Understanding and Detecting Abused Image Hosting Modules as Malicious Services

Geng Hong
Fudan University
ghong@fudan.edu.cn

Mengying Wu
Fudan University
wumy21@m.fudan.edu.cn

Pei Chen
Fudan University
peichen19@fudan.edu.cn

Xiaojing Liao
Indiana University Bloomington
xliao@indiana.edu

Guoyi Ye
Fudan University
yegy19@fudan.edu.cn

Min Yang
Fudan University
m_yang@fudan.edu.cn

ABSTRACT

As a new type of underground ecosystem, the exploitation of **A**bused **I**HMs as **M**al**I**cious s**E**rvices (*AIMIEs*) is becoming increasingly prevalent among miscreants to host illegal images and propagate harmful content. However, there has been little effort to understand this new menace, in terms of its magnitude, impact, and techniques, not to mention any serious effort to detect vulnerable image hosting modules on a large scale. To fulfill this gap, this paper presents the first measurement study of *AIMIEs*. By collecting and analyzing 89 *open-sourced AIMIEs*, we reveal the landscape of *AIMIEs*, report the evolution and evasiveness of abused image hosting APIs from reputable companies such as Alibaba, Tencent, and Bytedance, and identify real-world abused images uploaded through those *AIMIEs*. In addition, we propose a tool, called *Viola*, to detect vulnerable image hosting modules (*IHMs*) in the wild. We find 477 vulnerable *IHM upload APIs* associated with 338 web services, which integrated vulnerable *IHMs*, and 207 victim FQDNs. The highest-ranked domain with vulnerable web service is *baidu.com*, followed by *bilibili.com* and *163.com*. We have reported abused and vulnerable *IHM upload APIs* and received acknowledgments from 69 of them by the time of paper submission.

CCS CONCEPTS

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

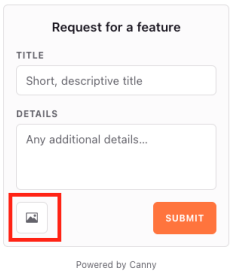
KEYWORDS

Web resource abuse, Vulnerability detection, Image hosting module, Cybercrime

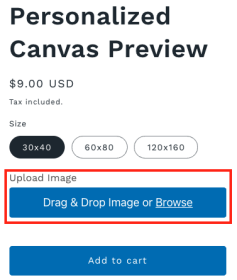
ACM Reference Format:
Geng Hong, Mengying Wu, Pei Chen, Xiaojing Liao, Guoyi Ye, and Min Yang. 2023. Understanding and Detecting Abused Image Hosting Modules as Malicious Services . In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 17 pages. *https://doi.org/10.1145/3576915.3623143*

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623143



(a) Service request form



(b) Product customization

Figure 1: Examples of web services integrated Image Hosting Modules (*IHMs*).

1 INTRODUCTION

In the digital era we live in, images are critical in facilitating communication and information sharing across diverse web services and platforms. Whether it's social media networks, e-commerce websites, or educational platforms, images enhance user engagement and improve the overall user experience. To enable seamless integration of images, many web services rely on image hosting modules (IHMs) to manage user image uploading, hosting, and sharing. For example, a Q&A platform may integrate an *IHM* to allow users to include images as supplementary materials for their questions, while online instant messaging services (*e.g.,* WhatsApp, Telegram) allow customized memes to be uploaded with the help of an *IHM*. An example of *IHM* is shown in Figure 1.

Abused *IHMs* as malicious services. However, there exist several reports [36, 49] about abused *IHMs* as malicious services, or *AIMIE*, where the adversary identifies and exploits vulnerable *IHM upload APIs* of image hosting modules (*IHMs*) to create a malicious service. The implications of such abused *IHMs* on security are substantial. First, miscreants can exploit vulnerable *IHMs* to store a considerable volume of images, wasting a large amount of storage space in the victim sites and a subsequent increase in storage expenses. Also, it introduces the risk of miscreants leveraging abused *IHMs* to store and reference illicit content, such as explicit images (as measured in Section 4.3). The association of such content with the company's domain will raise serious concerns regarding reputation and brand integrity. Despite previous research focusing on the abuse of various web resources, such as cloud computing services [13, 29, 48, 72], continuous integration pipelines [42] and DNS [5, 15, 41], *IHMs*,

which are a crucial web service infrastructure, have been largely neglected. To the best of our knowledge, little effort has been made to understand the security implications of *IHMs*.

To fulfill this gap, this paper presents the first measurement study on real-world *AIMIEs*. Specifically, we develop a systematic approach to collect a set of known abusive *open-sourced AIMIEs* from GitHub and examine their underlying infrastructure, including abused *IHM upload APIs* and victim *image hosting domains*. Armed with this methodology, we discover 89 *open-sourced AIMIEs*, along with 109 unique abused *IHM upload APIs* and 122 victim *image hosting domains*. We observed a significant increase in the number of *open-sourced AIMIEs*, which has grown from 18 to 89 in the last three years. Moreover, we found that newly emerging abused *IHM upload APIs* have increased 494% compared to three years ago. Further, our study reveals the high severity of *AIMIE* abuse: one *commercial AIMIE* `uomg` [68] has been requested 407 million times in three years based on the passiveDNS database, which has a similar popularity to *www.onenote.com* (see Section 4.2.2). Our study also sheds light on the connection between *AIMIEs* and real-world abused image hosting. Specifically, we investigated the extent to which reputable companies' servers were being used to host illicit images via *AIMIEs*. Particularly, we discover that 1,151 explicit images have been uploaded through 26 *IHM upload APIs* we identified in our study. We have reported our findings to the affected companies including Alibaba, Tencent, and Bytedance.

Detecting vulnerable *IHMs* in the wild. Apart from reporting the abuse of *IHM upload APIs* by *AIMIEs*, we are also interested in understanding the prevalence of vulnerable *IHM upload APIs* in the wild. To this end, we develop *Viola*, a tool to assess the security of *IHMs*. *Viola* first recognizes web services that implement *IHMs* in a given domain, and then analyzes each stage of the image upload lifecycle - presubmit, preview, submit, and callback - to determine if there are any interfaces that can be accessed and exploited by third parties to upload and host malicious images (see Section 5). *Viola* flagged 338 vulnerable *IHMs* associated with 477 *IHM upload APIs* in the wild. The most highly-ranked domain with vulnerable *IHMs* is *baidu.com* (ranked 8 based on Tranco List [39]), followed by *bilibili.com* (15) and *163.com* (61). We have reported vulnerable *IHM upload APIs* found in our study to 311 victim websites, and received acknowledgments from 69 of them as of paper submission. During our discussion with the developers about the root cause of such vulnerability, some of them admitted that they had not previously considered this new type of abusive service.

To aid website developers in addressing this vulnerability, we go beyond just outlining the abuse itself. We provide practical migration recommendations tailored to different web services. These suggestions encompass a range of strategies, such as opting for client-side caching as opposed to server-side caching, incorporating access control measures for uploaded images, ensuring the alignment of image hosting content and resources with specific purposes, and safeguarding uploaded image resource paths (see Section 6.1).

Contributions. This paper makes the following contributions:

- We conduct the first systematic study of *AIMIE* service, and uncover the characteristics of this malicious service including their infrastructure, workflow and abused artifacts (*e.g.*, abused *IHM upload APIs* and victim *image hosting domains*).

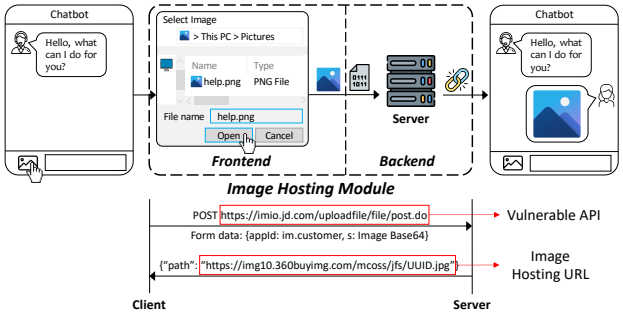


Figure 2: The *IHM* workflow. A user uploads an image via the *IHM*, which then communicates with the backend server to display the processed image in the chatbot interface.

- We model the image upload lifecycle of *IHMs* and propose a tool called *Viola*, which can effectively and accurately discover vulnerable *IHMs* and *IHM upload APIs* in the wild.
- We offer actionable recommendations to help developers minimize the threat of *AIMIE* and mitigate *IHM* abuse.
- We report vulnerable *IHM upload APIs* found in our study to 311 victim websites, including major platforms such as Alibaba, Baidu, Tencent, and Bytedance.
- We release our code and data at [8].

2 BACKGROUND

2.1 Image Hosting Module

Web image hosting service can be categorized as types of infrastructure-as-a-service or code module-as-a-service. For instance, image hosting platforms like Flickr [20], Imgbb [32] fall under the category of infrastructure-as-a-service, where users can purchase a *public* image hosting service to host and share their images. Meanwhile, image hosting module (*IHM*) like image upload function within a chatbot (Figure 1) can be considered examples of code module-as-a-service. Typically, such services provide an upload interface – a form in which users specify the location of an image file on their local computer file systems. Once the image is uploaded and hosted on the server, it can be accessed or shared.

It's important to note that, an *IHM* is *not* intended to function as a standalone image hosting platform. Instead, *IHM* typically serves as a functionality component/feature within a web service (*e.g.*, chatbot, account registration, Q&A platform) that incorporates it. Additionally, the hosting web service of an *IHM* sometimes restricts the type of images that can be uploaded or shared via *IHMs* in alignment with the rules and policies of the hosting web service. For example, some *IHM* might display a notification reminding users to only upload "appropriate" images, such as "upload your medical records", "uploaded image should not include personal contact information", or "no pornographic or terrorist images". Also, some websites outline their term of service [10, 60, 73], that explicitly prohibit the dissemination of pornography, gambling, intimidatory, racially discriminatory, or related content while utilizing their services.

Workflow. A typical workflow of an *IHM* is illustrated in Figure 2. In this example, we consider an *IHM* that supports the functionality of a customer service chatbot, allowing users to upload

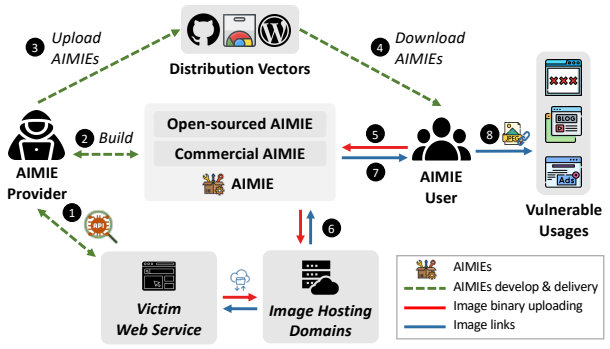


Figure 3: *AIMIE* workflow. Vulnerable *IHM upload APIs* in a web service's *IHM* are wrapped into *open-sourced AIMIEs* or *commercial AIMIEs* by *AIMIE* providers. *AIMIE* users discover *AIMIEs* in the distribution vectors and utilize them to upload and host images on victim *IHMs* servers.

images to supplement the information of their customer service requests. First, a user clicks the image uploading button to initiate the *IHM* and selects an image from their local device. This process is implemented through the HTML element `<input type="file">` on the client side. The *IHM* then submits a POST request to an image *IHM upload API imio.jd.com/uploadfile/file/post.do* with the payload of the base64 encoding of the selected image and associated chat information. Once the image is uploaded, it is hosted on the *image hosting domain img10.360buyimg.com* with the URL: *https://img10.360buyimg.com/mcoss/jfs/UUID.jpg*. Finally, the *IHM* resolves the image hosting URL from the POST response of *IHM upload API* and displays the image in the customer service chatbot. In this case, the customer service chatbot represents a web service, which is embedded with an IHM. A FQDN can consist of multiple web services on the same or different webpages. Note that we differentiate between the web service and the IHM, aiding in recognizing the non-image hosting web services and identifying *IHM* abuse (Section 5).

