This provides an overall summary of results by threat model and suspicious type, showing the number of affected extensions and their install counts. It also includes a column for 'Artifacts,' where detailed evidence supporting these findings can be accessed, highlighting the transparency and thoroughness of the analysis.

| Threat | Suspicious Type | Extension Count | Cumulative Install Count | Artifacts |
| --- | --- | --- | --- | --- |
| Malicious | Degrading the Security Posture | 69 | 2,809,972 | Results-Dynamic-Analysis-(Verified-behavior-with-manual-execution)  &  Results-API-Analysis/NODE\_TLS\_REJECT\_UNAUTHORIZED.xlsx |
| Critical File Access | 14 | 1,564,468 | Results-API-Analysis/suspicious\_extensions\_critical\_file\_access.xlsx |
| VT >= 4 Extensions | 26 | 385,408 | Results-VirusTotal-Analysis/vt\_analysis\_filtered.xlsx |
| VT >= 4 Network Requests | 8 | 6,239 | Results-VirusTotal-Analysis/VT-DNS-Check |
| Market Misuse | 42 | 254,232 | Results-Dynamic-Analysis-(Verified-behavior-with-manual-execution) |
| Concealed Operations | 18 | 145,047 | Results-Dynamic-Analysis-(Verified-behavior-with-manual-execution) |
| Vulnerable | Extensions with CVEs | 2,620 | 51,952,070 | Results-RetireJs-Analysis |
| API & Privacy | Tracking | 49 | 3,107,508 | Results-Dynamic-Analysis-(Verified-behavior-with-manual-execution) |
| Code Sharing | 108 | 560,666 | Results-Dynamic-Analysis-(Verified-behavior-with-manual-execution) |
| Data Sharing | 15 | 504,835 | Results-Dynamic-Analysis-(Verified-behavior-with-manual-execution) |

This table contains a summary of the details about each folder/file in the provided artifacts repository.

