Misdirection of Trust: Demystifying the Abuse of Dedicated URL Shortening Service

Zhibo Zhang, Lei Zhang, Zhangyue Zhang, Geng Hong, Yuan Zhang, Min Yang
Fudan University
{*zhibozhang19, zxl, ghong, yuanxzhang, m_yang*}*@fudan.edu.cn, zhangyuezhang23@m.fudan.edu.cn*

***Abstract*—Dedicated URL shortening services (DUSSs) are designed to transform *trusted* long URLs into the shortened links. Since DUSSs are widely used in famous corporations to better serve their large number of users (especially mobile users), cyber criminals attempt to exploit DUSS to transform their malicious links and abuse the inherited implicit trust, which is defined as *Misdirection Attack* in this paper. However, little effort has been made to systematically understand such attacks. To fulfill the research gap, we present the first systematic study of the *Misdirection Attack* in abusing DUSS to demystify its attack surface, exploitable scope, and security impacts in the real world.**

Our study reveals that real-world DUSSs commonly rely on custom URL checks, yet they exhibit unreliable security assumptions regarding web domains and lack adherence to security standards. We design and implement a novel tool, `Ditto`[1], for empirically studying vulnerable DUSSs from a mobile perspective. Our large-scale study reveals that a quarter of the DUSSs are susceptible to *Misdirection Attack*. More importantly, we find that DUSSs hold implicit trust from both their users and domain-based checkers, extending the consequences of the attack to stealthy phishing and code injection on users' mobile phones. We have responsibly reported all of our findings to corporations of the affected DUSS and helped them fix their vulnerabilities.

I. Introduction

Nowadays, to better attract mobile users, web service providers often employ shortened links to facilitate content sharing and user tracking [1]. The shortened link is a condensed version of a long and complex URL (Uniform Resource Locator) link with memorable and manageable features created by the URL shortening service (USS for short). However, a shared USS (SUSS) like Bitly [2] often becomes notorious for being used in spam and malware distribution [3]. To this end, famous corporations prefer to develop their dedicated USS (DUSS) to gain users' trust and offer greater user convenience while earning more benefits. The DUSS often uses the brand domain name[2] to host shortened links; therefore, these links inherit a good reputation from its corporation. Besides, the "dedicated" feature makes users implicitly perceive that accessing DUSS-shortened links is equivalent to accessing trustworthy web content. For example, *go.nasa.gov* is a domain used by NASA's DUSS to signal users that clicking on its links will lead to NASA-related web pages [4]. Moreover, eight leading corporations out of Fortune top 10 also set up their DUSSs, including Walmart (*walmrt.us*), Amazon (*amzn.to*), and Shell (*go.shell.com*).

Owing to its potential to "transform" a malicious domain into a reputable one, DUSS has garnered significant attention and preference among cyber attackers in past decades [5], [6], [7]. For example, in 2012, Symantec reported [5] that spammers abused the *1.usa.gov* (a DUSS of the USA government) to host their spam websites, having misdirected 40 thousand victim users to visit in six days. In this paper, we refer to this type of attack as a *Misdirection Attack*. The security implications of *Misdirection Attack* are substantial due to the *transitivity of trust*[3][8], [9]. Namely, social network users and domain-based checkers that trust DUSS-shortened links can be tricked into loading malicious websites into downstream software. First, scammers can abuse DUSS to generate shortened links that redirect users to their phishing websites to steal private information. Second, remote attackers can also leverage DUSS-shortened links to deploy malicious code to tamper with software integrity and confidentiality.

To the best of our knowledge, there is still a lack of thorough understanding of *Misdirection Attack*, and no prior work has studied them in the wild. On the one hand, prior works [10], [11], [12], [13], [14] mainly focus on identifying and detecting malicious shortened links, but essential problems, i.e., the root cause, detection, and consequences of *Misdirection Attacks*, have not been well studied. On the other hand, other prior works focus on the abuse of dedicated web services, such as cloud storage [15], [16] and web rehosting services [17], [18]. Thus, this paper focuses explicitly on dedicated USSs, the crucial web service infrastructures for famous corporations. These DUSSs set themselves apart from other USSs and web services by their highly ranked domains and security design that only serves trusted links. To fill this gap and conduct a systematic study on the entire attack surface, detection, and security impacts of *Misdirection Attack* in the real world, we formulate three-phased research questions:

RQ1. *What are the security design of DUSSs and their potential attack surfaces?*

[1]Our source code is available at https://github.com/seclab-fudan/Ditto.git.

[2]A brand domain give users a strong psychological and visual implication and indicate the relevance of the domain name to the corporation.

[3]If A trust B, and B trust C, then A trusts C

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230927
www.ndss-symposium.org

RQ2. *How do we automatically detect whether existing DUSSs are robust to the Misdirection Attack?*

RQ3. *What security implication does the Misdirection Attack have on social network users and domain-based checkers?*

Answering these questions is challenging due to the invisible, black-box, and customized implementation of DUSS. We present three studies to address these questions respectively:

Understanding security design and attack surfaces. By leveraging a top-down approach (see Section III), we locate 88 high-profile DUSSs exist on social media platforms. We further examine their underlying infrastructure, including the use of a *deployment model* and the adoption of *security enforcement*. Our research indicates that existing DUSSs typically support client applications (e.g., mobile apps) developed by their corporations accessing link shortening services through application programming interfaces (APIs) and implement customized checks for URLs to be shortened, such as checking whether the domain is in an allowlist. However, we find that *despite the adoption of security checks, the possibility of abusing DUSS still remains*, as these security checks are susceptible to various vulnerabilities. The reasons are two-fold. First, the lack of standardization in implementing URL checks can lead to security vulnerabilities. Second, the domain-based allowlist is unreliable because it mistakenly assumes that if the domain server is trustworthy, the URL is also considered trustworthy. Such an assumption overlooks other security attributes of a URL, such as domain ownership transfer or code injection on vulnerable websites. These weaknesses open loopholes in the security checks.

Detecting vulnerable DUSSs in the wild. Driven by the above security concerns, it is nontrivial to build an automated tool to locate the link shortening APIs of the 88 DUSSs and test their security in the wild. In this paper, we design and implement a tool, `Ditto`, which focuses on the mobile apps developed by these DUSS corporations (see Section IV). It faces unique challenges in identifying customized link shortening APIs and conducting black-box security assessment. `Ditto` has two key insights. On the one hand, it identifies the link shortening API by analyzing client-side Web API implementation logic, which outlines its general functionality. On the other hand, by simplifying all potential vulnerability-validation test cases and optimizing the testing sequence based on their dependencies, `Ditto` can avoid unnecessary tests and efficiently perform vulnerability detection. To this end, `Ditto` reports 50 link shortening APIs and finds that 22 can be abused by malicious attackers. The vulnerable DUSS corporations are widely distributed, from telecom (e.g., China Unicom) and e-commerce (e.g., Amazon) to news and media (e.g., Sina).

Security implications. To better understand the security threats of *Misdirection Attack*, we perform a survey study to analyze how they differ from traditional phishing techniques and impact domain-based checkers. Specifically, the compromised DUSS domain plays a critical role in users and domain-based checkers adopted by software due to the *trust transitivity*. That is, social network users or security checkers such as mobile applinks, OAuth allowlist, and link warning services can be deceived if they place excessive trust in the compromised DUSS domain. To illustrate these implications, we survey various security checkers based on domain allowlist and verify them (see Section V). We emphasize that DUSSs are more often used to serve mobile users, making mobile device users the most significant victims of *Misdirection Attack*. We also present several case studies to demonstrate the security consequences of compromising these checkers, including serious data leakage, code injection, and phishing. We have responsibly reported our findings to all 22 vulnerable DUSS corporations and received their acknowledgment.

Contributions. We summarize the contributions of this paper as below:

- *New understanding.* We conduct the first systematic study of the dedicated URL shortening services and uncover 88 high-profile DUSSs of real-world corporations, including their security design and attack surface.
- *Vulnerable DUSS detection.* We present a tool, `Ditto`, that can effectively and accurately discover DUSS with vulnerable link shortening API in the wild. We report all 22 vulnerable DUSSs to their developers and offer actionable recommendations to help developers minimize and mitigate the threat of *Misdirection Attack*.
- *Various security implications.* We perform a systematic study to illustrate the vast security implications of *Misdirection Attack*, including phishing, and bypassing downstream domain-based checker to abuse critical app functions and cause privacy leakage. The attack can affect millions of mobile and desktop users.

II. Background

In this section, we present the definition of DUSS and a motivating example to illustrate how a *Misdirection Attack* happens. Then, we thoroughly discuss our threat model and research scope.

A. Dedicated URL Shortening Service

A dedicated URL shortening service uses the brand name (provided by the corporation) as the domain name of shortened links. From a technical perspective, DUSS receives a relatively long URL in the query and generates a shortened link while maintaining the mapping relationship in a database and redirecting visitors of the shortened link to the original URL. The brand name is either an abbreviation or a sub-domain name of the corporation, which is easy for online users to recognize and remember. Moreover, DUSS can track its customers by generating individual shortened links and gain benefits from this. For example, sellers on Amazon shopping use shortened links (e.g., https://amzn.to/45TDYok) of their product URLs (e.g., https://www.aboutamazon.com/news/retail/.....?utm_source=SOCIAL&utm_medium=TWITTER&.....), which contain user tracking flags for different social media platforms (e.g., Twitter). Thus, Amazon sellers can collect the source of their visitors to build more precise user profiles.

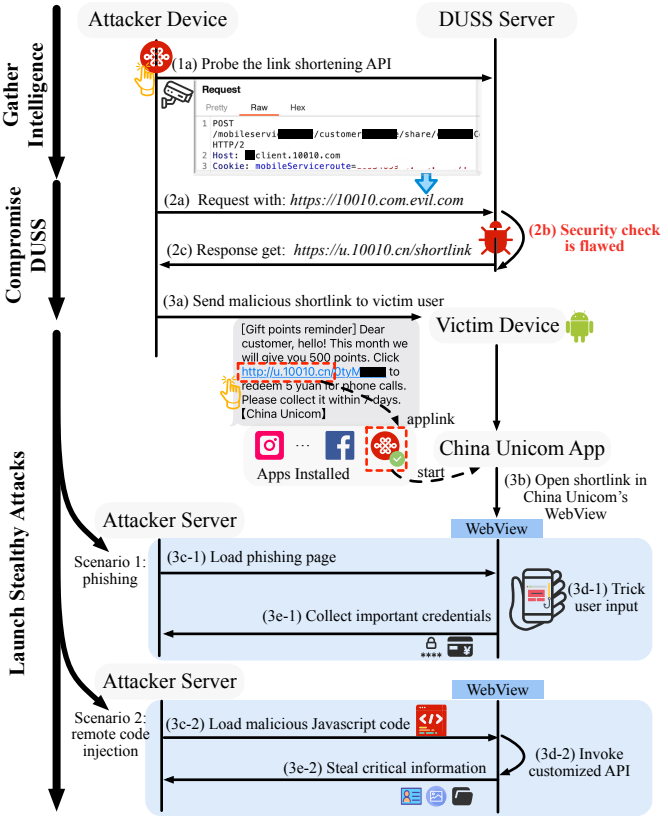

Fig. 1. Simplified motivating example for a remote attacker to exploit China Unicom's DUSS and launch phishing and code injection attacks.

