LeakyLinks: Measuring the Security and Privacy Risks of URL Scanning Services

Ali Mustafa*, Jannis Rautenstrauch*, Florian Hantke*, Shubham Agarwal†, Stefano Calzavara‡, Ben Stock*
* *CISPA Helmholtz Center for Information Security, Germany*
† *Max Planck Institute for Security and Privacy, Germany*
‡ *Università Ca' Foscari Venezia, Italy*
{*ali.mustafa,jannis.rautenstrauch,florian.hantke,stock*}*@cispa.de, shubham.agarwal@mpi-sp.org, stefano.calzavara@unive.it*

***Abstract*—URL scanning services are widely used in security workflows to detect malicious websites and protect users from online threats. However, their common practice of publicly indexing scanned URLs may unintentionally expose sensitive user information through URL-embedded access credentials. Although isolated accounts of such privacy incidents exist, a systematic assessment of their prevalence is still lacking.**

We present LeakyLinks, an automated analysis pipeline that combines URL filtering with LLM-driven semantic classification to identify URLs exposing Sensitive Personal Information (SPI). Using LEAKYLINKS, we analyze URLs collected from public feeds of six prominent URL scanning services over a period of three weeks. With the framework, we visited 332k URLs, identifying over 4k URLs which leak SPI with a precision of 97%.

To further assess the extent to which published URLs are actively accessed by third parties, we deploy honeypages and submit their links to the selected URL scanning services. Our measurements confirm that external entities access URLs submitted to these scanners, often from potentially suspicious IPs exhibiting behavior commonly associated with reconnaissance or opportunistic probing.

Taken together, these findings indicate that URL scanning services represent a valuable target for web adversaries and may already be subject to active exploitation in the wild.

1. Introduction

The increasing prevalence of malicious websites and other online threats has made URL scanning services a crucial component of modern security workflows [5, 13, 17, 30]. These services analyze URLs in real-time or on-demand, identifying potential threats such as phishing or malware distribution [28]. By examining various aspects of a web page (e.g., network requests, embedded scripts, and historical reputation), URL scanning services help security professionals and automated systems to detect risks before users are exposed to adversaries. Platforms like URLScan [39], VirusTotal [41], and Cloudflare Radar [10] play a prominent role in protecting individuals and organizations from online threats. Their integration into email security scanners and threat intelligence workflows demonstrates their importance in the broader security ecosystem. URLScan alone is being integrated into over 30 commercial products [37]. Cloudflare Radar's URL Scanner, launched in March 2023, executed nearly one million scans between its launch and March 2024 [11]. Moreover, ANY.RUN's Threat Intelligence Feeds process approximately 14,000 public sandbox analysis sessions daily between URLs and files, contributed by over 300,000 researchers worldwide, with data available for ingestion every two hours [42]. Researchers and practitioners widely use these services to detect threats and anticipate emerging risks, underscoring their role in both reactive and proactive defense strategies [8].

Although URL scanning is considered a sound security practice, it also poses potential risks due to specific implementation choices by individual service providers. In particular, one popular feature of URL scanning services is the maintenance of a public database of previously scanned URLs along with their analysis results, presumably to emphasize the effectiveness and popularity of these services among their users. This practice has the critical downside of publicly exposing any scanned URL that was intended to be private, as observed in prior case reports [6, 26, 40]. For example, sensitive URLs, such as password-reset links with embedded tokens or pre-signed file URLs, can be exposed to the public through these scan databases. This way, any Web user can visit these URLs to access associated information and perform unauthorized operations.

Some scanning services offer granular visibility controls and publish scan-visibility best practices to mitigate the risks of accidental information leaks [38]. Unfortunately, the granular visibility controls that these scanning services offer by default often fall short of safeguarding their users. For example, when the visibility setting of scans is public by default, sensitive URLs still leak if users or integrations do not actively opt into stricter settings. That is, misconfigured third-party integrations and even users could still inadvertently submit sensitive URLs for public scans, leading to unintended sensitive information leakage in the wild.

To quantify how prevalent these threats are in practice, we develop LeakyLinks, an automated, large-scale measurement pipeline that analyzes URLs publicly indexed by multiple URL scanning services. We leverage automated collection of several URL scanning services' public feeds, combined with near real-time visits to the corresponding pages, to capture URL content as an adversary could. Subsequently, we apply heuristics related to authorization and finally pass the content to a locally hosted LLM to assess its sensitivity. Unlike prior public case reports, which examined individual services in an ad-hoc, manual fashion, LEAKYLINKS enables automatic, at-scale analysis across several scanners, allowing us to measure the prevalence of publicly exposed sensitive URLs, significantly extending earlier reports. At the same time, our design goals for LEAKYLINKS ensure that we aim for low false positives, to provide a conservative lower bound for the prevalence of the problem. Our manual validation of the crucial LLM component shows that it correctly predicts a positive label in 97% of cases, giving high confidence to our findings.

Contributions. To summarize, we make the following key contributions:

- We identify the conditions under which URL scanning services may leak sensitive information. We survey 20 popular services and find that six satisfy these conditions, showing that the risk is not confined to a single service but appears across the URL scanning ecosystem.
- We design and implement LEAKYLINKS, an automated large-scale analysis pipeline that collects URLs from the public feeds of the six scanners of interest and combines URL filtering with LLM-driven semantic classification to detect Sensitive Personal Information (SPI). Using LEAKYLINKS, we collect and analyze 2,286,501 URLs from public feeds, detecting 4,417 URLs exposing SPI. Owing to its high precision and low false-positive rate (see Section 5.3.4), LEAKYLINKS can assist scanner operators in auditing and mitigating inadvertent data exposure. The LEAKYLINKS code is available at [25].
- We deploy honeypages and submit associated decoy URLs to each service to determine the extent and nature of third parties regularly scraping their public feeds. In doing so, we observe that entities, including those originating from IPs associated with malicious activities, are already actively scanning public feeds.

