Exploring the Unchartered Space of Container Registry Typosquatting

Guannan Liu, *Virginia Tech;* Xing Gao, *University of Delaware;*
Haining Wang, *Virginia Tech;* Kun Sun, *George Mason University*

https://www.usenix.org/conference/usenixsecurity22/presentation/liu-guannan

This paper is included in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

Exploring the Unchartered Space of Container Registry Typosquatting

Guannan Liu
Virginia Tech

Xing Gao
University of Delaware

Haining Wang
Virginia Tech

Kun Sun
George Mason University

Abstract

With the increasing popularity of containerized applications, container registries have hosted millions of repositories that allow developers to store, manage, and share their software. Unfortunately, they have also become a hotbed for adversaries to spread malicious images to the public. In this paper, we present the first in-depth study on the vulnerability of container registries to typosquatting attacks, in which adversaries intentionally upload malicious images with an identification similar to that of a benign image so that users may accidentally download malicious images due to typos. We demonstrate that such typosquatting attacks could pose a serious security threat in both public and private registries as well as across multiple platforms. To shed light on the container registry typosquatting threat, we first conduct a measurement study and a 210-day proof-of-concept exploitation on public container registries, revealing that human users indeed make random typos and download unwanted container images. We also systematically investigate attack vectors on private registries and reveal that its naming space is open and could be easily exploited for launching a typosquatting attack. In addition, for a typosquatting attack across multiple platforms, we demonstrate that adversaries can easily self-host malicious registries or exploit existing container registries to manipulate repositories with similar identifications. Finally, we propose CRYSTAL, a lightweight extension to existing image management, which effectively defends against typosquatting attacks from both container users and registries.

1 Introduction

Container technology such as Docker has been widely adopted in cloud computing [1–4], edge computing [5], high-performance computing [6], and serverless computing [7, 8] for its favorable multi-tenancy isolation [9], near-native performance [10], and excellent support on continuous integration/-continuous delivery (CI/CD) [11]. One of the distinct features is that Docker wraps applications and their dependencies (e.g., code, runtime, system tools and libraries) into an image, enabling fast software delivery and easy portability. This attracts developers to publish images on container registries for maintaining and managing images. Many enterprises, including Docker Hub, Google, and Amazon, provide container registry services targeting both open-source community and private software developers. For example, Docker Hub, one of the largest container registries in the market, maintains hundreds of terabytes of data with a growing rate of about 1,500 new public repositories (i.e., one or more versions of a specific Docker image) daily [12].

With the ever-increasing demand of container services, security incidents inevitably exist in container registries. Users might immediately suffer various attacks if they download and execute malicious container images. It has been reported that 60% of the organizations utilizing containers are affected by container security issues [13]. In 2018, Docker Hub uncovered 17 malicious images containing backdoors for mining cryptocurrency using a victim's system [14]. These images attract more than 5 million pull counts, accumulating approximately $90,000 worth of cryptocurrency for attackers. Palo Alto Networks [15] has reported 117 unsecured container registries on the market, allowing unauthorized push, pull, and delete operations on their hosted repositories. Adversaries can also infect victims' systems by deploying images with malicious code or directly downloading malicious software during runtime [16].

One prerequisite for the threats above is that adversaries need to trick users to download malicious images on registries. This is not a simple task, as registries (e.g., Docker Hub) often rank the images based on the download counts, and users also tend to pull those popular images. However, we reveal a new vulnerability that adversaries can exploit to easily distribute malicious images at a low cost by abusing the typosquatting in container registries. In particular, container registries utilize a fully qualified image identification (FQID) to identify an image. Users also rely on this unique identity to pull images from registries. Adversaries can create an FQID similar to those of benign images, and then users could accidentally download malicious images if they make a typo. Such a typosquatting attack on popular images could pose a serious

threat of spreading malicious container images.

In this paper, we systematically investigate the feasibility of launching typosquatting attacks on container registries in three scenarios: public registries, private registries, and across platforms. To form the basis, we first conduct a measurement study on two widely used public registries, Docker Hub and Quay.io, to understand the existing image naming mechanism. We discover that tens of thousands of repositories have remarkably similar FQIDs, implying that the current naming system of container registries is vulnerable to typosquatting. To demonstrate the severity of typosquatting, we further consider five types of common typosquatting and upload about 4,000 typosquatting images targeting 10 usernames in Docker Hub with the approval from the Internal Review Board (IRB) of our institution. We observe that our typosquatting images are constantly downloaded by users, and attract more than 40,000 pull counts by the end of a 210-day experiment. Meanwhile, out of 10 targeted users, only one user's typosquatting images (contributing 3,867 downloading counts) are deleted by the Docker Hub on day 105. Such an observation indicates that typosquatting attacks pose a serious threat to public registries.

We further uncover attack vectors in private registries, exploiting the image management policy adopted by five large private registries. The typosquatting attacks might happen involving both the project-ID (e.g., username) of the image identification and the region code defined by each registry. We find that most existing private registries allow users to directly name their repositories, and thus adversaries can intentionally upload images with the exact same image name but typosquat their project-IDs similar to that of an existing repository. We also conduct a 60-day experiment demonstrating that the typosquatting attacks could pose security threats to private registries. In addition, the typosquatting attack might also occur across different registries. Particularly, users may unintentionally obtain container images from undesired registries when they mistype or forget to provide the hostname of the image FQIDs. Adversaries can then exploit it by registering typosquatting domain names and host malicious registries.

Finally, we propose a lightweight defense system named CRYSTAL (Container RegistrY SquaTting ALarm) to mitigate the container registry typosquatting threat from both registry and user sides. Specifically, CRYSTAL can be used on the registry end to detect potentially conflicting usernames. Based on the detection result, registries can prevent co-existence of similar usernames. The same process can also be utilized on the user end to inform users about the potential conflicting and typosquatting image names by executing an FQID comparison and keyword search prior to the downloading process. It enables users to further confirm the FQIDs of their desired images. We develop a prototype on the user end based on top of the Docker *pull* command and evaluate the tool on Docker Hub. The evaluation results show that CRYSTAL can achieve high accuracy with minimal overhead.

2 Background

2.1 Container Registry

Container registry serves as a centralized storage that allows many users to facilitate, manage, and share their images. In this paper, we focus on Docker containers where users interact with a registry by using the Docker Command-Line Interface (CLI). Particularly, for downloading container images, users issue the Docker *pull* command followed by the FQID of the desired image from a configured registry.

Typically, an FQID consists of three major fields: hostname, username (also referred to as project-ID), and image name, with the format of *example.com/username/image*. The hostname is defined by the container registry to distinguish the hosting server. It contains a mandatory domain name of the hosting registry and some additional information, such as region code and account ID. Docker Hub is the default registry used by the Docker CLI. If the hostname is not specified in the FQID, users will interact with Docker Hub directly. A username (project-ID) serves as a unique collection of image repositories established by users. When downloading a container image from a registry, a user should provide the correct FQID of the desired image.

Public Registry. Public registries, such as Docker Hub and Quay.io, usually provide unlimited image storage for free as long as the repositories are publicly accessible. Public registries require developers to provide unique usernames when registering their accounts. Users can then upload Docker images under their unique usernames. By default, uploaded images are set as public repositories and are free to download by any users without any authentication or authorization.

Private Registry. Private registries have different repository management policies. In private registries, account owners are charged based on the amount of storage used to host their images and the network bandwidth for both uploading and downloading container images. Usually, each account contains one or more project-IDs (similar to usernames), in which developers can further establish image repositories. The account owner can authorize other users to download or modify the repositories. Some private registries also allow developers to make their repositories publicly available, allowing access to the image without any authentication or authorization.

2.2 Typosquatting

Originally, typosquatting refers to a URL hijacking attack targeting Internet users who make typing mistakes when inputting domain names [17]. While significant efforts have been devoted to understanding domain name typosquatting [18, 19], the current definition of typosquatting has been expanded to include more scenarios sharing similar attack behaviors and characteristics, such as email typosquatting [20] and mobile app typosquatting [21].

Squatting Type	Example
Duplication (AD)	alice ⇨ al**ii**ce
Deletion (DE)	alice ⇨ al**ie**
Permutation (SW)	alice ⇨ a**il**ce
Misinterpretation (MI)	alice ⇨ a**1**ice
Fat-Finger (FF)	alice ⇨ alic**3**

Table 1: Different types of typosquatting with abbreviations and examples.

It has been shown that people are more likely to make typos that involve a one-character distance, also called as Damerau-Levenshtein (DL) distance one [22]. Specifically, DL-1 typos can be categorized into four common types: character-addition, character-omission, character-permutation, and character-substitution. While the first two types are self-explained, character-permutation refers to swapping two adjacent characters, and character-substitution means replacing one character in a string with another character.

Misinterpretation (MI) represents the potential incidents when users misinterpret two characters that look like each other. For example, users misinterpret the "l"(el) in "alice" as "1"(one) and use "a1ice" in the download process. More recent studies suggest that fat-finger [23] (two characters are close to each other on the keyboard) may also yield higher chances of typing mistakes. While other types of typosquatting (e.g., the missing-dot typo [24]) are also common in many scenarios, they are not necessarily suitable for container registry typosquatting. Many container registries, such as Docker Hub and Quay.io, do not support special characters (such as ".", "_", or "-") in either `username` or `image name` of FQIDs. In this work, we only focus on typosquatting for English letters and numeric numbers. Table 1 shows examples of the different types of typosquatting considered in this paper.

