in a wide range of applications, from productivity software to video games. The availability of a range of programming languages has made it easier for developers to create plugins for a variety of platforms and use cases.

## 2.2 Integrated development environment(IDE)

An Integrated Development Environment (IDE) is a software application that provides comprehensive facilities for software development. IDEs generally consist of a source code editor, build automation tools, and a debugger.[16] IDEs help developers write, compile, and debug their code more efficiently, with features like syntax highlighting, code completion, and error checking. IDEs also provide support for version control systems, testing frameworks, and other tools that help developers write better software.

There are several popular IDEs used in plugin development, including the IntelliJ platform and VS Code. The IntelliJ platform is a popular Java IDE developed by JetBrains. It provides a range of tools for Java developers, including support for multiple languages, debugging, and code completion.[17] The IntelliJ platform also includes a plugin development kit, which allows developers to create custom plugins for IntelliJ.[18] It has a modular architecture that allows it to be extended using plugins, making it a popular choice for plugin development.

VS Code, on the other hand, is a lightweight, cross-platform code editor developed by Microsoft. It has quickly become one of the most popular code editors in the world, thanks to its ease of use and wide range of extensions. It supports a range of programming languages, including JavaScript, TypeScript, and Python.[19] VSCode has a powerful extension system that enables developers to customize the IDE's features and functionality.[20] This makes it a popular choice for plugin development, especially for web and cloud-based applications.

Both IntelliJ and VS Code provide powerful tools for plugin development, but they also introduce new security challenges. For example, plugins can be a source of vulnerabilities in an IDE. If a plugin is not properly designed or implemented, it can provide a way for an attacker to gain access to sensitive information or perform unauthorized actions on the user's system. IDEs also introduce new attack surfaces, since they have access to a wide range of system resources and can interact with third-party libraries and services.

## 2.3 Access control vulnerabilities in plugin development

### 2.3.1 Principles of access control in software systems

Access control is the process of managing the permission to access resources in a system. Access control mechanisms are typically based on two key concepts: authentication and authorization.[21] Authentication is the process of verifying the identity of a user or system, while authorization is the process of granting or denying access to specific resources based on the user's role or permissions.

Access control vulnerabilities in plugins arise when developers fail to implement effective access control mechanisms, leading to unauthorized access and manipulation of data and functionality. Attackers can exploit access control vulnerabilities in plugins to gain elevated privileges, execute unauthorized actions, and access sensitive data. This kind of vulnerabilities can occur due to a variety of reasons, such as weak or default passwords, improper authentication mechanisms, and insecure communication channels.[22] These vulnerabilities can be exploited by attackers to gain unauthorized access to sensitive data, modify or delete critical files, or execute malicious code on the system.[23]

One common access control vulnerability in plugin development is inadequate input validation. Developers must ensure that all input received from external sources, such as user input, is thoroughly validated to prevent unauthorized access and manipulation. Failure to validate input can result in attacks such as injection attacks, where attackers inject malicious code into the plugin and execute it with elevated privileges.

Another access control vulnerability in plugin development is inadequate authentication and authorization mechanisms. Developers must ensure that plugins authenticate and authorize users properly before granting them access to sensitive data and functionality. Failure to implement proper authentication and authorization can result in attacks such as privilege escalation, where attackers gain access to higher levels of privilege than intended.

### 2.3.2 Consequences of access control vulnerabilities

Access control vulnerabilities in plugins can have serious consequences on the security of software systems. Attackers can exploit these vulnerabilities to gain unauthorized access to sensitive data or system resources, modify or delete data, or execute arbitrary code.

One of the most common consequences of access control vulnerabilities in plugins is data theft. When an attacker gains unauthorized access to sensitive data, they can steal it for their own purposes or sell it on the black market. This can have serious implications for individuals, businesses, and even governments, as it can lead to identity theft, financial losses, and other types of fraud.

Another consequence of access control vulnerabilities in plugins is system compromise. Attackers can use these vulnerabilities to gain control of a system and execute arbitrary code, which can be used to launch further attacks or cause damage to the system. For example, an attacker could use a plugin vulnerability to install malware on a system, which could then be used to steal data, monitor user activity, or launch other types of attacks.

The consequences of access control vulnerabilities in plugins can be severe, leading to data theft, system disruption, and financial loss. Access control vulnerabilities in plugins have been exploited in several high-profile attacks and several high-profile security incidents have highlighted the importance of secure access control mechanisms in plugin development in recent years, such as the 2017 Equifax breach, where attackers exploited a vulnerability in a plugin to steal sensitive data of over 147.9 million customers[24].

