Developers Are Victims Too : A Comprehensive Analysis of The VS Code Extension Ecosystem

Shehan Edirimannage[1*], Charitha Elvitigala[1*], Asitha Kottahachchi Kankanamge Don[1], Wathsara Daluwatta[1], Primal Wijesekara[2], Ibrahim Khalil[1]

[1] *School of Computing Technologies, RMIT University, Melbourne, VIC 3000, Australia*
[2] *International Computer Science Institute & University of California, Berkeley, CA, 94704, US*
[*] The first two authors contributed equally to this work.
{shehan.edirimannage, charitha.elvitigala, asitha.kottahachchi.kankanamge.don, wathsara.daluwatta}@student.rmit.edu.au
primalw@icsi.berkeley.edu, ibrahim.khalil@rmit.edu.au

Abstract—With the wave of high-profile supply chain attacks targeting development and client organizations, supply chain security has recently become a focal point. As a result, there is an elevated discussion on securing the development environment and increasing the transparency of the third-party code that runs in software products to minimize any negative impact from third-party code in a software product. However, the literature on secure software development lacks insight into how the third-party development tools used by every developer affect the security posture of the developer, the development organization, and, eventually, the end product. To that end, we have analyzed 52,880 third-party VS Code extensions to understand their threat to the developer, the code, and the development organizations. We found that 5.6% of the analyzed extensions have suspicious behavior, jeopardizing the integrity of the development environment and potentially leaking sensitive information on the developer's product. We also found that the VS Code hosting the third-party extensions lacks practical security controls and lets untrusted third-party code run unchecked and with questionable capabilities. We offer recommendations on possible avenues for fixing some of the issues uncovered during the analysis.

Index Terms—Privacy-invasive software, Trust management, Software tools,

I. INTRODUCTION

Accepting the *Turing Award* in 1984, Thompson projected that reliance on third-party software components involves a significant amount of trust [1]. Despite this warning and subsequent research and incidents stemming from supply chain issues [2]–[7], developers' and organizations' reliance on third-party components continues to increase. At the same time, these components create security risks through undocumented functionality and settings, bugs, or malicious code.

Supply chain attacks have emerged as a pervasive and ubiquitous threat vector in recent years [8]–[10]. The main reason why these types of attacks can be catastrophic is due to the nature of their coverage: numerous systems and software products may be vulnerable, often without users', administrators', or organizations' knowledge, as most software products do not enumerate their constituent third-party components. Mounting an attack on several hundred organizations would otherwise be daunting; however, if the attacker can compromise one of the tools or systems used by all those organizations, then the economics suddenly shift heavily to the attacker. This has become quite apparent in the wake of several high-profile attacks [2], [6], [11]. The sheer scale of these attacks is unprecedented. However, the security community has been aware of these risks for almost 40 years.

Standards, like the Software Bill of Materials (SBoM), increase transparency and awareness of the third-party code that may be executed in an organizational environment. A few of the current literature on supply chain security includes securing CI/CD [12], [13] or securing update channels [14] focusing on securing the product even before it ships out of the development organization. Security literature has looked into how developers write secure code [15] or how they look for security advice [16], which is essential in securing the whole ecosystem. However, the recent high-profile attacks on a development organization emphasize the importance of securing the roots of the supply chain: the developer [17].

While SBoM helps to articulate the composition of the software code consisting of third-party software libraries, there has been an oversight on one aspect of the development activity that involves third-party code: developer tools. Developers use tools for a wide range of activities that have a direct impact on the final software code. There is already documented evidence [18], [19] of malicious developer tools that could jeopardize the integrity of the development environment, which will lead to catastrophic events such as recent attacks [17]. However, security and privacy literature lacks systematic knowledge of how third-party developer tools behave in the wild and their impact on the security posture of the developer, the organization, and the end product.

To that end, we analyzed 52,880 third-party VS Code extensions using static and dynamic analysis methods. We used static analysis to examine the code and narrow the extension list with suspicious code segments. We then executed those selected extensions using an instrumented VS Code environment, logging all their execution aspects. We also used VirusTotal [20], Retire.js [21] to scan for malicious content and known vulnerabilities. With this holistic analysis

approach, we found that 5.6% [1] of our collected extension set has suspicious behavior that could jeopardize the integrity of the development environment and/or leak sensitive information such as code and personally identifiable information.

We contribute the following:

- Systematically show that the third-party developer tools pose a serious security threat to the developer, the host computer, and the organization.
- VS Code, one of the most popular developer tools, has a lax security architecture that lets third-party extensions run unchecked, resulting in serious security lapses.
- To the best of our knowledge, the paper presents the first holistic (52, 880) analysis of developer tools and their security and privacy implications.

II. VISUAL STUDIO CODE

Visual Studio Code (VS Code) [22] is a free, open-source code editor developed by Microsoft. VS Code is built on the Electron framework [23], which facilitates the creation of cross-platform desktop applications. Electron leverages the Chromium engine [24] for rendering web pages within applications. The Stack Overflow 2023 Developer Survey [25] reveals that Visual Studio Code continues to be the preferred IDE among developers. VS Code supports various programming languages, including JavaScript, TypeScript, Python, PHP, C++, and C#.

A. Extensions

A Visual Studio Code (VS Code) extension is a piece of third-party software installed into the VS Code editor to enhance its functionality. The development of VS Code extensions is a collaborative effort involving the community and technology providers. A VS Code extension can be defined as a Node.js application. Each Node.js application contains a `package.json` file, which holds descriptive and functional metadata about the application. Similarly, a VS Code extension includes a `package.json` file that contains information specific to the extension [26]. This file holds additional details pertinent to the extension, akin to the Manifest files found in Android apps and Chrome extensions.

A portion of a sample package.json is shown in the Listing 1, each extension has its own `package.json`. The extension can define a set of other extensions to be installed along with its installation, and `extensionPack` ② is used to configure that option. `extensionDependencies` is also used to define other extensions that are required and runtime dependencies to the other extension. Note that there is no consent taken from the user to install extensions that are described in the `extensionPack` ② and `extensionDependencies` ③ contexts. The term `capabilities` is used to define the runtime scope of the extension. Restricted workspace is a VS Code feature [27] in which extensions are not treated as a trusted source.

```json
{
    "main": "./out/extension",
    "scripts": {
        "compile": "tsc -p ./"
    },
    "dependencies": {    ←①
        "@types/vscode": "0.10.x",
        "axios": "0.0.1"
    },
    "extensionPack": [    ←②
        "chrismarti.regex"
    ],
    "extensionDependencies": [    ←③
        "zobo.php"
    ],
    "repository": {
        "url": "https://me.me"
    },
    "capabilities": {
        "untrustedWorkspaces": {    ←④
            "supported": "true"
        }
    }
}
```

Listing 1. This listing shows a portion of a sample package.json file with keys colored in green and values in blue.