In our study, we observe that miscreants offer an abusive service that allows users to upload, host and access arbitrary images in vulnerable *IHMs* (see Section 3). Here we consider this abuse as an exploitation of *IHM* functionality, and the *IHM upload API* is an abused API that can be exploited by miscreants.

2.2 Threat Model

In our study, we focus on investigating "image hosting modules" as functional components or features within specific web services (*e.g.,* chatbot, account registration, Q&A platform). In this context, we consider an adversary who seeks out vulnerable *IHMs* that can be exploited to host and retrieve arbitrary images for malicious purposes. To achieve this goal, the adversary identifies and exploits vulnerable *IHM upload APIs* of *IHMs* to create a malicious service, *i.e., AIMIEs*. We assume that the adversary has the capabilities of an ordinary user, requiring no special privileges or access to the victim website. Additionally, the uploaded images do not need to carry any malicious payload to exploit vulnerabilities in victim servers.

AIMIEs. In our paper, we categorize *AIMIEs* into two types: commercial AIMIE, where the malicious service is created for profit, and open-sourced AIMIE, where the malicious service is developed and distributed as an open-source project. Our preliminary study of *AIMIEs* (see Section 3), indicates that the service follows a typical workflow, as depicted in Figure 3.

An AIMIE provider identified vulnerable *IHM upload APIs* of an *IHM* associated with a web service (❶). After that, the AIMIE provider wrapped those vulnerable *IHM upload APIs* to create an *open-sourced AIMIE* or a *commercial AIMIE* (❷), and released it through multiple distribution vectors, such as third-party library, docker image, or browser extension (❸). AIMIE users seeking to host arbitrary images discovered these *AIMIEs* in package-management systems (*e.g.,* PyPI [1], Maven [21]) or extension stores (*e.g.,* Chrome Web Store [26]) (❹). After that, users will interact with the *AIMIEs* (❺) via GUI or API to upload and host arbitrary images on victim *IHMs*' servers (❻). The *AIMIE* will return the image hosting URLs to users (❼), who might embed them on their own websites, such as explicit images for adult sites or advertising for online gambling sites (❽).

2.3 Scope of Problem

Our research focuses on the exploitative misuse of the *IHMs*, which are tailored to support specific functionalities of their associated hosting web services, rather than catering to public image hosting platforms. It's worth noting that, in this study, the term "abuse" refers to any usage of these modules that deviate from their intended purposes, *i.e.,* not for public image hosting, regardless of whether the hosted images contain explicit content or not, or adhere to content policies that restrict uploaded materials.

We acknowledge that image hosting platforms (*e.g.,* Flickr), which are designed for public image hosting, could also potentially be misused for unintended image hosting purposes. However, our study does not focus on an analysis of user compliance with the terms of service for these public image hosting platforms. Our scope remains limited to the abusive exploitation of specialized *IHMs* within their hosting web service environments.

3 ABUSED *IHM* AS MALICIOUS SERVICE

In this section, we present our analysis of a collection of open-sourced *AIMIEs*. We first provide an overview of these *AIMIEs*, followed by a detailed explanation of our methodology for gathering and profiling these entities.

3.1 Overview of *AIMIE*

To describe the workflow, we define key artifacts in an *AIMIE*:

• AIMIE upload API: an AIMIE upload API is an interface offered by an *AIMIE* service, facilitating users in uploading images. Typically, these upload APIs are associated with vulnerable *IHMs*, image hosting platforms, or other open-sourced/commercial *AIMIEs*. An upload API is usually in the form of domain and path, *e.g., victimhost.com/uploadfile/file/post.do*, specifying in the `HOST` and `PATH` fields in the POST request, respectively. *In our study, we specifically focus on the abused upload APIs of vulnerable IHMs in AIMIEs. Those upload APIs are intended for internal use by the IHM and not typically accessible to the public for image hosting, sharing and storage.*

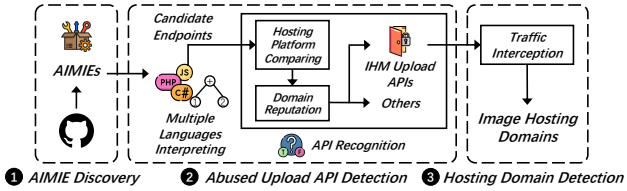


Figure 4: Methodology overview. We begin by identifying *open-sourced AIMIEs* on GitHub, then locate APIs using AST-based approach. Next we triage abused *IHM upload APIs* of vulnerable *IHMs*, and detect the image hosting domains by intercepting traffic.

• Image hosting domain: the domain of a victim image hosting server (*e.g., IHM*), which hosts and stores uploaded images.

The *AIMIE* ecosystem is flourishing with various entities, including *open-sourced AIMIEs*, such as `auxpi` [3], `CDNDrive` [9] and *commercial AIMIEs*, *e.g.,* `yum6` [77]. Specifically, *commercial AIMIEs* usually offer a variety of hosting plans and flexible service models. Users can upload an image through the base64 encoding or with the binary format, even an online image by specifying the resource URL. However, rare *commercial AIMIEs* set restrictions for image content to prohibit child pornography or terrorism content, while the image size limitation of the uploaded images varies from 5MB to 10MB. Also, *commercial AIMIEs* allow users to specify the victim *image hosting domain* they want to host images. In terms of pricing, compared with legal image hosting platforms (*e.g.,* Imgur [43] and Imgbb [32]), *commercial AIMIEs* offers affordable hosting plan, *e.g.,* `ALAPI` [6] provides a free plan for uploading 1,000 images per day, $5 for 100 million images, and $15 for unlimited images. To improve the usability, some *open-sourced AIMIEs* allow users to integrate their codes as third-party libraries, *e.g.,* `CDNDrive` [9] can be downloaded and installed via the official Python Package Manager (PyPI) while `wbp4j` [17] via Maven package managers. In our study, we also observed *commercial AIMIEs* as Firefox Add-ons, WordPress Plugins and Docker Images.

3.2 Methodology

Figure 4 illustrates our measurement methodology to collect and profile *open-sourced AIMIE*, which consists of three components: the discovery module for collecting *open-sourced AIMIEs* in the wild, the abused API recognition module for identifying abused upload APIs of vulnerable *IHMs* (*i.e., IHM upload APIs*), and the abused hosting domain identification module for examining abused image hosting domains of victim *IHMs*. We elaborate them as follows.

3.2.1 AIMIE discovery. To understand the design and implementation of the *AIMIE*, we started with collecting a set of *open-sourced AIMIEs* for analysis. Specifically, searching keywords, *e.g.,* "image hosting", and "picture free upload", on GitHub, the largest open-source code-sharing platform [25], we found 508 *open-sourced AIMIE* candidates. We scrutinized documentation, git commit logs, and comments to confirm both the abusive behavior and the services provided by the *AIMIE*. For example, `auxpi` [3] is an *open-sourced AIMIE* on GitHub with a code repository and detailed documentation. Its README file points out that its functionality is to upload and monitor the lifespan of images, and the repository contains multiple abused IHM upload APIs from popular reputable companies (but not image hosting platforms), such as *JD [35], Sina [57]*. We also find that, in Jan 2019, it committed with the message – "add many picture upload interfaces".

Results. To this end, we confirmed 89 *open-sourced AIMIEs* (see [8]) by examining 508 candidates. We elaborate on the analysis of these *open-sourced AIMIEs* in Section 4.1.

3.2.2 Abused upload API detection. After confirming 89 open-sourced *AIMIEs*, we then engaged in the profiling of these *AIMIEs* to effectively identify the abused upload APIs of vulnerable *IHMs*. Specifically, to extract abused *IHM upload APIs* from *open-sourced AIMIEs*' code repositories, we develop a language interpreter that utilizes a set of parsers to extract upload APIs from source codes written in multiple programming languages, as well as differentiating abused *IHM upload APIs* from other AIMIE upload APIs.

Abused API recognition. In this step, we develop a multi-language interpreter to extract abused APIs scattering over the *open-sourced AIMIE*. Specifically, the interpreter will parse the source code of each *open-sourced AIMIE* and generate the Abstract Syntax Tree (AST). After that, the interpreter extracts all strings by interpreting the string-related operations in the ASTs, and validates whether the strings points to network-related APIs. For these APIs, the interpreter further examines whether they relate to image uploading, as elaborate below.

To retrieve all the strings on the ASTs, we build our interpreter on top of open-source parsers (*i.e.,* py-tree-sitter [67]) to the five most popular programming languages (JavaScript, PHP, Python, Golang, and Java) in *open-sourced AIMIEs*. After that, the interpreter traverses the AST, performing semantic analysis, especially for string construction and assignment operations. To achieve this, we model different string operations of multiple programming languages, such as the plus binary operator (*e.g.,* +) and the format function (*e.g.,* `string.format()`). The interpreter then analyzes the subtree rooted at these operations to extract the relevant information. Additionally, to retrieve variable values during concatenation, we store the results of all variable assignment operations. To locate the network-related URLs from the strings, we extract the URLs that start with *http(s)*. The intuition behind this is that most programming languages (*e.g.,* Python) and popular network libraries (*e.g.,* Requests [2]) require the developer to explicitly specify network protocols, *e.g.,* HTTP, and HTTPS, when initiating network requests.

To determine whether those URLs are candidate APIs that can be used for uploading, we investigate semantic information (*e.g.,* FQDN, path) of each URL. Our approach is based on the insight that context in static code provides sufficient semantic information. For example, a string representing the uploading destination is likely to be assigned to a variable named *"uploadAPI"*. Thus, we investigate whether each URL, including its FQDN, path, corresponding variable names, function names and comments, contains uploading-related keywords (such as "upload" or "image") to determine whether it can be used for image uploading.

***IHM upload APIs* triage.** With a set of API candidates output by the above multi-language interpreter, our approach further triage abused *IHM* upload APIs and other AIMIE upload APIs. More specifically, we recognize an API as *IHM upload APIs* if an API resides on

a website that was not designed for image uploading while having the image-uploading capability. Specifically, we compile a list of image hosting platforms by utilizing search engines, resulting in a list of 98 domain names associated with such platforms. The full list of domains can be found at [8]. Meanwhile, as *IHM upload APIs* come from reputable companies, we take abused APIs from the top 500k domains of the Tranco list as *IHM upload APIs*.

Results. Among 89 *open-sourced AIMIEs*, we found most of them implemented in JavaScript (46), followed by PHP (16), Python (7) and Golang (7), and Java (6). We implemented the interpreters of the above top-five languages, which cover 92.1% of *open-sourced AIMIEs*. To this end, we collect 265 candidate APIs. Further, we detected 109 unique abused *IHM upload APIs* (267 in total).

Evaluation. To assess the performance of our approach in identifying *IHM upload APIs*, we randomly selected 10 *open-sourced AIMIEs* and evaluated the precision and recall of the identified *IHM upload APIs*. We manually reviewed the semantics of the source codes of 10 *open-sourced AIMIEs* to label all *IHM upload APIs*, resulting in 51 *IHM upload APIs* as the ground truth. After applying our interpreter, we extracted 56 *IHM upload APIs*, of which 51 *IHM upload APIs* were confirmed as true positives, yielding 91.07% precision/100.00% recall. False positives (FPs) mainly come from the following two scenarios. Firstly, in certain instances, the *IHM upload APIs* necessitate users to invoke an API in order to acquire an upload token. This has led to three FPs due to the semantic similarity between this specific API and the *IHM upload API* itself (*e.g.,* "https://upload.domain/token.php"). Secondly, within certain *open-sourced AIMIEs*, there exist two hard-coded image hosting URLs (*e.g.,* "https://hosting.domain/"+id). These URLs include the anticipated keywords; however, they are not upload APIs but associated with hosting images.