3 Container Registry Typosquatting

3.1 Threat Overview

As mentioned above, to download an image from a registry, users need to initiate a *pull* Docker CLI command followed by the FQID of the target image. Some developers (e.g., developers who use Linux Server that only supports command-line input) may download container images by manually typing their FQIDs. These developers are vulnerable to container typosquatting attacks.

Figure 1 presents an example of pull image `reg.com/alice/linux`, representing the target image (`linux`) from the desired hosting registry (`reg.com`) under the username `alice`. The problem is that the Docker pull command is often issued in a console environment, and thus it is not protected by any spelling check functionality. In many cases, it does not require any authentication or authorization for users to pull an image. Thus, users might unintentionally

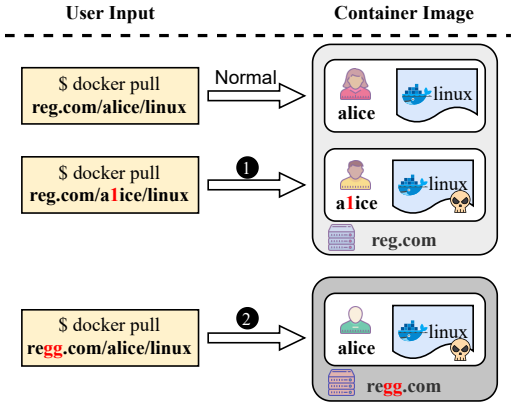

Figure 1: Overview of two cases of container registry typosquatting.

download other images when typos are made in the FQID. In the existing container registry infrastructure, all repositories under the username `alice` are managed by the same user and cannot be modified by others. As a result, even if a typo is made in the image name, the user will obtain an unwanted container image, but it will still be managed by the same owner (e.g., `alice`). Unless attackers can gain access to the account with username `alice` in registry `reg.com`, they cannot upload malicious images. However, there are two cases (❶❷ in Figure 1) in which users might pull an image maintained by malicious developers when typos happen.

Case ❶ represents the case of when users mistype in the username field of the FQID (`a1ice` instead of `alice`). This pull request still reaches the desired hosting registry (`reg.com`), but it enters a different username to pull the image `linux`. If the image `linux` exists in `a1ice`, the CLI will download the image automatically; otherwise, an error message will be reported. Therefore, adversaries can intentionally obtain the username `a1ice` to bait users to download malicious images. Also, this type of typosquatting is possible in both public and private registries.

Case ❷ illustrates a cross-platform typosquatting scenario, where a user makes a typing mistake in the hostname field. The download request reaches an undesired container registry (`regg.com`), instead of `reg.com`. As a result, the user unknowingly obtains a container image with the same name (e.g., `linux`) under the same username `alice`, but from an undesired registry.

Typosquatting case ❷ can occur in existing container registries that share similar hostnames. Additionally, adversaries can utilize previously effective domain typosquatting techniques [25] to reserve a typosquatting domain hosting malicious registries by themselves.

3.2 Threat Model

The goal of container registry typosquatting attacks is to distribute malicious container images by exploiting the potential typos made by container users. Adversaries attempt

Registry	Hostname	Repository	Username	Typosquatting Pairs	DL-1 Username		DL-2 Username	
					Identical Image	Total	Identical Image	Total
Docker Hub	docker.io (default)	416,087	246,080	22,386	16,660 (22.1%)	75,312	37,816 (15.6%)	242,998
Quay.io	quay.io	21,409	6,475	85	47 (39.5%)	119	168 (8.3%)	2,042

Table 2: Overview of public container registries.

to generate multiple typosquatting FQIDs similar to those of existing benign images, and bait users to pull images from a malicious repository. The target can be images in both public and private registries, with popular images being preferable. Adversaries can be simply normal container registry users without any privileges/inside information, and they can utilize the public information provided by the registry to select targets. For some private registries used by teams and organizations, adversaries might be one of the members who know the FQIDs of existing images. In all cases, adversaries only upload images into their own repositories, without compromising the registry or others' accounts.

To launch a typosquatting attack, adversaries first need to obtain the necessary information, such as the FQIDs of targeted (existing) images. This can be achieved by directly browsing or searching repositories inside the existing container registries. For example, Docker Hub allows users to search repositories with particular names and provides the pull count information. Adversaries can also register the target FQID in the registries to check the availability. Then, adversaries can generate and register related typosquatting FQIDs and further push malicious images.

Registering a username typically requires a unique and verifiable email address, which is free to do as most email service providers offer free email accounts. Adversaries can also utilize disposable emails [26] for registration. To launch a large-scale typosquatting attack, attackers can use multiple email addresses to register many typosquatting usernames. For some private registries or hosting attackers' own registries, attackers might need to pay a fee for a storage service or domain registration. Thus, in some scenarios, there is a cost related to launching a container registry typosquatting attack. However, while the cost is trivial (as discussed later), the damage could be severe. Once tricking victims downloading malicious images, attackers can (1) generate financial profits (such as mining cryptocurrency using a victim's system [14]), (2) disclose sensitive information of the victim [9]), (3) harvest computing resources of the hosting server that may lead to denial of services [27]), or even (4) take control over the hosting server [28]).

Adversaries may further inject malicious code (e.g., backdoor) into a container image and circumvent the target registries' checking mechanisms. While the creation of such images is outside the scope of this paper, previous research has shown that existing registries contain many malicious or vulnerable images [29,30]. To make the attack even stealthier, attackers can repackage a target image with malicious code, so that users may not notice the difference, as the malicious image has the exact same functionalities.

3.3 Ethics

To have a better understanding of the exploitability of typosquatting threats in container registries, we conduct our experiments by intentionally uploading multiple typosquatting images to the public. Since our experiments involve human interactions, we take ethical considerations seriously. Our experiment procedure and protocols are carefully designed with the collaboration of our institution's Internal Review Board (IRB) to minimize potential risks to users and ourselves.

First, our experiment procedure should not be considered as a live phishing attempt with the use of honeypot. Watson et al. [31] described the behavior of phishing as a technique to lure victims into revealing personal information. Han et al. [32] discussed the ethical issues of conducting research with phishing honeypots. Szurdi et al. [20] studied email typosquatting by registering typosquatting domains as honeypots to collect mis-delivered emails. They examined the content of the received emails to check if any sensitive information is included in these emails. Moreover, they sent "honey emails" to existing typosquatting domains to scrutinize the responses. Our experiment protocol differs from such a procedure: we do not have access to any information about the users who download our images, nor do we attempt to collect any data from within our typosquatting images. We only gather information about the pull counts of our uploaded images. Such download statistics are available on the Docker Hub as public information and the Google Cloud Console in the category of registry API usage. Therefore, ethical considerations of protecting sensitive information collection and storage are not needed in our experiments.

Second, all of our experiments are conducted in legitimate manners. We do not compromise any accounts. We register all of our accounts legally through Docker Hub and Google Container Registry. The image we uploaded is the latest release of a bare-bone Ubuntu image without any modifications or code injections. We scan our image through Docker Scan [33] and Anchore [34], confirming that our uploaded image contains zero security threat or CVE. We never advertise our images anywhere and all images have no description to avoid being searched by users. At the end of our study, we have manually deleted all of our uploaded typosquatting images and disclosed our findings to Docker Hub and Quay.io via emails.

We acknowledge that our experiments cause inconveniences on users' side, mainly waste of time and confusion. Users waste their time if they download and execute our uploaded images. It might also cause a psychological effect of

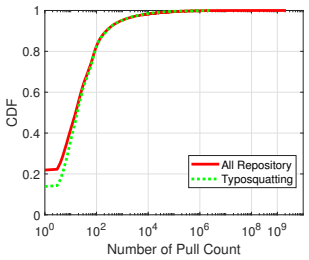

Figure 2: CDF distribution over different numbers of pull counts for our identified images and typosquatting images.

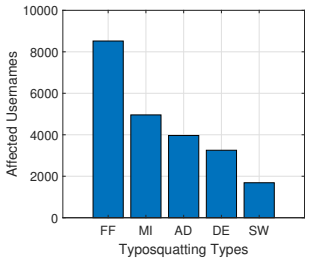

Figure 3: Total number of affected username pairs satisfying the five types of typosquatting.

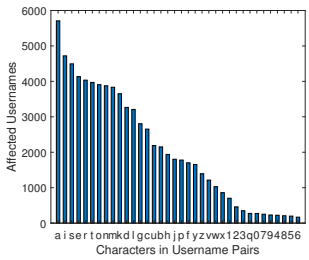

Figure 4: Number of existing typosquatting username pairs for alphabets and numeric numbers.

confusion if users are not aware of their typos. To minimize these concerns, we carefully design our experiments by including a clear warning message in our uploaded images, which informs users of their typos and further provides the correct download commands. Thus, after seeing such a warning message, users who execute our image can correct their typos as quickly as possible and then download the legitimate images. In addition, our experiments can educate users about the security risks of container registry typosquatting, increasing their awareness of such typosquatting attacks, which is similar to phishing training tests that have often been conducted inside an organization.