B. Extension Development

To develop a VS Code extension, developers use a combination of JavaScript or TypeScript and Node.js, leveraging the extensive VS Code API [28]. The API allows access to various aspects of the editor, including its UI components, workspace data, settings, and more. The VS Code APIs, organized into intuitive namespaces, are crucial for extension development. Key among them are `vscode.workspace`, for workspace functionalities; `vscode.debug`, for debugging; `vscode.scm`, for source control management; `vscode.extensions`, enabling interactions with other installed extensions; and `vscode.tasks`, for task management and automation within VS Code.

C. Marketplace

The Visual Studio Marketplace [29] serves as a central hub for developers to discover, download, and publish extensions and other add-ons for a suite of development tools, including Visual Studio Code (VS Code). To ensure the safety and reliability of extensions, the Visual Studio Marketplace scans for viruses on every submitted extension or update [30]. Furthermore, the Marketplace has implemented measures to deter extension authors from name-squatting, safeguarding the integrity of extension names [30]. Verifying publishers for Visual Studio Code (VS Code) extensions is a fundamental security measure in the Marketplace. This verification process entails adding a TXT record for domain verification, confirming the publisher's authority over their domain [31].

III. THREAT MODEL

The extensibility of VS Code allows for a rich development experience but also presents a significant attack surface. Malicious actors can leverage extensions to execute arbitrary code, access sensitive data, or compromise the integrity of the development environment. The threat model of the current analysis

is guided through the main three questions we mentioned in the beginning:

1) Can an extension be malicious and pose a threat to the host computer? Developer organization?
2) Can an extension introduce vulnerabilities to the code?
3) Can an extension leak sensitive information?

A. Malicious Extensions

In the first scenario (Figure 1), the likely process begins with the attacker publishing a malicious extension to the marketplace. After an unknowing victim installs the malicious extension, the extension can communicate with other installed extensions, including vulnerable extensions, to escalate privileges gain unauthorized access, or directly access underlying unrestricted APIs to carry out the malicious attack. These extensions can then interact with the runtime and renderer contexts, performing malicious operations such as modifying workspace content or running background processes. Crucially, they can leverage Node.js to interact with the victim's operating system and share data with an external server, compromising user privacy and system integrity. These extensions are malicious by nature and designed to harm the host computer, the developer, and/or the developer organization.

B. Vulnerable Extensions

The Vulnerable Extensions scenario (Figure 2) illustrates how external attackers can exploit vulnerabilities in legitimate and benign extensions. The attacker may not directly publish a malicious extension but instead uses vulnerabilities in existing extensions to penetrate the developer's host system. These activities include injecting code into workspace tabs and executing background operations, culminating in exploiting npm library vulnerabilities. While the developer of the vulnerable extensions might not be an accomplice to the potential hack, the developer's mistakes are a crucial contributing factor enabling external attackers to compromise the host.

C. Privacy-Invasive Extensions

In the Privacy-Invasive Extensions scenario (Figure 3), the threat arises from extensions that overreach their intended scope. They can monitor changes in the workspace, execute unauthorized operations, and transmit data to external servers without user consent, potentially leading to significant privacy breaches. Such breaches carry a significant weight as they could potentially include confidential organizational information. The biggest contributing factors is again the architectural limitations in VS code with a lack of proper developer consent and permissions guarding sensitive information.

IV. METHODOLOGY

Our main objective is to understand the threats posed by VS Code extensions – we selected the VS Code for two reasons: a) it is the leading IDE/Code Editor used by developers [25], and we can instrument the Electron-based IDE/Code Editor to understand its execution [32]. Due to the vast array of capabilities available through the platform, we deployed different
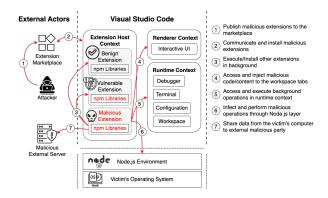

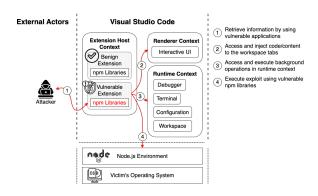Fig. 1. Malicious Extensions

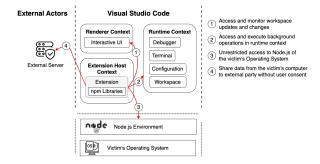Fig. 2. Vulnerable Extensions

Fig. 3. Privacy Invasive Extensions

techniques to filter out extensions and verify the suspicious behavior. We used a mix of static and dynamic analysis. While dynamic analysis exposes the execution behavior, static analysis helps to narrow down the initial questionable list, increasing the scalability of the analysis considerably. We collected a dataset of 52,880 extensions over three months and analyzed them using various tools. This analysis also thoroughly examined the extension code, focusing on the Manifest (`package.json`) file.

A. Data Collection

To comprehensively analyze the Visual Studio Marketplace, we developed a crawler to identify and download all extensions listed in the Marketplace. To capture the most up-to-date

data, we ran a daily crawler from July 15, 2023, to October 16, 2023, explicitly targeting recently published extensions. After three months of crawling, we ended up downloading 52,880 extensions. To the best of our knowledge, the extension marketplace has no geo-restrictions; hence, we are confident that our initial set of 52,880 is a representative sample of the entire VS Code extension ecosystem.

B. Analysis pipeline: Static

In the static analysis phase of our study, we rigorously evaluated all extensions using a combination of tools and code analysis methods: VirusTotal [20], Retire.js [21], and a combination of VS Code API Usage Analysis and Manifest Analysis. This multi-faceted approach allowed us to capture different threats with a holistic view. Figure 4 illustrates the static analysis pipeline, delineating the sequential steps and methodologies employed in this phase.
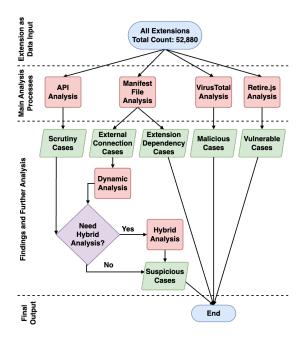

Fig. 4. Execution Flow of the Analysis

VS Code API Usage Analysis: We analyzed the VS Code API usage of each extension to identify poor security practices or malicious behaviors introduced by the extension developers, potentially raising security threats and risks. We then generated an Abstract Syntax Tree (AST) for every JavaScript file within the extracted extension. The AST enabled us to scrutinize each file for VS Code extension API usage and other questionable behaviors.

Manifest Analysis: We processed the Manifest (`package.json`) file for all extensions to extract various fields, specifically on `dependencies`, `extensionPack`, `extensionDependencies`, and `capabilities` for further analysis. We aimed to understand how they influence the security posture of each extension and any hidden behavior. We also used manifest analysis to filter any extensions that could share data over the network.