| File Name (with file path) | Content Description |
| --- | --- |
| filtered\_extension\_details.json | Basic details about the identified extensions on the VS Code Extension Marketplace that are either malicious, vulnerable, or suspicious. |
| Results-API-Analysis/Category wise API calls.xlsx | Analysis of all extensions on how developers used API methods to develop the extensions, based on the extension category as listed in the extension manifest file. |
| Results-API-Analysis/settings.json file updates.xlsx | Analysis of extensions that modify the settings.json file of VSCode during runtime. |
| Results-API-Analysis/NODE\_TLS\_REJECT\_UNAUTHORIZED.xlsx | Analysis of extensions that degrade the TLS/SSL certificates while the extension is running or has started to run. |
| Results-API-Analysis/suspicious\_extensions\_critical\_file\_access.xlsx | This file contains the results of the analysis on critical file access, indicating whether an extension accesses critical files in the operating system. |
| Results-Dynamic-Analysis-(Verified-behavior-with-manual-execution)/ICSE Dynamic Analysis - Report.xlsx | This file contains the identified suspicious extensions details and summary of their behaviors. |
| Results-Dynamic-Analysis-(Verified-behavior-with-manual-execution)/Logs | This folder contains logs and flows related to several suspicious extensions.  Note that we have not included all the logs and flow files to maintain the anonymity of the authors, as these files contain information that could reveal their identities. The steps to generate the flows and logs are provided later in this document, should you need to create them. |
| Runtime-Instrumented-VSCode | VSCode application which we instrumented to find extensions' API usages in the VSCode extension API interface. Currently, we only support macOS and Windows applications. |
| Results-Manifest-File-Analysis/Extension-Dependency-Analysis/extension\_dependency\_list.json | A JSON file containing extension details (extension ID, name, and dependency extension names in the format publisher:extension name) on the extension and the dependent extensions that are required to be installed prior to the given extension. The JSON file contains an array of Python dictionaries with the above details. |
| Results-Manifest-File-Analysis/Extension-Dependency-Analysis/extension\_pack\_dependency\_list.json | A JSON file containing extension details (extension ID, name, and the set of other extensions to be installed with the given extension, listed in the format publisher:extension name). The JSON file contains an array of Python dictionaries with the above details. |
| Results-Manifest-File-Analysis/extension\_code\_repository\_data.json | This file contains details about each extension’s code repository. |
| Results-Manifest-File-Analysis/manifest\_file\_capabilities\_access\_list.json | This file contains details about extensions that use the `capabilities` option in the manifest file to access the VSCode runtime in untrusted mode. |
| Results-Manifest-File-Analysis/possible\_external\_link\_creating\_npm\_packages.xlsx | Identified npm packages used in VSCode extension development that may create external connections via the internet. |
| Results-Radar-Reports/extension\_dns\_report.json | This file contains the identified extensions that have external links associated with them. |
| Results-Radar-Reports/radar\_report\_http\_urls.json | This file contains the Cloudflare Radar report for the HTTP links identified in each extension, as listed in the `Radar Reports/extension\_dns\_report.json` report. |
| Results-Radar-Reports/radar\_report\_https\_urls.json | This file contains the Cloudflare Radar report for the HTTPS links identified in each extension, as listed in the `Radar Reports/extension\_dns\_report.json` report. |
| Reported-Cases/Dynamic-Testing-Reported-Cases | We reported the identified threats to the VSCode Marketplace, requesting their removal. This folder contains the report we provided concerning the identified threats discovered during our dynamic testing, which involved manually testing the extensions installed in our instrumented VSCode application. |
| Reported-Cases/Extension-Dependency-Installation-Cases | We reported the identified threats to the VSCode Marketplace, requesting their removal. This folder contains extensions that depend on such malicious or harmful extensions and will install these problematic extensions along with them. |
| Reported-Cases/Static-Analysis-Reported-Cases | We reported identified threats to the VSCode Marketplace, requesting their removal. This folder contains the reports we sent to the VSCode Marketplace team regarding the vulnerable packages used in the extension development process, which are packaged with the extensions without addressing those vulnerabilities. |
| Reported-Cases/VT-Reported-Cases | We reported identified threats to the VSCode Marketplace, requesting their removal. This folder contains screenshots of harmful VSCode extensions that were submitted to VirusTotal and returned with 4 or more VirusTotal engines indicating that the extensions are harmful. |
| Reported-Cases/Action-Taken-Cases | This document contains the extensions that were removed from the marketplace after we reported their malicious behavior to the VSCode Marketplace team. |
| Results-RetireJs-Analysis/retirejs\_vulnerability\_report.json | A JSON file containing extension details along with the vulnerable packages used in the extension's development and their associated vulnerabilities with CVE numbers. The JSON file is formatted with the extension ID as the key to hold these details. |
| Results-RetireJs-Analysis/CVEs-and-Details | A folder containing summarized CVE details along with the npm package names. Note that these are raw data files, which include download counts for the extensions but lack keys to identify each extension’s download count. |
| Results-RetireJs-Analysis/POC-on-cve-exploits.zip | We verified whether the high CVEs could be exploited. This sample extension addresses the high CVEs we identified based on RetireJS analysis. |
| Results-VirusTotal-Analysis/vt\_analysis\_filtered.xlsx | This file contains details about the identified extensions based on VirusTotal analysis. The document has two sheets: one for the VSIX file itself and another for the URLs that are benign and used in the extensions for external communications. |
| Data-VSIX-Files | A ZIP file containing the list of extensions selected for dynamic analysis. |

**How to Work with Instrumented VSCode**

1. Install the Application:
   1. Download and install the relevant application for either Windows or macOS.
2. Start the Application:
   1. Launch the application and grant full permission to trust the workspace.
3. Install Extensions:
   1. Download the extension(s) from the Data-VSIX-Files/v6Extensions.zip and install them via a .vsix file installation in the extension tab of the application.
4. Monitor Network Traffic (if required):
   1. Install mitmproxy to monitor external access.
   2. Configure the proxy settings in the instrumented VSCode.
5. View API Access Logs:
   1. Access each extension's API access logs in the /tmp/extensionLogs/ folder. Log files are created for each installed extension.
   2. Note: On a Windows machine, the temporary folder will be located in the disk partition where the application is installed.