Ethics. Our research is conducted on live systems involving data from various individuals and organizations. To uphold academic ethics standards, we considered ethics in every step of the project. We conducted a stakeholder analysis (see Appendix A), as recommended by the Menlo Report [4], before beginning our study. In addition, we discussed our proposal with our ERB and received formal approval.

Our stakeholder analysis led to two primary outcomes. First, we aim to minimize manual analysis wherever possible to protect the privacy of affected users. While we need to sample some data points to validate the efficacy of our detection tool, we keep this to a minimum. Moreover, we decided to rely on a locally hosted LLM running on secure internal servers to identify and validate findings. This ensures that we do not need to view all data ourselves, nor is any data exposed to third parties. While this strategy risks false positives and negatives (as discussed in Section 5.3.4), it provides a reasonable balance between accuracy and privacy.

The second key point is the design of a responsible disclosure policy. We choose to notify both individual owners of websites leaked and providers of the analyzed services about our findings, as later detailed in Section 7.2. At the same time, to prevent reputational harm, we do not name any affected companies in our paper, but only describe high-level examples. We expand on ethics in Appendix A.

2. Background on Web Authorization

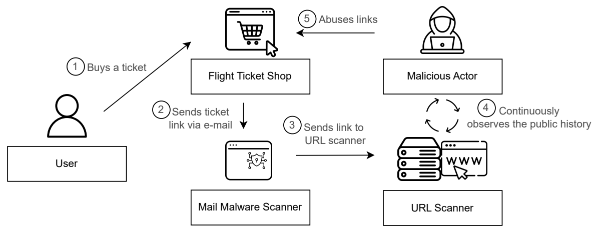
The HTTP protocol implements the client-server paradigm and is stateless by design. To protect resources from being accessed by unauthorized parties, HTTP/1.0 introduced the notion of authentication through an HTTP header [16]. This mechanism, though, is cumbersome to handle logouts, which is why modern Web applications instead rely on application-level authentication and authorization checks. One prominent example is encoding authentication and authorization into URLs themselves.

When considering links that may contain sensitive information, any URL that carries a secret, high-entropy value functions as token-based authorization. These tokens are server-generated and can be validated to prove that a client is authorized to access a resource. Examples include random document identifiers (such as in Google Docs). Servers may use such tokens in different ways: they may set a cookie from it, validate it once and redirect, or keep rewriting links to include it. As long as the token appears in the URL, it effectively acts as a bearer credential.

The second form of authorization ties in with *authentication*. Here, a user is first identified (e.g., by being logged into an application) and can then use the rights associated with that account to be authorized to access a given resource. This may occur if a secret link is used to establish such *login state* in the process. A common example of this is "login with email" functionality, where an application sends a link that embeds an access token to the user's email. Visiting this link then establishes a state (through cookies or by setting Local Storage values) within the browser. Since the browser now has a state established, this is automatically (in case of cookies which are sent to the server) or programmatically through JavaScript (if saved in Local Storage) used to identify the user towards the server for any further interaction with the application.

In both cases, the authorization step is meant to protect a resource from being accessed by parties not privy to the credentials. While the paradigm holds for both cases, the second form of authorization frequently involves redirections; once the user is authorized through the secret token in the URL, they are redirected to a different page, such as a dashboard or profile page, which may no longer contain any secret token in the URL.

In this context, any request that successfully conveys valid access credentials to the server can be considered an *authorized request*. Such credentials may be embedded



1 Buys a ticket
Flight Ticket Shop
5 Abuses links
Malicious Actor
User
2 Sends ticket link via e-mail
3 Sends link to URL scanner
4 Continuously observes the public history
Mail Malware Scanner
URL Scanner

Figure 1: Motivating example of how sensitive URLs can get exploited via URL scanning services

directly in the URL, as in token-based authorization, or may be implicitly carried by the browser through an authenticated session established earlier (for example, after a redirection that sets cookies or Local Storage). In both cases, the request grants access to protected resources, and a leak of the corresponding credentials, whether in the URL or in session state, would enable unauthorized access.

3. Risks of Leaking Links

We clarify here the risks associated with URL scanning services, define our threat model, and present the key research questions of our study.

3.1. Motivating Example

To illustrate the risks connected to URL scanning services, we consider the example in Figure 1. A user purchases a flight ticket from an airline website (1). After completing the purchase, the airline sends back a direct access link for the boarding pass to the user's corporate email address (2). The user's company, highly concerned about security and privacy risks, automatically scans all incoming emails to detect potential threats. As part of their vetting procedure, all links contained in incoming emails are automatically submitted to a URL scanning service to verify their trustworthiness (3). Unfortunately, as is common for such services, the URL scanner publicly indexes recent scans by default as part of its threat-intelligence features; as a result, links submitted by the company become discoverable. Thus, malicious actors can continuously monitor recent scans to identify URLs leading to pages containing private information of high-profile websites (4). Upon finding the boarding-pass link, the attacker can retrieve it directly because the URL itself confers access without additional authentication.

We emphasize that an automated email scanner is just one option for sensitive links to be published on such URL scanners. Users may also send links to such services manually, unaware that these links are published in the publicly accessible feed of the given scanner.