Traditional SUSSs often use a shared domain name (e.g., *bit.ly*), but is notorious for being abused in cyber crimes [11], [12], [14]. This is primarily due to the public nature of SUSS, which brings uncertainty to the behind-the-back URLs. In stark contrast, DUSS is designed with a strong focus on trustworthiness, a key feature that sets it apart from SUSS. This emphasis on trustworthiness requires significant efforts in the security design of DUSS. To this end, the DUSS establishes its trust through two crucial properties: (i) a reputable domain with a high web rank and (ii) a strict policy of only serving trusted URLs. Therefore, the security of DUSS is critical.

B. Misdirection Attack

1) Definition: A *Misdirection Attack* is a deceptive strategy that misdirects security attention and reliance from an untrusted entity to one perceived as trustworthy, leading to a compromise. In DUSSs, their brand domains are often granted with great reputation, thus being trusted by users in social networks and developers who deploy domain-based checkers in the applications or systems. However, if the landing web pages are malicious, they can inherit the trust due to the *trust transitivity*. To this end, a malicious attacker can exploit the vulnerable DUSS and transform malicious URLs into dedicated shortened links, abusing its trust.

2) Motivating Example: In this section, we illustrate *Misdirection Attack* by describing a motivating example of China Unicom corporation[4]. This attack exploits China Unicom's vulnerable DUSS to shorten malicious URLs, misdirecting its mobile app users to visit and load these malicious links. This eventually leads to phishing and code injection attacks on millions of users, abusing critical app functions such as arbitrary APK download and installation on the Android platform and theft of sensitive files from users' mobile devices.

Let us describe the steps of the end-to-end attack, as shown in Figure 1. This attack can be detailed in eight steps, divided into three major phases: (i) gathering intelligence about DUSS of China Unicom, (ii) compromising and abusing the DUSS to transform malicious URLs into the shortened links, (iii) launching stealthy attacks for phishing and code injection.

First, an attacker needs to identify the entry point to abuse China Unicom's DUSS, which is non-public and hidden. This requires reverse engineering and network traffic analysis while dynamically exploring China Unicom's client-side applications. A detailed use case can be found in Appendix A. The China Unicom mobile app [20] is the only one that can generate shortened links with the domain name *u.10010.cn* while sharing its advertising pages. As shown in Step (1a), this sharing behavior indicates that a Web API is invoked to access the DUSS. The API's network traffic is encrypted using TLS protocol. The request body contains a long URL (e.g., an advertisement), and a shortened link to the long URL is returned in response. At this stage, the attacker has only gained access to DUSS, but is not yet able to abuse it.

Second, let us examine the second phase of compromising the vulnerable DUSS, which involves abusing the DUSS to shorten malicious links hosted on the attacker's server. In Step (2a), the attacker modifies (see Appendix A) the Web API traffic, setting a malicious long URL in the request body. In Step (2b), the DUSS performs a security check on the received URL. However, the check is flawed and leads to compromise. Specifically, the DUSS checks if *10010.com* is in the URL host received for shortening. This ad-hoc check can be bypassed with attacker-controlled domain names (e.g., *10010.com.evil.com*). Finally, Step (2c) successfully retrieves a shortened malicious link from the response body.

Third, let us examine the third phase: launching stealthy attacks on victim users via shortened links. The attack begins with a tactic that tricks users into clicking the shortened link in phishing SMS messages[5] (Step 3a). For example, the attacker mimics the daily award notice message of China Unicom [21], prompting users to click the shortened link. Additionally, the attacker can send the message through a fake base station [22], [23], enhancing its deceptiveness. Interestingly, the China Unicom app registers *u.10010.cn* as a hyperlink (i.e., applink [24]) to Android system, directing users to a specific WebView (an embedded browser) component within the app[6] when they click the shortened links (Step 3b). The stealthy attack then

[4]China Unicom is one of the largest communications companies in China with more than 969 million users [19].

[5]Attackers can also use email and social platforms as alternatives.

[6]If the app is not installed, the mobile browser will load a web page provided by China Unicom to help users download and install it.

divides into two parallel scenarios: phishing and code injection into the victim's mobile device.

In the phishing scenario, the malicious website is presented within the China Unicom app (Step 3c-1), tricking users into entering sensitive credentials such as bank card numbers and passwords (Step 3d-1). Attackers can then stealthily collect this information (Step 3e-1). It is difficult for users to identify fake websites because the WebView component hides the address bar and allows the website to customize its title.

In the parallel scenario, malicious JavaScript is injected into China Unicom's customized WebView runtime (Step 3c-2). China Unicom adds rich custom Java APIs to the WebView runtime via `addJavascriptInterface` [25]. Thus, the attack can cause serious consequences, such as installing malicious APK or stealing sensitive files from the victim's mobile device (Step 3d-2). For example, the `uploadFile` API, which inherits the file access permissions granted to the app, can be exploited by the attacker to transfer sensitive files to their server, including account token files and photos from the user's gallery (Step 3e-2).

Bottom line. The most important fact is that the *Misdirection Attack* happens due to vulnerable security checks in the DUSS. Therefore, the key is to identify which DUSS implementations have such weak security mechanisms.

3) Threat Model: Let us describe the threat model adopted in this paper. We consider the attacker a remote web attacker who exploits vulnerable DUSS to transform malicious URLs into shortened links with implicit trust. In this context, the attacker exploits the "link shortening service," a functional component supported by the vulnerable DUSS in the client-side applications (e.g., websites, and mobile apps). We assume the attacker has no specific privileges or access to the DUSS server. Additionally, the shortened malicious URL may carry phishing web content or malicious JavaScript code, damaging the victim users. As victim users, they are either ordinary online social media users or those using specific downstream software affected by *Misdirection Attack*. The software includes client-side applications and web services that can visit and load shortened links. For instance, mobile apps that use DUSS to generate shortened links may also load these links. These software may use a domain-based checker to decide whether the URL can be loaded or visited; thus, they can be bypassed by malicious shortened links due to trust transitivity. More details on exploitation can be found in Section V.

C. Scope of Problem

Note that our research scope focuses on the DUSSs with good reputations but can be abused for *Misdirection Attack*; therefore, the privately developed USSs for malware distribution and spam are out of scope. The reason is that the malicious USS intends to host malicious resources from various hidden servers, which do not need a security guarantee for serving trusted content. Also, popular SUSS (e.g., "t.co" in Twitter) is out of the scope because it serves links from anyone without limitation and lacks trust.

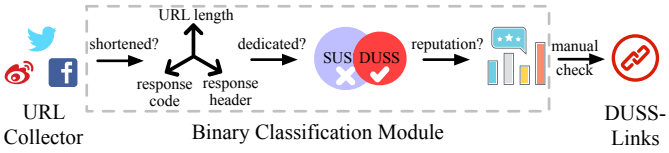

Fig. 2. Top-down approach overview for locating real-world DUSSs.

III. DUSS: A Preliminary Study

In this section, we perform a preliminary study on existing DUSSs. We aim to understand their popularity, security designs (including deployment models and security checks), and potential attack surfaces.

A. DUSS Popularity

We bootstrap our study by collecting real-world DUSS-shortened links as a data set. The key insight is that the DUSS-shortened links can reflect the real-world usage of DUSSs despite these DUSSs being hidden. However, it is challenging to determine which links are shortened by DUSSs, considering the vast amount of URLs on the Internet. To address this, we present a top-down approach using binary classification modules for different URL features, as illustrated in Figure 2.

First, we use a web crawler to collect public user-posted URLs from three leading social media platforms: Twitter, Facebook, and Weibo. For example, we manually search the verified accounts of Fortune Top 500 companies on Facebook, collect their Page IDs, and submit these IDs to the Facebook Graph API [26] using our authorized access token to collect their historical posts. Finally, we use regular matching to extract URLs from the full text of these posts.

Second, we dynamically visit them and classify whether they are shortened links based on two features: short length and link redirection. We can learn the baseline of these features from SUSS-shortened links because of their similarity. Specifically, we collect and analyze 1,000 shortened links from 654 SUSSs extended from previous work [10]. Our observations show that a shortened link has: (i) a maximum URL path length of less than 17 characters; (ii) an HTTP response header contains "30x" HTTP code or uses "Location" or "Refresh"; (iii) or a response body contains navigation code, such as modifying the "location" property of the DOM.

Third, if they are shortened links, we exclude SUSS-shortened links by comparing their domain with the above SUSSs. After that, we identify whether the domain name has a good reputation by matching it with the top 1M SLD[7] domains from Tranco [27]. Because the Tranco list ensures the domain name has a good reputation, thus malicious domains can be filtered out. Finally, by manually checking their website ownership, we verify whether they are DUSS-shortened links.

We successfully find 1,038 DUSS-shortened links from 130 million URLs (in a month experiment in August 2023), which can be grouped into 88 unique DUSSs by their FQDNs (fully qualified domain names). These popular DUSSs are provided

[7]Second-level domain represents the website and its owner.

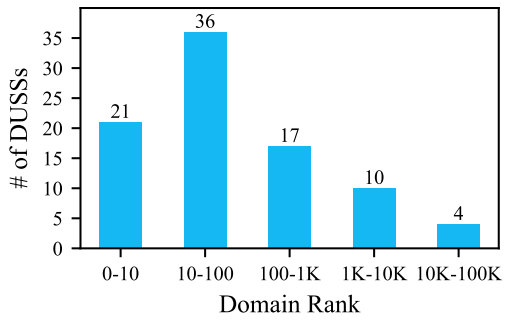

Fig. 3. Domain category ranking statistic of DUSSs.

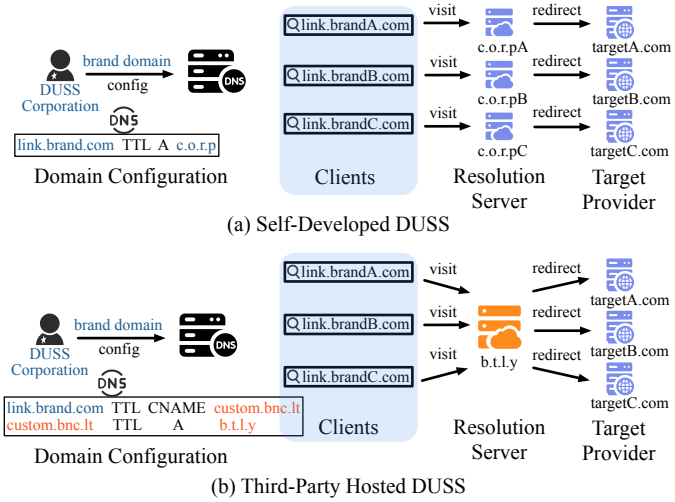

Fig. 4. Two typical deployment models of DUSS.

by famous corporations such as Baidu, Reddit, and YouTube. Figure 3 illustrates the website ranking in their corresponding category[8]. There are 64.8% DUSS domains (57 out of 88) ranking within 100, and 90% of the domains ranking within 4K. These statistics explain the great popularity and reputation of DUSS in daily life.