In addition, we compared our AST-based method with a naive regex-based method. In the regex-based method, we employed regular expressions to match all URLs present in the source codes. Subsequently, we applied the same keyword-based filter and triage strategy as utilized in our proposed method to ascertain whether these URLs constituted abused *IHM upload APIs*. In particular, we evaluate the regex-based method over the same groundtruth dataset we used above. Running on this set, the regex-based method reported 45 *IHM upload APIs*, of which 30 were confirmed as true positives, resulting in a precision of 66.67% and a recall of 58.82%. This indicates that our AST-based method outperforms the naive regex-based method. One key reason is that some *IHM upload APIs* were implemented as complex structures that require semantic analysis. For example, the URL of an *IHM upload API* is formed by combining a base URL (*e.g.,* "http://upload.domain/upload.php") with additional parameters or query strings (*e.g.,* "?mime=image&parameters"). Such intricacies can often escape simple regular expression matching.

3.2.3 Abused hosting domain detection. To detect the abused host domains, we deploy 14 *open-sourced AIMIEs*, which covered all 109 *IHM upload APIs* found in our study, in virtual machines with Ubuntu 20.04. Here we manually examine configuration files and documentation to set up all necessary dependencies and components, such as Nginx, MySQL, and Redis services. After that, we trigger each *IHM upload API* in *open-sourced AIMIEs* and log response traffic using `burpsuite` [52] and `Proxychains` [28]. Here we used `Proxychains` to instruct the *open-sourced AIMIEs* to communicate with its backends through the `burpsuite`, which is used for logging the network traffic. To validate the response traffic, we inspect the traffic payload to ensure it indeed consists of the image we upload.

We further confirm the abused *IHM upload APIs* we found (see Section 3.2.2) and build the mapping between abused *IHM upload APIs* and the associated image hosting URLs with the purpose of helping track and analyze more malicious activities. Particularly, we parse the payload of each abused *IHM upload API*'s POST response to extract the URL within it. This extracted URL is then considered as the associated image hosting URL for the respective abused *IHM upload API*. Subsequently, we manually examined the results to confirm that these APIs contain specific semantics, indicating that they are not employed for storing content originating from other APIs.

Results. We confirm 76 valid abused *IHM upload APIs* and map them to 122 *image hosting domains*. We found that the 33 invalid *IHM upload APIs* are expired due to the update from the victim companies, which will be discussed in Section 4.3.

Table 1: Prominent *open-sourced AIMIEs* with over 100 stars.

open-sourced AIMIE	# of Stars	# of Forks	Language	# of Abused APIs	# of Host Domains
0xDkd/auxpi	2,636	377	Go	12	22
apachecn/CDNDrive	668	90	Python	11	34
Mikubill/transfer	630	91	Go	9	12
ShareX/CustomUploaders	372	228	sxcu	2	0
szvone/imgApi	183	47	PHP	2	13
iAJue/Alibaba_pic	173	88	PHP	1	4
BlueSkyXN/KIENG-FigureBed	119	153	PHP	9	0

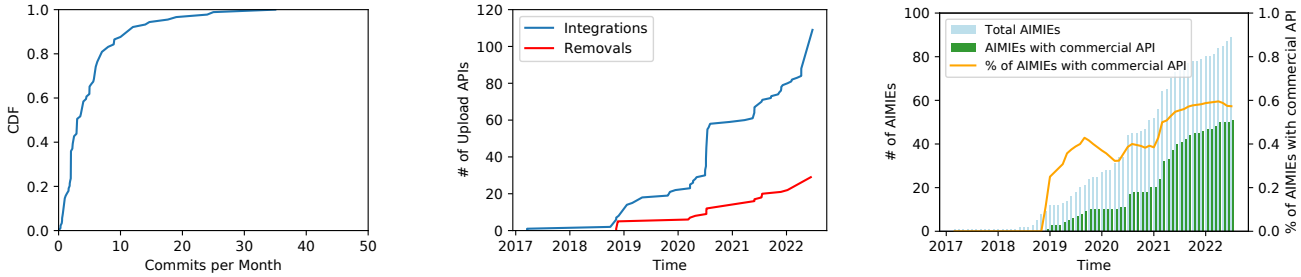
3.3 Limitations

In our study, we acknowledge that there may be some abused *IHM upload APIs* that we have missed. One reason for this is that the *AIMIEs* used in our study were collected from open-source platforms, and it is possible that there are *IHM upload APIs* that we have not discovered in the wild. To address this limitation, we have developed and open-sourced a vulnerable *IHM* scanner (see Section 5). Another reason why we may have missed some abused *IHM upload APIs* is that our collection process may have overlooked repositories that do not contain specific keywords that we searched for. Additionally, some *AIMIEs* may use countermeasures to bypass security auditing, such as dynamically loading APIs from the server side, which may evade our interpreter. Despite these limitations, we believe that our measurement approach provides valuable insights into the open-sourced *AIMIEs*.

4 MEASUREMENT

4.1 Open-sourced *AIMIEs*

In total, we collect and analyze 89 unique *open-sourced AIMIEs*, which exploit 109 *IHM upload APIs* and abuse 127 *image hosting domains*. The *open-sourced AIMIEs* are built with various languages, such as JavaScript, PHP, and Golang. Their lines of code (LoC) range from 717 to 206,891. We observe *open-sourced AIMIEs* as a thriving ecosystem: the earliest *open-sourced AIMIE* found in our study was in 2017. The number of *open-sourced AIMIEs* has risen rapidly, about five times than three years ago (18 in Aug. 2019 vs 89 in Aug. 2022). Throughout the duration of our study, we found that none



(a) *open-sourced AIMIE* Update Frequency **(b) *open-sourced AIMIEs* API Integration and Removal** **(c) *commercial AIMIE* Used in *open-sourced AIMIEs***

Figure 5: *AIMIE* Landscape & Abused *IHM upload API* Analysis

Table 2: Top 10 *IHM upload APIs* integrated by *open-sourced AIMIEs.*

Domain	Path	# of *open-sourced AIMIE*	% of *open-sourced AIMIE*
kfupload.alibaba.com	/mupload	23	25.84%
you.163.com	/xhr/file/upload.json	13	14.61%
search.jd.com	/image	11	12.36%
mp.toutiao.com	/upload_photo	11	12.36%
pic.sogou.com	/pic/upload_pic.jsp	10	11.24%
cdn-ms.juejin.im	/v1/upload	9	10.11%
review.suning.com	/imageload/uploadImg.do	9	10.11%
picupload.service.weibo.com	/interface/pic_upload.php	8	8.99%
prntscr.com	/upload.php	7	7.87%
changyan.sohu.com	/api/2/comment/attachment	7	7.87%
shopapi.io.mi.com	/homemanage/shop/uploadpic	7	7.87%

of the *open-sourced AIMIE* were delisted by GitHub. To address this concern, we have brought this matter to the attention of GitHub's security team, who are currently investigating these issues.

Developers of *open-sourced AIMIEs.* To understand how many developers contributing to the *open-sourced AIMIE*, we refer to authors in the commit logs. Particularly, here we use the email address of the commit author to pinpoint an individual code contributor. In total, we found 294 developers among the 89 *open-sourced AIMIEs*. Interestingly, there are nine code contributors contributing to at least two *open-sourced AIMIEs*. Examining their commit logs, we found these active code contributors add abused *IHM upload APIs* into multiple *open-sourced AIMIEs*.

Evolution. To understand how often they update and for which reasons they update, we study the commit history of the *open-sourced AIMIEs* by inspecting the update frequency and content. As shown in Figure 5a, we found about 56.2% of *open-sourced AIMIEs* were updated more than three times per month, and 13.5% were updated more than 10 times per month. Among the updates, 4.5% of them are related to abused *IHM upload APIs*. The frequent and long-lasting updates indicate most of *open-sourced AIMIEs* are carefully maintained by its developers. These changes are related to functionality upgrades, such as bug fixes, and better UI, other than updating abused APIs. This difference may indicate that the abused API is relatively stable and victim websites are unaware of this abuse (see Section 4.2).

Impacts of *open-sourced AIMIEs.* To evaluate the popularity of *open-sourced AIMIEs*, we study the number of stars and forks of *open-sourced AIMIEs*' GitHub repositories. Among the 89 *open-sourced AIMIEs*, seven have a minimum of 100 stars on GitHub, 15 have received between 10 and 100 stars, and the remaining 67 have less than 10 stars. Table 1 lists the seven *open-sourced AIMIEs* with more than 100 stars. The most popular *open-sourced AIMIE* is `auxpi` [3], which provides the image hosting service with both API and the easy-to-use web GUI (graphical user interface). It received over 2.7k stars. `auxpi` integrated 12 abused *IHM upload APIs* of *IHMs* from big tech companies, including Alibaba, Bytedance, Xiaomi, *etc.* Even worse, the `auxpi` library has been forked over 380 times. Piles of abusive repositories are created based on it for various purposes, such as online shopping or content management system (CMS).

4.2 Abused *IHM Upload API* Analysis

In this subsection, we profile the abuse situation of abused *IHM upload APIs*' installation, spread, and deletion to understand the evolution of *open-sourced AIMIEs*.

4.2.1 Abused IHM upload APIs. In total, 89 *open-sourced AIMIEs* recruit 109 *IHM upload APIs*, belonging to 65 popular companies. Table 2 list the top-10 *IHM upload APIs* integrated by the 89 *open-sourced AIMIEs*. The *kfupload.alibaba.com/mupload* is the most popular *IHM upload API*, which is integrated by 25.84% *open-sourced AIMIEs*, followed by *you.163.com/xhr/file/upload.json* (14.61%) and *search.jd.com/image* (12.36%). Also, we found that none of the abused *IHM upload APIs* were on a blocklist of Google Safe Browsing [33] and VirusTotal [70], implying that the *IHM upload APIs* can avoid detection while hosting inappropriate content.

Abused *IHM upload API* integration and removal. To investigate the evolution trend of abused *IHM upload APIs* in *open-sourced AIMIEs*, we study when the abused *IHM upload API* was integrated and removed. Specifically, we parse the git commit logs associated with *IHM upload APIs* in *open-sourced AIMIE* code repositories by inspecting the collected abused *IHM upload APIs* that appeared in the git changes between adjacent versions. To this end, among 2,682 git logs from 89 *open-sourced AIMIEs*, we identify 121 git logs (109 integrations and 21 removals) associated with 109 *IHM upload APIs*. As shown in Figure 5b, during *open-sourced AIMIE* integrates newly discovered vulnerable *IHM upload APIs* frequently, but only a few repositories remove them. The majority of removals are to remove the out-of-service APIs. The significant disparity between API integrations and removals suggests that this threat has become widespread over time. The average integrated rate of newly-found *IHM upload APIs* is experiencing phases of fast rise

from December 2018 to January 2019. The *open-sourced AIMIEs*, `Figure-bed` [64] and `upimg-mirror` [69], integrated three and six newly-found *IHM upload APIs*, respectively, in the two months. Furthermore, an extremely rapid growth of newly-found *IHM upload APIs* was seen around July 2020. The reason for such growth is the `TransparentLC/free-img`, `likeyun/likeyun-imgupload` and `chwl66/image`, added/updated a total of 22 *IHM upload APIs* in the same month. Besides that, they actively mined 39 newly-found APIs during their lifespan, efficiently facilitating others to abuse such vulnerabilities.