3.2. Threat Model

The attacker's goal is to get access to **Sensitive Personal Information (SPI)**, i.e., information specific to an entity and intended to remain private or confidential. It includes, but is not limited to, data such as private contact details, personal identifiers, financial or transactional records, and other non-public personal content. Because many string types (emails, names, order IDs) can appear on legitimate public pages without implying private disclosure, we do not treat the presence of such strings as SPI by itself. In other words, SPI is the subset of Personally Identifiable Information (PII) that is meant to stay non-public by intention or policy, hence must be protected against unauthorized access. This implies that the resource that contains the data must be protected by means of an *authorized request* as discussed in Section 2. We consider a passive adversary who aims to extract SPI from URL scanning services. This adversary acts opportunistically, continuously monitors public feeds of URL scanning services, and exploits whatever information is publicly visible without requiring authentication or user interaction. The adversary has the ability to view and access all public URLs, including query strings and paths. They may visit the URLs themselves, potentially within seconds of publication, to capture the rendered page. In our example (Figure 1), the direct access link generated by the airline company may include personal details of the boarding pass. Anytime the attacker observes a URL that looks like a promising target for abuse, for example, if it points to a known file-sharing service or includes what appears to be an access token, the attacker navigates to the URL in an attempt to cause harm. This also includes security-sensitive links such as password-reset or login-via-email URLs, whose leakage puts the associated accounts at risk. Since anyone on the Web can visit these URLs with potentially sensitive content, we consider any unprivileged Web user with access to these URLs included in our threat model.

We assume that the adversary operates entirely through public interfaces and does not access private or unlisted scans. They do not perform authentication bypasses or manipulate scanning infrastructure. Their behavior reflects realistic automated OSINT-style collection, operating at scale and near real-time.

3.3. Research Questions

SPI leaks enabled by URL scanning services have previously been observed in public case reports and industry blog posts [26, 40], suggesting that a few public scan archives may inadvertently expose sensitive content. This prior work raised awareness about potential misuse of public URL scanning services; however, there is still little systematic understanding of how often such platforms leak sensitive data. In particular, prior studies are based on *dorking*, i.e., researchers manually enumerate patterns corresponding to likely sensitive content (file sharing links, API keys, etc.) and look for such patterns using the search facilities and historical indices of URL scanning services when available. This approach thus relies on known sensitive patterns and on the existence of rich search interfaces, which necessarily underestimate the prevalence of the problem in the wild and limit the scope of the analysis, because it biases it towards specific services of interest.

To systematically characterize this threat landscape, we consider three research questions:

1) **RQ1:** *Which public URL scanning services are prone to SPI leakage?* To answer this, we collect and analyze existing services to identify those that provide publicly accessible streams of scanned URLs.
2) **RQ2:** *How prevalent is SPI in scan data publicly disclosed by URL scanning services?* To answer this, we implement a fully automated pipeline that combines URL filtering with LLM-driven semantic classification to detect sensitive information at scale. This way, we can automatically scrape hundreds of thousands of URLs from multiple public feeds and quantify room for exploitation without relying on dorking.
3) **RQ3:** *What are the characteristics of visitors of leaked links and do they show signs of malice?* For this, we set up a honeypot experiment in which we submit sensitive-looking URLs and observe characteristics of those who visit the URLs. While this does not allow us to reason about their malicious nature, it provides indications of parties acting in line with our threat model.

4. Who Leaks Links?

To systematically characterize the threat landscape, we analyze existing URL scanning services to identify those potentially vulnerable to our threat model, thereby answering RQ1: *Which public URL scanning services are prone to SPI leakage?*

The threat model implies that an attacker must be able to extract URLs from a service, hence, any service that does not publish scanned URLs cannot leak SPI. In addition, if a service only provides the list of scanned *base-domain* URLs (i.e. without path and query string) rather than complete URLs, the service is also not prone to being abused by the attacker. This is due to the fact that some form of authorization from the URL must exist for SPI leakage, which is impossible with base-domain URLs. Finally, since we study unintentional leakage through public scan feeds, services whose public feeds are dominated by malicious or suspected-malicious submissions are outside our scope. Our goal is to analyze services that publicly expose URLs submitted through ordinary security workflows, rather than repositories primarily intended to collect phishing or other malicious indicators.

Thus, we define three criteria when analyzing whether an identified URL scanning service may be prone to abuse:

1) The scanner must publish a publicly accessible feed of resources they scanned.
2) The scanner publishes full URLs in its feed.
3) The scanner's public feed must not be predominantly malicious.

To apply these criteria, we first compiled a list of candidate services by conducting an extensive lookup via search engines. Then, we determined if they fulfill the above criteria and considered the matching services for further analysis. All our observations are based solely on publicly accessible features and free or trial plans of these services; we did not purchase or use any premium or enterprise offerings.

We extensively searched the Web, using key terms like *public URL scanning services* or *live threat intelligence feeds*. We examined the first 10 pages of Google search results for each key term. Additionally, we recursively followed any references to other scanning services embedded within the collected search results to identify additional candidates for our analysis. Through this process, we identified 20 URL scanning services from our Google search and assessed whether they met the three criteria for SPI leakage.

Table 1 reports all the identified URL scanning services. For each service, we mark whether it satisfies the preconditions required by the threats under analysis. As a result of this analysis, six services fully match the criteria for SPI leakage: Anyrun [1], Cloudflare Radar [10], Hybrid Analysis [18], Joe Sandbox [19], URLQuery [36], and URLScan [39]. These six scanners are therefore the focus of our subsequent analysis and, during our experiment, they collectively contributed more than two million published URLs in their public feeds, providing a large and diverse corpus for studying SPI leakage in practice.

5. What Links Are Leaked?

After identifying candidate services that Web adversaries may target, we now turn to RQ2: *How prevalent is SPI in scan data publicly disclosed by URL scanning services?*

To systematically evaluate SPI leakage, we develop LEAKYLINKS, a modular framework that processes the live public URL feeds from scanning services and estimates the prevalence of SPI by analyzing the retrieved content. LEAKYLINKS processes live URL feeds to maximize the likelihood of capturing sensitive links before they expire.