4 Typosquatting in Public Registry

Public registries, such as Docker Hub and Quay.io, offer free storage services, and thus attract numerous software developers to publish and share their container images. The downstream traffic is also significant: some images reach more than 2 billion downloads. Meanwhile, a large number of usernames exist since each developer needs to register an online account with a unique username. These characteristics make public registries ideal platforms for launching container registry typosquatting attacks (case ❶ in Figure 1).

4.1 Measurement on Public Registries

To comprehensively understand the potential occurrence of typosquatting in public registries, we conduct a measurement study on both Docker Hub and Quay.io. Since neither registries provide complete information of existing usernames and images, we build several crawlers to automatically query a large number of keywords and then record those uncovered usernames and image repositories.

Data Collection. We use three methods for crawling: (1) the search function provided by the Docker Hub API, which returns a maximum of 10,000 results from Docker Hub; (2) the Docker search CLI command that shows 100 results per query for Quay.io; and (3) a web crawler to grab search results from their websites directly. We obtain the keyword database from the latest dump of all entries in Wiktionary [35], which contains a total of 6,760,665 words, numbers, and special characters.

We exclude common/popular image names such as "test," "docker," "hello-world," and combine the results from all methods. As listed in Table 2, in total, we find 416,087 and 21,409 container image repositories, with 246,080 and 6,475 unique usernames in Docker Hub and Quay.io, respectively. The red solid line in Figure 2 illustrates the CDF for the number of pull counts of all our mined images, with about 1.8% of images downloaded more than 100k times.

Results and Analysis. Table 2 presents the analysis on DL distances for both registries. We find 75,312 username pairs in Docker Hub and 119 pairs in Quay.io that meet the condition of DL-1. The green dotted line in Figure 2 represents the CDF distribution for the number of pull counts of those images with DL-1 (referred to as typosquatting images). The distribution is similar to that of all mined images, showing that some of those DL-1 images are popular. Specifically, within those identified username pairs, 16,660 in Docker Hub and 47 in Quay.io contain at least one image repository with an identical name. We conduct vulnerability scanning using docker scan and Anchore to further investigate these images. The scanning result reveals that 4,861 (29.1%) images in Docker Hub and 11 images (23.4%) in Quay.io contain at least one medium or high severity CVE. The existence of these DL-1 container images could potentially pose high security risks to users who obtain an unwanted image if typos are made.

Within the 75,312 DL-1 username pairs in Docker Hub, there are 22,386 pairs that fall into the five different types of typosquatting: duplication (AD), deletion (DE), permutation (SL), misinterpretation (MI), and fat-finger (FF). Figure 3 shows the number of usernames corresponding to each type of DL-1 typosquatting. We find that fat-finger has the most affected cases. Furthermore, Figure 4 shows the frequency of all English letters and numeric numbers involved in the existing DL-1 username pairs. In general, the occurrence of affected English letters is significantly larger than that of affected numeric numbers. The top 8 most affected alphabets {a,i,s,e,r,t,o,n} account for 34,841 or 46.3% of the total number of DL-1 username pairs.

In addition, we expand our analysis to include the occurrence of DL-2 and present the results in Table 2. Compared to DL-1, the total number of username pairs with DL-2 is significantly larger, with 242,998 in Docker Hub and 2,042

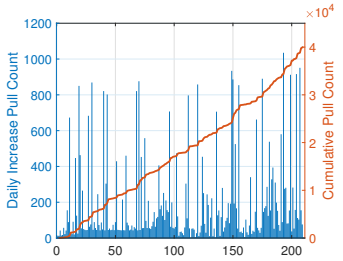

Figure 5: Daily and cumulative pull count of our uploaded images during a 210-day experiment.

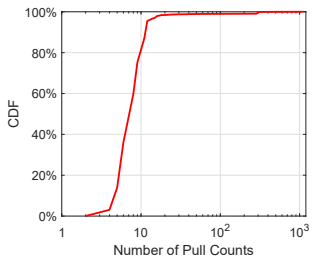

Figure 6: CDF distribution over different numbers of pull counts for our uploaded images.

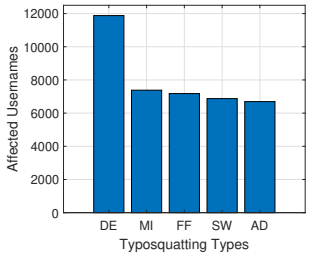

Figure 7: Total pull counts for five types of typosquatting.

in Quay.io. For image repositories with DL-2 FQIDs, we uncover 37,816 and 168 pairs in Docker Hub and Quay.io, respectively. To conclude, these measurement results demonstrate that typosquatting attacks could pose a serious threat in current public container registries.

4.2 Exploitation

We further conduct a proof-of-concept exploitation experiment to demonstrate the severity of a container registry typosquatting attack in public registries. We target 10 existing Docker Hub usernames and create multiple typosquatting usernames with an exact same image name. Then we upload Ubuntu images and track the pull counts on a daily basis to check whether users download our images. The experiment lasts 210 days. Overall, we observe a total of 40,009 pull counts for 4,787 uploaded images. For 10 target usernames, only one username is detected by Docker Hub; all the typosquatting usernames related to that single target are removed on day 105.

Methodology. The first task is to select target usernames for launching typosquatting attacks. To maximize the probability of our uploaded images being downloaded, we heuristically choose two criteria for selecting usernames: (1) at least one container image reaches a pull count of 10 million or more, and (2) at least 20% of the container images are updated within a one-week period. In total, we identify 2,436 usernames that meet the requirements, and we randomly select 10 of them as our target usernames.

Next, we generate typosquatting counterparts for the 10 target usernames. We generate two typosquatting usernames in each of the five types of typosquatting mentioned in Section 2 (i.e., AD, DE, SW, MI, and FF). Particularly, we randomly choose characters and perform the corresponding modifications. Then, for each image under 10 target usernames, we upload our bare-bone Ubuntu images with the exact same repository (image) name. In total, we generate 100 typosquatting usernames and upload 4,787 images on Docker Hub with typosquatting FQIDs. We record cumulative pull counts on daily basis using the Docker Hub registry API.

Results and Analysis. Our experiment lasts 210 days. In total, we record 40,009 pull counts for our uploaded images. While the highest downloading count reaches more than 1,000, most images are pulled less than 5 times. Such an observation indicates that the number of intentional pulling of our uploaded images (from particular organizations or researchers who might constantly pull images from Docker Hub) is small (if any). On day 105, we find that 10 of our typosquatting usernames are deleted by Docker Hub, along with all 1,290 images under those usernames. Those 1,290 images contribute 3,867 downloading counts. Looking into the details of those 10 usernames, we confirm that all deleted typosquatting usernames are targeting a single username. The fact that these usernames are removed indicates that Docker Hub might have realized the attack behavior and thus have taken proper actions. However, all other typosquatting usernames (with images pulled 36,142 times) targeting the other nine usernames remain active. This result suggests that a container registry typosquatting attack is difficult to detect.

Figure 5 shows the trend of the total pull count as well as a detailed breakdown of its daily increases. Overall, the cumulative pull count indicates a linear increase trend. This observation is reasonable and expected, since the occurrence of users mistyping a character in the username field of FQID should be random. We also observe that the daily increase has multiple spikes: there are hundreds of pulls on some days. For example, on day 27, we record the daily increase with a total of 641 downloads. A breakdown of these 641 download shows that 33 images are downloaded three times, 45 images are downloaded two times, and the rest are downloaded once. Another example is on day 70, we record 834 total downloads, in which 278 downloads are generated by a single image. While the root cause of those phenomenons remains unknown, one possible explanation is that users keep pulling the image because the downloaded image does not function as expected and they fail to catch the typos.

Figure 6 presents the CDF distribution of our uploaded images with respect to the number of pull counts. Of all uploaded images, 37 images have more than 100 pull counts by the end of our experiment. In total, these 37 images account for 10,209 of the total number of pull counts (about 25.5%), with the largest one being pulled 1,094 times. To investigate the reason that these images are particularly attractive, we further check the characteristics of their targeting repositories.

We find that 35 out of 37 targeting images are updated on a daily basis. Also, four of them have pull counts of more than 100 million, two images are pulled more than 5 million times, and the rest also have at least 100k pull counts. Meanwhile, about 80% of our uploaded images have pull counts of less than 10. However, these images still attract 21,614 pull counts or 54% of the total number of pull counts. Even when targeting less popular benign images, typosquatting container repositories might still be downloaded due to random typing errors. Those results suggest that popular and frequently updated images are more attractive targets for launching a container registry typosquatting attack. Meanwhile, our target images cover many aspects, with different username length and different base platform (e.g., Ubuntu, Java, and SQL related). However, we observe that, except for image popularity and update frequency, other characteristics have little impact on the effectiveness of typosquatting attacks, implying that typos occur randomly when downloading container images.

Furthermore, Figure 7 breakdowns the pull counts of five typosquatting types (i.e., AD, DE, SW, MI, and FF). We find that usernames with character-omission (DE) typosquatting attract the largest percentage (29.7%) of user downloads, indicating that users are more likely to miss characters when typing the image FQIDs. Meanwhile, the download counts for the other four types are relatively equivalent, with the occurrence rate between 16% to 18%, which are still alarming numbers. Overall, while our experiments are small scale targeting only 10 usernames, the result still suggests that the container registry typosquatting attack is indeed effective in tricking users into downloading unwanted images.