VirusTotal Analysis: VirusTotal [20], an online platform, facilitates the analysis of files and URLs to identify potential threats such as viruses, worms, trojans, etc. It operates by aggregating a range of antivirus engines and website scanners. We utilized VirusTotal's API to submit all examined extensions for analysis. The generated reports encapsulated the detection outcomes from each participating antivirus engine.

Vulnerability Scan using Retire.js: Retire.js [21] is employed to detect known vulnerabilities, specifically CVEs (Common Vulnerabilities and Exposures) [33], in Node.js packages used by VS Code extensions. This tool scans the Node.js dependencies in each extension, revealing their security status. These reports highlight any known vulnerabilities in the Node.js dependencies.

C. Analysis pipeline: Dynamic

We ran the extensions in a customized VS Code editor with network monitoring capabilities using MITM proxy [34]. Our dynamic analysis pipeline is depicted in Figure 5, which provides a detailed illustration of the process. This approach enabled us to capture and analyze the in-situ behavior of the extensions meticulously.

Extensions Selection for Dynamic Analysis: We selectively focused on extensions that create external connections (network requests), recognizing their greater potential to inflict arbitrary damage on code, developers, host systems, and organizations, as detailed in Section 3.4. Since VS Code does not have a built-in API for establishing external connections, extensions commonly utilize third-party Node.js packages. Based on these findings, we selected 2,365 extensions for in-depth dynamic analysis. Additionally, we chose several other extensions suspected of harboring malicious behaviors, through the static analysis stage. In total, we selected 2,698 extensions for dynamic analysis.

VS Code Instrumentation: We developed an instrumented version of VS Code, utilizing VS Code 1.80 [35] as the base. This instrumented version was designed to capture all VS Code API calls comprehensively. We implemented monitoring mechanisms for terminal access and other resource accesses. These monitoring capabilities were active during both the installation and execution phases of each extension.

Network Request Monitoring: We utilized a pre-configured mitmproxy proxy [34] to monitor web traffic generated by the extensions, encompassing protocols like HTTP, WebSockets, and other SSL/TLS-protected protocols. We installed a mitmproxy certificate on the hosting environment to enable the decryption of HTTPS traffic.

D. Analysis pipeline: Summary

We first ran the static analysis to detect suspicious code or unusual coding patterns. These findings were then further scrutinized through dynamic analysis for confirmation. During dynamic analysis, we captured network requests from extensions, including IPs and URLs to which the extensions connect. We then evaluated their maliciousness using threat intelligence resources. Conversely, insights gained from dynamic analysis

also informed subsequent static analysis - doing threat analysis using VT for URLs uncovered in the dynamic analysis.
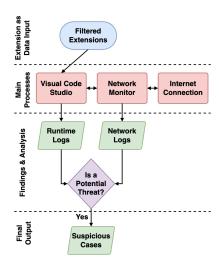

Fig. 5. Dynamic Analysis Flow

V. VISUAL STUDIO CODE SECURITY ANALYSIS

This section analyses the security measures in VS Code and their limitations. Figure 6 shows the current architecture of the Processes in VS Code. VS Code is based on Electron framework [23]. VS Code architecture, however, has a significant influence from the Chromium browser [24] and its' V8 JavaScript engine [36]. However, VS Code extensions are executed on top of a Node.js environment.

A. Sandboxing

Sandboxing enables untrusted code to provide functionality in a controlled manner containing any harmful behavior of the untrusted code [37]. In the current VS Code architecture, only the *Renderer Process* (Figure 6) is sandboxed containing any harmful behavior within the *Rendered Process* which hosts all the VS Code functionalities and the VS Code UI.
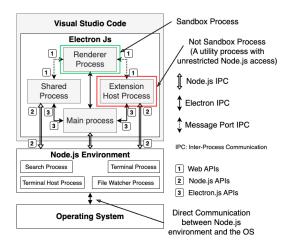

Fig. 6. Sandboxed Process Architecture of VS Code.

The current VS Code architecture does not sandbox *Extension Host Process*, which hosts all the extensions. As per official documentation [38], providing a scalable solution with full Node.js support is the reason for separating the extension process from the sandboxed *Rendered Process*. The main caveat that comes with that decision is the *Extension Host Process* can access everything from the VS Code data to the host computer files and data.

As per Electron documentation [39], the *Extension Host Process* hosts a variety of functions, including management of untrusted services, execution of CPU-intensive tasks, and oversight of crash-prone components. This suggests that VS Code extensions, hosted within the *Extension Host Process*, are, rightly so, treated as 'untrusted services' – common wisdom suggests that any third-party code execution should be treated with caution.

Untangling the Sandbox: Security researchers have reported instances where malicious extensions have caused damage in these areas since 2018 [18], [19], [40]. Letting extensions have a free pass is likely a strategic and marketing decision to promote a robust extension ecosystem that will eventually positively impact the VS Code market share. We also believe that fixing the issues presented in this work is non-trivial which will require a complete re-design of the VS Code architecture, preserving the current functionality of the extensions.

B. Perils of Unchecked Execution

VS Code adopts a trust-based model in its extension ecosystem, characterized by broad access privileges granted to extensions. These privileges extend to potential interactions with the host system and network, all granted without any notice to the developer. Such sensitive access, while facilitating significant functional enhancement and customization capabilities, simultaneously introduces potential security risks especially if the code is not properly vetted.

This operational paradigm of VS Code stands in stark contrast to the permission-based models employed in platforms like Android and Chrome. In these systems, explicit user permissions are a prerequisite for apps or extensions to access certain functionalities or data.

Upon installation, a VS Code extension typically gains unrestricted access to the VS Code API. This level of access can be perilous if the extension is either malicious or becomes compromised due to its vulnerabilities. Possible threats include arbitrary code execution, unauthorized file manipulation, and data theft or corruption. The unregulated access to APIs raises the possibility of extensions misusing these interfaces, performing actions well beyond their advertised scope. For instance, an extension ostensibly designed for code formatting could hypothetically access and modify unrelated files in the user's workspace or interact with other extensions in a manner not initially intended – we have found similar use cases in the current marketplace.

The following list is some of the sensitive API capabilities that are not guarded by any checks or permissions.

- Run potentially sensitive terminal commands.

- Modify source code or the commit history without any developer knowledge.
- Capture authentication tokens or credentials.
- Interfere with other installed extensions.
- Modify the behavior of the app during debugging.

The absence of any sort of permission in VS Code keeps developers completely in the dark about extension's execution.

C. Third Party Dependencies

VS Code extensions, similar to standard Node.js applications, are capable of incorporating third-party Node.js dependencies. While this integration undoubtedly enhances flexibility and promotes the reuse of code, it also introduces a spectrum of security concerns. Literature has documented the escalation of security threats associated with third-party Node.js dependencies (npm) [41]–[44].

D. Network Requests

There is no specific built-in VS Code API for handling network requests. Extensions utilize various third-party Node.js packages to perform network operations. Network operations can be used for potential misuse like data exfiltration, downloading malicious payloads, or establishing command and control communication. `axios`, `node-fetch`, `request`, are Node.js packages commonly used for network requests in VS Code extensions.