In the Web setting, SPI should only appear in responses to *authorized requests* (per Section 2): either because the URL itself embeds an authorization token or because prior redirects established client-side state. Thus, we focus on leaked URLs that already present indicators of potentially credentialed access and measure how often a subsequent case returns content that contains SPI. Because identifying credentialed access is a prerequisite to measuring SPI leakage, we design LEAKYLINKS to first check whether a URL shows potential token- or state-based access, and only then proceed with analyzing its retrieved content for SPI.

5.1. Overview of LEAKYLINKS

Figure 2 shows the overall architecture of LEAKYLINKS, which determines whether URLs publicly shared by scanning services expose SPI. The system is organized into modular components, each addressing a specific concern.

Given the identified URL Scanning Services, we extract URLs from their respective public feeds. The URLs pass

Service	Public URL feed	Publishes full URLs	Not Predom. Malicious	Used in this study
app.any.run (Anyrun)	✓	✓	✓	✓
radar.cloudflare.com (Cloudflare Radar)	✓	✓	✓	✓
hybrid-analysis.com (Hybrid Analysis)	✓	✓	✓	✓
joesandbox.com (Joe Sandbox)	✓	✓	✓	✓
urlquery.net (URLQuery)	✓	✓	✓	✓
urlscan.io (URLScan)	✓	✓	✓	✓
phishtank.org	✓	✓	✗	✗
criminalip.io	✓	✗	✓	✗
hypestat.com	✓	✗	✓	✗
immuniweb.com/websec	✓	✗	✓	✗
safeweb.norton.com	✓	✗	✓	✗
emailveritas.com	✗	–	–	✗
check.lionic.com	✗	–	–	✗
checkphish.bolster.ai	✗	–	–	✗
developer.mozilla.org/en-US/observatory	✗	–	–	✗
scanurl.me	✗	–	–	✗
sitereport.netcraft.com	✗	–	–	✗
sitecheck.sucuri.net	✗	–	–	✗
transparencyreport.google.com	✗	–	–	✗
virustotal.com	✗	–	–	✗

TABLE 1: Overview of URL scanning services identified through extensive Web lookup and whether they were included in our study. ("✓" & "✗" indicates whether the corresponding service passes or fails the selection criteria. "–" indicates that the criteria is *not applicable*.)

through the *Live Crawl* stage, which first discards bare base-domain URLs (e.g., https://example.com/) and entries flagged as malicious, as these trivially cannot encode user-specific authorization credentials. Next, *Live Crawl* visits each non-filtered URL to resolve its final destination, verify that it is reachable, and capture two controlled views of the page: one after following all potential redirections (including potentially established state) and another after clearing any client-side state. This way, LEAKYLINKS collects paired snapshots that later components use to assess whether URL access may depend on an *authorized request*.

Building on the previous step, the *URL Token Checker* analyzes each final (after redirects) URL to identify cases where access might be tied to a secret embedded directly in the URL itself. It scans URL components for high-entropy tokens that could indicate credential-bearing parameters. This step acts as a necessary check: URLs with token-like parameters are directly routed to SPI detection, while the others undergo an additional check to infer whether access may have been restricted through *authentication* from the redirection flow.

For pages without token indicators in the final URL, the *Page Difference Checker* evaluates whether content access might depend on client-held state established during browsing. It compares the two snapshots collected by the Live Crawl: one with the state intact and one after clearing all states to detect meaningful page changes. Such changes suggest that access to the page may rely on potential client-state or an authenticated context. Like the Token Checker, this component provides a conservative signal: it cannot confirm that the final page is authenticated, but indicates when state-based access is likely involved.

Finally, the *SPI Detection* stage examines all the candidate URLs that passed the prior checks to verify whether they actually expose SPI. Its goal is to separate actual instances of SPI leakage from pages that merely exhibit token-like URL parameters or state-dependent behavior, but do not reveal SPI. This stage uses a content-aware, locally hosted, large language model with vision capabilities that operates directly on rendered page screenshots. By reasoning over layout and surrounding cues rather than isolated strings, this stage distinguishes genuinely sensitive content from generic or public information, providing the final confirmation of SPI exposure in the scanned data.

5.2. Implementation Details

We now describe each stage of LEAKYLINKS – as outlined in the previous section – in more detail.

5.2.1. Live Crawl. The goal of the live crawl stage is to obtain two near-live, comparable views of each URL scraped from the URL scanning services. Before crawling, this stage filters the incoming feed to remove entries unlikely to yield meaningful results. It discards bare base-domain URLs (e.g., https://example.com/) and excludes any domains flagged as malicious using Google Safe Browsing. These filters reduce noise and focus the measurement on legitimate, user-facing pages.