B. Popular DUSS Infrastructure

In this section, we perform a preliminary study on how the 88 DUSSs are deployed and used for link shortening. The purpose is to understand the typical infrastructure of DUSS and further support the automatic mining of these link shortening APIs (see Section IV). Our methodology is as follows. First, we cluster the domain DNS records (i.e., CNAME, A record) of 88 DUSSs and search them in online engines to unravel their deployment models. Second, we manually sample the top-ranked DUSSs to analyze their access methods. Specifically, we collect the landing web pages from their shortened links and explore their functionalities to verify the use of link shortening services.

Deployment Model. Figure 4 illustrates two typical models in existing DUSSs. The details are described below:

[8]The ranking data is collected from Alexa web ranking [28], which provides more information about the ranked domain than Tranco.

- *Self-Developed.* In this model, the corporation provides the servers and binds the domain name DNS record directly to the IP address of the link resolution server.
- *Third-Party Hosting.* In this model, commercial USS provides the server, and the corporation configures the domain name DNS record (e.g., CNAME, A) to point to the alias name or IP address of the third-party resolution server. We find five popular commercial USSs, including Bitly PREMIUM [2], TinyURL pro [29], Rebrandly [30], Branch.io [31], and AppsFlyer [32].

Our statistic shows that over 73.9% of DUSSs (65 out of 88) are self-developed, while 23 are third-party hosted. We list the top-ranked 15 DUSSs and five third-party hosting services in Appendix Table III and IV, respectively.

Link Shortening. Typically, there are two ways to use the DUSS for link shortening, as described below:

- *Access from client-side applications.* In this scenario, the client-side applications invoke the link shortening API to dynamically generate the shortened links. We find that at least 53.3% of DUSSs (8 out of 15) can be accessed by their client-side applications, which are usually initiated by users during sharing activities. Among these, four are accessible by mobile and web apps, while the remaining four can only be accessed by mobile apps.
- *Access from the admin portal.* In this scenario, the web content provider can directly log into the DUSS admin portal, and manually generate the shortened links. We assume the 7 (out of 15) DUSSs shortening links in their admin portals since we do not find any shortened links generated when manually exploring their applications.

Over half of the DUSSs use the former method, exposing a broad attack surface. Interestingly, the link-shortening server often uses a private domain name and IP address, distinct from those used by the shortened link resolution server. This design may reduce the attack surface of link shortening services, but it also makes it more challenging for security researchers to discover and test their security.

C. Security Design and Attack Vector

1) Security Design: In this section, we perform a survey study on what security checks exist in DUSSs. Our methodology is as follows: We first analyze the security-related description from documentation of five third-party commercial services about how to secure the link shortening service. Then, we also manually test the link shortening service in the top 15 self-developed DUSSs to cross-validate our findings.

The URL contains three parts: scheme, domain, and path, which naturally categorize three parallel security checks:

- Scheme Check. This checks whether an allowed URI scheme (e.g., http, https, or developer configured) is contained in the URL.
- Domain Check. This checks whether the FQDN matches with the allowlist configured by developers. The comparison can be strict equals or vague matching (e.g., endswith or contains).

- Path Check. This checks whether the URL strictly matches the prefix configured by developers, e.g., only the URL that starts with "https://www.reddit.com/r/" can be shortened by Reddit.

However, the implementation of security checks can be diversified in different DUSSs. Not only is there no standard for security checks, but DUSS security depends on how developers configure these allowlists, indicating that various security holes could exist in the DUSS ecosystems.

2) Attack Vector: For each type of security check, we summarize the potential vulnerabilities that could lead to the abuse of DUSSs. These vulnerabilities are reviewed from academic work [33], [34] and non-academic report [35] related to URL check bypass. Then, we manually test the 15 popular DUSSs and five third-party hosting services using our accounts and servers. In particular, we systematized the different attack vectors that can be exploited to bypass these checks in DUSS.

Flawed Scheme Check. This flaw is that DUSS does not check the URI scheme. Critical URI schemes such as javascript can cause malicious code injection in the mobile WebView [36]. For example, the "javascript://allowed.com/%0a%0dalert(1)" can bypass the domain check and execute JavaScript code in WebView. Customized malicious schemes can be recognized as deeplinks by Android or iOS that open specific applications. For example, "evilscheme://poc" can launch the malicious apk installed on the mobile device.

Flawed Domain Check. We identify three flaws that may lead to a domain check bypass:

- Flawed Domain Parsing. This flaw happens when parsing URLs and extracting their FQDNs. For example, "malicious.com\.benign.com" is recognized as a subdomain of *benign.com*; however, it eventually loads the *malicious.com*. For another example, "allowed.com:x@evil.com" may be mistakenly parsed, and *allowed.com* is checked as the domain name, but *evil.com* is visited eventually in the browser.
- Flawed Domain Matching. This flaw is that the DUSS uses a broad matching algorithm, thus being able to serve potential malicious URLs. The main reason is the lack of balance between business and security requirements. For example, DUSS may use `endswith` or `contains` to allow different sub-domains within the same host domain. Therefore, when the allowlist contains *benign.com*, an adversary can bypass the check using a domain like *evilbenign.com* or *benign.com.evil.com*.
- Flawed Domain Asset. Even if the matching algorithm and URL parser are correct, the domain check may still be vulnerable to other web content manipulation. First, the allowed domain may be susceptible to subdomain takeover [18], domain expiration, or domain abuse in public hosting services, such as a SUSS or content delivery network (CDN). Second, URLs under the domain could have vulnerabilities that allow an attacker to control its content, such as open redirect flaws (CWE-601) and cross-site scripting (XSS).

Flawed Path Check. This flaw is a subset of the flawed domain assets in the domain check. The path check is flawed if the vulnerable URL includes the legitimate path suffix.

To summarize, the *lack of standardization and diversified implementation of security checks in link shortening service* indicates a worrying situation about the security of DUSSs. Considering that DUSSs are frequently accessed from client-side applications of their corporations, the attack surfaces are widely exposed. The results of this preliminary study urge us to build an automatic tool to inspect the real-world prevalence of vulnerabilities in DUSS (RQ2) in depth.

IV. Vulnerable DUSS in the wild

In this section, we present our measurement study on the prevalence of DUSSs that can be abused in the real-world by leveraging a novel DUSS interface testing tool, `Ditto`. We start by describing the overview technique challenges and `Ditto`'s solution in Section IV-A and present three stages of `Ditto` and their implementation in Sections IV-B– IV-E. The results of this study are presented in Section IV-F.

A. `Ditto` Overview

We bootstrap our study by recognizing and testing client-side DUSS link shortening APIs from the mobile applications developed by the DUSS corporation. The reasons are twofold. On the one hand, the client-side link shortening API is the only attack entry of the web attacker without the need to hack into the DUSS server, as defined in our threat model. On the other hand, mobile apps developed by DUSS corporations are most likely access to DUSSs (see Section III). However, this is challenging due to the following reasons.

1) Technique Challenge: First, it is hard to identify the link shortening APIs, considering various web service APIs based on HTTP/HTTPS communication in the client-side applications. Specifically, it is hard to identify the link shortening APIs of self-developed DUSSs, which are often customized in the URL hosts, paths, and queries. Second, additional security measures, such as cryptographic protection, are implemented to safeguard the integrity of dedicated API requests. We manually sample the network traffic of eight DUSSs identified in Section III-B and find that six are strictly protected by API signature or body encryption. These defenses make intuitive API testing approach, such as replaying the reconstructed API requests and modifying their required values as done in previous works [37], impractical in this scenario as it compromises integrity guarantees. Last but not least, ensuring the effectiveness of vulnerability detection while minimizing the ethical burden on DUSSs poses a challenge. That is, conducting random, exhaustive black-box testing, as seen in previous work [33], where enumerating a large amount of possible illegal characters is not feasible.

2) Solutions Overview: Now let us describe how `Ditto` addresses the aforementioned challenges. The overall architecture of `Ditto` is illustrated in Figure 5, which consists of three major stages.

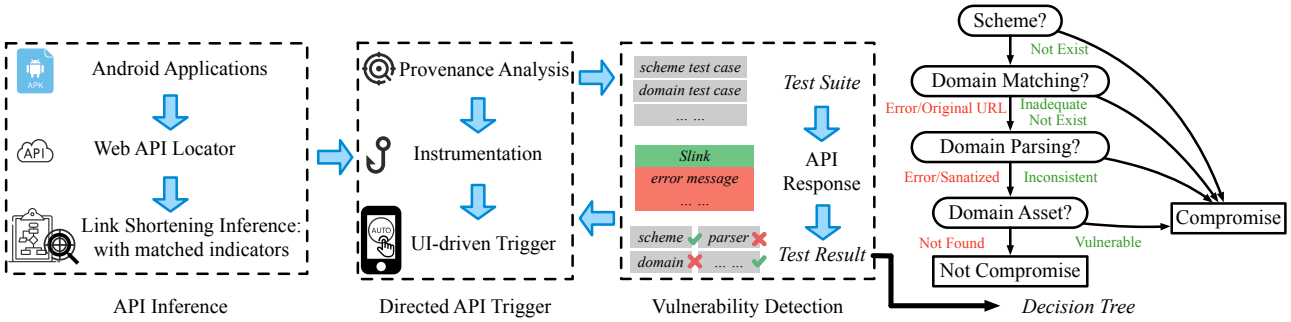

Fig. 5. The architecture overview of Ditto.

First, Ditto infers the link shortening behavior from the implementation logic of a Web API. Specifically, Ditto statically identifies the Web API call sites and analyzes their call context along control- and data-flow graphs to find indicators that reflect the functional behavior of link shortening, e.g., request and response traffic for self-developed DUSS or usage of standard SDK methods for third-party hosted DUSS. If any indicators are detected, Ditto reports the API as a potential link shortening API. The correctness of inference is verified in the directed API trigger stage.

Second, Ditto uses a dynamic API trigger to generate legal and modifiable API requests while bypassing API integrity safeguards. The key insight is that integrity safeguards become active only when the API request body is formed; therefore, we can bypass them by modifying the source of the URL to be shortened. Specifically, Ditto backtraces the URL object that populates the request body to find its provenance–typically a function due to programming conventions–and alters its value before the construction of the request body. To achieve this, Ditto leaves instrumentation on the provenance method for the convenience of modifying URL value in vulnerability detection. With all the work done, Ditto uses the UI-driven application testing technique to trigger the UI event that leads to the call of link shortening API. To this end, we could bypass cryptography defenses and generate legal API requests in vulnerability detection.