E. Untrusted Workspaces

The VS Code has a restricted mode where all extensions are blocked. For an extension to run on the restricted mode, one must have this property as *true*. If it is *false*, then the extension becomes non-operational in Restricted Mode until Workspace Trust is granted, or *limited*, it will be partially functional in Restricted Mode with certain features disabled. The default value will be *false* if there is no entry. Given that this is a flag the developer voluntarily sets, a malicious developer can set it to true so that the malicious extensions can still be active in the restricted mode of the VS Code. Our testing found 542 extensions with the property set as *true*.

F. Visual Studio Marketplace

The Visual Studio Marketplace, as the central hub for VS Code extensions, implements security measures to safeguard the integrity and safety of its extensions.

a) Virus Scanning: : The marketplace has an extensive virus scanning protocol [30]. Every new submission and update undergoes a thorough virus scan before public release. Despite these scans, our analysis found dozens of potentially malicious extensions in the marketplace.

b) Preventing Name Squatting: : The Marketplace also prevents "extension name squatting" [30] to protect users from deceptive practices where unauthorized parties use similar names to popular extensions. This measure is crucial for protecting users from downloading counterfeit extensions.

TABLE I
RESULTS OF THE VT ANALYSIS OF EXTENSIONS

Min VT Positive Engine Count	Number of Extensions	Cumulative Install Count
1	835	122,489,729
2	163	1,840,317
3	62	693,147
4	26	385,408

c) Publisher Verification: : This confirms the publisher's authority over their domain, and verified publishers receive a badge in the Marketplace, denoting their verified status. However, we found that 7% of the tagged extensions for harmful content had the domain verified badge.

d) Reactive Protocols - Reporting and Kill List: : In response to security threats, the Marketplace removed such malicious extensions and the implementation of a "kill list". This list ensures that compromised extensions are automatically uninstalled from user environments.

G. Not so Helpful Extension Page

It's important to consider several key aspects that are often not explicitly detailed on the extension page:

- Many extensions do not provide specific information about their security or privacy practices. Concerned Developers might appreciate the knowledge of who is getting what data about them, their code, or the organization.
- It is vital that the developers have full transparency on resource accesses before they make an informed decision to install an extension.
- Listing third-party dependencies is also critical to make an informed decision.

It is imperative that developers are given all the possible information to make informed decisions to make sure the development environment is not compromised.

VI. RESULTS

This section presents the results of our static and dynamic analysis of the 52,880 extensions. We aim to delineate the security risks, threats, practices, and any malicious usage identified in the extensions. These findings focus on aspects that could cause arbitrary damage to code, host systems, developers, or organizations.

A. Malicious Extensions

Extension File: Our primary discovery is identifying malicious extensions by analyzing VirusTotal (VT) data. Upon scrutinizing the VT results, we initially identified 835 extensions flagged as potentially harmful by at least one antivirus engine. However, for the scope of this research, we have adopted a stringent criterion for classifying an extension as malicious: it must be marked by a minimum of four engines on VirusTotal. This methodology is in line with the best practices recently advocated in the malware research community [45]. Table VI illustrates the results of our VirusTotal analysis.

TABLE II
OVERALL SUMMARY OF RESULTS BY THREAT MODEL AND SUSPICIOUS
TYPE.

Threat	Suspicious Type	Extension Count	Cumulative Install Count
Malicious	Degrading the Security Posture	69	2,809,972
	Critical File Access	14	1,564,468
	VT >= 4 Extensions	26	385,408
	VT >= 4 Network Requests	8	6,239
	Market Missuse	42	254,232
	Concealed Operations	18	145,047
Vulnerable	Extensions with CVEs	2,620	51,952,070
API & Privacy	Tracking	49	3,107,508
	Code Sharing	108	560,666
	Data Sharing	15	504,835
Total		2969	61,290,445

TABLE III
EXTENSIONS COMMUNICATING WITH VT COUNT GREATER THAN OR
EQUAL 4 DOMAINS

Extension Name	Domain	Install Count
search-pawn-package	sampctl.com	1,977
FridaExtension	fridaplatform.online	1,823
OI Wiki	oi-wiki.org	1,723
Hey	hey.network	559
SnippetDrop	snippetdrop.com	106
Get rich overnight	gateio.ch	27
VSQuote	type.fit	21
Code Naming Conventions	openai-proxy.com	3

Out of the 835 extensions initially flagged, we discovered that 667 had at least one previous version. Our analysis of these versions revealed that 102 extensions had at least one prior malicious version.

Network Request: During our dynamic analysis, we captured the network requests made by the extensions. We separated IPs and URLs from the dataset and applied a set of heuristics to filter them for further analysis. We filtered out all the local IP addresses from the IP set, resulting in 72 direct external IPs for further examination. Regarding the URLs, we extracted the domains and filtered them using the Cloudflare Radar [46] Top 50,000 Domains. We selected an exclude list from the Radar Top 50,000 Domains for further analysis, which yielded 493 URL domains in the dataset. We then submitted all filtered IPs and URLs to VirusTotal to obtain reports, which were analyzed to check for any malicious behavior associated with the IPs and URLs.

After the VT analysis, we found 8 recipients from 8 extensions with more than 3 VT engines flagging them as malicious (Ref Table III). Some of these extensions have a significant number of downloads, risking a sizable population of developers. We found nothing sensitive shared with those domains except for one domain receiving search terms.

B. Vulnerable Extensions

One of our principal findings stems from identifying vulnerable extensions through the Retire.js vulnerability scanning process. We discovered 54 distinct vulnerabilities, each

TABLE IV
RESULTS OF SILENT EXTENSION INSTALLATION CASES.

Suspicious Cases of Silent Installation	Number of Extensions
Extensions that are installing other extensions	4,317
Extensions that are installed by other extensions	4,618
Chain extensions (E1 → E2 → E3)	2,327
With external connections (for 'installed by' cases)	325
VT Positive Cases (for 'installed by' cases)	92
CVE Positive Cases (for 'installed by' cases)	325

associated with a CVE, within Node.js packages used by these extensions. The severity levels of these vulnerabilities are categorized as low (1), medium (34), and high (19). We identified 5,775 instances of these CVEs across 2,620 extensions with a cumulative install count of 151,952,070. Further analysis revealed that these CVEs originated from 28 unique Node.js packages. All VS Code extensions are installed in the ~/.vscode/extensions/ folder. If a malicious developer has installed any of the above-mentioned vulnerable extensions, a malicious extension could access this folder, import vulnerable Node.js packages, and exploit them, potentially causing arbitrary damage to the code, host systems, developers, or organizations. We verified that 19 CVEs with a high severity level are exploitable via a demo VSCode extension that can import vulnerable Node.js packages from vulnerable extensions. Additionally, the ease with which malicious entities can access and install these vulnerable extensions exacerbates the issue. This lack of proactive alerts or guidance poses a significant risk, potentially exposing codebases, developers, host systems, and organizations to security threats inherent in these extensions.