1. Rodrigo. Santos, Cludia. Werner, Olavo. Barbosa and Carina. Alves, "Software Ecosystems: Trends and Impacts on Software Engineering," 2012 26th Brazilian Symposium on Software Engineering, Natal, Brazil, 2012, pp. 206-210, doi: 10.1109/SBES.2012.24.

2.Rupp, S. & Daum, V.. (2006). Design and implementation aspects for plugin-base software frameworks. 5. 425-432.

3.Konstantinos Manikas, Klaus Marius Hansen, Software ecosystems – A systematic literature review, Journal of Systems and Software, Volume 86, Issue 5, 2013, Pages 1294-1306, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2012.12.026.

4.Inamdar, Danish & Gupta, Shyam. (2020). A Survey on Web Application Security. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 223-228. 10.32628/CSEIT206543.

5.Li, X., & Xue, Y. (2011). A survey on web application security. Nashville, TN USA, 25(5), 1-14.

6. Fonseca, José & Vieira, Marco. (2013). A Survey on Secure Software Development Lifecycles. Software Development Techniques for Constructive Information Systems Design. 1. 57-73. 10.4018/978-1-4666-3679-8.ch003.

7. Benzekki, K., El Fergougui, A., & Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): a survey. Security and communication networks, 9(18), 5803-5833.

8. Tosi, Christian & Zanoni, Marco & Maggioni, Stefano. (2009). A Design Pattern Detection Plugin for Eclipse.

9. Huerta M, Caballero-Hernández JA, Fernández-Ruiz MA. Comparative Study of Moodle Plugins to Facilitate the Adoption of Computer-Based Assessments. Applied Sciences. 2022; 12(18):8996. https://doi.org/10.3390/app12188996

10. Y. Wang et al., "An Empirical Study of Usages, Updates and Risks of Third-Party Libraries in Java Projects," 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, SA, Australia, 2020, pp. 35-45, doi: 10.1109/ICSME46990.2020.00014.

11. Singh, Dr. Tejinder. (2015). JAVA WEB DESIGN FRAMEWORKS: REVIEW OF JAVA FRAMEWORKS FOR WEB APPLICATIONS. IJARSE. 592-595.

12. Gerber, Adam & Craig, Clifton. (2015). Gradle. 10.1007/978-1-4302-6602-0_13.

13. Cardoso, M., De Castro, A. V., & Rocha, Á. (2018, June). Integration of virtual programming lab in a process of teaching programming EduScrum based. In 2018 13th Iberian Conference on Information Systems and Technologies (CISTI) (pp. 1-6). IEEE.

14. Ciccarelli, E. (1978). An introduction to the emacs editor.

15. Levy, J. (1996, July). A Tk Netscape Plugin. In Tcl/Tk Workshop.

16. Hausladen, J., Pohn, B., & Horauer, M. (2014, September). A cloud-based integrated development environment for embedded systems. In 2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA) (pp. 1-5). IEEE.

17. JetBrains. (2021). IntelliJ IDEA. Retrieved from https://www.jetbrains.com/idea/

18. Jackson, W., & Jackson, W. (2017). An Introduction to the Android Studio Integrated Development Environment. Android Apps for Absolute Beginners: Covering Android 7, 33-57.

19. Microsoft. (2021). Visual Studio Code. Retrieved from https://code.visualstudio.com/

20. Prakash, M. G., Ponmalar, A., Deeba, S., Akilandeswari, A., Rasool, S. B. M., & Lavanya, P. (2022, December). VSCODE-Code With Voice Using Natural Language Processing (NLP). In 2022 International Conference on Computer, Power and Communications (ICCPC) (pp. 571-574). IEEE.

21. Sindre, Guttorm & Opdahl, Andreas. (2005). Eliciting security requirements with misuse cases. Requirements Engineering. 10. 34-44. 10.1007/s00766-004-0194-4.

22. Shostack, A. (2014). Threat modeling: Designing for security. John Wiley & Sons.

23. Venson, E., Guo, X., Yan, Z., & Boehm, B. (2019, August). Costing secure software development: A systematic mapping study. In Proceedings of the 14th International Conference on Availability, Reliability and Security (pp. 1-11).

24. Gressin, S. (2017). The equifax data breach: What to do. Federal Trade Commission, 8.