Background Noise. Our approach might record user downloads that are not caused by typing mistakes (e.g., random image crawling). To understand the impact of the background noise on our 210-day experiment, we attempt to find the existence of images with only a few or no downloads but listed for a long time. If the background noise is large, we expect to observe only a very small number of such images existed in Docker Hub, as most images will be affected by the background noise. We examine the download counts of existing container images that are listed for more than 7 months, since our images were placed in Docker Hub for exactly 7 months. In total, we have identified 61,757 images that are listed longer than 7 months. Among them, a large number of images (14,160, 23%) have less than 2 downloads. More specifically, 8,904 images (14.5%) have been downloaded only once (we believe one is a reasonable number as developers might download their images once for testing), and 5,256 images (8.5%) have no download at all. This result indicates that our recorded pull counts in the 210-day experiment are less likely caused by random crawling.

Furthermore, we conduct an additional experiment as a control group of our exploitation study. In this experiment, we target the same set of 10 container images used in our 210-day experiments. For each of the 10 target container images, we generate 3 different usernames using DL distances of 3, 4, and 5, respectively, leading to 30 images in total. Also, we upload another 10 images with randomly generated usernames. We monitor the download counts of these 40 images for 100 days. The result shows that the images containing DL-3, 4, and 5 usernames attract an average of 0.57 downloads per image with a total download count of 17; the 10 images with randomly generated usernames receive 5 downloads in total, with an average of 0.5 downloads per image. For comparison, on day 100 of our exploitation experiment, we record 17,076 download counts for the 4,787 images that we upload to Docker Hub, with an average of 3.57 downloads for each DL-1 typosquatting image. Such a result confirms that the background noise during our download collection is low. It further demonstrates that users are more likely to make DL-1 typing errors, and thereby DL-1 typosquatting images attract more user downloads.

Cost Analysis. Finally, we discuss the cost for launching the container registry typosquatting attack in public registries. The attacks generally require three steps: (1) acquire information of targets; (2) generate typosquatting usernames by registering accounts; and (3) store images in public registries. The first step has no cost as all information is available online. The account registration requires adversaries to have a large number of email addresses, as both Docker Hub and Quay.io use valid, unique, and verifiable email addresses as account identities. Since many email service providers (like Gmail or Hotmail) are free, attackers can easily obtain many valid email addresses. Users can also utilize temporary email addresses such as disposable email services [26] for account registration. Thus, the cost for obtaining accounts in public registries is still zero. Finally, all of the image repositories hosted in public registries are free of charge. As a result, we conclude that the financial cost for launching container registry typosquatting attacks is zero.

5 Typosquatting in Private Registry

Many private registries allow users to open or share repositories, thus making a typosquatting attack possible. We investigate the potential typosquatting opportunities in five existing private registries (listed in Table 3).

5.1 Attack Vectors

Typosquatting threats, as shown in cases ❶ and ❷ of Figure 1, might also exist in private registries. Attackers can intentionally upload images to private registries with typosquatting FQIDs. While private registries have different project-ID management policies, our investigation reveals three potential attack strategies to achieve DL-1 typosquatting FQID.

User-defined Project-ID Typosquatting. Launching a typosquatting attack in private registries with user-defined project-IDs is fairly straightforward. Once attackers identify their targeting project-IDs, they can generate a swarm of typosquatting counterparts in the same private registry and

Registry	Hostname	Project-ID	Permitted Characters	Region	Search	Public	Price /10GB/year
Alibaba	registry.[region].aliyuncs.com	User Define	Alphabet, Number,"_","-"	✓	✓	✓	0
Amazon	[client_id].dkr.ecr.[region].amazonaws.com	Random	Number	✓	✗	✗	$12
Azure	[username].azurecr.io	User Define	Alphabet, Number, "_","-",".","/"	✗	✗	✗	$61
Google	[region].gcr.io	User Define	Alphabet, Number, "-"	✓	✗	✓	$3.12
IBM	[region].icr.io	User Define	Alphabet, Number, "_","-"	✓	✗	✓	$6

Table 3: Overview of private container registry.

upload their malicious images. In this study, we systematically investigate the possibility of project-ID typosquatting in four of the private container registries that allow users to customize their project-IDs, including Alibaba, Microsoft Azure, Google, and IBM.

We obtain 407 unique project-IDs from Alibaba Container Registry, and 158 unique project-IDs in Azure Registry. In Google Container Registry, we find a total of 407 occupied project-IDs. For IBM Container Registry, we focus on the `us.icr.io` server and find 584 project-IDs. We randomly select 50 project-IDs from each registry for further testing. For each selected project-ID, we generate a full list of potential DL-1 project-IDs satisfying the five types of typosquatting shown in Table 1. This list covers all DL-1 project-IDs that can potentially be used to launch a project-ID typosquatting attack. In total, we generate 35,861, 32,629, 25,183, and 29,636 project-IDs in Alibaba, Azure, Google, and IBM, respectively. For Azure, Google, and IBM, we use the same trial-and-error approach in the data collection procedure to check the existence of our generated project-IDs. For Alibaba, since the search functionality cannot reveal any project-IDs in private repositories, we also employ the trial-and-error approach to check the existence of our typosquatting project-IDs.

Figure 8 shows the percentage of the available typosquatting project-IDs that we generate in each registry. In all four private registries, we observe that more than 90% of the DL-1 project-IDs are available for registration. With a large percentage of availability, attackers can intentionally register many typosquatting project-IDs to host malicious images, and therefore, any users who accidentally download these malicious images could be at risk for system compromisation due to typing mistakes.

Randomly Generated Project-ID Typosquatting. Whereas registries including Alibaba, Azure, Google, and IBM allow users to customize the names of project-IDs, Amazon uses a randomly generated 12-digit client-ID as the project-ID for each user. With this policy adopted, attackers can repeatedly create a large amount of accounts and hope that some of them satisfy the DL-1 typosquatting preference. We attempt to investigate the practicality of launching project-ID typosquatting attacks on Amazon.

In order to generate a large number of AWS accounts in Amazon, we first register a piloting AWS account and establish an organization group inside our account. This allows us to create more Amazon accounts using the AWS Command-Line Interface so that the whole process can be automatically streamlined using a script. We raise the limitation of the total allowed users in our organization to 20,000 by contacting AWS customer services. Using the increased user quota, we spawn 20,000 AWS accounts within our organization.

We compare and calculate the DL distance across all possible client-ID pairs from the 20,001 obtained client-IDs. Figure 9 shows the number of client-IDs pairs we obtained with a DL distance less than 6. Unfortunately, we do not obtain any DL-1 client-IDs, and the closest distance we achieved is DL-2, with only one client-ID pair. Given the low probability of obtaining DL-1 pairs, we conclude that it is difficult for attackers to launch attacks in private registries using randomly generated project-IDs.

Region Typosquatting. Many private registries allow users to choose a specific data center location to host their container repositories. A unique region code is used to represent the location information inside the hostname. Users are expected to correctly type the region codes (and the rest of the FQID fields) to obtain the desired images. Thus, if a registry utilizes a similar, preferably DL-1, region codes, it might be vulnerable to region typosquatting attacks.

We conduct a case study on the IBM Container Registry, which contains five regions that are widely spread across the world, including two data centers in Asia Pacific, one in Europe, one in the U.K., and another one in the U.S. In particular, the U.K. data center is assigned with hostname `uk.icr.io`, and the US data center uses a very similar one, `us.icr.io`. Apparently, these two URLs contain DL-1 region codes. Further analysis shows that, within the 584 project-IDs we identify in IBM Container Registry, 132 usernames exist in both regions, which account for 76.3% in `uk.icr.io` and 22.6% in `us.icr.io`. This result indicates that users who mistype the region code of those 132 usernames may unintentionally pull unwanted images from another region of the registry.

5.2 Proof-of-Concept Exploitation

We also attempt to evaluate the feasibility and effectiveness of launching typosquatting attacks on private registries. As mentioned in the threat model, since enterprises typically isolate their container images from the public, the key information about target images, such as the popularity, total download count, and latest update time, is not publicly available. As a result, the scale of our experiment in private registries is limited. To launch effective attacks, attackers might be one of the internal members who can obtain an insider's knowledge (e.g., FQIDs of existing images and relevant information).

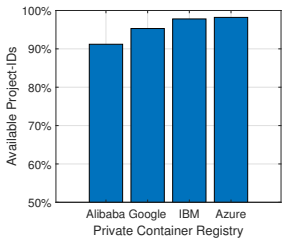

Figure 8: Percentage of available typosquatting project-IDs in private registry.

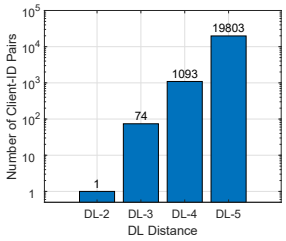

Figure 9: Total number of client-ID pairs identified for different DL distances.