C. Silent Extension Installation

We have discovered that in VS Code, extensions can install additional extensions through three different methods. These include specifying dependencies in the manifest file using extensionPack or extensionDependencies, and utilizing the VS Code Command API. A notable command within this API is vscode.commands.executeCommand, which can invoke workbench.extensions.installExtension. This function allows for the silent installation of extensions without the need for user consent. Our findings indicate that 4,317 extensions engage in this behavior, installing other extensions covertly. This includes the creation of extension chains (E1 → E2 → E3) and instances where an extension without external connections installs another with such connections. Crucially, we also examined extensions that silently install malicious or vulnerable extensions. Table IV provides a detailed overview of these silent extension installation cases with likely harmful content.

D. Suspicious Extensions

Degrading the Security Posture: In our VS Code API usage analysis of extensions, we identified several suspicious coding patterns that may pose security risks. With the recent wave of supply chain attacks, much of the attention has shifted towards securing the development environment [14]. We found several cases where extensions have severely degraded the security posture of the host machine, which will negatively impact not only the host's machine but could affect the organizational security posture as well. The only reason for this occurrence is the lack of proper containment – sandboxing – in the VS Code for untrusted extensions.

We found that 14 extensions directly access SSH private keys or cloud access tokens via file access paths, posing additional security vulnerabilities. This is the most severe threat posed by malicious extensions. Accessing SSH private keys and cloud access tokens has repercussions far beyond an individual's development machine. These tokens can be used to log into highly sensitive servers in the organizations, polluting production artifacts and/or update channels. Recent supply chain attacks have demonstrated that they have successfully targeted developers to exploit their weakness to access tokens, but this way, it can be done with the near certainty of success, which is dreadful.

Another severe threat was intentionally degrading the TLS setup in the environment `process.env.NODE_TLS_REJECT_UNAUTHORIZED = 0` This particular code snippet turns off TLS/SSL certificate validation, which exposes the extension to severe security risks, including man-in-the-middle attacks and potential data breaches. Our study found that 60 extensions contained this pattern, compromising security. These extensions have a median install count of 314.5. To corroborate our findings, we conducted a dynamic analysis, further verifying the significant risks posed by this practice. We identified several interesting extensions, one of which is 'Black Box' [47]. This extension interacts with the OpenAI API and notably disables SSL/TLS certificate verification(130,913 downloads).

We identified eight extensions that implement local proxy servers to enhance their functionality. These proxies are used primarily because they can circumvent the stringent network security measures established in most corporate environments. Such security measures typically include firewalls, intrusion detection systems, and content filtering policies. A VS Code extension could bypass these security protocols by setting up a local proxy, thereby creating a backdoor. This backdoor allows data to be transferred in and out of the network without undergoing standard scrutiny and security checks.

We also discovered that 329 extensions update the VS Code `settings.json` file, either through the API (using `vscode.workspace.getConfiguration("config").update("new value"))` or by directly modifying the `settings.json` file. This practice raises significant security risks and threats. An extension can use this feature to turn off any security feature in the VS code again without any developer's consent.

Data Sharing: Our dynamic analysis of 13 VS Code extensions revealed that a significant amount of data is transmitted over the internet.

VS Code extensions share a variety of information, including device specifics like unique Device ID, language settings, screen dimensions, and timezone. Operating System (OS) details are also communicated such as the username, hostname, OS name, architecture, version, and home directory path. Moreover, details about the working project are transmitted, such as the project name, directory, source code, timestamp, and analytic data which include Google Analytics, workspace dependencies, user actions, GitHub account information, log events, and stack traces.

These kinds of information at the hands of the wrong actors can be devastating for the developers and the organizations. These metadata can reveal a lot of information about the proprietary developments that are yet to be released.

Tracking: Our investigation identified 46 extensions that actively monitor and transmit data regarding user, device, project, session, coding time, user IP, user actions, commit logs, and session details. Alarmingly, 28 of these extensions fail to disclose their tracking practices to developers. The 'Code Time' [48] extension, with 425,386 downloads offers programming metrics, infringes upon privacy by clandestinely collecting information such as usernames, device hostnames, timezones, and detailed project directories alongside project names, all without user awareness.

Code Sharing: In the course of our research, we identified a total of 108 extensions within the VS Code ecosystem that engage in the sharing of source code over the internet.

We discovered a wide range of code-sharing practices with external services. Notably, 78 extensions actively transmit data to public and private large language models, specifically focusing on two extensions that share code with locally run language model servers. Recently, a few cases appeared to have leaked sensitive code to LLMs due to inaccurate configuration; with an increasing number of extensions now exploiting LLMs, it is worth noting that for almost all of the extensions, developers lack any control over configuring the LLMs that the extension use. However, within this subset, 27 extensions offer limited information about the recipients of this code on the VS Code Marketplace extension's page. We also identified three extensions transmitting selected code to servers for synonym finding and 17 extensions sending code to translation APIs. Concerningly, in both these groups, several extensions (two in the former and 12 in the latter) lack explicit disclosure regarding the destinations of the transmitted code.

Furthermore, three extensions designed for code deployment clearly state the destination of the deployed code. In code snippet sharing, four extensions stand out for their core functionality of sharing code, with one using an encryption scheme for the source code. An extension facilitating interaction with an online code playground and another intended for code vulnerability scanning must adequately disclose their data-sharing practices on their marketplace pages. Finally, an extension

focused on variable naming, transmitting selected text without proper validation, is also reticent about the destination of this data. These findings underscore the diverse nature of code-sharing functionalities among VS Code extensions, marked by varying degrees of transparency and leading to potential security and privacy concerns in the ecosystem.

Further, we found extensions transmitting personal information and logs over the internet. Out of 12 such extensions analyzed, 11 shared sensitive data, including stack traces with error logs, GitHub usernames, user emails, and extension logs, without prior disclosure to the developers. This is equally concerning as code sharing since stack traces and execution logs can still expose sensitive information.

Suspicious APIs: Our API analysis revealed that the misuse of certain APIs can cause arbitrary damage to host codebases, developers, host systems, and organizations. We utilized dynamic analysis and runtime logs to verify the extent of API usage in suspicious extensions. Detailed information for these APIs is provided in Table V, which highlights the potential security risks associated with their misuse.

Concealed Operations: In our dynamic analysis, we observed several extensions with concealed activities. One extension was identified as redirecting developers to a malicious webpage, consequently leading to the installation of an unwanted browser plugin. Another extension enabled access to a private extension marketplace within VS Code. Moreover, our investigation revealed 12 extensions clandestinely downloading third-party libraries, command-line tools, and executables without obtaining user consent or providing any notification. In addition, we discovered three extensions initiating terminals within VS Code stealthily by employing the `hideUser` parameter in the `createTerminal` API, thus operating unbeknownst to the user.