Lastly, Ditto performs vulnerability detection guided by a minimal decision tree with little overhead to the back-end server. The decision tree contains a sequence of test suites, which follows the control variate principle to test different categories of vulnerability mentioned in Section III. When the target link shortening API is triggered, Ditto mutates the URL to be shortened to generate a crafted one that lands at our server (i.e., untrusted web server) or contains the attacker-controlled script. By leveraging the decision tree on the API response, Ditto decides whether to conduct further testing or conclude the procedure and generate a vulnerability report.

B. Link Shortening API Inference

Ditto searches Android applications to identify potential DUSS link shortening APIs. Specifically, Ditto takes two important steps to recognize the link shortening API: (i) collecting Web API call sites, and (ii) identifying behavior indicators of link shortening.

1) Web API Locator: Ditto identifies the Web API call sites of various network request methods as our points of interest (POIs). Specifically, the network request methods are summarized from previous work [38], [37] that analyze Web APIs in Android applications, which includes the standard network request libraries in Android and Java (e.g., android.net.http.execute, java.net.Socket) and popular third-party SDKs (e.g., Okhttp and Retrofit2). To reduce the search scope, we collect UI event handlers (e.g., methods that override View.onClick, onLongClick, and AdapterView.onItemClick) as entry points, analyze their reach-ability to Web API call sites, and retain only the reachable call sites as our POIs. Because link shortening behavior is passively triggered by users when using the application (Section III).

2) Link shortening inference: We consider a Web API to have link shortening behavior if any of the following two indicators are found in the call context:

- **Both the request body and response body contain a URL object.** This indicator represents the functionality of the link shortening API. To shorten a link, such an API sends the request with a long URL and returns the shortened link to the client side. Note that this is a necessary but insufficient condition, which needs further verification that the returned URL domain matches our known DUSS domain name in the dynamic trigger stage.
- **A POI is invoked by known SDKs provided by third-party hosting services.** This indicator strongly represents the POI is related to a third-party hosted DUSS. Specifically, the POI is either invoked from the SDK method or uses a third-party hosting service URL endpoint as the communication server address. The SDK methods and URL endpoints are collected from official documents, as listed in Appendix Tables V and VI.

To detect the first type of indicator, Ditto needs to analyze the data flow of the request and response body of the Web API to find the reachable definitions of the URL string objects. Note that we manually model the request body assignment methods and response callback methods for various network libraries, as done in previous work. Using this information, Ditto can identify the request body, a parameter of the

assignment method (e.g., `post(body)` in OkHttpClient), on the backward control-flow graph from the POI. Similarly, it can locate the response body in the response callback method (e.g., the return value of `execute()` in OkHttpClient) on the forward control-flow graph. Then, `Ditto` conducts backward and forward data-flow analysis on the request and response bodies. During this analysis, `Ditto` identifies two types of definitions for URL string objects: (a) the string conversion of Android URI and Java URL class; (b) methods that store or retrieve data from a key-value pair object (e.g., JSON), where the key object contains a constant string (i.e., "URL," "link"). We also manually model frequently used data structures with key-value pairs, which include the native org.json, as well as four external libraries: Gson, Moshi, Protobuf, and Fastjson.

To detect the second type of indicator, `Ditto` collects the method names from the call graph of POIs and string values from the data-flow graph to match with the predefined SDK methods or URL endpoints. If the method name matches predefined SDK methods (Table V), `Ditto` determines whether the POI contributes to the SDK method rather than an unrelated network event (such as log collection). We establish the contribution of the POI through taint analysis, identifying whether the propagation of SDK method parameters taints the request body. If the POI contributes to the SDK method, `Ditto` reports it as a call to the link shortening API. Otherwise, `Ditto` conducts backward taint analysis from the request endpoint object (tainted in the request method) to extract the constant strings assigned to it. If the collected string values match predefined URL endpoints (Table VI), `Ditto` reports that the POI is a call to link shortening API.

C. Directed API Trigger

The directed API trigger leverages the program logic to generate legal and modifiable link shortening API requests. It contains three steps: (i) statically locate the provenance method of URL object; (ii) add instrumentation on the provenance method to dynamically modify the value of the URL object; (iii) use UI-driven testing to trigger the API.

1) URL Object Provenance Analysis: `Ditto` performs a static backward data-flow analysis on the identified URL object that compose the request body of link shortening API. The purpose is to find the provenance method that returns the URL object. Specifically, we take a step back in our analysis of the object that was previously assigned to the identified URL object and locate its provenance method as the target in instrumentation. This is because the initial identification of the URL object's provenance, like "android.net.URI.toString," may be extensively referenced throughout the program. Such granularity in instrumentation could introduce significant overhead or disrupt the program logic.

2) Application Instrumentation: We add the instrumentation code using the process hook technique, which can dynamically modify the provenance method execution (i.e., return value). The hook technique receives the package name of the target application and method signature and intercepts the target method when it is invoked during the execution of the target application. Thus, we can replace the return value of the provenance method (i.e., URL to be shortened) with our test cases in vulnerability detection. We also add instrumentation to the caller of link shortening API for monitoring its invocation and to the response callback method for monitoring response value (i.e., whether a shortened link is returned).

3) UI-driven API Trigger: We use the dynamic UI-driven testing technique to explore the UI buttons that may trigger the link shortening API. The UI-driven testing iteratively explores clickable buttons on the current UI and auto-generates input texts. The exploration stops when the link shortening API is triggered or the one-hour exploration time limit is exceeded. If the API is triggered and its response callback contains the domain value of a known DUSS-shortened link, it is verified as a true link shortening API. In the meantime, the UI exploration sequence and button location will be saved and replayed to repeatedly trigger the API in vulnerability detection. To this end, we can leverage the code logic to generate legal API requests while modifying the URL to be shortened.

D. Vulnerability Detection

`Ditto` performs two main tasks to validate whether a link shortening API is flawed: (i) test case generation and (ii) API request modification following the decision tree guidance.

1) Test Case Generation: First, we summarize a set of proof-of-concept URLs, each exploiting one of the five types of known URL check vulnerabilities (Section III-C2) found in previous work [33], [34], [35]. Inspired by the principle of equivalence partitioning [39] in software testing, we simplify these cases by removing redundant ones that validate equivalent URL check vulnerabilities. Besides, we prioritize retaining those that can validate the widest array of URL checks. For example, we retain the test case "evilallowed.com" because it can validate both flawed domain matching algorithms using "contains" or "endswith". Then, we compile a test suite that contains unique test case generation templates. These templates represent the formula for constructing PoC URLs. They involve mutating or replacing the scheme, domain, and path of the intercepted URL (triggered by directed API calls) with predefined payloads. We register a domain (e.g., *evil.com*) as the untrusted domain payload. We also collect exploitable URLs under the domain assets of *allowed.com* by leveraging web vulnerability detection tools. More details of these templates are listed in Appendix Table VII.

2) Decision Tree Guidance: Let us describe how we build the decision tree to prioritize our test sequence. A basic rule here is to validate the consequent before its preconditions, avoiding unnecessary validations. For example, validating the parser will only become necessary if domain matching exists. To this end, we organize the security validation scenarios and their expected outcomes into a tree-like structure, as shown in Figure 5. It considers four security test scenarios: scheme check, domain matching, domain parsing, and domain asset validation. The rounded square tree nodes represent tested security scenarios, and each edge represents the expected execution results. `Ditto` iterates each testing scenario from the

root node, uses corresponding test cases to replace the original URL in the triggered API request, and checks if shortened links have been successfully generated in its response message. It then decides whether to continue or end the testing based on the expectations from the decision tree. The details of the procedure are described below:

- *Test Scheme Check*. We first validate whether a scheme check exists. If not, `Ditto` reports a compromise. The validation then continues for the domain check.
- *Test Domain Allowlist Matching*. We first test whether the domain check exists. If not, `Ditto` reports a compromise and ends the testing. Otherwise, it further tests whether the DUSS adopts an inadequate domain-checking algorithm. Only if our test cases fail will the validation continue for domain parsing.
- *Test Domain Parsing*. This round uses crafted URL test cases to explore whether the domain parser is flawed. If DUSS successfully shortens one of the test cases, `Ditto` reports a compromise. Otherwise, the validation continues to validate vulnerable domain assets.
- *Test Domain Asset*. The vulnerable domain assets are the minimal granularity in vulnerability detection. If one vulnerable URL is not blocked by DUSS, it reports that DUSS is compromised.

E. Ditto Implementation

Our static analysis module is implemented based on Flowdroid [40], a popular Android application decompiler and static analysis framework. It provides precise construction on the data-flow and control-flow graphs and supports taint analysis. The network request libraries used in locating Web API call sites are summarized from related work [38] and popular third-party libraries [41], [42], [43] in GitHub. The API instrumentation is implemented based on Frida [44], which provides a command line tool to load JavaScript instrumentation code dynamically. Note that, during vulnerability detection, we only use the most popular tool, Xray [45], for detecting XSS and open redirect, and Aquatone [46] for detecting subdomain takeovers. The main reason is that scanning for vulnerabilities in websites or domains is not the focus of this paper. These well-known corporations themselves should conduct more comprehensive scans for the security of their websites and domains. In UI-driven testing, we utilize TextExerciser [47] for exploring and clicking UI buttons, which is implemented based on Appium [48] and can automatically handle complex UI events (e.g., user input for login). `Ditto` saves each test case that successfully leads to a compromise in the vulnerability report; thus, our tests are controllable and can be accurately reported to DUSS corporations as proof of concept.

Data Set. We collect DUSS-related apps from two leading Android app stores (i.e., Google Play [49] and WanDouJia [50]) by manually searching developer names. We check whether the app's developer name matches one of the DUSS corporations and collect all apps under that developer tag. Our data set contains 377 Android apps related to 88 DUSS corporations.

F. Evaluation

This section presents `Ditto`'s overall evaluation result.

Effectiveness. We list the overall vulnerabilities in the 15 top-ranked DUSSs reported by `Ditto` in Table I. Specifically, `Ditto` scans 377 apps and finds 58 apps with link-shortening APIs, containing a total of 50 unique APIs. In vulnerability detection, `Ditto` reports 22 vulnerable APIs. We manually verify the reported vulnerable APIs, and all of them are true positives. Here is the breakdown of their vulnerabilities: 11 DUSSs do not check the scheme, 16 DUSSs have flawed domain matching, four have flawed URL parsing, and three have flawed domain assets. In vulnerability detection of the 50 link shortening APIs, `Ditto` totally generates 373 test cases, with 34 leading to a compromise. This not only shows that our tool can effectively detect vulnerable DUSSs but also shows no unacceptable overhead to the server.