**Dynamic Analysis (Verified behavior with manual execution)**

This project demonstrates the manual execution of Visual Studio Code (VS Code) extensions, logs the extensions' actions, captures network requests using the [mitmproxy tool](https://mitmproxy.org/), and checks for suspicious extensions. Each sheet corresponds to a result section in the paper, found in the Results section "Suspicious Extensions."

When replicating these experiments, you need to [download the extensions](https://drive.google.com/drive/u/4/folders/1faRAX9sdzxGsrUx_dOpXsFpqdMkvzg1n) using Extension Id (ex: f2fd462f-f1bd-4c62-b54f-59a4e5ffc6a3) and run them through instrumented VS Code.

Each column in the excel sheet represents the following:

* **Id**: Unique extension ID we created to search extensions
* **Extension Name**: Extension's name
* **Usage**: VS Code marketplace page description or basic functionality
* **Download Counts**: Download count at the VS Code marketplace page
* **Comment**: Suspicious behavior of the extension
* **Transparency**: Whether they describe the actual behavior in the VS Code marketplace page
* **Flow/Logs**: mitmproxy log file (Flow), and the logs captured when executing VS Code extensions locally.
* **Sharing Reason**: Purpose of the code sharing (Only in the Code Sharing sheet)
* **Tracking Data**: Types of data used for tracking (Only in the Tracking sheet)

To demonstrate code sharing, we have used the following sample javascript code file, which you might find in the flow files:

*"const forge = require('node-forge');\r\nconst fs = require('fs');\r\n\r\n// Generate a key pair (public and private keys)\r\nconst keys = forge.pki.rsa.generateKeyPair({ bits: 2048 });\r\n\r\n// Convert the keys to PEM format (for demonstration purposes)\r\nconst privateKeyPem = forge.pki.privateKeyToPem(keys.privateKey);\r\nconst publicKeyPem = forge.pki.publicKeyToPem(keys.publicKey);\r\n\r\n// Save the keys to files (in practice, keep these keys secure)\r\nfs.writeFileSync('private\_key.pem', privateKeyPem, 'utf8');\r\nfs.writeFileSync('public\_key.pem', publicKeyPem, 'utf8');\r\nconsole.log('Keys generated and saved.');\r\n\r\n// Function to sign a message\r\nfunction signMessage(privateKey, message) {\r\n const md = forge.md.sha256.create();\r\n md.update(message, 'utf8');\r\n const signature = privateKey.sign(md);\r\n return signature;\r\n}\r\n\r\n// Function to verify a signature\r\nfunction verifySignature(publicKey, message, signature) {\r\n const md = forge.md.sha256.create();\r\n md.update(message, 'utf8');\r\n return publicKey.verify(md.digest().getBytes(), signature);\r\n}\r\n\r\nconst message = 'This is a signed message';\r\n\r\n// Sign the message using the private key\r\nconst privateKey = forge.pki.privateKeyFromPem(fs.readFileSync('private\_key.pem', 'utf8'));\r\nconst signature = signMessage(privateKey, message);\r\n\r\nconsole.log('Message:', message);\r\nconsole.log('Signature:', signature);\r\n\r\n// Verify the signature using the public key\r\nconst publicKey = forge.pki.publicKeyFromPem(fs.readFileSync('public\_key.pem', 'utf8'));\r\nconst isSignatureValid = verifySignature(publicKey, message, signature);\r\n\r\nif (isSignatureValid) {\r\n console.log('Signature is valid.');\r\n} else {\r\n console.log('Signature is not valid.');\r\n}\r\n\r\n"*