Network Capabilities: We discovered the F5 NIM extension [49] by F5DevCentral, serving as an NGINX Instance Manager with 1502 installs. The Visual Studio, Code extension marketplace, is described as a tool enabling the discovery and management of NGINX instances and configuration files. Upon executing the "Start Scan" command, developers initiate a scan of NGINX instances, launching an underlying network port scan. This was never explicitly mentioned.

Market Misuse: In our analysis, we encountered several instances indicative of marketplace misuse. We identified 16 extensions published predominantly for testing purposes, as evidenced by their lack of specific functionalities and apparent experimental nature. Additionally, our investigation revealed 69 extensions presenting multiple identical versions, characterized by shared logos, identical descriptions, and only minor variations in titles or publishers, hinting at potential duplication issues. After analyzing the `repository` field in the `package.json`, which contains the repository URL, we identified that 9,185 extensions were published without a repository. Of these, 75 extensions were flagged as malicious by VirusTotal (VT), and 142 were associated with known Common Vulnerabilities and Exposures (CVEs). Further analysis of the repository URLs revealed that 120 extensions were

clones of other extensions. Furthermore, we observed four extensions that bundled Node modules, EXE, JAR files, and JRE folders, leading to abnormally large extension sizes, a concern we've labeled 'oversized extensions.'

VII. RELATED WORK

There is limited research on the security and privacy of IDEs and code editors, particularly concerning their plugins or extensions. Elizabeth Lin [50], identified 716 dangerous data flows and 21 verified extension vulnerabilities with proof-of-concept exploits in VS Code extensions. The investigation did not cover the marketplace for malicious or suspicious extensions. A. David proposed implementing a permission system for VS Code extensions [51]. They only evaluated 56 extensions and faced challenges in fully mapping npm package usage to specific permissions. Jin et al. [52] identify security vulnerabilities in the VS Code Markdown editor.

VS Code extensions are based on Node.js, and a few studies have investigated the security risks and vulnerabilities associated with Node.js packages [41]–[44]. The authors have highlighted a few issues, such as a lack of regular updates, compromised dependency trees, and widespread vulnerabilities in NPM packages.

Numerous studies have analyzed various third-party integration tools for applications [12], [13], [53]–[55]. Authors have looked into different plugin architectures such as WordPress, Team Chat, and Android Marketplace.

Ladisa et al. scrutinize OSS supply chains, identifying their complexity as a source of security vulnerabilities [56]. Insights from 30 organizations are synthesized, spotlighting challenges like updating vulnerable dependencies, leveraging SBoM for enhanced security [14]. A study on the use of open source components (OSCs) in software projects reveals that their selection often depends on superficial metrics like downloads or GitHub stars disregarding security [57]. The significance of Reproducible Builds (R-Bs) in software projects, particularly following the 2020 SolarWinds attack [2], is highlighted [58].

There is an extensive corpus of literature on the analysis of malicious and vulnerable extensions [59]–[68]. Studies such as [69] and [70] utilize static analysis to identify extensions that potentially abuse privileges, including improper access to APIs and sensitive user data. Similarly, research like [71], [72], and [64] use static analysis to detect misuses of permissions.

VIII. DISCUSSION

We have analyzed 52,880 extensions and found 2969 (5.61%) extensions to be potentially harmful to the developer under five different categories. These extensions have a cumulative install count of 613 Million installs, exposing a significant number of developers worldwide. VS Code being the most popular IDE/Code Editor in the world [25], the 6% likely malicious extensions from the VS Code marketplace is a significant number. This work presents the first holistic analysis of the VS Code extension ecosystem.

Our analysis uncovered issues along two main axes: security of the host, the network, and the organization, as well as

API	Summary	Extension Count	Usage Count	Install Count (Median)
workspace.fs	This file system API can read and write files. If misused, it could lead to unauthorized file access or modification.	469	35,782	648
workspace.createFileSystemWatcher	Monitoring file system changes can be exploited if the data is used maliciously.	335	7,179	1,235
workspace.applyEdit	This can modify files in the workspace, posing a risk if used to make unauthorized edits.	314	7,597	865
workspace.findFiles	This API can access various files within the workspace, which could potentially expose sensitive information.	230	5,947	1041
window.activeTextEditor	Improper use of this API could lead to unauthorized access or manipulation of the active text editor content.	1,136	73,073	287
window.createTerminal	This API allows the creation of a terminal within VS Code with full access to the host machine. A potential risk is executing arbitrary commands through the terminal.	267	3,743	491
window.createWebviewPanel	Webviews can display arbitrary content, including potentially malicious HTML or JavaScript.	681	17,488	346
authentication.getSession	Accessing authentication sessions can pose risks if the session data is mishandled or exposed.	103	2,978	572
extensions.getExtension	Involves accessing details about installed extensions, which could be misused to target specific extensions or behaviors.	488	11,559	1,252
env.openExternal	This API is used to open URLs or files in the default external application. If not properly validated, it could be exploited to open malicious links or files.	554	17,703	527
env.clipboard	Access to the clipboard could lead to the exposure of sensitive data copied by the user.	301	8,060	804
env.sessionId, env.machineId, env.uiKind, env.remoteName, env.appRoot, env.appHost	These environment variables provide system and session information which, if exposed, can be sensitive	254	9,702	5972

problems of privacy and IP of the developer and the organization. Implications of a compromised extension with an ability to steal pretty much anything out of the host machine are dangerous and require a thorough examination to fix it before a real-world attack occurs. Supply chain attack on SolarWinds [2], key exposure in Microsoft [17] exposes the ugly truth of being complacent on developer security.

With the recent high-profile wave of IP theft and supply chain attacks, having untrusted extension code roaming in the host with unparalleled access should be a nightmare for organizations and developers. These unintended ID leaks can lead to sophisticated spear phishing or, even worse, targeted attacks on the host to steal code or other sensitive materials. With extensions having high severity vulnerable code executing with unchecked capabilities, these extensions themselves could be weaponized for IP theft, geopolitical advantages, or criminal gangs. The first and foremost step would be to educate developers and get the relevant stakeholders to act upon it.

Security and privacy literature has been focusing on consumer and organizational security and privacy. It is high time that we also devote our focus to developer security. Literature has looked into secure CI/CD [12], [13] and secure update channels [14], but threat vectors such as developer extensions are usually overlooked. But as the analysis suggests, developer tools and extensions present an equally important threat vector to be concerned with. The current analysis opens up two avenues for further examination: a secure extension/tool architecture with visibility and the ability to audit.

Microsoft, the VS Code team, knew about the potential dangers of the current extension architecture since 2018 [40]. The online documentation suggests their design choices were largely based on creating a capable extension ecosystem with minimal restrictions [38]. This is a sensible decision for creating a vibrant ecosystem. Still, security has to be a critical factor in designing the architecture. It raises the question of the inaction by Microsoft for 5 years, potentially underestimating the grave ramifications.