False Positive Analysis. The static analysis may report incorrect link shortening APIs, but these can be verified by the dynamic API trigger. Since `Ditto` identifies a link shortening API is vulnerable only because it observes a shortened link is generated for the crafted malicious URL, there is no false positive on the vulnerability report.

False Negative Analysis. In reviewing false negatives of the directed API trigger, two apps are reported as having potential link shortening APIs but not being triggered by `Ditto`. We manually reverse-engineer and test their API endpoints, confirming they are false negatives. Specifically, one app fails because its link shortening API is called from an unreachable activity. Another app has a complex UI layout using nested WebView; thus, the existing UI-driven testing tool cannot precisely identify the button to click. As for the false negatives in vulnerability detection, we manually submitted crafted URLs following our templates (Table 7) to the 28 link shortening APIs reported by `Ditto` as not compromised and two non-triggered APIs and observed they could not be shortened in the API responses. The results indicate that these DUSSs enforce strict domain-based allowlist configuration, allowing only links from their main website domain, and no vulnerable URLs are detected on the main website either. Thus, none of them is a false negative.

Detection Scope. `Ditto` has some limitations despite our best efforts to design and develop its methodology. First, testing the DUSS with no directly exposed link shortening API is limited. For example, the DUSS might be accessed by the application server when its app user posts a message containing a long URL, but the corresponding response to the app may not include the shortened link. Such a case can be addressed by fully automatic UI-driven testing with optical character recognition to recognize whether shortened links are displayed on the screen. Second, `Ditto` can analyze hybrid apps by further modeling the JavaScript-to-Java communication functions (e.g., annotated with "@JavaScriptInterface"). While our tool may not achieve a comprehensive test of vulnerable domain

TABLE I
VULNERABILITY BREAKDOWN IN THE TOP 15 VULNERABLE DUSSS. SYMBOL "✔" MEANS THE DUSS IS COMPROMISED IN THIS URL CHECK SCENARIO. "-" MEANS NO VULNERABILITY IS FOUND IN THIS CATEGORY. "M/P/A" STANDS FOR VULNERABLE MATCHING/PARSING/ASSET.

#	Corporation	Website	# Visiting	Brand Domain	Visiting	Tranco Rank	Development	Scheme	Domain(M/P/A)
1	Huawei	huawei.com	92.9M	url.cloud.huawei.com	-	295	Self-developed	-	✔(A)
2	Lazada	lazada.com	1.6M	s.lazada.sg	231.6K	6k	Self-developed	-	✔(P)
3	CastBox	castbox.fm	1.2M	castbox.fm	1.2M	6k	Self-developed	✔	✔(M)
4	Sina	sina.cn	107.7M	t.cn	1.4M	8k	Self-developed	-	✔(M)
5	Amazon	amazon.com	2.4B	a.co	17.3M	8k	Self-developed	✔	✔(M)
6	ixigo	ixigo.com	12.3M	f.ixigo.com	188.7K	13k	Self-developed	-	✔(M)
7	Weilai	nio.cn	345.6K	l.nio.com	2.8K	22k	Self-developed	-	✔(M)
8	Flipboard	flipboard.com	4.5M	flip.it	852.6K	42k	Self-developed	✔	✔(M)
9	YamiBuy	yamibuy.com	1.1M	u.yamibuy.com	-	56k	Self-developed	✔	✔(M)
10	Yelp	yelp.com	134.8M	yelp.to	1.6M	183k	Self-developed	-	✔(P)
11	Xiaohongshu	xiaohongshu.com	162.1M	xhslink.com	928.9K	295k	Self-developed	✔	✔(M)
12	China Unicom	10010.com	3.4M	u.10010.cn	58.9K	526k	Self-developed	-	✔(P)
13	momo shopping	momoshop.com.tw	33.6M	momo.dm	237.8K	694k	Self-developed	-	✔(M)
14	flipp	flipp.com	2.8M	click.flipp.com	124.8K	14k	Third-party Hosted	-	✔(M)
15	TubiTv	tubitv.com	36.3M	link.tubi.tv	202.8K	49k	Third-party Hosted	-	✔(M)

assets, our approach can guide developers to conduct more comprehensive security testing in a white-box scenario.

Although `Ditto` has been developed and implemented to analyze Android apps, its core solution can be extended to analyze websites. Specifically, in API inference, static analysis can be adapted for JavaScript code using frameworks like JSWALA [51] or AST-based method[52]; in API trigger, UI-driven testing can be replaced with Puppeteer [53] and the intercept of link shortening API can be done easily with the help of Chrome Developer Protocol [54].

V. SECURITY IMPACT: A SURVEY STUDY

In this section, we perform a survey study to demonstrate the real-world effect of the *Misdirection Attack* on social network users and domain-based checkers (for answering RQ3).

A. If Social Users Hold Implicit Trust

It is already known that social network users prefer to trust URLs with well-known domain names [13]. Besides, current secure mechanisms such as link preview [55] can be unreliable in helping users decide the URL's trustworthiness. Therefore, we can conclude that the better the reputation of a URL domain name, the more likely it is to gain users' trust and carry out phishing attacks.

To better understand the security impact of *Misdirection Attack* on social network users, and whether it gains more trust than other phishing techniques, we compare the domain rank of vulnerable DUSSs with other phishing URLs. Specifically, on January 22, 2024, we downloaded phishing URLs from PhishTank [56], which contains 41,643 online and verified[9] URLs. Then, we compute the SLD rank based on the Tranco.

Figure 6 presents the unique SLD rank distribution of the vulnerable DUSSs shortened links and existing phishing URLs. Although a few PhishTank URLs exploit vulnerable web services hosted on Google and achieve a very high rank (top 1 in Tranco), the majority (72.1%) of PhishTank

[9]The "online and verified" tag is provided by PhishTank.

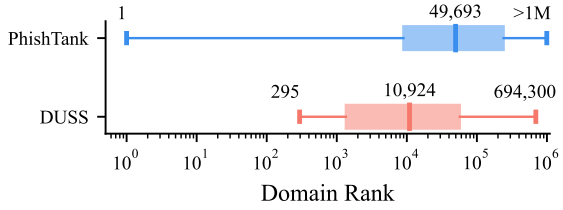

Fig. 6. Domain rank distribution of PhishTank and vulnerable DUSSs.

URLs have a domain rank lower than the median of DUSS-shortened links. In summary, the *Misdirection Attack* demonstrates greater effectiveness in gaining the trust of social network users.

B. If Security Developers Hold Implicit Trust

In this section, we first summarize the types of domain-based checkers and then conduct an independent security analysis for them. The purpose is to understand how security developer design the policy of the domain-based checker, specifically, whether they hold implicit trust in DUSS shortened links. We also enrich our study with three case studies.

1) Domain-based Checkers: We systematically classify the domain-based checkers from multiple sources, including the academic literature [57], [58], [59], [55], [34], the CVE database [60], the CNVD database [61], and non-academic resources (see, i.e., [62]). Specifically, we study the checkers that can be affected by domain-related vulnerabilities listed on OWASP [63], including open redirection [64], subdomain takeover [65], and domain expiration [66].

We find five scenarios of using a domain-based security checker, as detailed below:

- **Applink verification.** In this scenario, apps configure "applink" as a special deeplink recognized by mobile devices to automatically launch the app when mobile users click the link.
- **External link warning.** In this scenario, mobile apps or websites use a domain-based allowlist to check and alert their users before navigating to external links.

• **OAuth redirection.** In this scenario, the websites check the redirection URL using a domain-based allowlist after the OAuth [67] is completed.

• **SSRF protection.** To mitigate the Server-Side Request Forgery (SSRF), web servers may adopt domain-based allowlists [68], [69], e.g., verifying the domain of a user-posted URL to retrieve its preview content.

• **Link security checker.** Online services offer link security checks may include domain-based blocklists to filter out links hosted on malicious servers, including popular spam filters like Spamhaus [70] and MxToolBox [71], phishing detectors such as Google Safe Browsing (GSB) [72] and PhishTank [56], and malware detectors like VirusTotal [73] and Malwareurl [74].

2) Analysis Methodology: Here, we present our methodology for identifying the real-world domain-based checkers and testing whether they can be exploited by compromised DUSS-shortened links discovered by `Ditto`. In the meantime, we also conduct a comparative experiment using other standard URLs, such as the SUSS-shortened links. The purpose is to better understand the security implications and advantages of *Misdirection Attack*. Specifically, the experiment group contains 22 DUSS-shortened links redirecting to the landing page hosted on our server. The control group contains URLs pointing to our server (as untrusted URLs) and their shortened links generated by five commercial URL-shortening services. Note that we leave the analysis of SSRF to Appendix D, which carries out the theoretical analysis for ethical reasons.

Analysis of applink configuration. We aim to test whether the DUSS-shortened untrusted URL can be recognized as an applink and loaded into the WebView component of the mobile app to launch the *Misdirection Attack*. To do this, we analyze the mobile apps with link-shortening API identified in Section IV because these apps are more likely to configure DUSS-shortened links as applinks. We use static analysis to identify the apps with WebView and extract their applink and deeplink configurations from the AndroidManifest file. For applink configurations containing the DUSS domain, we directly trigger the app using the Android Debug Bridge (adb), sending an intent to launch the app with applink as input. We also add the test links as a parameter on the deeplink path and dynamically trigger them using adb to test whether they can be loaded. Lastly, we observe whether our landing URL is loaded into the WebView.

In this attack scenario, the remote attacker injects malicious JavaScript or phishing websites into the victim user's mobile device, leading to serious remote code injection or phishing attacks. The WebView often contains sensitive runtime APIs to access the underlying file system, GPS, and rich app functions, which are exposed via JavaScript bridge (e.g., `addJavascriptInterface`). Therefore, the effect of such an attack will vary in different applications, such as user private information leakage, privilege escalation, and malicious apk installation.

Analysis of external link warning service. This experiment tests whether a warned external link triggers the warning after being transformed into DUSS-shortened links. We manually collect the official websites and apps that contain warning services from each vulnerable DUSS corporation. The websites and apps allow users to send URLs to the screen, and navigate when clicking the link. For example, we post a comment as the user along with a shortened link and then use another account to click on the link.

In this attack scenario, remote attackers can lure victim users to malicious websites, leading to stealthy phishing attacks.

Analysis of OAuth redirection. This experiment measures whether an attacker-controlled URL can bypass the OAuth redirection check after being shortened by vulnerable DUSS. For each DUSS corporation, we manually collect the user login URLs with OAuth protection on their official websites. Next, we manually complete the login procedure while modifying its callback URL in the address bar to an untrusted URL on our server and its DUSS-shortened and SUSS-shortened links, respectively. If the websites navigate to the untrusted URL after OAuth completion, it leads to the *Misdirection Attack*.

In this attack scenario, scammers can lure victim users to their phishing websites after user login. If the token is carried in the redirect URL, this can also lead to token leakage. Remote attackers can access and manipulate the victim users' accounts on target websites.