Other repositories with abused *IHM upload APIs.* As mentioned earlier, we discovered 109 *IHM upload APIs* from the *open-sourced AIMIEs*. To understand how such APIs are being used in other code repositories, we search the 109 abused *IHM upload APIs* through the GitHub searching API [25]. Particularly, we search the FQDN of the APIs, and leverage multiple language interpreter (see Section 3.2) to reconstruct the incomplete URL, and subsequently verified if the URL corresponded to any *IHM upload APIs*. We exclude *open-sourced AIMIEs* and non-code repositories (*e.g.,* personal blogs). This process revealed 193 repositories with abused *IHM upload APIs*. We classified these repositories into three categories based on GitHub topic tags: toolbox, website (CMS), and mini-program.

• *Toolbox* repositories provide various functionality to users, *e.g.,* video slicing, image compression. The abused *IHM upload APIs* served the image hosting functionality of these repositories. For example, the `WechatMomentScreenshot` [66], a widely-used counterfeit screenshot generation tool with 400 forks and 2.6k stars, stores generated screenshots Alibaba's servers via abused *IHM upload APIs*.

• *Website* repositories are those for content management systems (CMS), personal blogs or web platforms (*e.g.,* online malls). Those repositories customize Mac-CMS templates [45] by implementing their image hosting functions using abused *IHM upload APIs*.

• *Mini-program* are mini-applications [44] that run within mobile host apps. Most of the mini-programs with abused *IHM upload API* are online shopping malls, implementing their image hosting for displaying merchandise using abused *IHM upload APIs*.

4.2.2 Commercial AIMIE services. Apart from the abused upload APIs of vulnerable *IHMs*, looking into the other image uploading APIs, interestingly, we observed that open-sourced *AIMIEs* integrated image upload APIs provided by commercial AIMIEs as their own upload APIs (see Section 3.1). Specifically, we looked into the remaining APIs from our triage tool (see Section 3.2.2), which did not design for image uploading but not from top 500K Tranco domains. Particularly, we visited those FQDNs and checked whether they offered *AIMIE* services. The outcome was the confirmation of 89 upload APIs provided by 13 *commercial AIMIEs* and integrated by 51 *open-sourced AIMIEs*. As shown in Table 3, the upload APIs of several prominent commercial AIMIEs (such as yum6, kieng, and hualigs) have been incorporated into multiple distinct open-sourced AIMIEs. Also, these commercial AIMIEs often offer either free or partially free services, which facilitates their adoption by various open-source AIMIE projects and enables a more extensive reach of the abusive behavior. Additionally, we observe the trend of incorporating upload APIs of *commercial AIMIEs* into *open-sourced AIMIEs*. As shown in Figure 5c, following the initial observation of a *commercial AIMIE*, it has subsequently been embraced by over 15.7% (14 out of 89) of the *open-sourced AIMIEs* over a span of four years.

Query volume of *commercial AIMIE*. The query volume of *IHM upload APIs* is an important metric that indicates the abuse severity of *AIMIE*, however, we can hardly tell the query volume from abusers since they are hiding in a large number of normal query requests. Luckily, as the *commercial AIMIE* have their own domains to forward the abused *IHM upload API* query traffic, we can estimate the query volume those APIs using passiveDNS databases.

We use a commercial passiveDNS database [34] to gauge DNS activity for a given *commercial AIMIE*. This database contains the DNS resource records from all successful DNS resolutions observed at a main ISP, from July 1st, 2019 to the present. We sum up the query volume of each *commercial AIMIE* from the first time it was observed in the passiveDNS database to December 2nd, 2022, as shown in Table 3. The total query volume ranges from 827 to 407,785,813, with daily average ranges from 5.82 to 326,228.65. For instance, `uomg`, one of *commercial AIMIEs*, has been queried up to 407M times in three years, has a similar popularity to *www.onenote.com*, which accepts 388M queries within the same time frame.

4.3 Images hosted via *AIMIEs*

In our study, we estimate the number of abused images hosted via *AIMIEs* by looking into explicit images which are hosted on the *image hosting domains* of the *AIMIEs*, which are reputable domains that prohibit explicit images. To accomplish this, we relied on the fact that images uploaded through specific *IHM upload APIs* are typically stored in the same resource directory path, which helps developers maintain an organized file structure. Using this observation, we checked whether URLs of explicit images matched the patterns of URLs of uploaded images that we detected in Section 3.2.3. If so, we inferred that the explicit image was uploaded through an *AIMIE*.

Specifically, we collected 813,577 webpages from the three sources (PhishTank [27], ultimate-blocklist [11], and a commercial list from an anonymous company). Utilizing an explicit content classifier ([76]), which utilizes a convolutional neural network (CNN) trained on a large-scale dataset of labeled images to analyze the visual features, we find 243,447 explicit images. We then generate the patterns from the URLs of uploaded images. Specifically, we first cluster the URLs through the Levenshtein distance into groups. For URLs within a group, we mask out the differences between them and treat the rest as abused image hosting patterns. For example, on Tencent clouds customer service [61], images uploaded through the abused API *yzf.qq.com/fsnb/kf-file/upload_wx_media* will be hosted on the victim server with the following pattern: *yzf.qq.com/fsnb/kf-file/kf_pic/UUID.jpg*. If the hosting URL of the explicit image matches the pattern, we report it as an abused image through *AIMIEs*.

Results. We developed 83 patterns to identify abused images. By examining 243,447 unique explicit images collected from 813,577 webpages of the three blocklists, we discovered that 1,151 explicit images were uploaded through 26 *IHM upload APIs* and hosted on 50 vulnerable *image hosting domains*. These abused images were distributed across 39,414 blocklist websites, with each image displayed on an average of 129.37 websites. For example, Tencent is a

Table 3: All 13 *commercial AIMIEs* identified in this study.

commercial AIMIE	# of APIs used	# of APIs alive	# of image hosting domains	Charges	# of *open-sourced AIMIE* Integration	Integration Time of *open-sourced AIMIE*	Start Time of PDNS	Daily Avg. Query Volume	Total Query Volume
apis.yum6.cn	4	0	N/A	free	14	2018/10/29	2019/07/01*	36.18	45,225
www.yanwz.cn	1	0	N/A	unknown	2	2019/01/19	2019/07/01*	648.70	810,870
api.uomg.com	7	0	N/A	free	12	2019/04/09	2019/07/01*	326,228.65	407,785,813
api.169740.com	1	0	N/A	unknown	1	2019/10/19	2019/07/01*	5,048.80	6,310,998
v1.alapi.cn	9	0	N/A	partial (free within 1,000 images)	2	2020/04/11	2019/07/08	11,859.13	14,740,894
api.nikolatesla.top	2	0	N/A	unknown	2	2020/07/09	2020/05/14	35.07	32,685
image.kieng.cn	11	0	N/A	free	22	2020/12/18	2019/07/23	1,037.63	1,274,207
www.hualigs.cn	23	2	3	free	20	2021/01/14	2019/07/01*	58,834.14	73,542,676
pic.onji.cn	9	2	3	free	3	2021/05/10	2019/07/01*	46.32	57,898
tool.lq520.club	10	0	N/A	unknown	2	2021/11/12	2019/07/20	100.11	123,239
pic.ihcloud.net	15	0	N/A	free	3	2021/12/26	2021/11/20	309.24	116,584
api.kinh.cc	14	9	21	free	1	2022/04/19	2020/08/05	49,426.93	41,963,467
xkx1.herokuapp.com	4	0	N/A	unknown	1	2022/07/24	2022/07/13	5.82	827

* The passive DNS database supplies data post-2019/07/01; consequently, we compute the data from this date onward.

Table 4: Top 10 abused companies.

Company	Sample FQDN	# Related FQDNs	# Image Links	# Blocklist Domains
Tencent	p.qlogo.cn	7	428	56,421
Alibaba	ae05.alicdn.com	16	443	51,201
Ctrip	dimg04.c-ctrip.com	1	314	29,502
Toutiao	p1.toutiaoimg.com	7	35	13,340
Baidu	wkphoto.cdn.bcebos.com	34	352	11,834
JD	dd-static.jd.com	13	399	3,170
Sina	tvax1.sinaimg.cn	19	297	2,034
Sohu	i2.itc.cn	19	130	687
Meituan	p0.meituan.net	2	51	519
360	p1.qhimg.com	17	65	34

leading IT company in China, that owns domains including *qq.com*, *qlogo.com*, *etc.* Tencent's term of service [60] prohibits users to upload explicit content under any conditions. However, in our study, we observed around 56k blocklisted domains (*e.g.,* illegal gambling, pornographic websites) abuse Tencent's private *IHM upload APIs*. Note that this approach is a conservative estimation since we only consider explicit images on blocklisted webpages. It's possible that *AIMIE* users upload non-explicit images, which are not included in our lower-bound estimation.

Discussion. The above conservative result is based on the *AIMIEs*, which only reveal the tip of the iceberg of the real-world abuse landscape. To conduct a more general estimation, we study how many explicit image links are hosted on Tranco top 1M non-pornographic domains. Here we determine the non-pornographic domains based on the tags provided by Similarweb [56]. In our study, we observed 30,416 unique explicit images from 87,195 blocklist domains. The details are listed in Table 4. Note that the number of associated FQDNs differs for each company, as a single company might have multiple vulnerable APIs that store abusive images on distinct FQDNs.

5 VULNERABLE *IHMS* IN THE WILD

Apart from reporting *IHM upload APIs* have been abused by *open-sourced AIMIEs*, we also wonder about the prevalence of vulnerable *IHM upload APIs* in the wild. In this section, we elaborate on the design and implementation of our approach for vulnerable *IHM* assessment, *Viola*. Figure 6 shows the architecture of *Viola*, consisting of a *semantic analyzer*, an *upload lifecycle assessor*, and a *longitudinal analyzer*. *Viola* first recognizes all IHM-enabled web services within the domain using *semantic analyzer*. Next, through *upload lifecycle assessor*, it assesses each stage of the image upload lifecycle (presubmit, preview, submit, and callback) to identify vulnerable *IHM upload APIs*, while using the *upload lifecycle assessor* to track the lifecycle of the uploaded images.

5.1 Semantic Analyzer

We bootstrapped our study by recognizing web services with *IHMs* on a large scale. Identifying web services with *IHMs* on a website is nontrivial, *IHMs* are widely used in modern web services, and their design and implementation vary significantly. Our methodology for detecting such web services is based on the observation that *IHM* always incurs a semantic gap between the module itself (*e.g.,* image upload) and its host web service (*e.g.,* chatbot).

Locating host web services. Our approach for identifying a host web service of an *IHM* starts with locating the client-side user interface of the *IHM*. Using this interface as an anchor, we traverse the DOM tree to determine the associated host web service. We use the signature of `<input type="file">` to locate a client-side user interface. Specifically, we use `document.querySelectorAll` to query all the `input` tags with the field `type` equals to *file*.

To establish a clear boundary for a web service with an *IHM* and facilitate subsequent processing (*e.g.,* pre-filling fields in the table prior to triggering submit and callback stages), we adopted a methodology proposed by previous work [38] for web service localization. In particular, we leveraged low-level text properties instead of DOM structure to construct the segmentation model. The central idea lies in the fact that text density variance within the same web service is considerably lower than the variance observed between different web services. Based on this insight, we applied the concept to our web service localization task. The specifics of threshold selection for segmentation can be found in Section 5.4.