We obtain the first view of each page through the natural client-side state that the site establishes, possibly after redirects, and the second view after we deliberately remove all the associated stateful data on the client-side. As

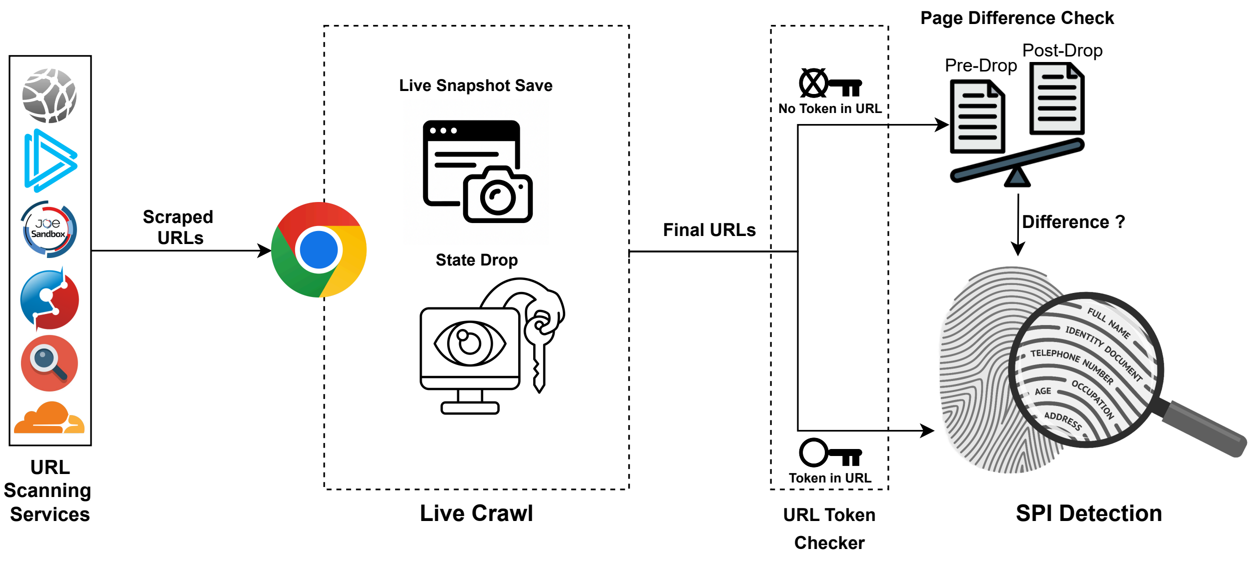


Figure 2: LEAKYLINKS Architectural Diagram

we discuss further, having both views allows us to determine whether the final content depends on credentials carried in the request or held in the client state. For this, we crawl each given URL in a headful Chromium-based browser with a timeout of 30 seconds on both pre- and post-state drop visits. We follow all top-level redirects until we reach the stable final URL, and remain in the same browser context. We then clear all session-bearing client data: *cookies*, *Local Storage*, and *Session Storage*, by reloading the final URL in a new context. We store the DOM snapshot before and after the state drop for downstream components to use. We only crawl the given URLs without interacting with the respective pages, to avoid triggering side effects due to ethical reasons.

5.2.2. URL Token Checker. The URL Token Checker inspects the final URL of the first visit from the Live Crawl to decide whether the URL embeds any credential tokens and should undergo scrutiny for SPI detection. Indeed, recall that the presence of random tokens might suggest that access depends on credentials embedded in the URL. For example, starting from a scraped link, we may reach a URL like https://example.com/app/⟨auth-token⟩, which is a promising target for SPI detection.

Concretely, we parse the URL into its standard components (path, query string, and fragment) and scan each of them for ASCII substrings of length at least 8 whose Shannon entropy is at least 2.0; these thresholds were chosen empirically (see Appendix C). The path is further split on "/", so a path like `/a/path/like/this` is treated as four separate segments (`a`, `path`, `like`, `this`), each checked independently. The query part is split into key–value pairs (e.g., `a=foo&b=bar` becomes parameters `a` and `b`, and we check both their names and their values). The fragment, if present, is also checked.

5.2.3. Page Difference Checker. Even if the final URL does not contain any tokens, access to it may nevertheless be linked to client-side state. For example, consider a "magic link" login where visiting https://service.example/login?tok=⟨token⟩ authenticates the user and then redirects to a dashboard at https://service.example/profile. The final URL contains no token, but the browser stores a session cookie before the redirect, which then provides the authorization to access the specific page and user-specific content.

The Page Difference Checker estimates whether the content of a page is likely to depend on client-side state. It compares the DOM snapshots recorded before and after the state drop in the Live Crawl and checks whether the two versions are similar enough to be treated as the same page. In our example, reloading the profile page after dropping client state yields a login prompt instead of the dashboard, indicating that access depended on stored state.

To quantify similarity, we reuse the page-similarity scoring method of Roth et al. [31], which combines several content-level features, such as overlap of script hosts, counts of loaded scripts, title similarity, and response size, into a single score in $[0, 1]$. Following their evaluation, we treat pages with a similarity score below 0.8 as *dissimilar*.

This component complements the URL Token Checker: while the token checker handles cases where access credentials are embedded directly in the URL, the Page Difference Checker targets pages protected through stored client state (e.g., cookies). If dropping the state causes a significant change in the page, we treat this as a signal that the page may have been protected through an authorization flow, and we forward the page for SPI analysis. We exclude pages that remain effectively unchanged from further inspection.

5.2.4. SPI Detection. The SPI Detection stage is the component that ultimately checks whether a page exposes Sensitive Personal Information (SPI) as defined in Section 3.2. The preceding stages (URL Token Checker and Page Difference Checker) only identify pages whose access appears to depend on tokens or client-side state, i.e., pages that may be reached through an authorized request but are not necessarily sensitive. SPI Detection then applies a content-aware check to decide whether the rendered page actually contains SPI.

Detecting SPI is not as simple as matching personal-looking strings, such as email addresses, via regular expressions. The same types of identifiers often appear in public or templated content. Thus, the key challenge is to distinguish benign occurrences from genuinely private contexts such as account dashboards or user-specific application views. Prior PII- or string-based methods aim to detect whether a page contains PII-like strings, whereas our task is to decide whether the rendered page exposes sensitive, user-specific, non-public information in context. While regex-based techniques can work well for well-structured fields, they are inflexible with respect to page context and may therefore miss SPI exposures whose sensitivity depends on page context [23]. We therefore rely on a vision–language model over full-page screenshots, allowing the detector to combine semantic understanding with layout and visual grouping when deciding whether a page truly exposes SPI.