Analysis of link security checkers. This experiment tests whether a URL identified initially as malicious by the online checker becomes innocent after being shortened by DUSS. We choose the most popular link security checker in each scenarios: VirusTotal (malware), Spamhaus (spam), and GSB (phishing). Then we collect the malicious URLs from public blocklists [75], [76] and submit them to these scanners to verify if they are reported as malicious. For ethical considerations, we build a USS on our server as the DUSS to shorten these links. We also generate SUSS-shortened links for these blocked URLs using our own accounts. Finally, we send these shortened links to online checkers to test whether they are reported as malicious.

In this attack scenario, web criminals can abuse vulnerable DUSS to distribute malware, spam sites, and phishing sites.

3) Evaluation: Table II illustrates the overall affected domain-based checkers, including popular customized checkers used in applications developed by vulnerable DUSS corporations (found in Section IV) and popular online checkers. These applications include the most popular Android applications and websites from each vulnerable DUSS corporation – 22 apps and 22 websites.

Let us provide a detailed breakdown of the result. In testing applink, we find three apps directly configure the DUSS-shortened links as applink are vulnerable, leading to remote code injection. Seven apps can be attacked by both DUSS-shortened links and SUSS-shortened links due to the lack of security check. In testing external link warning services, four apps and Sina's website employ checks and give warnings

TABLE II
OVERALL RESULTS OF THE VULNERABLE DOMAIN-BASED CHECKERS. "A" FOR MOBILE APP; "W" FOR DESKTOP WEBSITE.

# Corporation	Applink (A)	External Warn (A)	External Warn (W)	OAuth (W)	Online Checker
1 Flipp	⊙	⊙	-	○	⊙
2 Xiaohongshu	⊙	●	-	⊙	⊙
3 China Unicom	●	⊙	⊙	-	⊙
4 Weilai	⊙	⊙	-	-	⊙
5 Huawei	⊙	●	-	●	⊙
6 Flipboard	●	⊙	⊙	○	⊙
7 YamiBuy	-	-	-	●	⊙
8 Gojek	⊙	●	-	-	⊙
9 Fox Sports	-	-	⊙	⊙	⊙
10 ixigo	⊙	⊙	-	-	⊙
11 CastBox	-	⊙	⊙	●	⊙
12 Adidas	⊙	⊙	⊙	○	⊙
13 Sina	●	●	●	○	⊙
14-22 Others	○	○	○	○	⊙

●: Only *Misdirection Attack*. ⊙: Vulnerable to all attacks.
○: Not vulnerable. "-": Scenario not found.

for untrusted links and their SUSS-shortened links; however, there is no warning for DUSS-shortened links. In contrast, the websites of five corporations and apps from eight corporations lack such checkers, making them susceptible to all untrusted links. In testing OAuth, seven out of nine corporations adopt the domain-based checker in their website login procedure. However, Huawei, Yamibuy and CastBox are vulnerable to the compromised DUSS-shortened links. While Xiaohongshu and Fox Sports have no such check, making them vulnerable to all untrusted links. In testing online link security checkers, both DUSS- and SUSS-shortened links can bypass the GSB, Spamhaus, and VirusTotal, similar to previous findings [77].

Note that we also find data over-collection, which is a privacy concern in which a corporation may collect more data than it needs from users. Specifically, Sina Weibo logs detailed user activity, including their visited web pages and clicked links, and sends them to Sina's server.

4) Case Studies: Now we illustrate three interesting cases:

Example 1 [Xiaohongshu]: One-click attack for information leakage. This example uses the compromised DUSS-shortened link and combines the applink and external link warning bypass in the Xiaohongshu mobile app, a social app with over two billion downloads, to launch *Misdirection Attack*. An adversary can further abuse its sensitive WebView runtime APIs to steal private information (e.g., phone numbers) of victim users and spy on them.

Xiaohongshu has a self-developed DUSS using the domain name *xhslink.com*, with more than 162 million visits. However, the DUSS lacks of URL checks, thus is compromised for shortening any attacker controlled URLs. Then, let us look at bypassing the URL checks in Xiaohongshu's mobile app. The attacker crafts a phishing deeplink "xhsdiscover://webview/[shortened-link]" and spreads it on social platforms. When a mobile user clicks the link, Xiaohongshu will start and check whether the link is external. However, Xiaohongshu configures the DUSS domain in its allowlist. To this end, the shortened link is loaded. It redirects the WebView to load and execute malicious JavaScript from the attacker server without warning the victim user. The malicious code can abuse sensitive runtime APIs, e.g., `getUserInfo` to get the user's phone number or `getCurrentGeolocation` to monitor the precise location.

Example 2 [Huawei]: Phishing in OAuth redirection. Huawei, the largest mobile device vendor in China, uses its sub-domain *url.cloud.huawei.com* to host shortened links, which has more than 92.9 million visiting. However, `Ditto` finds it has a vulnerable link shortening API in Huawei App Gallery, a mobile app preinstalled on Huawei devices. This vulnerability is found in testing the domain asset, which allows URLs with a common domain name *bit.ly* to be shortened. Thus attacker can first shorten the phishing URL using Bitly free account, then further compromise the DUSS of Huawei using the malicious *bit.ly* link.

Next, Huawei uses the OAuth redirection link, i.e., https://uniportal.huawei.com/.../login.html?redirect=[redirectURL], in redirecting the user to the redirectURL. This redirection is checked, and only allows its official website. Specifically, Huawei configures "*.huawei.com" in its allowlist. Therefore, the web criminal can craft a user login URL redirecting to malicious shortened links and publish such links to social network platforms. When victim users click these links and log into their Huawei accounts, their browser will be stealthily redirected to the phishing site, leading to phishing attack. For example, the attacker can fake a login failed web page and require victim users to re-enter their account and password to steal their credentials.

Example 3 [VirusTotal]: Nested shortened links to cheat link safety checker. This case illustrates an interesting strategy in cheating link safety checker, and keeping malicious URL live longer. Specifically, VirusTotal aggregates multiple antivirus engines to help users understand whether the submitted URL is potentially harmful. However, with the help of USS, malicious URLs can evade partial engines in VirusTotal, including the one uses domain blocklist [78]. In our experiment of using our USS to shorten the malicious link, only one engine in VirusTotal issues a warning. Moreover, if we double-shorten these malicious URLs, VirusTotal no longer reports them as malicious. Thus, web criminals can use USS to generate nested shortened links, and keep their malicious URLs to live longer.

VI. DISCUSSION

A. Lessons learned and mitigation

The *Misdirection Attack* exploiting the abuse of trust transitivity can be initiated through DUSS and other reputable link hosting services (e.g., Google AMP) with vulnerable URL checkers. Thus, the most important lesson from our research is that the URL checker's security design should obey the least privilege principle to prevent potential misdirection in URL shortening or loading. Essential mitigation should include the

secure design of API protection, URL check, and allowlist configuration. We present three mitigation rules:

• *The link shortening API should avoid getting URLs directly from the front end to reduce the attack surface.* For example, DUSS can use IDs to map trusted long URLs, enabling DUSS to receive only the ID from the front-end application, retrieve the long URL on the server side based on this ID, shorten it, and return the result to the front-end application.

• *Use consistent and precise URL check algorithm.* First, the URL parser adopted in client-side and server-side URL checks should be consistent and strictly follow one standard, such as using standard parser libraries (e.g., whatwg-url, urllib.parse). Second, URL-based security checks must cover each URL's landing pages. The policy should combine URL schemes, domains, and paths, restricting untrusted or unnecessary entities. Learning traditional browser defense mechanisms such as content security policy could benefit this.

• *Update and verify the allowlist configuration regularly.* URL checkers in DUSSs, apps, and websites may have different security constraints; thus, communication between different checkers' developers is required to reach a consensus on security constraints. For example, the DUSS often has broader constraints on the URLs they trust (for various first-party resources). However, the domain-based checkers (e.g., OAuth, link warning) should trust only limited resources. The developers need to work with each other more frequently to update the security constraints. Also, the allowlist configuration can benefit from regularly using web security scanners to verify the security of domain assets.

B. Ethics Concerns

We discuss the ethical considerations of our study, including URL collection, vulnerability testing, and disclosure.

Data Collection Ethics. We adhered to ethical guidelines from "The Menlo Report [79]," "Ethics and Internet Measurements [80]," and "Ethics in Cybersecurity Research and Practice [81]." Our data collection followed Facebook's rate limits (<200 API calls per user per 60 minutes), similar to previous studies [82], [83], [84], [85]. We collected only publicly accessible URLs without personal information, ensuring no adverse effects on Facebook users or services.

Vulnerability Testing Ethics. Our vulnerability testing avoided harm to remote servers. The API trigger ran at low speed, simulating regular user interactions with a limit of 12 requests per server. Xray payloads [86] used in testing were harmless, ensuring no impact on server databases. To avoid affecting other users, we conducted all tests with our test accounts on our devices.

Vulnerability Disclosure. Following the OWASP Vulnerability Disclosure Cheat Sheet [87], we notified all 22 DUSS corporations of their vulnerabilities and provided a 45-day fix deadline. Reports included PoCs, details of vulnerabilities, and affected services. We received confirmations from 13 corporations, resulting in 16 CVE-ids and 6 CNVD-ids, with 12 corporations having fixed their issues by the time of writing. One corporation fixed its vulnerabilities without a formal reply.

Our research can help DUSS vendors prevent abuse and protect users, enhancing security and user experience on social media platforms like Facebook and Twitter.

VII. Related Work

URL shortening service. As the USS has become popular in recent years, many researchers have addressed the security and real-world abuse problem of shared USS in social media. Neuman et al. [10] presented an analysis of the security and privacy risks of shortened links. Maggi et al. [11] collected 25 million shortened links belonging to 622 distinct SUSSs and measured the malicious URLs. Li et al. [88] proposed using the channel of USS for delivering messages. Gupta et al. [12] studied the Bitly service and found it failed in detecting malicious web URLs such as spam and pornographic content. Albakry et al. [13] conducted a survey and revealed the low abilities of users to predict the URLs' landing page. Fukushi et al. [14] studied and revealed mal-advertising behavior in ad-based USSs. As a comparison, our paper focuses on a special type of USS, only for dedicated usage, with a different threat model from prior works, which has not been studied before. Also, previous work can not identify the hidden link shortening APIs and perform security tests on them.

Domain Security. Domains, as the core entry point to the Internet, have garnered significant attention from security researchers. Previous studies have thoroughly analyzed domain infrastructures [89], [90], [91] and proposed various domain-oriented attacks [17], [18], [92], [93]. Wei [90] analyzed the content delivery network and proposed a domain shadowing attack that can bypass censorship to access blocklisted websites. Liu et al. [91] uncovered 467 exploitable dangling DNS records that could lead to domain hijacking and proposed defense mechanisms. Li et al. [89] analyzed the DNS resolvers and proposed the Phoenix Domain attack, making malicious domains irrevocable. Watanabe et al. [17] showed the web rehosting service (e.g., Google translator) can be abused to break the same origin policy. Squarcina et al. [18] thoroughly studied the same site attacker and illustrated its impact on web application security, such as CSP, CORS, and website cookies. So et al. [92] and Level et al. [93] discussed the security implications of expired but re-registered popular domains.