With the IRB approval, we conduct a 60-day experiment targeting the official container images provided by Google. We randomly select 10 images and generate 100 DL-1 usernames with typosquatting categories, including AD, DE, SW, MI, and FF. We use the same Ubuntu image mentioned in Section 4 for the typosquatting image. We record 62 pull counts for our uploaded images, with the highest download count of 14. We further conduct a control group experiment in which we upload 50 typosquatting images with DL-3 usernames and 50 images with randomly generated FQIDs. At the end of our 60-day experiment, we record 6 downloads for these 100 images.

Overall, the results of the two experiments demonstrate that private registries are also vulnerable to typosquatting attacks, and the images with DL-1 usernames attract more user downloads.

5.3 Cost Analysis

The cost primarily depends on the financial policy adopted by each registry. Alibaba provides an always-free service for hosting container repositories. Thus, the financial cost of launching a typosquatting attack on Alibaba is zero. Other registries listed in Table 3, including Azure, Google, and IBM, charge a service fee based on the amount of storage used to host container repositories, as well as the network traffic induced by downloading container images. Since container images are usually lightweight and range from tens to hundreds of megabytes (our testing image is only 20MB), we take an image with 100MB as an example. The price for hosting one image for one year is $0.031, $0.61, and $0.06 for Google, Azure, and IBM, respectively. Compared with the potential financial gains, the cost is relatively low and affordable.

6 Typosquatting Across Platforms

Cross-platform typosquatting may happen when users pull images from a different registry. This scenario is shown as case ❷ in Figure 1.

6.1 Hostname Typosquatting

The key for launching such a cross-platform typosquatting attack is to identify or establish a container registry with a hostname similar to that of an existing registry, preferably with DL-1 distance. This can be achieved by either self-hosting container registries with typosquatting domain names or utilizing existing registries.

6.1.1 Domain Typosquatting

Attackers can self-host container registries by registering typosquatting domain names, where attackers have complete control on the configuration of the entire registries. With self-hosted registries, adversaries can host as many malicious images as they need, and they can enable public access to all of the hosted repositories.

To understand the feasibility, we investigate the availability of the typosquatting domains on the market. For each container registry in Tables 2 and 3, we generate a complete list of all hostnames with DL-1 distance under the five categories. In total, we generate 2,692 DL-1 typosquatting domain names for the six container registries. We then utilize the bulk domain search function provided by Namecheap.com [36] to check the availability of all generated hostnames and their registration prices. Note that in this experiment, we exclude the typosquatting domains for Docker Hub since the `docker.io` hostname can be omitted in the Docker pull command. We also exclude the scenario of when typosquatting occurs in the top-level domains (TLDs) and only focus on the domain name part of the registry hostnames.

Table 4 presents the number of generated typosquatting domain names for each repository and the current availability of these typosquatting domains on the market. Among all generated typosquatting domain names, 2,258 (83.9%) of them are available for purchase. This result indicates that such an attack is feasible as attackers have a large range of typosquatting domain names available to register.

Table 4 also includes the purchasing price distribution for our generated typosquatting domain names. For most domains, the registration fee is under $30, with 72.5% (1,637 out of 2,258) of them less than $10. In general, the domain names under ".com" TLD is cheaper than the domains with ".io" in term of regular registration price (<$10 vs. <$30). Only 26 domains have a purchasing price over $30. Overall, the financial cost for such attacks is reasonable, as attackers no longer need to pay a monthly storage fee for hosting images.

6.1.2 Existing Registry

Attackers can also utilize two existing registries to launch cross-platform typosquatting attacks. For example, as listed in Table 3, Google Container Registry and IBM Container Registry share similar hostnames: Google uses `[region].gcr.io` as its registry hostname, and IBM employs `[region].icr.io`. In addition, both registries operate hosting servers in the mainland U.S. with the region code "`us`". Specifically, the hostname for Google server in the U.S. is `us.gcr.io`, while the southern U.S. region of the IBM server holds a hostname `us.icr.io`. Obviously, the `us.gcr.io`

and us.icr.io satisfy the requirement of cross-platform typosquatting attacks with a DL-1 character-substitution.

We further investigate the potential for attackers to launch typosquatting attacks across Google Container Registry and IBM Container Registry. To establish DL-1 typosquatting FQIDs, attackers need to register the same project-ID as their targeting repositories. For example, if the cross-platform typosquatting target has the FQID us.icr.io/alice/linux at IBM, attackers need to register the project-ID alice in Google and upload a malicious image named linux. In this case, the malicious image holds a typosquatting FQID of us.**g**cr.io/alice/linux, which becomes a DL-1 counterpart for the targeting image us.**i**cr.io/alice/linux.

We analyze the same occupied project-ID lists in Section 5, which contains 407 project-IDs in Google and 584 in IBM. From the list, we identify a total of 82 project-IDs existing in both registries, which account for 20% in Google and 14% in IBM. We also confirm that the rest of the project-IDs (325 in Google and 502 in IBM) uniquely exist in their own registry, and the same project-IDs can be registered in the other system for typosquatting attack purposes. Since the number of overlapping project-IDs is relatively small, it is very likely for attackers to successfully establish malicious images with DL-1 FQIDs and launch cross-platform typosquatting attacks.

6.2 Missing Hostname

A special case for container image FQIDs is that the hostname can be omitted if the container image is hosted in Docker Hub. This introduces yet another potential threat: if users forget to include a hostname in the Docker pull command, they might obtain an unwanted image from Docker Hub, instead of from their desired registries.

To shed light on the missing-hostname typosquatting attack, we utilize the same username lists we gathered in Section 4 for both Docker Hub and Quay.io. By comparing both lists together, we identify 960 usernames that exist in both Docker Hub and Quay.io. Our analysis also shows that Quay.io contains 5,515 unique usernames that are not available on Docker Hub, and Docker Hub contains 245,120 unique usernames that are not available on Quay.io. A further analysis shows that 117 (12.2%) of these shared usernames have uploaded container images with the exact same image names on both registries, which account for a total of 282 repositories. This may indicate that some container developers have already been aware of the fact that users might potentially mistype the hostname and have taken proactive actions by uploading the same images on both registries.

To further illustrate the effectiveness of the missing-hostname attack, we also perform a 30-day experiment with the IRB approval of our institution. We randomly select 10 unique usernames on Quay.io with high activity ratings, and then manually register these usernames on Docker Hub. We also upload our container images with the same name as the original image name in Quay.io. We use the same bare-bone Ubuntu image as mentioned in Section 4. We record the cumulative pull count for these images for 30 days.

Domain	Available (Total)	Price		
		<$10	<$30	>$30
aliyuncs.com	582 (619)	578	0	3
amazonaws.com	552 (692)	550	0	2
azurecr.com	511 (546)	509	0	2
quay.io	292 (327)	0	291	1
gcr.io	162 (254)	0	154	8
icr.io	159 (254)	0	149	10
Total	2,258 (2,692)	1,637	594	26

Table 4: Domain typosquatting for container registries with registration price distributions.

During our 30-day experiments, we record a total of 93 pull counts, with the highest pull counts for a single image being 24. The pull count result suggests that the missing-hostname might be exploited by container registry typosquatting attacks.

7 Mitigation

In this paper, we propose CRYSTAL (Container RegistrY SquaTting ALarm), a lightweight extension on the existing infrastructure of container registry, to mitigate the threat from both container registry and user sides. Working from the registry side, CRYSTAL can be seamlessly integrated into existing container registry platforms to detect typosquatting behaviors and prevent users from registering extremely similar usernames. Working from the user side, CRYSTAL extends the image pull method to provide typing suggestions and corrections to container users before the download process. Overall, CRYSTAL can effectively mitigate the container registry typosquatting problem with minimal overhead and without changing the current container architecture.

7.1 Design

The primary goal of CRYSTAL is to identify possible alternative FQIDs that could potentially lead to typosquatting behaviors on both registry and user sides. Figure 10 illustrates the overall architecture of the CRYSTAL tool. In general, CRYSTAL contains three major modules: FQID Analyzer, Alternative Finder, and Result Presenter. The FQID Analyzer module is used on the user side to analyze the key information, including hostnames, usernames, and image names from the user inputs. The extracted information is then input into Alternative Finder for the detection of images with potential typosquatting FQIDs, and then Result Presenter presents potential alternatives. When CRYSTAL is deployed on registries, Alternative Finder is fed with the account registration information, and the identified usernames from Result Presenter further help the registry to decide whether the registration request should be permitted or denied.