To preserve this context and ensure privacy, we use a locally hosted large language model (`qwen3-vl-30b-a3b-instruct`), which supports image inputs, instruction tuning, and is efficiently executable on our hardware [3]. We directly feed the snapshotted image from Live Crawl (Section 5.2.1) to this LLM. Here, we prompt the model to act conservatively: it must base its decision only on visibly rendered content in the screenshot, treat form labels and example/placeholder values as non-sensitive, ignore public or generic contact blocks, and return "sensitive" only if it can quote an on-screen artifact that clearly represents private user data (e.g., a prefilled personal email in an account view, a document showing a user identifier, or a displayed secret). The prompt is explicitly cautious to skip public or ambiguous pages as SPI. We supply the used prompt in Appendix E.

All inference runs locally so that screenshots with potential SPI do not leave our controlled environment. Pages that do not contain any legible text (as detected by an OCR check) are directly flagged negative to avoid wasting computational resources and energy.

5.3. LeakyLinks Results

We now present the main results of our prevalence measurement. We first provide an overview of the collected and filtered data, followed by a detailed analysis of SPI prevalence and its privacy and security implications, including case studies. Finally, we conclude with a validation of our detection pipeline.

5.3.1. Dataset Collection and Filtering. We subscribed to the public feeds of the six URL scanning services identified in Section 4 between October 10 and November 1, 2025, and collected a total of 2,302,812 URLs across all services (per-service insights in Table 2). We identified 15,727 duplicated URLs, each appearing under multiple scanning services. A breakdown of these duplicated URLs reveals that 15,208 appear in exactly two services (96.7% of the duplicate URLs), while only a small number span three (458 URLs), four (57 URLs), or five services (4 URLs). After deduplication, the final dataset comprised 2,286,501 unique URLs, as summarized in Table 2.

Scanning Service	Collected URLs	Non-Base Domains	Non-Malicious
Anyrun	36,811	23,480 (63.8%)	23,169 (62.9%)
Cloudflare Radar	443,576	7,474 (1.7%)	7,382 (1.7%)
Hybrid Analysis	25,522	16,301 (63.9%)	16,061 (62.9%)
Joe Sandbox	5,049	3,563 (70.6%)	3,497 (69.3%)
URLQuery	336,940	134,890 (40.0%)	124,386 (36.9%)
URLScan	1,454,914	188,980 (13.0%)	164,315 (11.3%)
Total (de-dup.)	**2,286,501**	**368,361 (16.1%)**	**332,647 (14.5%)**

TABLE 2: Filtering pipeline statistics per scanning service. Percentages are relative to collected URLs.

The first step of the Live Crawl component (Section 5.2.1) discards URLs that are just base domains, i.e., URLs without any path or query string. As Table 2 shows, this filtering step removes a large portion of the raw feed. After this preliminary step, we retain 368,361 URLs (16.1% of total scraped URLs) for further processing. In the next step, Live Crawl filters out URLs flagged as malicious by Google Safe Browsing. Overall, this leaves us with 332,647 (14.5%) URLs for analysis in our subsequent visits.

For these filters, we observe that the largest URL sources (URLScan and Cloudflare Radar) have the lowest share of URLs surviving the filter steps. The differing nature of collected URL contributions across services can explain this discrepancy. High-volume platforms like URLScan and Cloudflare Radar contribute vast numbers of URLs. However, a disproportionate share of these submissions consists solely of base domains, i.e., URLs without any meaningful path or query string. Specifically, 436,102 out of 443,576 Cloudflare Radar URLs (98.3%) and 1,265,934 out of 1,454,914 URLScan URLs (87%) fall into this category and are therefore filtered out (see Table 2). In contrast, smaller services such as Anyrun and Hybrid Analysis contribute fewer URLs overall, but a much larger share of full URLs that contain non-base-domain URLs. These are more likely to pass the preliminary filter, explaining their comparatively higher pass rates despite lower total volume.

5.3.2. Crawl Execution and Routing. After these initial filtering steps inside Live Crawl, 332,647 URLs proceeded to the crawling phase of the component. For 27,108 URLs, the visit attempt returned no retrievable page (dead link, DNS resolution failure), so we exclude them from the routing breakdown. The remaining 305,539 URLs produced a usable snapshot and form the basis of the analysis below; all percentages are relative to this set.

Sensitivity Class	Account Pages	Auth & Verif.	Billing, Payments & Orders	Booking, Travel & Sched.	Compliance & Consent	Docs, Records & Reports	Logistics, Delivery & Tracking	Messaging, Contact & Support	Prefs & Subscriptions	Security & Alerts	Sharing, Access & Invites	System State & Utility	**Total**
Anyrun	86	182	40	10	10	99	5	36	259	40	10	1	778
Cloudflare Radar	7	35	8	1	2	17	0	3	144	3	3	1	224
Hybrid Analysis	55	143	28	9	4	56	3	29	113	98	8	4	550
Joe Sandbox	12	46	5	1	2	14	1	3	9	6	8	0	107
URLQuery	106	199	168	26	10	102	3	39	222	4	7	11	897
URLScan	231	156	142	33	38	222	18	146	664	191	12	8	1,861
Total	**497**	**761**	**391**	**80**	**66**	**510**	**30**	**256**	**1,411**	**342**	**48**	**25**	**4,417**

TABLE 3: Sensitivity classes of pages detected by LEAKYLINKS, by URL scanner

Among these URLs, 226,338 (74.1%) contained at least one *token* and were routed directly to the SPI Detector (Section 5.2.4). The other 79,201 (25.9%) had no token and followed the alternative branch, where the deciding step is a *page change* after the state drop (Section 5.2.3); a top-level redirect is a prerequisite check.