Checking semantic inconsistency. After retrieving the web service with *IHMs*, we build the semantic profile for each web service by constructing the semantic vector with the state-of-the-art embedding technique. Specifically, we extract the context of the *IHM*. Here for each element, we extract its text and its attribute as our context. After concatenating these text contexts, we remove all the punctuation and stop-words and translate them into English. We use the Google pre-training BERT model [16] and generate feature vectors. To check whether the semantics of the web service is

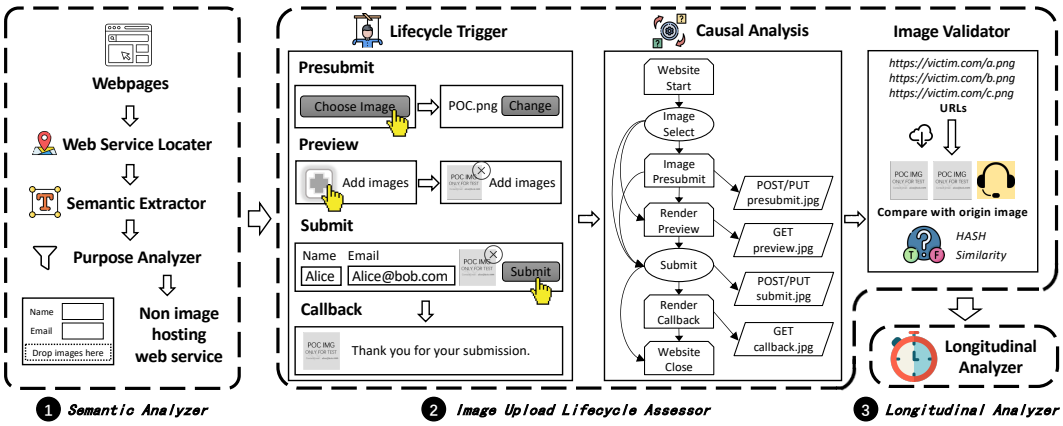


Figure 6: *Viola* overview. *Viola* first recognizes all IHM-enabled web services within the domain (❶). Next, it then evaluates vulnerability across each image upload stages - presubmit, preview, submit, and callback - by uploading and hosting an image within IHMs (❷), while tracking the lifecycle of the uploaded images (❸).

aligned with the image hosting, we compare the semantic distance between them. Specifically, we manually collect the image hosting service on reputable image hosting platforms and generate their semantics (*IHMs* semantics for short). Then, we compare the Euclidean distance between the candidate web service and the *IHMs* semantics, if the minimum distance is larger than a threshold, we take it semantics of such web service is inconsistent with *IHMs*. The evaluation of threshold selection can be found in Section 5.4.

5.2 Upload Lifecycle Assessor

Targeting web services that are not intended to share or store images while having image upload capability, *Image Upload Lifecycle Assessor* dynamically interacts with image hosting modules in web service and identifies the image upload vulnerabilities.

Modeling image upload lifecycle. We take the qualitative open-coding technique [58] on 50 web services with *IHMs* to model the workflow of the *IHM*. Specifically, we focus on monitoring how users interact with the *IHM* and the corresponding network traffics.

For a random subset comprising 10 web services (about 20% of the total), two cybersecurity professionals independently trigger and label the stage of *IHMs*. Then they discussed with each other to resolve inconsistencies while generating initial codebooks. After that, they independently coded the remaining 40 web services and compared their coded results by Krippendorff's alpha coefficient, a widely used statistical measure of the agreement achieved when coding a set of units of analysis. Krippendorff's alpha of this study is 0.87, which is higher than the threshold of reliability in previous work [30]. After that, they resolved all disagreements in coding phrases for each case to generate the final codes, as shown in Table 5. The ultimate codebook developed provides labels for how people interact with modern *IHMs* and which procedure of the *IHMs* process with the uploaded images. In total, it took two human laborers around three days to complete the procedure.

As shown in Table 5, in general, there are two phases during the interaction with web services with an *IHM*. 1) Image Selection: a user triggers the *IHMs* and selects the image from the local device. 2) Image Submission: a user fills in the required fields and submits the forms. During the interactions, they also monitor the network traffic, especially paying attention to the request method, payload, and event timestamp. Based on the traffic analysis, four stages related to different *IHM* functionality are summarized as follows:

- *Before submission – Presubmit Stage.* To speed up the form submission process, websites typically upload images as soon as they are selected, rather than waiting until the submit button is pressed. Since this early submission is before the user-triggered submission, thus we named it Presubmit.

  A key step of Presubmit is uploading the image to the server. According to RFC 2616 [19], clients should use the HTTP POST/PUT method to create a resource (*i.e.,* image) on the server. Consequently, uploading the image binary to a remote server via the POST/PUT method prior to image submission is regarded as Presubmit.
- *Before submission – Preview Stage.* To provide users with "What You See Is What You Get" [74] experience, web services render the user-chosen image on the screen after selecting. Since the HTML `<img>` element [47] is compatible with all modern web browsers, regardless of desktop or mobile, it's intrinsic for developers to render the image with `<img>` element. There are two common approaches to host the preview images: the *server-side cache* and *client-side cache*. The *server-side cache* directly uploads the user-selected image to the remote server, then get the remote image links; the *client-side cache*, which transforms the user-selected image to client-side cache objects, such as a blob object or data URL, then render this local object with `<img>`. Thus browsers initiate the GET requests before submission be considered as Preview.
- *After submission – Submit Stage.* After the preview and presubmit, users submit content, including the chosen images and the texts from blogs, posts, or reviews, to the remote server.

  Image-related submission implementations can be divided into two categories: the merged approach and the sequential approach. In the merged approach, the client side only sends a single request to the server, containing both the upload images binary and corresponding text. However, some web applications utilize a separate image hosting API (*e.g.,* third-party image CDN). As a result, developers adopt the sequential approach, where the client

first uploads images to the image hosting servers. Upon receiving the image links, the client fires a second request containing the links along with the text.

• *After submission – Callback Stage.* Websites demonstrate the submission result to confirm the status and check the information. We name this information display behavior as Callback. Developers return the link to the user-submitted image and render it through the `src` attribute, while others leverage the blob object or data URL (mentioned in the Preview stage) instead. Thus browsers initiating GET requests after submission are considered as Callback.

Dynamic analysis. To automatically trigger the potential pitfalls in different stages, here we instruct the *IHM* to perform full-cycle actions. The actions can be categorized into the following two steps:

1) Image Selection: the lifecycle trigger initiates the process by clicking the upload element within the image hosting module, which is located in the Semantic Analyzer (Section 5.1). Next, it selects and uploads the beacon image. It is important to note that the beacon images used in our tests are explicitly labeled with the experiment's purpose and contact information, both in the file name and embedded within the file content (see Section 6.2 for further ethical considerations).

2) Image Submission: dynamic triggering of the submission stage is much more challenging than it seems to be. In real-world web development, the image components are usually built within the `<form>` elements. In most cases, to trigger the following interaction, websites require that multiple fields in the form be filled in before submitting (*e.g.,* email address for user feedback services). To deal with this challenge, for web elements requiring inputting text content, we pre-define well-equipped user information based on previous work, such as the users' names, email, addresses, comments, *etc.* As for web elements that require user selection, *e.g.,* radio box and check box, the dynamic trigger selects one or more of them according to the element type and prompts information around the forms. After filling in the fields, we then locate and trigger the submission process by inspecting the elements' semantic information. Specifically, we examine the *id* and *class* attributes and text of the *button* element, and trigger them if contain specific keywords, such as "upload, confirm, and submit".

In our study, we implement the dynamic analysis based on Puppeteer [53], which automatically triggers the *IHMs* related interactions and monitors web traffic with Chrome Developer Protocol [14]. Additionally, we open source the tool in [8].

Vulnerable *IHM* detection. In our study, we recognize vulnerable *IHM upload APIs* if the implementation and design of *IHM* have the following pitfalls.

• *Presumbit Stage.* If the server side returns the URL of presubmit image to the client and did not set proper access control for the presubmit image, which leads to the attacker may access the image longer than it should be. Thus, attacks can massively initiate presubmit requests to upload and host the long-lived abusive content.

To detect this pitfall, we consider image URLs in requests or responses via HTTP POST/PUT method after beacon image selection and before submission as potential vulnerabilities in Presubmit stage. Since the returned image URLs may contain noise other than the beacon image, we locate the abused image URL using the *Image Validator* to exclude the noise by assessing image similarity. Specifically, in the *Image Validator*, we first compare the image

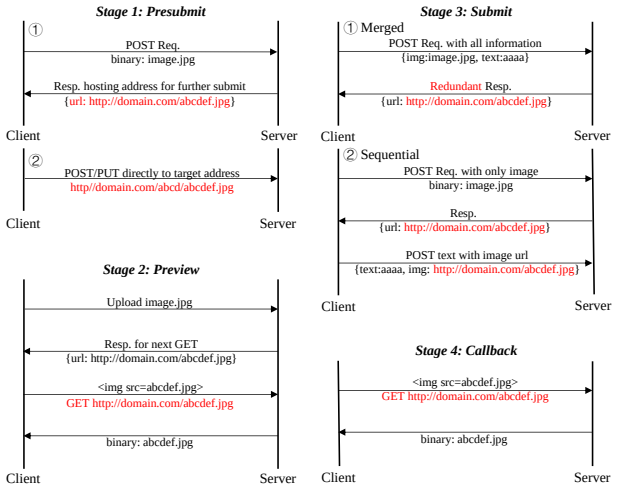


Figure 7: Image upload lifecycle. An *IHM* may have a full four-stage process or omit certain stages.

hash consistency, if failed, we then compare the visual similarity of returned images with beacon image. In particular, we leverage hamming distance between the perceptual hashes of the downloaded image and the beacon. If the difference is below the threshold, we consider the returned image to be identical to the beacon image. The evaluation of the similarity algorithm can be found in Section B.2.

• *Preview Stage.* For the developers who utilize server-side cache to host preview images, they should set a proper expired time for cached files. Otherwise, attackers may initiate massive image preview requests, which upload images to the server-side cache. If the "preview" image is without a proper lifespan limitation, which can be abused by the attacker as an image hosting service.

To detect this pitfall, we take the image URLs in HTTP GET requests after the beacon image selection and before submission as the potential vulnerability in the Preview stage, then apply the *Image Validator* as in *Presubmit Stage*, to collect abused URLs.

• *Submit Stage.* The root cause of flaws in submission is that the server side should not expose resource addresses to the client. Specifically, there exist two flaws in the implementations. First, in the merged approach, if the server-side returns redundant responses containing the uploaded images links to the client side, which leads to the vulnerability. Second, in the sequential approach, if the server does not encrypt or encode the remote address of the returned image URLs, the attacker can capture and abuse the links of the uploaded images. We take the image URLs in requests or responses via HTTP POST/PUT method after forms filling, and submission as the potential vulnerability in Submit stage, then apply the *Image Validator* as in *Presubmit Stage*, to collect the abused URL.

• *Callback Stage.* Similar to the flaws in Preview Stage, developers who utilize server-side cache to host callback images should set a proper expired time for cached files. Otherwise, such *IHMs* can be abused by the attacker as an image hosting service.

Table 5: Number of web services for each lifecycle stage

UI Interaction	Stage	Definition	#(%)
Image Selection	Presubmit	Submit the image to remote server after selecting but before user submitting	27(54%)
	Preview	Render the user-chosen image on the screen after selecting	31(62%)
Image Submission	Submit	Submit the image and text to the remote server after user submitting	50(100%)
	Callback	Render the user-chosen image on the screen after user submitting	25(50%)

We take the image links in HTTP GET requests after beacon image selection, form filling, and submission as the potential vulnerability in the Callback stage, then apply the *Image Validator* as in *Presubmit Stage*, to collect the abused URL.

5.3 Longitudinal Analyzer

The *AIMIE* users abuse the *IHM upload APIs* to host their images for free and/or illegal purposes. Thus, the stability and robustness of uploaded images are critical to them. However, we notice that a few images will be inaccessible after a fixed period due to the mitigating of websites (discuss in Section 6.1). We apply the longitudinal monitor to examine the lifespan of uploaded images. In our approach, for each image URL confirmed by *Image Validator*, we continuously monitor the image URL at regular intervals within a period under a trade-off between precision and ethical concerns (*e.g.,* server resource consumption). We set the maximum period to seven days and use the following intervals: one hour in the first three days, and 24 hours for the next four days.