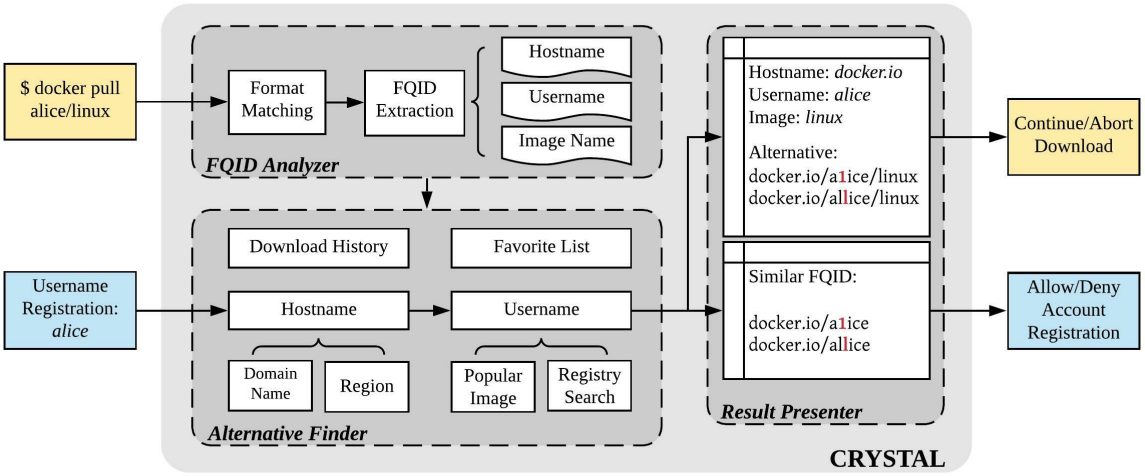

Figure 10: Design overview of CRYSTAL.

7.1.1 FQID Analyzer

FQID Analyzer first extracts key information from a Docker pull command, including the hostname, username, and image name for the target image. Since different container registries adopt different FQID formats (especially for the hostname), FQID Analyzer also identifies the FQID information based on the format of each registry. For example, registries like Google and IBM include the region code inside the hostnames, while other registries like Azure do not. Also, the hostname `docker.io` can be omitted if the corresponding image repository is hosted in Docker Hub. Thus, FQID Analyzer maintains a list of FQID templates for each supported container registry. Every time the user executes a Docker pull command, the FQID Analyzer module performs a table lookup to match the provided FQID with one of the recorded templates. Also, the analyzer adds the `docker.io` hostname back to the FQID if the hostname is missing.

7.1.2 Alternative Finder

Once all three fields of the target FQID are obtained (either from FQID Analyzer on the user side or the username registration request on the registry side), Alternative Finder further searches for other container images with similar FQIDs. Since users are more likely to make DL-1 typing mistakes, Alternative Finder is currently configured to focus on alternative usernames with a one-character difference.

Download History and Favorite List. On the user side, Alternative Finder maintains a record of the user's download history and a list of favorite container images. The download history contains the FQIDs of all previously obtained container images. Every time the user executes a Docker pull command, Alternative Finder compares the FQID provided by the user to the history record and identifies any container images with DL-1 FQIDs. The favorite list is further designed so that users can manually configure their favorite images by themselves. It also enables CRYSTAL to maintain multiple cache files to accelerate the search process.

Hostnames. The hostname of an FQID usually consists of two types of information: region code and domain name. Alternative Finder searches both fields independently to identify potential typosquatting targets. For example, it first identifies any potential usernames residing in a different region code, such as `us.icr.io` versus `uk.icr.io`. Then, Alternative Finder locks the region code and seeks other container registries with a DL-1 domain name to reveal alternative domain names with the same region code.

Usernames. To identify alternatives for the provided username, Alternative Finder performs a lightweight image search within the desired container registry. If CRYSTAL is deployed on the desired container registry, the image search can be performed directly on the repository database within the registry. Otherwise, for registries whose search functionality is publicly accessible, Alternative Finder utilizes the image name as a query term and executes a search command to find other existing images with DL-1 usernames. If the search functionality is prohibited or disabled, Alternative Finder relies on cache files, which can be periodically updated offline. For instance, CRYSTAL can utilize the trial-and-error method mentioned in Section 5 to update the favorite list.

Cache files. To enable fast lookups and reduce network traffic, CRYSTAL maintains four cache files, including lists of download history, favorite images, supported hostnames, and popular images. Alternative Finder first searches typosquatting images in both download history and favorite images with DL-1 FQIDs. To search for alternative DL-1 hostnames, we maintain another cache file containing all the domain names

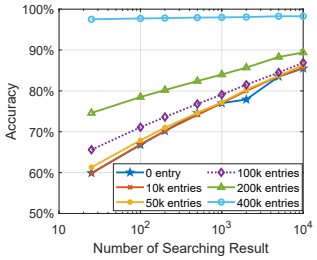

Figure 11: Accuracy for different search results and sizes of popular image files.

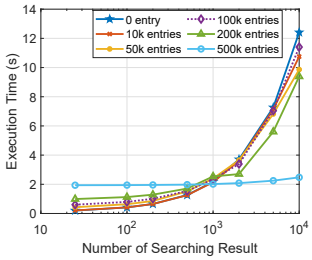

Figure 12: Execution time for different search results and sizes.

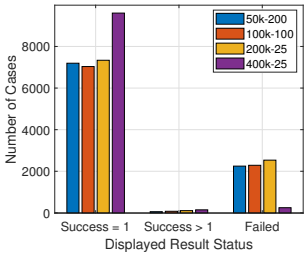

Figure 13: Comparison of displayed result statuses in four case scenarios.

and their associated region codes of the container registries (as shown in Tables 2 and 3). The Alternative Finder module performs a table lookup to distinguish any similar domain names or region codes. The last cache file consists of a list of popular images with a large number of pull counts from the container registries that we support. The maximum FQID entries included in each cache file is configurable and limited by default to prevent over-growth. Users can also control the number of results returned from Alternative Finder's search command to balance the overall performance and accuracy.

7.1.3 Result Presenter

After the lookup procedure, the Result Presenter module presents the search result to the user by providing both the user-provided image and potential alternatives. First, Result Presenter lists the hostname, username, and image name from the image FQID that users provide in the Docker pull command. Particularly, `docker.io` is added by FQID Analyzer to enable users to self-correct any typing mistakes if the result of FQID Analyzer does not match their intentions.

Next, Result Presenter shows the alternative images identified by Alternative Finder. In general, it favors container images recognized from the download history and the favorite list. All DL-1 alternative images identified from these cache files are always presented at the top of the result list. Additional queries are also performed to filter other images based on their popularity. We currently utilize the total download counts and star counts as the popularity. The results are listed in descending order of image popularity (i.e., images with larger download counts or star counts are displayed first). If multiple results exist with the same popularity, Result Presenter shows the results by the order of different hostname, region code, username, and image name. Meanwhile, any other metrics can be easily integrated into CRYSTAL to enhance its effectiveness. If no alternative images are identified, Result Presenter automatically allows users to download the image without presenting any warnings.

The format is slightly different when CRYSTAL is deployed on the registry side. Result Presenter shows similar usernames obtained from FQID Analyzer, allowing the registry to adopt further policies, such as permitting/denying the corresponding username registration.

7.2 Evaluation

We implement a preliminary prototype of CRYSTAL tool on the user side by expanding the functionality of the Docker Command-Line Interface. We modulize our implementation so that CRYSTAL can also be easily integrated into the registry server side. In our experiments, we focus on typosquatting images hosted in Docker Hub for demonstration purposes, while our tool can be easily applied across different registries. Also we limit our computing resources to simulate an average computing environment for most general container users. Our testing system is a virtual machine running a Ubuntu 18.04 system containing 4 CPU cores and 8G of memory. We evaluate the performance of CRYSTAL on the following three aspects: accuracy, overhead, and usability.

To investigate the performance of CRYSTAL, we randomly select 10,000 container repositories in Docker Hub, and we generate one typosquatting FQIDs for all of these repositories. We execute the CRYSTAL tool to process all generated typosquatting FQIDs and record the average execution time. We adjust different number of entries in the popular image cache file, and set different limitation numbers for search results. The cache file selects and stores popular images based on their pull counts. We also calculate the overall accuracy under different configurations.

Accuracy. Figure 11 presents the result of the accuracy for finding correct image repositories. If the cache file contains 400,000 entries, CRYSTAL can maintain a high accuracy of more than 97.5% in all cases. The size of the cache file is about 10 megabytes, which is acceptable. In other cases, the accuracy gradually increases with a higher number of entries in the popular image file and more limitations on search results. But even with the minimum search result limitation (e.g., 25 in our experiment) and zero entries in the popular image file, our tool can still correctly detect 60% of typing errors in the download command. With the cache file containing 200k entries, the accuracy is about 74.5% for the 25 searching result limitation. To obtain an accuracy of 71%, it takes an average of 100 search results with 100k entries, and 200 search results with 50k entries in the cache file.

Overhead. We first measure the download time for users to pull a container image from Docker Hub. We randomly pull 1,000 container images on Docker Hub and record an average download time about 51.2 seconds with 100 Mbps network

bandwidth. Figure 12 shows the average execution time of CRYSTAL. If the popular image cache file contains 400k entries which accounts for 96% of the entire list of recognized repositories, the average execution time is relatively stable at 2 seconds, which only accounts for less than 4% of the overall image download time. This is reasonable as almost all names are already cached. In other cases, the execution time dramatically increases with a high limitation of search results. Particularly, when the limitation of searching results reaches 1,000, the execution time is larger than the case caching 400k entries, which has a much higher accuracy.

Otherwise, to reach at least 70% accuracy, the execution times for 200k entries with 25 search result limit, 100k entries with 100 search result limit, 50k entries with 200 search result limit are 0.989s, 0.793s, and 0.857s, respectively. Combined with the accuracy results, users can configure CRYSTAL based on their preferences.