Within the no-token branch (79,201 URLs), 50,449 showed a top-level redirect. From this set of 50,449 URLs, only 2,918 (1.0% of all 305,539 visited URLs) satisfied the page-change criterion and were routed to the SPI Detector. This step was useful as it discarded 25.0% (76,283) of all 305,539 visited URLs, thus saving time and resources in the SPI step. In total, 229,256 URLs were sent to SPI detection: 226,338 from the token branch and 2,918 from the no-token branch.

5.3.3. Prevalence of SPI. In this section, we now report the numbers and discuss cases of SPI-containing pages detected by LEAKYLINKS. The per-service and type breakdown is shown in Table 3.

A total of 4,417 pages were detected to contain SPI. Of these, 4,387 came from the branch of URLs that contained a token, and 30 from the no-token branch. Our detection pipeline is explicitly optimized for high precision to avoid over-reporting sensitive pages; therefore, these counts should be interpreted as conservative lower bounds on the number of exposed SPI pages observed in our collection. Any proportions or distributions derived from these counts should be read in that light.

These detected SPI pages extend over 1,843 domains (considering the final landing page), with an average of 2.4 URLs per domain, a minimum of 1, and a maximum of 273. It is worth noting that URLs leading to these pages take a highly diverse range of forms and structures. That means that they are not practically discoverable by a manually crafted set of dorks. Notably, the affected domains include many highly popular sites. According to the Tranco ranking, 69 domains appear in the Top 1K and 210 in the Top 10K. Looking more broadly, 1,098 of the 1,843 affected domains (59.6%) are listed in the Tranco Top 1M, indicating also that the issue is not limited to obscure or abandoned sites, but also affects widely visited parts of the Web.

Comparing across the URL scanning services, Table 3 shows that the largest contributor to these SPI pages is *URLScan* with 1,861/4,417 (42.1%), followed by *URLQuery* 897 (20.3%), *Anyrun* 778 (17.6%), *Hybrid Analysis* 550 (12.5%), *Cloudflare Radar* 224 (5.1%), and *Joe Sandbox* 107 (2.4%). Relating to initial collection volumes (Table 2), *Cloudflare Radar* collected the second most URLs overall yet contributes relatively few SPI pages in the final set, while dynamic analysis services like *Anyrun* and *Hybrid Analysis* collect far fewer URLs but contribute a larger share of SPI pages. This indicates that the prevalence of SPI among collected URLs varies substantially by service.

The detected SPI can surface in many different kinds of pages across the Web (e.g., invoice views, order tracking, document signing, profile editing). To make the results interpretable, we categorized the labels returned by the SPI detector into twelve reasonable classes, shown in Table 3. Notably, five classes account for most pages: *Preferences & Subscriptions* (e.g., unsubscribe, manage email preferences) has 1,411/4,417 (31.9%), followed by *Auth & Verification* (e.g., prefilled login, password reset) 761 (17.2%), then *Docs, Records & Reports* (e.g., document viewer, e-signature or certificates) 510 (11.5%), *Account Pages* (e.g., dashboards or settings) 497 (11.3%), and *Billing, Payments & Orders* (e.g., invoice views, checkout) 391 (8.9%).

While SPI exposure affects a wide range of domains and classes, certain domains appear particularly often in our dataset. This is primarily due to large providers whose leaks span either multiple users or multiple product-specific subdomains. The most prevalent case is an email security provider with 273 URLs, which shows email spam detections and thus leaks users' email addresses. This provider is followed by a major productivity suite accounting for 162 affected URLs across several of its services (e.g., document editing, file storage, calendar, and confidential mail). The leaked data for this domain encompasses a broad variety, including private photos, contracts, and survey results. Here, we notice that even within the same provider, exposed URLs span a wide range of paths, subdomains, and URL structures, making it difficult for manually crafted dorks to comprehensively capture the full landscape of SPI exposure. Other frequently appearing domains are a marketing/CRM platform with 105 URLs, a cloud office provider with 83 URLs, and a document-signing service with 52 URLs. Taken together, these high-volume cases show that SPI exposure can occur across multiple product-specific subdomains of the same large provider, suggesting a systemic issue. The phenomenon is not driven only by small or poorly maintained sites.

Besides the high-volume cases, we also observed a number of particularly severe cases during manual validation and pipeline development. We encountered both privacy-harming and security-relevant cases that underline how sensitive these exposures are. For example, pages belonging to SPI class *Auth & Verification* include prefilled login pages or password reset forms exposing users' email addresses; many of these cases could plausibly have enabled account takeovers based on their content, although we did not attempt to log in or modify accounts for ethical reasons. We also identified *Billing, Payments & Orders* and *Booking/Logistics* cases having high-value financial invoices, personalized tracking links, or hotel bookings leaking private and corporate-sensitive information, as well as detailed time and location patterns, to third parties. In addition to the payments, we also saw pages in the *Docs, Records & Reports* class that were highly sensitive, including visa applications, government documents, and files marked as military-classified. One particularly severe case was an e-visa page from this class where both the online application and the issued visa were visible to the visitors. The page contained detailed personal information such as name, address, phone number, and place and date of birth, together with passport and travel details, and still exposed active editing controls. We did not attempt to submit any changes (see Appendix A), but the presence of these controls suggests that an attacker could potentially modify, not just view, the data.

5.3.4. Evaluating Precision of SPI Detection. To facilitate large-scale analysis, our methodology relies on an LLM to determine whether content shown in a particular page is sensitive or not. This is a crucial component to judge the accuracy of our results. In this section, we validate the precision of the LLM component in correctly predicting both positive (i.e., how many pages flagged as containing SPI actually do so) and negative labels (i.e., how many pages flagged as not containing SPI actually do so). For this, we focus exclusively on URLs that are passed on from the check for authenticated requests.