Application Vulnerability Analysis. Open Redirect vulnerabilities are a significant concern in applications and have been the subject of extensive research in the academic community. Rastogi et al. [57] developed a methodology to investigate malware infections and scams that target mobile users in the interface between mobile apps and the web. Ghozali et al. [94] used the OWASP Risk Rating to detect security vulnerabilities, including open redirects, in PHP web applications, emphasizing the importance of regular updates. Riadi et al. [95] discussed the susceptibility of framework-based websites to injection attacks, including open redirects. Gadient et al. [38] analyzed web communications in mobile apps, including

open redirects, highlighting the prevalence of insecure HTTP connections in closed-source apps.

VIII. Conclusion

In this paper, we perform the first systematic security study of dedicated URL shortening services. Specifically, we identify a significant security concern—weaknesses in DUSS URL validation, leading to a novel attack called a Misdirection Attack. We discover 22 vulnerable DUSSs out of 88 popular services, resulting in severe consequences, including phishing and code injection in millions of users' mobile devices.

Acknowledgement

We would like to thank Haipei Wang, Pan Huang, and Congcong Zhang for providing technical assistance and for their help with the preparation of figures in this paper. We also 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), National Natural Science Foundation of China (62172104, 62172105, 62472096, 62102093, 62102091, 62302101, 62402114, 62402116, 62202106), and Shanghai Pilot Program for Basic Research - FuDan University 21TQ1400100 (21TQ012). Min Yang is the corresponding author, and a faculty of Shanghai Institute of Intelligent Electronics & Systems, and Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, China.

References

[1] (2023) Beware of linkedin slinks! [Online]. Available: https://wp.nyu.edu/itsecurity/2022/02/10/beware-of-linkedin-slinks/

[2] (2023) Bitly — url shortener. [Online]. Available: https://bitly.com

[3] (2023) Apt28. [Online]. Available: https://www.secureworks.com/research/iron-twilight-supports-active-measures

[4] (2023) Nasa.gov: Behind the page. [Online]. Available: https://blogs.nasa.gov/nasadotgov/tag/recently-on-nasa-gov/

[5] (2023) Spammers abuse .gov url shortener service in work-at-home scams. [Online]. Available: http://www.pcworld.com/article/2012800/spammers-abuse-gov-url-shortener-service-in-workathome-scams.html

[6] (2024) Dropbox's url shortener abused by spammers — infoworld. [Online]. Available: https://www.infoworld.com/article/2619285/dropbox-s-url-shortener-abused-by-spammers.html

[7] (2024) Beware of linkedin slinks! – nyu it security news and alerts. [Online]. Available: https://wp.nyu.edu/itsecurity/2022/02/10/beware-of-linkedin-slinks/

[8] S. Hamdi, A. L. Gancarski, A. Bouzeghoub, and S. B. Yahia, "Tison: Trust inference in trust-oriented social networks," *ACM Transactions on Information Systems (TOIS)*, vol. 34, no. 3, pp. 1–32, 2016.

[9] G. Liu, Y. Wang, and M. Orgun, "Trust transitivity in complex social networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, no. 1, 2011, pp. 1222–1229.

[10] A. Neumann, J. Barnickel, and U. Meyer, "Security and privacy implications of url shortening services," in *Proceedings of the Workshop on Web 2.0 Security and Privacy*, 2010.

[11] F. Maggi, A. Frossi, S. Zanero, G. Stringhini, B. Stone-Gross, C. Kruegel, and G. Vigna, "Two years of short urls internet measurement: security threats and countermeasures," in *proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 861–872.

[12] N. Gupta, A. Aggarwal, and P. Kumaraguru, "bit. ly/malicious: Deep dive into short url based e-crime detection," in *2014 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2014, pp. 14–24.

[13] S. Albakry, K. Vaniea, and M. K. Wolters, "What is this url's destination? empirical evaluation of users' url reading," in *Proceedings of the 2020 CHI conference on human factors in computing systems*, 2020, pp. 1–12.

[14] N. Fukushi, T. Koide, D. Chiba, H. Nakano, and M. Akiyama, "Analyzing security risks of ad-based url shortening services caused by users' behaviors," in *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part II 17*. Springer, 2021, pp. 3–22.

[15] G. Hong, M. Wu, P. Chen, X. Liao, G. Ye, and M. Yang, "Understanding and detecting abused image hosting modules as malicious services," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 3213–3227.

[16] X. Liao, S. Alrwais, K. Yuan, L. Xing, X. Wang, S. Hao, and R. Beyah, "Lurking malice in the cloud: Understanding and detecting cloud repository as a malicious service," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1541–1552.

[17] T. Watanabe, E. Shioji, M. Akiyama, and T. Mori, "Melting pot of origins: Compromising the intermediary web services that rehost websites." in *NDSS*, 2020.

[18] M. Squarcina, M. Tempesta, L. Veronese, S. Calzavara, and M. Maffei, "Can i take your subdomain? exploring {Same-Site} attacks in the modern web," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2917–2934.

[19] (2023) China unicom operating data. [Online]. Available: https://www.chinaunicom.com.hk/sc/ir/operating.php?type=monthly

[20] (2024) China unicom mobile app. [Online]. Available: https://iservice.10010.com/e4/PublicitychannelView/phone_new.html

[21] (2024) China unicom: Corporate social responsibility report. [Online]. Available: https://www.chinaunicom.com.hk/en/esg/csr2017/csr2017_14.pdf

[22] Z. Li, W. Wang, C. Wilson, J. Chen, C. Qian, T. Jung, L. Zhang, K. Liu, X. Li, and Y. Liu, "Fbs-radar: Uncovering fake base stations at scale in the wild." in *NDSS*, 2017.

[23] Y. Zhang, B. Liu, C. Lu, Z. Li, H. Duan, S. Hao, M. Liu, Y. Liu, D. Wang, and Q. Li, "Lies in the air: Characterizing fake-base-station spam ecosystem in china," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 521–534.

[24] (2023) Applink. [Online]. Available: https://developer.android.com/studio/write/app-link-indexing

[25] (2023) Webview. [Online]. Available: https://developer.android.com/reference/android/webkit/WebView

[26] (2024) Graph api: Meta for developers. [Online]. Available: https://developers.facebook.com/docs/graph-api

[27] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, ser. NDSS 2019, Feb. 2019.

[28] (2023) Similarweb: Website traffic - check and analyze any website. [Online]. Available: https://www.similarweb.com/

[29] (2023) Tinyurl — url shortener. [Online]. Available: https://tinyurl.com/app/pricing

[30] (2023) Rebrandly — url shortener. [Online]. Available: https://www.rebrandly.com/

[31] (2023) Branch — url shortener. [Online]. Available: https://www.branch.io/

[32] (2023) Appsflyer — url shortener. [Online]. Available: https://www.appsflyer.com/

[33] J. Reynolds, A. Bates, and M. Bailey, "Equivocal urls: Understanding the fragmented space of url parser implementations," in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 166–185.

[34] X. Han, Y. Zhang, X. Zhang, Z. Chen, M. Wang, Y. Zhang, S. Ma, Y. Yu, E. Bertino, and J. Li, "Medusa attack: Exploring security hazards of In-App QR code scanning," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 4607–4624. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/han-xing

[35] (2023) Url format bypass. [Online]. Available: https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/url-format-bypass

[36] T. Li, X. Wang, M. Zha, K. Chen, X. Wang, L. Xing, X. Bai, N. Zhang, and X. Han, "Unleashing the walking dead: Understanding cross-app remote infections on mobile webviews," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 829–844.

[37] A. Mendoza and G. Gu, "Mobile application web api reconnaissance: Web-to-mobile inconsistencies & vulnerabilities," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 756–769.

[38] P. Gadient, M. Ghafari, M.-A. Tarnutzer, and O. Nierstrasz, "Web apis in android through the lens of security," in *2020 IEEE 27th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2020, pp. 13–22.

[39] T. Badgett, G. J. Myers *et al.*, *The Art of Software Testing*. Wiley Online Library, 2023.

[40] (2024) Flowdroid static data flow tracker. [Online]. Available: https://github.com/secure-software-engineering/FlowDroid

[41] (2024) Okhttp. [Online]. Available: https://github.com/square/okhttp

[42] (2024) Moshi. [Online]. Available: https://github.com/square/moshi

[43] (2024) Retrofit. [Online]. Available: https://github.com/square/retrofit

[44] (2023) Frida. [Online]. Available: https://github.com/frida

[45] (2024) xray. [Online]. Available: https://github.com/chaitin/xray

[46] (2019) Aquatone - a tool for domain flyovers. [Online]. Available: https://github.com/michenriksen/aquatone#installation

[47] Y. He, L. Zhang, Z. Yang, Y. Cao, K. Lian, S. Li, W. Yang, Z. Zhang, M. Yang, Y. Zhang, and H. Duan, "Textexerciser: Feedback-driven text input exercising for android applications," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1071–1087.

[48] (2023) Appium. [Online]. Available: https://appium.io/docs/en/2.4/

[49] (2023) Google play. [Online]. Available: https://play.google.com/

[50] (2023) Wandoujia. [Online]. Available: https://www.wandoujia.com/

[51] (2024) Wala. [Online]. Available: https://github.com/wala/WALA

[52] A. Fass, D. F. Somé, M. Backes, and B. Stock, "Doublex: Statically detecting vulnerable data flows in browser extensions at scale," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1789–1804.

[53] (2020) Puppeteer. [Online]. Available: https://pptr.dev/

[54] (2024) Chrome debugging protocol interface. [Online]. Available: https://github.com/cyrus-and/chrome-remote-interface

[55] G. Stivala and G. Pellegrino, "Deceptive previews: A study of the link preview trustworthiness in social platforms," 2020.

[56] (2023) Phishtank. [Online]. Available: https://phishtank.org/

[57] V. Rastogi, R. Shao, Y. Chen, X. Pan, S. Zou, and R. D. Riley, "Are these ads safe: Detecting hidden attacks through the mobile app-web interfaces." in *NDSS*, 2016.

[58] C. Lever, R. Walls, Y. Nadji, D. Dagon, P. McDaniel, and M. Antonakakis, "Domain-z: 28 registrations later measuring the exploitation of residual trust in domains," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 691–706.

[59] S. Wang, M. Almashor, A. Abuadbba, R. Sun, M. Xue, C. Wang, R. Gaire, S. Nepal, and S. Camtepe, "Doitrust: Dissecting on-chain compromised internet domains via graph learning." in *NDSS*, 2023.