Usability. CRYSTAL does not change the existing workflow of the Docker CLI, except that users might be required to confirm the correctness of their typed FQIDs and make any adjustments if necessary. We further measure the number of lines needed in the Result Presenter to present the target image for users. If it requires a large number of lines, it will degrade the usability. We configure CRYSTAL to present at most 100 lines. We consider the configurations that have an execution time of less than 2 seconds and an accuracy of more than 70%. We illustrate the required number of lines in Figure 13. The results show that, with the exception of failed cases, almost all of the remaining cases need only one line to present the target FQID correctly to users. Thus, we believe CRYSTAL has little impact on usability.

7.3 Limitation and Future Work

Users who receive many warnings when downloading container images could experience warning fatigue. To mitigate this issue, we implement a preliminary approach: CRYSTAL allows the download process without showing any warning messages for an image having comparably high popularity. We will fully address the fatigue issue in our future work.

Our CRYSTAL prototype is implemented on the user side as an extension of the Docker CLI and focuses on Docker. Docker also supports using a *FROM* statement in Dockerfile to automatically pull an image when building a new container image. We also observe an increasing popularity of some Docker alternatives, such as Singularity [37] and Podman [38]. The overall performance would be similar or even better if CRYSTAL was deployed on the registry side. If CRYSTAL is running on a container registry consisting of datacenter-grade servers, the total execution time should be further reduced. In addition, the image search function can be integrated with the registry's database, further reducing network traffic and latency. Accuracy and user experience can also be enhanced if the registry searches the entire repository database for username registration requests and caches more elements from other registries. In the future work, we will extend CRYSTAL to the registry side and support more methods for mitigating potential container registry typosquatting threats.

8 Related Work

8.1 Container Performance and Security

Extensive research efforts have been conducted on both performance [10, 39–41] and security [42–45] of containers. Gupta [46] presented a brief overview of container security. Grattafiori [47] further explored potential vulnerabilities in containers, including container escaping, cross-container attacks, and inner-container attacks. Lin et al. [28] demonstrated that exploiting container vulnerabilities can cause problems such as sensitive information leakage, remote control, denial of service attacks, and privilege escalation. While covert channels are possible in containers [48], Gao et al. [9, 49] further revealed the existence of information leakage channels from which containers can obtain information about their hosting servers. Gao et al. [27] also suggested that malicious containers can exhaust the computing resources of their host by escaping the resource control of Linux control groups.

Containers are also widely adopted in serverless computing [50, 51]. While most existing works attempt to enhance performance [52, 53] such as reducing startup time [54–56], few works focus on the security aspect of serverless computing, including potential vulnerabilities [57] and defenses [58]. Our work differs from all previous works in that our study focuses on revealing new vulnerabilities that attackers exploit to distribute malicious container images. Moreover, the uncovered container registry typosquatting attack complements previous studies on feasible and practical container attacks.

8.2 Container Registry Security

While many studies focus on improving the performance of container registries [59, 60], security issues in container registries have also received much attention. Gummaraju et al. [29] showed that around 40% of the container images uploaded by general users in Docker Hub are vulnerable to various cyber-attacks. Shu et al. [30] proposed a Docker Image Vulnerability Analysis (DIVA) tool and scanned a large amount of images in Docker Hub. Their study reveals that an average of 180 vulnerabilities exist in both official and community repositories. Furthermore, Zerouali et al. [61] demonstrated that outdated container images pose serious security threats due to those vulnerable, buggy packages. To the best of our knowledge, we are the first to study the typosquatting problem and its security threats to container registries.

8.3 Typosquatting

Our work shares the fundamental concepts of domain typosquatting. Early research can be traced back to 1999 when the Anticybersquatting Consumer Protection Act (ACPA) [62] was published. Since then, much research effort has been

devoted to studying domain name typosquatting, including the semantic features that can cause more typing errors [63] and the potential consequences caused by a typosquatting attack [64]. Szurdi et al. [19] demonstrated that the vast majority of typosquatting attackers are targeting lower ranking domains. Agten et al. [65] suggested that the majority of domain names are not properly protected against typosquatting attacks. In addition to the traditional domain names, Liu et al. [18] uncovered that typosquatting attacks also exist in International Domain Names (IDN). Le Pochat et al. [66] explored the possibility of typosquatting on non-English keyboard layouts. Tian et al. [67] revealed the wide existence of typosquatting domains and uncovered their high effectiveness on avoiding detection. Based on traditional domain typosquatting, many studies also revealed other squatting techniques targeting domain names, including combosquatting [68], bitsquatting [69, 70], and abbreviation squatting [71].

Moreover, typosquatting has been applied to scenarios beyond domain names, such as email [20] and mobile apps [21]. Studies also revealed potential squatting vulnerabilities in voice recognition systems [72–74]. Inspired by previous works, we reveal the container registry typosquatting threat.

9 Conclusion

In this paper, we conducted a systematic study on the container registry typosquatting threat. We demonstrated that adversaries can impersonate FQIDs of benign repositories to spread malicious container images. This attack can be launched within both public and private registries as well as across different platforms, posing realistic security threats to the container ecosystem. Our exploitation experiments for public and private registries show that users indeed make typing errors and download unwanted container images. We also validated that a large amount of typosquatting usernames, project-IDs, and domain names are currently available for public registration. To mitigate the threat, we proposed CRYSTAL that assists container registries to discover potential typosquatting FQIDs and alerts users about potential typing errors when downloading container images. We implemented a prototype of CRYSTAL on Docker CLI and achieved a high detection accuracy of more than 97.5% with low overhead.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful and constructive comments, which helped us to improve the quality of this paper. This work was supported in part by the National Science Foundation (NSF) grants DGE-1821744, CNS-2054657, and CNS-1815650, the Army Research Office (ARO) grant W911NF19-1-0049, and the Office of Navy Research (ONR) grant N00014-20-1-2153.

References

[1] AWS Elastic Container Servicee. https://aws.amazon.com/ecs/.

[2] Google Kubernetes. https://cloud.google.com/kubernetes-engine/.

[3] IBM Cloud Container Service. https://www.ibm.com/cloud/container-service.

[4] Zhanibek Kozhirbayev and Richard O Sinnott. A Performance Comparison of Container-Based Technologies for the Cloud. *Elsevier Future Generation Computer Systems*.

[5] Lele Ma, Shanhe Yi, Nancy Carter, and Qun Li. Efficient Live Migration of Edge Services Leveraging Container Layered Storage. *IEEE Transactions on Mobile Computing*, 2018.

[6] Miguel Xavier, Marcelo Neves, Fabio Rossi, Tiago Ferreto, Timoteo Lange, and Cesar De Rose. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In *IEEE Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013.

[7] Amazon Lambda. https://aws.amazon.com/lambda/.

[8] Azure Serverless Computing. https://azure.microsoft.com/en-us/overview/serverless-computing/.

[9] Xing Gao, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. ContainerLeaks: Emerging security threats of information leakages in container clouds. In *IEEE/IFIP DSN*, 2017.

[10] Jörg Thalheim, Pramod Bhatotia, Pedro Fonseca, and Baris Kasikci. Cntr: Lightweight OS Containers. In *USENIX ATC*, 2018.

[11] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. Cloud Container Technologies: A State-of-the-Art Review. *IEEE Transactions on Cloud Computing*, 2017.

[12] Ali Anwar, Mohamed Mohamed, Vasily Tarasov, Michael Littley, Lukas Rupprecht, Yue Cheng, Nannan Zhao, Dimitrios Skourtis, Amit S Warke, Heiko Ludwig, et al. Improving Docker Registry Design Based on Production Workload Analysis. In *USENIX FAST*, 2018.

[13] 60% of Organizations Suffered a Container Security Incident in 2018, Finds Study. https://www.tripwire.com/state-of-security/devops/organizations-container-security-incident/.

[14] Cryptojacking Invades Cloud. How Modern Containerization Trend is Exploited by Attackers. https://kromtech.com/blog/security-center/

cryptojacking-invades-cloud-how-modern-containerization-trend-is-exploited-by-attackers.

[15] Unit 42 CTR: Leaked Code from Docker Registries. https://unit42.paloaltonetworks.com/leaked-docker-code/.

[16] Attacker's Tactics and Techniques in Unsecured Docker Daemons Revealed. https://unit42.paloaltonetworks.com/attackers-tactics-and-techniques-in-unsecured-docker-daemons-revealed/.

[17] What is Typosquatting? https://www.mcafee.com/blogs/consumer/family-safety/what-is-typosquatting/.

[18] Baojun Liu, Chaoyi Lu, Zhou Li, Ying Liu, Hai-Xin Duan, Shuang Hao, and Zaifeng Zhang. A Reexamination of Internationalized Domain Names: The Good, the Bad and the Ugly. In *IEEE/IFIP DSN*, 2018.

[19] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich. The Long "Taile" of Typosquatting Domain Names. In *USENIX Security Symposium*, 2014.

[20] Janos Szurdi and Nicolas Christin. Email Typosquatting. In *ACM IMC*, 2017.

[21] Yangyu Hu, Haoyu Wang, Ren He, Li Li, Gareth Tyson, Ignacio Castro, Yao Guo, Lei Wu, and Guoai Xu. Mobile App Squatting. In *The Web Conference 2020*, 2020.

[22] Fred J Damerau. A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 1964.