The time the image lives determines whether it can be abused in the wild, *e.g.,* the corresponding image hosting module is vulnerable. A short lifespan may be a design feature while an unexpectedly long lifespan may be a vulnerability. To gain an inside view of the lifespan threshold for *AIMIE* users, we refer to the minimum lifespan of image hosting domains during our dynamic profiling. Specifically, if an uploaded image through the *IHMs* exists for more than 3 days, we consider it a vulnerability. The detail of the ethical discussion can be found in Section 6.2.

5.4 Evaluation

We run all the experiments on a Ubuntu 18.04 server with Intel Xeon 2.8G, 32 cores CPU and 192 GB memory. The *Semantic Analyzer* and *Image Upload Lifecycle Assessor* cost 17 hours and 25 hours, respectively. Generally, utilizing the Tranco [39] top 10k domains as seed input, we adopt the Depth-First Search (DFS) methodology as the navigation technique, limiting the maximum traversal depth to four layers. To minimize redundancy and cover more domains, we stop crawling a webpage when encountering its fully qualified domain name (FQDN) more than five times. Finally, we successfully crawled 426,702 webpages in Tranco top 1M, found 5,668 image hosting modules by *Semantic Analyzer* and discover 397 vulnerable *IHMs* in *Upload Lifecycle Assessor* and found images uploaded through 338 vulnerable *IHMs* exist for more than three days. Only 5% vulnerable *IHMs* found by *Viola* exhibit overlap with those found by *open-sourced AIMIE*, suggesting that there are more potentially vulnerable *IHMs* in the wild that could be exploited by miscreants.

Table 6: Classifier F1-score for Semantic Analyzer

		Semantic Distance Threshold				
		4.5	5.0	5.5	6.0	6.5
Segmentation Threshold	0.30	48.48%	67.53%	71.43%	77.55%	79.25%
	0.34	48.48%	70.00%	74.42%	80.00%	83.63%
	0.38	52.17%	72.29%	74.73%	**84.40%**	80.00%
	0.42	51.43%	71.43%	73.91%	83.63%	78.63%
	0.46	51.43%	71.43%	73.92%	83.63%	78.63%

Effectiveness of *Viola*. We measure the performance of *Viola* based on its *precision* and *recall*. Due to the absence of the off-the-shelf ground truth dataset, we assess precision by manually inspecting 100 randomly selected cases from the Viola results. We find that 97 cases are true positive, and three are false positive. More specifically, in the three false positives, one of them is introduced by the Semantic Analyzer, which fails to exclude the image uploading components; while the other two of them are raised by the Lifecycle Assessor, which attributes the vulnerable stage from presubmit to submit due to the accidental network latency. To evaluate the coverage of *Viola*, we manually identified image upload vulnerability on 20 popular websites (randomly selected from Tranco [39] Top 10k websites) with our best efforts. In this process, we found seven vulnerabilities on these websites, while *Viola* reported four of them. Upon reviewing false negatives, we found one instance where the failure occurred due to IHM interaction requiring account login, while the other two cases demanded CAPTCHA-solving. Given the known complexities of automating login and CAPTCHA processes in web testing, we propose that integrating Viola with a more robust dynamic triggering module could enhance coverage.

Effectiveness of Semantic Analyzer. To comprehensively assess the performance of the semantic analyzer, we performed an evaluation of the threshold selection for both the web segmentation module and the semantic checking module. To achieve this, we manually labeled 50 web services with *IHMs* and 50 image hosting modules from reputable image hosting platforms, which served as our ground truth. The outcomes of this evaluation are presented in Table 6. With a threshold of 0.38 in the web segmentation module and 6.0 in the semantic checking module, our *Semantic Analyzer* exhibited a precision of 77.97%, recall of 92.00%, and an F1-score of 84.40%. By manually inspecting the result, we conclude that false positives occur as web services are developed using widely-adopted templates, which are also utilized by *IHM* web services. False negatives are mainly because of the lack of semantic information about the specific purpose (*e.g.,* chatbot, feedback) in web segmentation.

Limitations. We acknowledge that there are still some limitations worth noting, despite our best efforts to develop the methodology of *Viola*. As previously mentioned, triggering the full lifecycle of image uploads is not a trivial task. Websites may implement CAPTCHA checks before submission. Although *Viola* has been designed to handle various form-filling tasks and button-clicking strategies, we admit that there may be cases that it does not cover. Besides, due to the computing resource limitation, we only crawl webpages of an FQDN no more than five times, which may result in *Viola* miss some vulnerable APIs. We believe that future research could achieve higher coverage by improving the triggering module or providing additional computing resources.

5.5 Findings

In this subsection, we perform a measurement study to understand the 477 vulnerable stages and their associated 338 web services and 207 FQDNs in the wild. Due to the large number of vulnerabilities found, we are communicating with the developers on a case-by-case basis (see detail in Section 6.2).

Scope and magnitude. Our study reveals the prevalence of vulnerable *IHM upload APIs* in reputable websites. Specifically, websites with vulnerable web services have a median ranking of 20,520. The most highly-ranked domain with vulnerable *IHM upload APIs* is *baidu.com*, followed by *bilibili.com* and *163.com* . Notably, Baidu Mobile Statics (mtj.baidu.com) stood out as the domain with the highest number of vulnerable stages in our study. Particularly, it has two vulnerable web services: online chatbot and customer satisfaction survey, resulting in a total of six vulnerable stages. The vulnerable upload APIs associated with this domain include *https://mtj.baidu.com/web/demo/ajax/post* and *https://mtj.baidu.com/web/ajax/post*.

Image lifespan. Figure 9 illustrates the lifespan of images uploaded through these 397 vulnerable *IHMs* with 550 vulnerable stages, as monitored by the Longitudinal Analyzer. Our analysis indicates that over a three-day span, around 13.27% (73/550) of the images become inaccessible. However, the larger portion, constituting 79.63% (438/550) of the images, remains accessible even after seven days. Notably, only 20.56% of the vulnerable *IHMs* that exposed image URLs during the Presubmit stage had become inaccessible within the same week.

Vulnerable web services and case studies. Regarding vulnerable web service categories, Figure 8 shows that product review services exhibit the most vulnerable *IHM upload APIs*. We observe 106 product review services contain 127 vulnerable stages; also 69 online chatting services expose 81 stages to abuse. Looking into those vulnerable web services, our study further revealed a set of widely-used web service plugins or templates with vulnerable *IHMs*. For example, we find that 21 vulnerable product review services are built with *Judge.me* plugin [37], which ranked 20 in Shopify app store [55], and received a rating of 5.0 under more than 12k comments. The workflow is as follows: when users intend to write a review, they initially select the relevant image of their goods within the *Judge.me* web service. Subsequently, to expedite the submission of complete review forms and enhance the user experience, the *IHM* utilizes the vulnerable *IHM upload API* to send the chosen image to a remote storage space that they've acquired. Moreover,

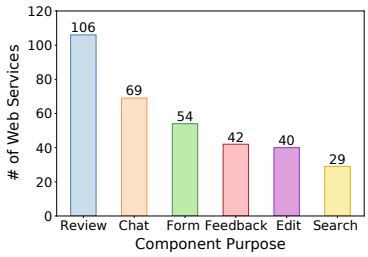


Figure 8: # of vulnerable web services per purpose. The web services of product review showcase the most vulnerable *IHM upload APIs*.

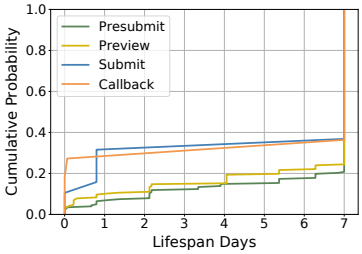


Figure 9: CDFs of image lifespan in each vulnerable stage. 79.63% images have a lifespan longer than seven days. Images uploaded via the Presubmit stage display the highest survival rate, with only 20.56% expired within seven days.

in order to display the selected images to users, the *IHM* retrieves these images from the remote server and then presents them within the web service interface. Throughout this process, users can extract the image link either from the response of the presubmit stage or from the request of the preview stage. These APIs provide the image link in plain text format, and the link remains active even if the user opts not to finalize the review submission.

Besides, information collection services display numerous vulnerable *IHMs*. For example, the account registration of *baishiyunda.cn* requires users to submit ID card, real name, and address. Once users select the photo of ID card from local storage, the website will immediately post the image to the remote server and return the link of images in the response package. This upload action is done without user notification or confirmation. Additionally, the *IHM* not only can be abused to massively as *AIMIE*. Moreover, users have no means of deleting the uploaded images, which violates "right to be forgotten" in privacy laws and regulations, *e.g.*, GDPR [23]. Also interestingly, our analysis on the support page of informer.com *(support.informer.com)* reveals that this web service employs CAPTCHA to impede abusive *form* submissions. However, this protection method only affects the form submission process and does not effectively protect image uploading. As a result, attackers can circumvent the CAPTCHA and proceed with uploading abusive images through the *Presubmit* and *Preview* stages.

6 MITIGATION AND DISCUSSION

6.1 Root Cause and Mitigation

The root cause of vulnerable *IHM* mainly lies in the following two aspects. First, concerning the upload API aspect, developers often lack adequate control over user-uploaded images. This deficiency manifests in various ways, such as the absence of access control, IP-based rate limiting for request frequency, the frequency of image uploads, inadequate implementation of CAPTCHA systems, and more. Second, the backend handling of images frequently overlooks the investigation of abnormal usage patterns, despite the metadata associated with the uploaded images. For example, in cases where images are uploaded during online chats with customer service on an e-commerce platform, such images should be accessible only from specific IP addresses and for a limited number of times. Developers could monitor unusual request behavior to detect instances of *IHMs* abuse. Given these root causes, we elaborate on mitigation strategies as below.

Leverage the client-side cache. In *Preview* or *Callback* stage, developers usually allow users to examine the result after selection or submission. In our research, we observe 291 web services (*e.g.,* online chatting, image searching) first upload the images, store them on the server-side and render with `<img>` tag. This server-side caching may be abused by attackers when a proper expiration time was not set. To mitigate this issue, we recommend using client-side caching, through the adoption of Blob object [71] and data URLs [46]. The Blob interface, which represents immutable raw binary data, in the File API [71], can be easily used to transform a user-uploading file to a URL. More specifically, developers can use *URL.createObjectURL(blob)* to construct a URL representing a blob object without uploading such a file to the website, and this blob URL can be referenced in web components. Alternatively, developers can utilize base64 encoding to represent chosen images and display them on webpages. The `readAsDataURL` API offers the functionality that converts user input image to Data URLs [46], which can be used in the `src` attribute of `<img>` element.

Set up the access control for uploaded images. In other use cases, such as online customer service chat or blog editing, the servers are usually designed to resume the interaction environments with the clients. In this context, the servers have to store the users' uploaded images with server-side caching. Thus, the attacker inevitably received the server-side image links. Thus, we recommend the developer set a proper expired time for the temporarily uploaded images (*e.g.,* 72 hours), or set the image request allowance. It's important to note that the wide variety of services provided by the *IHM* and the extensive use of NAT/DHCP means that an IP allowlist might not be suitable for certain use cases. Furthermore, for scenarios where an image is meant to be viewed only by the user who submitted it, we recommend implementing server-side access control, *i.e.,* user-specific image storage, which could involve naming the image files based on the user's ID or some other unique identifier to prevent conflicts or unauthorized access.