[60] (2023) Nvd - search vulnerability database. [Online]. Available: https://nvd.nist.gov/vuln/search

[61] (2023) Cnvd database. [Online]. Available: https://www.cnvd.org.cn/

[62] (2023) The real impact of an open redirect vulnerability. [Online]. Available: https://blog.detectify.com/industry-insights/the-real-impact-of-an-open-redirect/

[63] (2024) Vulnerabilities — owasp foundation. [Online]. Available: https://owasp.org/www-community/vulnerabilities/

[64] (2024) Cwe-601: Url redirection to untrusted site ('open redirect'). [Online]. Available: https://cwe.mitre.org/data/definitions/601.html

[65] (2024) Subdomain takeovers - security on the web — mdn. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Subdomain_takeovers

[66] (2024) Cwe-672: Operation on a resource after expiration or release. [Online]. Available: https://cwe.mitre.org/data/definitions/672.html

[67] (2023) Oauth. [Online]. Available: https://oauth.net/2/

[68] (2023) Cve-2021-25640. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-25640

[69] (2023) Cve-2022-24969. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-24969

[70] (2023) spamhaus. [Online]. Available: https://check.spamhaus.org/

[71] (2023) Mxtoolbox. [Online]. Available: https://mxtoolbox.com/blacklists.aspx

[72] (2023) Google safebrowsing. [Online]. Available: https://safebrowsing.google.com/

[73] (2023) Virustotal. [Online]. Available: https://www.virustotal.com/gui/home/url

[74] (2023) Commercial malicious website blacklist services. [Online]. Available: https://malwareurl.com

[75] (2023) Threat intelligence feeds dns blocklist. [Online]. Available: https://gitlab.com/hagezi/mirror/-/raw/main/dns-blocklists/domains/tif.txt

[76] (2023) Fake dns blocklist. [Online]. Available: https://gitlab.com/hagezi/mirror/-/raw/main/dns-blocklists/adblock/fake.txt

[77] P. Peng, L. Yang, L. Song, and G. Wang, "Opening the blackbox of virustotal: Analyzing online phishing scan engines," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 478–485.

[78] (2023) Virustotal += malware domain blocklist. [Online]. Available: https://blog.virustotal.com/2012/06/virustotal-malware-domain-blocklist.html

[79] M. Bailey, D. Dittrich, E. Kenneally, and D. Maughan, "The menlo report," *IEEE Security & Privacy*, vol. 10, no. 2, pp. 71–75, 2012.

[80] J. Van Der Ham, "Ethics and internet measurements," in *2017 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2017, pp. 247–251.

[81] K. Macnish and J. Van der Ham, "Ethics in cybersecurity research and practice," *Technology in society*, vol. 63, p. 101382, 2020.

[82] B. Li, J. Lin, F. Li, Q. Wang, Q. Li, J. Jing, and C. Wang, "Certificate transparency in the wild: Exploring the reliability of monitors," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2505–2520.

[83] A. Sathiaseelan, M. S. Seddiki, S. Stoyanov, and D. Trossen, "Social sdn: Online social networks integration in wireless network provisioning," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 375–376.

[84] M. S. Rahman, T.-K. Huang, H. V. Madhyastha, and M. Faloutsos, "Efficient and scalable socware detection in online social networks," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 663–678.

[85] G. Kontaxis, M. Polychronakis, A. D. Keromytis, and E. P. Markatos, "{Privacy-Preserving} social plugins," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 631–646.

[86] (2024) Xray documentation. [Online]. Available: https://docs.xray.cool/tools/xray/BasicIntroduction

[87] (2024) Vulnerability disclosure - owasp cheat sheet series. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Vulnerability_Disclosure_Cheat_Sheet.html

[88] D. Li, F. Zhang, and C. Liu, "Poster: Abusing url shortening services for stealthy and resilient message transmitting," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1451–1453. [Online]. Available: https://doi.org/10.1145/2660267.2662390

[89] X. Li, B. Liu, X. Bai, M. Zhang, Q. Zhang, Z. Li, H. Duan, and Q. Li, "Ghost domain reloaded: Vulnerable links in domain name delegation and revocation," in *Proceedings of the 30th Annual Network and Distributed System Security Symposium (NDSS'23). https://doi.org/10.14722/ndss*, 2023.

[90] M. Wei, "Domain shadowing: Leveraging content delivery networks for robust {Blocking-Resistant} communications," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3327–3343.

[91] D. Liu, S. Hao, and H. Wang, "All your dns records point to us: Understanding the security threats of dangling dns records," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1414–1425.

[92] J. So, N. Miramirkhani, M. Ferdman, and N. Nikiforakis, "Domains do change their spots: Quantifying potential abuse of residual trust," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2130–2144.

[93] C. Lever, R. Walls, Y. Nadji, D. Dagon, P. McDaniel, and M. Antonakakis, "Domain-z: 28 registrations later measuring the exploitation of residual trust in domains," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 691–706.

[94] B. Ghozali, K. Kusrini, and S. Sudarmawan, "Mendeteksi kerentanan keamanan aplikasi website menggunakan metode owasp (open web application security project) untuk penilaian risk rating," *Creative Information Technology Journal*, vol. 4, no. 4, pp. 264–275, 2019.

[95] I. Riadi, R. Umar, and W. Sukarno, "Vulnerability of injection attacks against the application security of framework based websites open web access security project (owasp)," *J. Inform*, vol. 12, no. 2, pp. 53–57, 2018.

[96] (2024) Burp suite - application security testing software. [Online]. Available: https://portswigger.net/burp

[97] (2024) Server side request forgery prevention - owasp cheat sheet series. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

APPENDIX

A. Use Cases for Motivating Example

Here, we give more detailed use cases for attackers analyzing the DUSS link shortening API. First, to get network traffic from the China Unicom application, the attacker must set up a man-in-the-middle proxy server, such as using Burpsuite [96], and configure a trusted certificate in the mobile device (e.g., "/system/etc/security/cacerts" in Android) to get traffic encrypted using TLS. Second, to manipulate the API request obtained by the tool, Burpsuite can save the intercepted request traffic and replay it after the attacker has changed specific fields (e.g., replaced the long URL with a malicious one).

B. Popular DUSS

TABLE III
Top 15 self-developed DUSSs.

#	DUSS Corporation	Link Domain	Tranco Rank
1	Baidu	j.map.baidu.com	8
2	Reddit	www.reddit.com	35
3	Youtube	youtu.be	40
4	Google Map	goo.gl	57
5	Wangyi	u.163.com	61
6	TikTok	www.tiktok.com	68
7	Skype	join.skype.com	87
8	Aliexpress	a.aliexpress.com	122
9	Booking.com	www.booking.com	166
10	Huawei	url.cloud.huawei.com	295
11	Snapchat	t.snapchat.com	337
12	Ozon	ozon.ru	347
13	Discord	discord.gg	539
14	Coupang	link.coupang.com	785
15	Facebook	fb.watch	1055

TABLE IV
Popular third-party hosting services.

#	Commercial Service	# of Users	Charge
1	Bitly	500,000+	paid
2	TinyURL	102,035+	paid
3	Rebrandly	30,000+	paid
4	Branch IO	100,000+	paid
5	AppsFlyer	12,000+	paid

C. POIs and SDK Endpoints

D. Analysis of SSRF protection.

We choose to analyze the security impact of compromised shortened links on SSRF protection theoretically. The reasons are two-fold. First, unlike bypassing previous checkers, the impact of SSRF may inevitably affect the backend server (where a resource loading happens). Second, there lacks of standard RFC or framework to implement our own SSRF protection for testing. However, the vulnerability described in OWASP [97] emphasizes that if the server enforces SSRF protection using a domain allowlist but fails to disable subsequent redirection, then SSRF can occur. Since the DUSS is intended to serve the web sites belong to the corporations, thus their domain is likely to be configured in the allowlist, leading to potential attacks. Bypassing SSRF protection exposes the remote server to various threats, including sensitive information leakage and remote code execution.

E. Test Case Design

Table VII illustrates our test case design details in mapping with different vulnerability categories. Given the original (first requested) URL `oURL` to be shortened as is trusted, `Ditto` either mutates or replaces the scheme, domain, and path of it with predefined payload. For example, `Ditto` mutates the host part of `oURL` *allowed.com* into *evil.com**.allowed.com* to test whether the domain parser has flawed sanitizer.

TABLE V
ANDROID SDK METHODS PROVIDED BY POPULAR THIRD-PARTY HOSTING SERVICES.

#	Commercial Service	Method Signature
1	Bitly	<com.bitly.Bitly: void shorten(java.lang.String, com.bitly.Bitly$Callback)>
2	TinyURL	Not Support
3	Rebrandly	Not Support
4	Branch IO	<io.branch.referral.Branch: String generateLinkSync(io.branch.referral.ServerRequestCreateUrl)>
5	AppsFlyer	<com.appsflyer.share.LinkGenerator: void generateLink(Context, CreateOneLinkHttpTask.ResponseListener)>

TABLE VI
HTTP URL ENDPOINTS IN POPULAR THIRD-PARTY HOSTING SERVICES.

#	Commercial Service	API
1	Bitly	https://api-ssl.bitly.com/v4/shorten
2	TinyURL	https://api.tinyurl.com/create
3	Rebrandly	https://api.rebrandly.com/v1/links
4	Branch IO	https://api2.branch.io/v1/url
5	AppsFlyer	https://onelink.appsflyer.com/short link/v1/onelink-id

TABLE VII
TEST CASE DESIGN TEMPLATES. GIVEN A LEGITIMATE URL (OURL) INTERCEPTED IN THE API TRIGGERING PROCESS, DITTO WILL MODIFY ITS DOMAIN, SCHEME, AND PATH ACCORDING TO THE TEMPLATE FOR VULNERABILITY TESTING.

Test Scenario	Test Case Template	Description
Scheme	oURL.scheme ← JavaScript	Test sensitive schemes; simultaneously, whether the checker is case-sensitive or not.
	oURL.scheme ← evilscheme	Testing customized evil scheme.
Domain Matching	oURL.host ← evil.com	Whether the checker performs the domain check.
	oURL.host ← {evil}{oURL.host}	Whether the checker uses the "endswith" or "contains" function.
Domain Parsing	oURL.host = {evil.com}\\.{oURL.host}	Whether the host extraction can be confused by "\".
	oURL.host = {oURL.host}@{evil.com}	Whether the checker has inconsistent host extraction.
	oURL.host ← {oURL.host}:x@{evil.com}	Whether the checker has inconsistent host extraction.
	oURL.host ← a:a@{oURL.host}:b@{evil.com}	Whether the checker has inconsistent host extraction.
	oURL.host ← {evil.com}%00@{oURL.host}	Whether the checker has inconsistent host extraction.
	oURL.host ← {evil.com}\@{oURL.host}	Whether the checker has inconsistent host extraction.
Domain Asset	oURL ← common domains	Test with attacker controlled shared domain names.
	oURL ← vulnerable(oURL.host)	Test with URLs vulnerable to XSS, open redirection; domain expiration or takeover.