[23] Tyler Moore and Benjamin Edelman. Measuring the Perpetrators and Funders of Typosquatting. In *Springer International Conference on Financial Cryptography and Data Security*, 2010.

[24] Yi-Min Wang, Doug Beck, Jeffrey Wang, Chad Verbowski, and Brad Daniels. Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting. *USENIX SRUTI*, 2006.

[25] Anirban Banerjee, Md Sazzadur Rahman, and Michalis Faloutsos. SUT: Quantifying and Mitigating URL Typosquatting. *Elsevier Computer Networks*, 2011.

[26] Hang Hu, Peng Peng, and Gang Wang. Characterizing Pixel Tracking through the Lens of Disposable Email Services. In *IEEE S&P*, 2019.

[27] Xing Gao, Zhongshu Gu, Zhengfa Li, Hani Jamjoom, and Cong Wang. Houdini's Escape: Breaking the Resource Rein of Linux Control Groups. In *ACM CCS*, 2019.

[28] Xin Lin, Lingguang Lei, Yuewu Wang, Jiwu Jing, Kun Sun, and Quan Zhou. A Measurement Study on Linux Container Security: Attacks and Countermeasures. In *ACSAC*, 2018.

[29] Jayanth Gummaraju, Tarun Desikan, and Yoshio Turner. Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities. *Technical Report, Banyan Ops*, 2015.

[30] Rui Shu, Xiaohui Gu, and William Enck. A Study of Security Vulnerabilities on Docker Hub. In *ACM CODASPY*, 2017.

[31] Know your Enemy: Phishing. http://honeynet.onofri.org/papers/phishing/index.html.

[32] Xiao Han, Nizar Kheir, and Davide Balzarotti. Phisheye: Live monitoring of sandboxed phishing kits. In *ACM CCS*, 2016.

[33] Vulnerability scanning for Docker local images. https://docs.docker.com/engine/scan/.

[34] Anchore. https://anchore.com/.

[35] Wiktionary Latest Dump. https://dumps.wikimedia.org/enwiktionary/latest/.

[36] Beast Domain Search. https://www.namecheap.com/domains/registration/results/?domain=&type=beast.

[37] Singularity. https://singularity.lbl.gov/.

[38] Podman. https://podman.io/.

[39] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An Updated Performance Comparison of Virtual Machines and Linux Containers. In *IEEE ISPASS*, 2015.

[40] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. Lightweight Virtualization: a Performance Comparison. In *IEEE International Conference on Cloud Engineering*, 2015.

[41] Ethan Young, Pengfei Zhu, Tyler Caraza-Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. The True Cost of Containing: A gVisor Case Study. In *USENIX HotCloud*, 2019.

[42] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, Mark L Stillwell, et al. SCONE: Secure Linux Containers with Intel SGX. In *USENIX OSDI*, 2016.

[43] Lingguang Lei, Jianhua Sun, Kun Sun, Chris Shenefiel, Rui Ma, Yuewu Wang, and Qi Li. SPEAKER: Split-Phase Execution of Application Containers. In *Springer DIMVA*, 2017.

[44] Yuqiong Sun, David Safford, Mimi Zohar, Dimitrios Pendarakis, Zhongshu Gu, and Trent Jaeger. Security Namespace: Making Linux Security Frameworks Available to Containers. In *USENIX Security Symposium*, 2018.

[45] Jaehyun Nam, Seungsoo Lee, Hyunmin Seo, Phil Porras, Vinod Yegneswaran, and Seungwon Shin. BASTION: A Security Enforcement Network Stack for Container Networks. In *USENIX ATC*.

[46] Udit Gupta. Comparison between Security Majors in Virtual Machine and Linux Containers. *arXiv preprint arXiv:1507.07816*, 2015.

[47] Aaron Grattafiori. Understanding and Hardening Linux Containers. *Whitepaper, NCC Group*, 2016.

[48] Yang Luo, Wu Luo, Xiaoning Sun, Qingni Shen, Anbang Ruan, and Zhonghai Wu. Whispers between the Containers: High-Capacity Covert Channel Attacks in Docker. In *IEEE Trustcom/BigDataSE/ISPA*, 2016.

[49] Xing Gao, Benjamin Steenkamer, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. A Study on the Security Implications of Information Leakages in Container Clouds. *IEEE TDSC*, 2018.

[50] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*. 2017.

[51] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Serverless Computation with Openlambda. In *Usenix HotCloud*, 2016.

[52] Vikram Sreekanti, Chenggang Wu, Saurav Chhatrapati, Joseph Gonzalez, Joseph Hellerstein, and Jose M Faleiro. A Fault-Tolerance Shim for Serverless Computing. In *ACM Eurosys*, 2020.

[53] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov, Feng Yan, and Yue Cheng. InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache. In *USENIX FAST*, 2020.

[54] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. SAND: Towards High-Performance Serverless Computing. In *Usenix ATC*, 2018.

[55] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. Catalyzer: Sub-millisecond Startup for Serverless Computing with Initialization-less Booting. In *ASPLOS*, 2020.

[56] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. SOCK: Rapid Task Provisioning with Serverless-Optimized Containers. In *USENIX ATC*, 2018.

[57] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. Peeking Behind the Curtains of Serverless Platforms. In *USENIX ATC*, 2018.

[58] Pubali Datta, Prabuddha Kumar, Tristan Morris, Michael Grace, Amir Rahmati, and Adam Bates. Valve: Securing Function Workflows on Serverless Computing Platforms. In *The Web Conference*, 2020.

[59] Tyler Harter, Brandon Salmon, Rose Liu, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Slacker: Fast Distribution with Lazy Docker Containers. In *USENIX FAST*, 2016.

[60] Nannan Zhao, Hadeel Albahar, Subil Abraham, Keren Chen, Vasily Tarasov, Dimitrios Skourtis, Lukas Rupprecht, Ali Anwar, and Ali R Butt. DupHunter: Flexible High-Performance Deduplication for Docker Registries. In *USENIX ATC*, 2020.

[61] Ahmed Zerouali, Tom Mens, Gregorio Robles, and Jesus M Gonzalez-Barahona. On the Relation between Outdated Docker Containers, Severity Vulnerabilities, and Bugs. In *IEEE SANER*, 2019.

[62] The Anticybersquatting Consumer Pretection Act. https://www.govinfo.gov/content/pkg/CRPT-106srpt140/html/CRPT-106srpt140.htm/.

[63] Rashid Tahir, Ali Raza, Faizan Ahmad, Jehangir Kazi, Fareed Zaffar, Chris Kanich, and Matthew Caesar. It's All in the Name: Why Some URLs are More Vulnerable to Typosquatting. In *IEEE INFOCOM*, 2018.

[64] Mohammad Taha Khan, Xiang Huo, Zhou Li, and Chris Kanich. Every Second Counts: Quantifying The Negative Externalities of Cybercrime via Typosquatting. In *IEEE S&P*, 2015.

[65] Pieter Agten, Wouter Joosen, Frank Piessens, and Nick Nikiforakis. Seven Months' Worth of Mistakes: A Longitudinal Study of Typosquatting Abuse. In *NDSS*, 2015.

[66] Victor Le Pochat, Tom Van Goethem, and Wouter Joosen. A Smörgåsbord of Typos: Exploring International Keyboard Layout Typosquatting. In *IEEE S&P Workshops (SPW)*, 2019.

[67] Ke Tian, Steve TK Jan, Hang Hu, Danfeng Yao, and Gang Wang. Needle in a Haystack: Tracking Down Elite Phishing Domains in the Wild. In *ACM IMC*, 2018.

[68] Panagiotis Kintis, Najmeh Miramirkhani, Charles Lever, Yizheng Chen, Rosa Romero-Gómez, Nikolaos Pitropakis, Nick Nikiforakis, and Manos Antonakakis. Hiding in Plain Sight: A Longitudinal Study of Combosquatting Abuse. In *ACM CCS*, 2017.

[69] Nick Nikiforakis, Steven Van Acker, Wannes Meert, Lieven Desmet, Frank Piessens, and Wouter Joosen. Bitsquatting: Exploiting Bit-Flips for Fun, or Profit? In *WWW*, 2013.

[70] Thomas Vissers, Timothy Barron, Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. The Wolf of Name Street: Hijacking Domains Through their Nameservers. In *ACM CCS*, 2017.

[71] Pin Lv, Jing Ya, Tingwen Liu, Jinqiao Shi, Binxing Fang, and Zhaojun Gu. You Have More Abbreviations Than You Know: A Study of AbbrevSquatting Abuse. In *ICCS*, 2018.

[72] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. Skill Squatting Attacks on Amazon Alexa. In *USENIX Security Symposium*, 2018.

[73] Nan Zhang, Xianghang Mi, Xuan Feng, XiaoFeng Wang, Yuan Tian, and Feng Qian. Dangerous Skills: Understanding and Mitigating Security Risks of Voice-Controlled Third-Party Functions on Virtual Personal Assistant Systems. In *IEEE S&P*, 2019.

[74] Yangyong Zhang, Lei Xu, Abner Mendoza, Guangliang Yang, Phakpoom Chinprutthiwong, and Guofei Gu. Life after Speech Recognition: Fuzzing Semantic Misinterpretation for Voice Assistant Applications. In *NDSS*, 2019.