Moving forward, the solution must be multi-faceted: how can VS Code create a highly capable extension architecture with correct checks and guards following defense-in-depth and least-privilege principles, and how can the developer make informed decisions before installing extensions? The solution should guard different capabilities with permission-guarded APIs so that the developer is also in the loop during the execution. VS Code marketplace should also have a strict structure (such as in Google Playstore) with permission and capability information so that developers can comprehend the extension before deciding which tool to pick.

This work also calls for more Developer studies on understanding their privacy and security expectations while using developer tools. There can be a lot of lessons learned from Android permission literature [73]–[75] on understanding different contexts and approaches to effectively seek permission before granting access to sensitive information. Developers are likely to have an opinion on when they want to be prompted for permissions, under what contexts, for what types

of capabilities, etc. Developers are also likely to want to have an audit mechanism where they can go back and audit extension history and change their future behavior at a finer level. All of these are major changes that will considerably affect the extension ecosystem. Thus, other factors such as adoption, usability, and retention will come into play in any decision around revamping the current extension architecture.

In conclusion, our analysis reveals that developer extensions pose a credible threat to the developer, the code, the host computer, and the organization. The analysis uncovers five types of questionable behavior: (a) developers have minimal visibility into what extensions are accessing behind VS Code, (b) extensions have unchecked access to the code and the host computer, (c) some extensions purposely degrade the host computers security posture opening up to the possibility of getting hacked, (d) extensions are sharing sensitive developer information, code, execution logs, and stack traces over the internet, and (e) some extensions are either very likely malicious or riddle with vulnerabilities. The work analyzed 52,000+ VS Code extensions, and 5.6% of them have suspicious behavior potentially putting over 500 million developers at risk.

REFERENCES

[1] K. Thompson, "Reflections on trusting trust," *Communications of the ACM*, 1984.

[2] Reuters, "Solar wind attack," https://www.reuters.com/technology/exclusive-wide-ranging-solarwinds-probe-sparks-fear-corporate-america-2021-09-10/, 2021.

[3] S. Magazine, "Rapid7 attack," https://www.securitymagazine.com/articles/95236-rapid7-victim-of-a-software-supply-chain-breach, 2021.

[4] DarkReading, "Dragongfy attack," https://www.darkreading.com/attacks-breaches/pharmaceuticals-not-energy-may-have-been-true-target-of-dragonfly-energetic-bear, 2021.

[5] Y. Finance, "Juniper attack," https://finance.yahoo.com/news/juniper-breach-mystery-starts-clear-130016591.html, 2022.

[6] S. Peisert, B. Schneier, H. Okhravi, F. Massacci, T. Benzel, C. Landwehr, M. Mannan, J. Mirkovic, A. Prakash, and J. B. Michael, "Perspectives on the solarwinds incident," *IEEE Security & Privacy*, 2021.

[7] CrowdStrike, "Crowdstrike falcon platform identifies supply chain attack via a trojanized comm100 chat installer." [Online]. Available: https://www.crowdstrike.com/blog/new-supply-chain-attack-leverages-comm100-chat-installer/

[8] D. Spinellis, "Reflections on trusting trust revisited," *Communications of the ACM*, 2003.

[9] S. Bratus, T. Darley, M. E. Locasto, M. L. Patterson, R. Shapiro, and A. Shubina, "Beyond planted bugs in "trusting trust": The input-processing frontier," *IEEE Secur. Priv.*, 2014.

[10] E. Levy, "Poisoning the software supply chain," *IEEE Security & Privacy*, 2003.

[11] "Apache Log4j Security Vulnerabilities," https://logging.apache.org/log4j/2.x/security.html, [Online; accessed: 01-August-2024].

[12] I. Koishybayev, A. Nahapetyan, R. Zachariah, S. Muralee, B. Reaves, A. Kapravelos, and A. Machiry, "Characterizing the security of github CI workflows," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[13] F. Moriconi, A. I. Neergaard, L. Georget, S. Aubertin, and A. Francillon, "Reflections on trusting docker: Invisible malware in continuous integration systems," in *IEEE Security and Privacy Workshops (SPW)*, 2023.

[14] W. Enck and L. Williams, "Top five challenges in software supply chain security: Observations from 30 industry and government organizations," *IEEE Security and Privacy*, 2022.

[15] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[16] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[17] R. Satter, "Microsoft says compromise of its engineer's account led to chinese hack of us officials," *Reuters*, 2023.

[18] Can you trust your vscode extensions? Accessed 01-August-2024. [Online]. Available: https://blog.aquasec.com/can-you-trust-your-vscode-extensions

[19] Malicious vscode extensions with more than 45k downloads steal pii and enable backdoors. Accessed 01-August-2024. [Online]. Available: https://blog.checkpoint.com/securing-the-cloud/malicious-vscode-extensions-with-more-than-45k-downloads-steal-pii-and-enable-bac

[20] Virustotal. Accessed: 08/01/2024. [Online]. Available: https://www.virustotal.com/

[21] Retire.js github repository. Accessed: 08/01/2024. [Online]. Available: https://github.com/RetireJS/retire.js

[22] Visual studio code. Accessed: 08/01/2024. [Online]. Available: https://code.visualstudio.com/

[23] Electron. Accessed: 08/01/2024. [Online]. Available: https://www.electronjs.org/

[24] Chromium. Accessed: 08/01/2024. [Online]. Available: https://www.chromium.org/Home/

[25] Stack overflow developer survey 2023. Accessed: 08/01/2024. [Online]. Available: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-integrated-development-environment

[26] Visual studio code - extension manifest. Accessed: 08/01/2024. [Online]. Available: https://code.visualstudio.com/api/references/extension-manifest

[27] V. S. Code, "Visual studio code workspace trust - restricted mode," https://code.visualstudio.com/docs/editor/workspace-trust#_restricted-mode.