Note that while many developers have adopted CAPTCHAs to guard against abusive form submissions, we've observed that this protection often centers on the form submission itself, rather than the image uploading process of the *IHM*. This distinction leads to two potential vulnerabilities in CAPTCHA protection: (1) If image uploading vulnerabilities arise prior to the CAPTCHA solving, e.g., vulnerability in preview/presubmit stage, while the CAPTCHA enforcing the submission stage. (2) Image uploads that occur after CAPTCHA validation, but image uploading can be triggered independently, are still at risk. In such scenarios, CAPTCHAs don't prevent the image-uploading API from being exploited. To effectively address these vulnerabilities, mandating CAPTCHA verification as a prerequisite for image uploading can establish a more robust defense against misuse.

Align the image hosting content and resource to purpose. Rate-limiting, or "request allowance", doesn't entirely prevent the hosting of explicit content, especially in one-to-one exchanges between users. Therefore, developers should ensure that the image hosting content and associated resources align with their intended purpose. As for the image content, a potential method to determine if the hosted content violates the policy of the upload API service is to launch a compliance check between uploaded images and terms-of-services. This could utilize NLP techniques to understand the texts in UIs [7, 40] or terms of service [51, 75], combined with ML/computer vision techniques to recognize the content of the upload images [31, 76]. For example, the information collection web service for an online hospital requires medical record images should be uploaded. The developer can apply the content classifier (*e.g.,* OCR-based classifier) to automatically detect abusive uploads. Respecting the metadata of the requested uploaded image, developers can also be alarmed by investigating the abnormal patterns. Thus, the developers can monitor the abnormal request behavior to detect *IHMs* abuse.

Hide image resource path. Even though the client-side cache can effectively reduce unnecessary binary transmission between client and server, and thus mitigate the possible vulnerability, in *Presubmit and Submit* stage, it's inevitable to send the image to the server. For the web service which did not render the image on the web page, *e.g.,* the *IHM* of the anonymous feedback collection service, the key to mitigating abuse is to not return the full genuine image URL to the client side. Thus, we suggest the developer leverages the image identifier (*i.e.,* salt hash of the image binary), which refers to the uploaded image, instead of the full genuine image resource paths. Only with this identifier, the abuser can not refer to the genuine image URL, and thus can not abuse such vulnerability.

6.2 Ethics and Disclosure

Ethical concerns. In our study, to gain an insider's view of *AIMIEs*, we engaged with *open-sourced AIMIEs* to upload images to victim host domains. However, we took several precautions to minimize the impact on victim websites. First, we limited the number of *open-sourced AIMIEs* we tested to only 14, ensuring that we tested each abused *IHM upload API* only once. Also for each abused *IHM upload API*, we strictly restricted the size of the demonstrated image used in the experiments to be 10 kB per file, to reduce the impact on target servers. In total, we uploaded 447 images, amounting to 4.4 MB. We also proactively informed victim domain administrators of our experiments and provided them with the file names and hashes of our uploaded images for corresponding removal actions. These experiments comply with the principles identified in the Menlo Report, and were approved by our organization's IRB.

As for the design and implementation of *Viola*, we meticulously crafted it to circumvent any harm to the analyzed targets. Our vulnerability assessment encompasses both static analysis and dynamic interaction. It's essential to note that websites lacking image uploading capabilities or those intended solely for image sharing remain entirely unaffected by our assessments. Furthermore, even when engaging in real-world analysis, we are stringent in restricting the number of dynamic testing actions performed on a single website, allowing us to upload just one image per API. Additionally, the beacon images deployed in our tests are conspicuously labeled with the experiment's purpose and contact information.

Also, to understand the performance of an explicit content classifier ([76]) and ascertain an optimal threshold (Section 4.3), we performed manual validation on a randomly selected pool of 400 images (comprising 200 explicit and 200 non-explicit ones). During this process, team members unavoidably encountered some explicit images. To uphold ethical standards, we adhered to the guidelines outlined in the Menlo Report, which dictate that we minimize unnecessary exposure of annotators to potentially explicit or illegal images. To ensure this, we distributed the annotation task among four team members, each tackling a subset of 100 images. This strategic allocation of work ensured that each individual was exposed to a limited number of explicit images, thereby mitigating any potential impact on their well-being. Moreover, we ensured that annotators were well-aware of their option to halt, discontinue, or opt out of the annotation task at any point. Additionally, throughout the labeling process, they were encouraged to take regular breaks, and their advisors consistently checked in on their well-being.

Responsible disclosure. In total, we have identified and reported abused and vulnerable *IHM upload APIs* across 311 victim websites (109 from *open-sourced AIMIEs*, 207 identified by *Viola*, and five appeared in both categories). Our disclosure reports include vulnerability details (POCs, demo videos), root cause analysis, and actionable steps for mitigating such vulnerabilities. Until August 2023, we received 69 acknowledgments and bounties from victim website owners. Moreover, we release our code and data of *Viola* at [8], offering a lightweight and reliable approach for discovering image uploading vulnerabilities. We hope that our research will enhance awareness regarding this vulnerability and guide developers toward constructing more secure web applications.

7 RELATED WORK

Web resource abuse. A number of works have studied the web resource abuse of the prevalent cybercrime. Previous work mainly focuses on the abuse of cloud computing services, and related web infrastructures and services, *e.g.,* domain name system (DNS). For cloud computing service, Nappa et al. [48] reported over 60% of the exploit servers belong to cloud hosting services in the drive-by download attacks. Wang et al. [72] showed that using public third-party services to perform amplification attacks is more advantageous than attacking the victims directly. Canali et al. [13] conducted a study of the web hosting providers about the prevention mechanism of malicious activities. Their results showed that most of the providers do nothing to detect malicious activities after registrations. Han et al. [29] reported that a million malware samples contact one public IP address on the Amazon EC2 Cloud. Lever et al. [41] conducted a longitudinal study of dynamic analysis traces of 26.8 million unique malware samples. They reported that the malware heavily abuses the dynamic DNS service to communicate with the C&C servers and host malicious content on the Content Delivery Networks (CDNs). Some studies focused on the upper-level cloud service with a specific purpose, *e.g.,* shortening links, and DevOps. Thomas et al. [65] examined the abuse of online social networks (Twitter). Their study reported that free blog hosting websites (WordPress, Blogspot) are heavily abused for the short links hosting of Twitter spam attacks. Li et al. [42] discovered that continuous integration (CI) platforms have been widely abused for illicit crypto mining, showing that this abuse spreads 1974 Cijacking instances, 30 campaigns across 12 different cryptocurrencies on 11 mainstream CI platforms. To the best knowledge of us, none of them have systematically investigated the vulnerabilities of image uploading and corresponding abusive activities.

Web application vulnerability scanning. Web application vulnerability scanners (WAVSs) are automated software tools used to identify security vulnerabilities in applications that run on web servers. Traditional commercial scanners, such as Acunetix Web Vulnerability Scanner [4], WebInspect [63], W3AF [54], ZAP [59], AppSpider [62], mainly target at the OWASP Top vulnerabilities, for instance, SQL injection, XSS and CSRF. Recently, research efforts have been made to improve the scanning performance [18, 22]. For example, Eriksson et al. [18] developed Black Widow, which identifies and builds on navigation modeling, traversing, and tracking inter-state dependencies. Apart from the traditional web vulnerability threats, many studies aim to discover the logic flaws in web components, *e.g.,* the account system and payment system. Ghasemisharif et al. [24] leveraged Single Sign-On (SSO) account and session management model, and guide an automated black-box auditing framework to conduct a large-scale study of flaws across the SSO ecosystem. Calzavara et al. [12] focused on web application insecurity due to cryptographic vulnerabilities, they used attack trees to specify attack conditions against TLS thus scanning the issues on page integrity, authentication credentials and web tracking. Existing works either focus on the general web security vulnerability or on the specific web components issues. However, these works do not address the new rising threat of image hosting abuse and cannot model the image life cycle. To the best of our knowledge, none of the existing work has yet detected image hosting abuse.

8 CONCLUSION

Miscreants are increasingly abusing image hosting modules as malicious services (*AIMIEs*) to host illicit images and disseminate harmful content. This paper presents the first measurement study of *AIMIE* services to provide an inside view of such vulnerability. By collecting and analyzing 89 *open-sourced AIMIEs*, we reveal the landscape of *AIMIEs* and report the evolution and evasiveness of 109 abused *IHM upload APIs* from reputable companies such as Alibaba, Tencent, and Bytedance. We find that 1,151 explicit images are uploaded through 26 *IHM upload APIs*. In addition, we model the image upload lifecycle and construct a vulnerability Scanner, which can effectively and accurately discover 477 vulnerable *IHM upload APIs* in the wild. We have reported abused and vulnerable *IHM upload APIs* to 311 victim sites, and received acknowledgments

from 69 of them until paper submission. Besides, we also provide actionable suggestions for mitigating such vulnerability.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful comments that helped improve the quality of the paper. This work was supported in part by the National Key Research and Development Program (2021YFB3101200), the National Natural Science Foundation of China (62302101, 62172104, 62172105, 61972099, 62102093, 62102091). Xiaojing Liao was partially supported by the Grant Thornton Institute and Indiana University Institute for Advanced Study (IAS). Min Yang is the corresponding author, a faculty of Shanghai Institute of Intelligent Electronics & Systems and Engineering Research Center of Cyber Security Auditing and Monitoring, and Shanghai Collaborative Innovation Center of Intelligent Visual Computing, Ministry of Education, China.

REFERENCES

[1] 2022. *PyPI · The Python Package Index.* https://pypi.org/.
[2] 2023. *requests · PyPI.* https://pypi.org/project/requests/.
[3] 0xDkd. 2018. *auxpi. https://github.com/0xDkd/auxpi.*
[4] Acunetix. 2023. *Acunetix | Web Application Security Scanner. https://www.acunetix.com/.*
[5] Yehuda Afek, Anat Bremler-Barr, and Lior Shafir. 2020. NXNSAttack: Recursive DNS Inefficiencies and Vulnerabilities. In *29th USENIX Security Symposium (USENIX Security 20).* 631–648.
[6] ALAPI. 2022. *ALAPI. https://alapi.cn/api/view/57.*
[7] Benjamin Andow, Akhil Acharya, Dengfeng Li, William Enck, Kapil Singh, and Tao Xie. 2017. Uiref: analysis of sensitive user inputs in android applications. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks.* 23–34.
[8] Anonymous. 2023. *aimie-artifacts. https://github.com/AIMIE-Group/AIMIE.*
[9] apachecn. 2019. *CDNDrive. https://github.com/apachecn/CDNDrive.*
[10] Baidu. 2023. *Baidu User Agreement. https://passport.baidu.com/static/passpc-account/html/protocal.html.*
[11] Ultimate Hosts Blacklist. 2022. Ultimate Hosts Blacklist. *https://github.com/Ultimate-Hosts-Blacklist/Ultimate.Hosts.Blacklist.*
[12] Stefano Calzavara, Riccardo Focardi, Matus Nemec, Alvise Rabitti, and Marco Squarcina. 2019. Postcards from the Post-HTTP World: Amplification of HTTPS Vulnerabilities in the Web Ecosystem. In *2019 IEEE Symposium on Security and Privacy (SP).* 281–298. ISSN: 2375-1207.
[13] Davide Canali, Davide Balzarotti, and Aurélien Francillon. 2013. The role of web hosting providers in detecting compromised websites. In *Proceedings of the 22nd international conference on World Wide Web.* 177–188.
[14] cyrus and. 2022. *chrome-remote-interface.* https://github.com/cyrus-and/chrome-remote-interface.
[15] Theresa Degroote. 2014. *Water torture: A slow drip dns ddos attack.* Technical Report. tech. rep., Secure64.
[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[17] echisan. 2018. *wbp4j. https://github.com/echisan/wbp4j.*
[18] Benjamin Eriksson, Giancarlo Pellegrino, and Andrei Sabelfeld. 2021. Black Widow: Blackbox Data-driven Web Scanning. In *2021 IEEE Symposium on Security and Privacy (SP).* 1125–1142.
[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. 1999. *RFC2616: Hypertext Transfer Protocol – HTTP/1.1.* USA.
[20] Inc. Flickr. 2022. *Flickr Blog. https://blog.flickr.net/.*
[21] The Apache Software Foundation. 2002. *Maven – Welcome to Apache Maven.* https://maven.apache.org/index.html.
[22] Adonis P.H. Fung, Tielei Wang, K. W. Cheung, and T. Y. Wong. 2014. Scanning of Real-World Web Applications for Parameter Tampering Vulnerabilities. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '14).* Association for Computing Machinery, New York, NY, USA, 341–352.
[23] GDPR. 2022. *Right to erasure. https://gdpr-info.eu/art-17-gdpr/.*
[24] Mohammad Ghasemisharif, Chris Kanich, and Jason Polakis. 2022. Towards Automated Auditing for Account and Session Management Flaws in Single Sign-On Deployments. In *2022 IEEE Symposium on Security and Privacy (SP).* IEEE Computer Society, 1524–1524.
[25] Inc GitHub. 2022. *Github REST API. https://docs.github.com/en/rest.*
[26] Google. 2023. *Chrome Web Store. https://chrome.google.com/webstore.*
[27] Cisco Talos Intelligence Group. 2022. *PhishTank. https://phishtank.org/.*
[28] haad. 2022. *proxychains. https://github.com/haad/proxychains.*
[29] Xiao Han, Nizar Kheir, and Davide Balzarotti. 2015. The role of cloud services in malicious software: Trends and insights. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, 187–204.
[30] Andrew F Hayes and Klaus Krippendorff. 2007. Answering the call for a standard reliability measure for coding data. *Communication methods and measures* 1, 1 (2007), 77–89.
[31] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision.* 2961–2969.
[32] Imgbb. 2022. *Imgbb. https://imgbb.com/.*
[33] Google Inc. 2022. *Google Safe Browsing. https://developers.google.com/safe-browsing.*
[34] Qianxin Inc. 2022. *Sinan. https://sinan.qianxin-inc.cn/.*
[35] jd.com. 2022. *jd.com. https://www.jd.com/.*
[36] jqbaobao. 2022. *Made a global CDN map bed for three major manufacturers' interfaces, and released the source code by the way. https://hostloc.com/thread-807552-1-1.html.*
[37] Judge.me. 2022. *Judge.me Product Reviews. https://apps.shopify.com/judgeme.*
[38] Christian Kohlschütter and Wolfgang Nejdl. 2008. A densitometric approach to web page segmentation. In *Proceedings of the 17th ACM conference on Information and knowledge management.* 1173–1182.
[39] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings of the 24th Network and Distributed System Security Symposium (NDSS).* 15 pages.
[40] Yeonjoon Lee, Xueqiang Wang, Kwangwuk Lee, Xiaojing Liao, Xiao Feng Wang, Tongxin Li, and Xianghang Mi. 2019. Understanding iOS-based crowdturfing through Hidden UI Analysis. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security).* 765–781.
[41] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. 2017. A lustrum of malware network communication: Evolution and insights. In *2017 IEEE Symposium on Security and Privacy (SP).* IEEE, 788–804.
[42] Zhi Li, Weijie Liu, Hongbo Chen, XiaoFeng Wang, Xiaojing Liao, Luyi Xing, Mingming Zha, Hai Jin, and Deqing Zou. 2022. Robbery on devops: Understanding and mitigating illicit cryptomining on continuous integration service platforms. In *2022 IEEE Symposium on Security and Privacy (SP).* IEEE, 2397–2412.
[43] Imgur LLC. 2009. *Imgur: The magic of the Internet. https://imgur.com/.*
[44] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. 2020. Demystifying Resource Management Risks in Emerging Mobile App-in-App Ecosystems. In *Proceedings of the 27th ACM SIGSAC Conference on Computer and Communications Security (CCS).* 569–585.
[45] mac cms. 2022. *Mac CMS. https://www.maccms.cn/.*
[46] MDN. 2022. *Data URLs. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URLs.*
[47] MDN. 2022. *The Image Embed element Service Contract. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img.*
[48] Antonio Nappa, M Zubair Rafique, and Juan Caballero. 2013. Driving in the cloud: An analysis of drive-by download operations and abuse reporting. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, 1–20.
[49] onji. 2022. *Share a self-writing image hosting API of reputable company. https://www.v2ex.com/t/770087.*
[50] OpenCV. 2023. *OpenCV: cv::img_hash::PHash Class Reference. https://docs.opencv.org/3.4/df/d4e/classcv_1_1img__hash_1_1PHash.html.*
[51] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. 2013. WHYPER: Towards automating risk assessment of mobile applications. In *22nd USENIX Security Symposium (USENIX Security 13).* 527–542.
[52] portswigger. 2021. *The Burp Suite family.* https://portswigger.net/burp.
[53] Puppeteer. 2020. *Puppeteer.* https://pptr.dev/.
[54] Andres Riancho. 2018. w3af-open source web application security scanner. *https://w3af.org.*
[55] Shopify. 2022. *Shopify App Store. https://apps.shopify.com/.*
[56] Similarweb. 2023. *Website Traffic - Check and Analyze Any Websites | Similarweb.* https://www.similarweb.com/.
[57] sina.com. 2022. *sina.com. https://weibo.com/.*
[58] Anselm Strauss and Juliet Corbin. 1990. *Basics of qualitative research.* Sage publications.
[59] The ZAP Dev Team. 2023. *OWASP ZAP. https://www.zaproxy.org/.*
[60] Tencent. 2022. *Tencent Service Contract. https://www.qq.com/contract.shtml.*
[61] tencent. 2022. *yzf.qq.com. https://yzf.qq.com/xv/html/login.*
[62] Open Text. 2023. *AppSpider DAST Tool - Rapid7. https://www.rapid7.com/products/appspider/.*
[63] Open Text. 2023. *Fortify WebInspect. https://www.microfocus.com/en-us/cyberres/application-security/webinspect.*
[64] ThedoRap. 2018. *Figure-bed. https://github.com/ThedoRap/Figure-bed/.*

[65] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. 2011. Suspended accounts in retrospect: an analysis of twitter spam. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. 243–258.
[66] TransparentLC. 2019. *WechatMomentScreenshot. https://github.com/TransparentLC/WechatMomentScreenshot/.*
[67] tree sitter. 2022. *py-tree-sitter.* https://github.com/tree-sitter/py-tree-sitter.
[68] Uomg. 2022. *UomgAPI. https://api.uomg.com/.*
[69] upimg backup. 2019. *upimg-mirror. https://github.com/upimg-backup/upimg-mirror.*
[70] VirusTotal. 2020. *VirusTotal. https://www.virustotal.com/.*
[71] W3C. 2021. *File API. https://www.w3.org/TR/FileAPI/.*
[72] Huangxin Wang, Zhonghua Xi, Fei Li, and Songqing Chen. 2016. Abusing Public Third-Party Services for EDoS Attacks. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. USENIX Association, Austin, TX, 13 pages.
[73] Weibo. 2023. *Weibo Service Usage Agreement. https://weibo.com/signup/v5/protocol.*
[74] Wikipedia. 2022. *WYSIWYG. https://en.wikipedia.org/wiki/WYSIWYG.*
[75] Yue Xiao, Zhengyi Li, Yue Qin, Xiaolong Bai, Jiale Guan, Xiaojing Liao, and Luyi Xing. 2023. Lalaine: Measuring and Characterizing Non-Compliance of Apple Privacy Labels. In *32nd USENIX Security Symposium (USENIX Security 23)*. 1091–1108.
[76] Yahoo. 2022. *Open nsfw model. https://github.com/yahoo/open_nsfw.*
[77] Youngxj. 2022. *yum6-Sina Img. https://tools.yum6.cn/Tools/sinaimg/.*

A DETAILS OF THE BASELINE METHOD

This regular expression (as shown in Figure 10) is employed in the regex-based baseline method to match all URLs in the AIMIEs.

```
1 http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9
      a-fA-F][0-9a-fA-F]))+
```

Figure 10: Regular expression to match all URLs in the source codes in AIMIEs

B *VIOLA*

B.1 *Viola* Method

Signature-based Locating Web Service Locating. The first step of locating the web service is to locate the IHMs. However, a comprehensive report detailing how web developers implement *IHMs* is currently lacking. To address this, we conducted a preliminary study to comprehend the underlying code patterns of *IHMs*, which involved the following steps:

1) For widely-used web development frameworks, we crafted demo websites featuring basic functionality and examined their underlying implementations.; 2) We reverse-engineered the implementation of *IHMs* on popular websites; 3) We conducted searches on online Q&A platforms and developer forums, such as GitHub and Stack Overflow, using keywords like "upload images", "submit images", *etc.*, to find relevant development instructions.

This process allowed us to compile the characteristic signatures of popular client-side code patterns for image uploading, notably the `<input type="file">` tag. We employed this signature to locate the client-side user interface of *IHMs*.

B.2 *Viola* Evaluation

Additionally, we compared our lifecycle-based method with a baseline method without ignoring upload lifecycle. Specifically, the baseline method first applies *Semantic Analyzer* to locate a host web service of an IHMs. It then stimulates the image upload behavior through the same method used by *Viola*. After that, it employs a regex, `https?://[^\s"']+?.jpe?g[^\s"']*`, on the network responses to check for the presence of content similar to the submitted content. Running this baseline approach on the same evaluation dataset (426,702 webpages in Tranco top 1M), the baseline approach found 226 vulnerable IHMs (vs 397 by *Viola*). We manually validated 100 of those cases and found 99 are true positives. Similar to the evaluation of *Viola*, for the 20 popular websites, this baseline approach found a total of vulnerable two vulnerable IHMs. In contrast, our lifecycle approach reports four vulnerable IHMs with the same targets. This result indicates that the modeling of the image hosting module lifecycle can significantly detect more vulnerabilities (4 vs 2).

Effectiveness of Image Validator. Here we compare the performance of Image Validator under eight different image similarity metrics (see Table 7). To construct a dataset for evaluation, we randomly sampled 100 images from popular websites and applied commonly-used image compression and converting techniques on them. Specifically, we resized each image by 0.25x, 0.5x, 0.75x, or compressed image quality by 0.2, 0.5, 0.8, and then converted the image into formats like JPEG, PNG, BMP, TIFF and WEBP (exclude its original format), resulting in the creation of 10 similar images per original image. For each pile of 100 images, we evaluated which similarity metrics can best distinguish the 10 similar images from the remaining 99 original images. The similarity threshold is determined by analyzing the distribution of similarity scores between the original image and its variations, as well as between the original image and the different images generated through different similarity metrics. Based on the results, we opted for DCT-based image perceptual hash [50] to generate and compare image hash values. This metric with a similarity threshold of 10, demonstrated superior performance in distinguishing variant images from unrelated ones.

Table 7: Evaluation of image similarity metrics at optimal thresholds. The method-specific threshold was selected within corresponding ranges.

Methods	Threshold	Range	Step	Precision	Recall	F1-score
DCT-based Perceptual Hash	10	(0,64)	1	98.5%	98.7%	**98.4%**
Block Mean Hash	21	(0,256)	1	98.3%	98.2%	98.1%
Structural Similarity	0.6	(-1,1)	0.05	97.1%	**99.4%**	98.0%
Marr-Hildreth Hash	186	(0,1000)	1	96.9%	98.9%	97.6%
Average Hash	5	(0,64)	1	**99.1%**	95.5%	97.1%
Color Moment Hash	6	(0,1000000)	2	88.4%	88.2%	86.4%
Difference Hash	16	(0,64)	1	69.7%	**99.4%**	77.2%
Radia Variance Hash	0.85	(0,1)	0.05	89.5%	73.1%	75.7%