[28] Visual studio code api reference - vs code api. Accessed: 08/01/2024. [Online]. Available: https://code.visualstudio.com/api/references/vscode-api

[29] Visual studio code marketplace. Accessed: 08/01/2024. [Online]. Available: https://marketplace.visualstudio.com/vscode

[30] Visual studio code documentation: Can i trust extensions from the marketplace? Accessed: 08/01/2024. [Online]. Available: https://code.visualstudio.com/docs/editor/extension-marketplace#_can-i-trust-extensions-from-the-marketplace

[31] Working with extensions - publishing extension (section: Verify a publisher). Accessed: 08/01/2024. [Online]. Available: https://code.visualstudio.com/api/working-with-extensions/publishing-extension#verify-a-publisher

[32] Visual studio code documentation: Robust and extensible architecture. Accessed: 08/01/2024. [Online]. Available: https://code.visualstudio.com/docs/editor/whyvscode#_robust-and-extensible-architecture

[33] Mitre cve. Accessed: 08/01/2024. [Online]. Available: https://cve.mitre.org/

[34] A. Cortesi, M. Hils, T. Kriechbaumer, and contributors, "mitmproxy: A free and open source interactive HTTPS proxy," 2010–, [Version 10.1]. [Online]. Available: https://mitmproxy.org/

[35] Visual studio code github repository. Accessed: 08/01/2024. [Online]. Available: https://github.com/microsoft/vscode/tree/release/1.80

[36] V8 javascript engine. Accessed: 08/01/2024. [Online]. Available: https://v8.dev/

[37] M. Backes, S. Bugiel, C. Hammer, O. Schranz, and P. von Styp-Rekowsky, "Boxify: Full-fledged app sandboxing for stock android," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[38] Vs code sandboxing - the new sandbox feature. Accessed: 08/01/2024. [Online]. Available: https://code.visualstudio.com/blogs/2022/11/28/vscode-sandbox

[39] Electron process model: The utility process. Accessed: 08/01/2024. [Online]. Available: https://www.electronjs.org/docs/latest/tutorial/process-model#the-utility-process

[40] Visual studio code issue: [feature request] extension permissions, security sandboxing & update management proposal. Accessed: 08/01/2024. [Online]. Available: https://github.com/microsoft/vscode/issues/52116

[41] A. Decan, T. Mens, and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018.

[42] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019.

[43] B. Pfretzschner and L. ben Othmane, "Identification of dependency-based attacks on node.js," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017.

[44] A. Zerouali, T. Mens, A. Decan, and C. De Roover, "On the impact of security vulnerabilities in the npm and rubygems dependency networks," *Empirical Software Engineering*, 2022.

[45] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and modeling the label dynamics of online Anti-Malware engines," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[46] Cloudflare radar. Accessed: 08/01/2024. [Online]. Available: https://radar.cloudflare.com/

[47] Visual Studio Marketplace, "Black box - visual studio marketplace," https://marketplace.visualstudio.com/items?itemName=nikhilmjeby.black-box, 2021.

[48] Software.com, "Code time," https://marketplace.visualstudio.com/items?\discretionary{}{}{}itemName=softwaredotcom.swdc-vscode, 2018.

[49] F5 nim. Accessed: 08/01/2024. [Online]. Available: https://marketplace.visualstudio.com/items?\discretionary{}{}{}itemName=F5DevCentral.vscode-nim

[50] E. Lin, I. Koishybayev, T. Dunlap, W. Enck, and A. Kapravelos, "Untrustide: Exploiting weaknesses in vs code extensions," in *31th Annual Network and Distributed System Security Symposium, (NDSS 2024)*, 2024.

[51] Å. David, "Implementation and evaluation of an emulated permission system for vs code extensions using abstract syntax trees," 2021.

[52] Z. Jin, S. Chen, Y. Chen, H. Duan, J. Chen, and J. Wu, "A security study about electron applications and a programming methodology to tame DOM functionalities," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023*, 2023.

[53] R. P. Kasturi, J. Fuller, Y. Sun, O. Chabklo, A. Rodriguez, J. Park, and B. Saltaformaggio, "Mistrust plugins you must: A Large-Scale study of malicious plugins in WordPress marketplaces," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[54] M. Zha, J. Wang *et al.*, "Hazard integrated: Understanding the security risks of app extensions on team chat systems," in *Network and Distributed Systems Security Symposium*, 2022.

[55] Y. Shen, P.-A. Vervier, and G. Stringhini, "A large-scale temporal measurement of android malicious apps: Persistence, migration, and lessons learned," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[56] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "Sok: Taxonomy of attacks on open-source software supply chains," in *IEEE Symposium on Security and Privacy (SP)*, 2023.

[57] D. Wermke, J. H. Klemmer, N. Wöhler, J. Schmüser, H. S. Ramulu, Y. Acar, and S. Fahl, ""always contribute back": A qualitative study on security challenges of the open source supply chain," in *IEEE Symposium on Security and Privacy (SP)*, 2023.

[58] M. Fourné, D. Wermke, W. Enck, S. Fahl, and Y. Acar, "It's like flossing your teeth: On the importance and challenges of reproducible builds for software supply chain security," in *IEEE Symposium on Security and Privacy (SP)*, 2023.

[59] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Extension breakdown: Security analysis of browsers extension resources control policies," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017.

[60] Q. Chen and A. Kapravelos, "Mystique: Uncovering information leakage from browser extensions," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[61] A. S. Buyukkayhan, K. Onarlioglu, W. K. Robertson, and E. Kirda, "Crossfire: An analysis of firefox extension-reuse vulnerabilities." in *NDSS*, 2016.

[62] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson, "Hulk: Eliciting malicious behavior in browser extensions," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[63] S. Bandhakavi, N. Tiku, W. Pittman, S. T. King, P. Madhusudan, and M. Winslett, "Vetting browser extensions for security vulnerabilities with vex," *Commun. ACM*, 2011.

[64] Y. Ling, K. Wang, G. Bai, H. Wang, and J. S. Dong, "Are they toeing the line? diagnosing privacy compliance violations among browser extensions," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2023.

[65] N. Jagpal, E. Dingle, J.-P. Gravel, P. Mavrommatis, N. Provos, M. A. Rajab, and K. Thomas, "Trends and lessons from three years fighting malicious extensions," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[66] A. Aggarwal, B. Viswanath, L. Zhang, S. Kumar, A. Shah, and P. Kumaraguru, "I spy with my little eye: Analysis and detection of spying browser extensions," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[67] X. Li, B. A. Azad, A. Rahmati, and N. Nikiforakis, "Good bot, bad bot: Characterizing automated browsing activity," in *IEEE Symposium on Security and Privacy (SP)*, 2021.

[68] P. Picazo-Sanchez, B. Eriksson, and A. Sabelfeld, "No signal left to chance: Driving browser extension analysis by download patterns," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022.

[69] N. Pantelaios, N. Nikiforakis, and A. Kapravelos, "You've changed: Detecting malicious browser extensions through their update deltas," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[70] D. F. Somé, "Empoweb: Empowering web applications with browser extensions," in *IEEE Symposium on Security and Privacy (SP)*, 2019.

[71] L. F. DeKoven, S. Savage, G. M. Voelker, and N. Leontiadis, "Malicious browser extensions at scale: Bridging the observability gap between web site and browser," in *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET 17)*, 2017.

[72] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy, "Verified security for browser extensions," in *IEEE Symposium on Security and Privacy (SP)*, 2011.

[73] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the eighth symposium on usable privacy and security*, 2012.

[74] A. P. Felt, S. Egelman, M. Finifter, and D. Wagner, "How to ask for permission," in *7th USENIX Workshop on Hot Topics in Security (HotSec 12)*, 2012.

[75] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011.