To estimate the precision of the positive cases, we manually reviewed a domain-stratified random sample of 300 of the 4,417 SPI-positive URLs (at most one URL per domain). Of these, 291 indeed contained sensitive personal information, while 9 were false positives. This corresponds to a precision of 97.0% (9/300 false positives; 95% Wilson Score Interval [7] 94.4%-98.4%).

Examining cases where the pipeline provided a negative label, we also apply a sampling approach. However, given the significant skew to this class of prediction, we need a large sample size. Specifically, out of the 301,122 URLs which were not flagged by our end-to-end pipeline, 224,839 were passed on to the LLM in the first place. The remaining 76,283 were filtered out by the authorization heuristics. Therefore, we decided to sample about 1% of the set of URLs for which the LLM predicted a negative label, yielding 2,250 cases for manual inspection. In total, of these, we found that 12 pages contained SPI. This implies that the precision of the negative labels is 2,238/2,250 (99.47%) (95% Wilson Score Interval: 99.07–99.69%).

These two measures of precision now allow us to reason about the likely prevalence of SPI in the wild. Under the assumption that our sampling was truly random and thus generalizes to the entire dataset, of the 4,417 findings identified in Section 5.3.3, around 133 may have been false positives. On the other hand, considering the fact that 224,839 pages were flagged as non-SPI-bearing and that our sampling showed a precision for negative predictions of 99.47%, it is highly likely that approximately 1,191 additional URLs in the dataset contained SPI. In conclusion, the total number of security or privacy-critical URLs in the dataset is estimated to be around 5,475 (4,417 - 133 + 1,191) URLs.

6. Who Looks at Leaked Links?

In Section 5, we showed that public feeds of URL scanning services expose SPI. Our goal in this section is to study who visits links available on public feeds. In particular, we aim to characterize these visitors based on their observed behavior and criteria, such as the IP address they use to visit the leaked links. Specifically, we answer RQ3: *What are the characteristics of visitors of leaked links and do they show signs of malice?*

To investigate this, we set up a honeypot infrastructure and submitted decoy pages through the six selected URL scanning services described in Section 4. This setup allows us to passively monitor any visits to these URLs. By analyzing the origin, access patterns, and technical characteristics of the requests, we assess whether they are consistent with benign indexing behavior or suggest more targeted interest. We explicitly note that these URLs cannot be accidentally visited (e.g., by crawlers seeded from Certificate Transparency logs [21]) given that they need to be aware of the specific paths; that is, any visitor to the URL must necessarily originate from the feeds of the URL scanning services.

6.1. Honeypage Experiment Setup

We designed a honeypot page (*honeypage*) (see screenshot in Appendix D) that includes multiple hyperlinks, embedded images, a dropdown menu, a button, and a form. As evident from the screenshot, we have designed the page to explicitly solicit interactions from an automated crawler. The combination of information present embedded into the page (e.g., MySQL dumps alongside credit card information) also makes it trivially detectable as a honeypage for a human. Instead, the dropdown menus and QR codes are meant to infer whether the visitor automatically interacts with the page or scans the QR codes.

To detect more advanced or potentially malicious activity, we rely on the service *Canarytokens.org* [33] to create and embed various *tokens* in our page. These tokens include fake secrets (e.g., a credit card number in a pre-filled form, an email address, and fake AWS credentials). We also deploy instrumented triggers designed to detect the opening of decoy documents, loading Web resources, and attempting to access network paths. Additionally, we include a QR code unique to each submission, linking to a controlled endpoint to detect attempts to extract or interact with visual elements.

We clone the same base HTML content in all honeypages. Each instance is hosted under a unique subdomain and accessed via a distinct URL. The only variation in the page content itself is the embedded QR code. We refer to each submitted honeypage URL as a *honeylink*.

Since feeds may expose URLs with different visible structures (e.g., generic paths, login-like paths, or paths with long tokens), we also examine whether this URL shape alone affects who later accesses them. We therefore compare accesses to honeylinks that differ only in their path/query structure. Websites, in practice, may include URLs with varying levels of sensitivity. Subsequently, the adversary may also prioritize or react to them differently. That is, plain URLs corresponding to static content (`/static/articles`) may be less enticing for adversaries when compared to the URLs indicating towards more sensitive content (e.g., `/login`).

Thus, to systematically test how different types of URLs may attract different kinds of accesses, we structured the submitted URLs into four categories, each crafted to resemble a different level of sensitivity. The *plain* URLs followed the flat path `/news/`, simulating generic public content. The *sensitive-login* URLs used the path `/secret/login/`, suggestive of access-controlled pages. The *entropy-path* category embedded a hard-to-guess identifier directly in the path; `/secret/login/562210be-067c-4a62-b8a8-df27f3893a80/`. Finally, the *entropy-query* URLs used structured paths of the form `/dashboard/?authCode=7e73...&key=U2Vj...`, where entropy appears in query parameters, to emulate dynamic endpoints or authentication tokens.

We submitted honeylinks to the six services listed in Section 4. All submissions used freely accessible channels and public visibility settings to ensure the URLs were visible to external parties. To track which service each honeylink was submitted to, we embedded a unique service code into the URL path. These codes (e.g., `u` for URLScan or `r` for Cloudflare Radar, ...) allowed us to later attribute crawler activity to the correct scanning service. We note that all URLs at least contain a numerical identifier and an indicator of the tested service (e.g., `/news/2/u`). This not only allows us to attribute visits correctly, but more importantly, ensures that automated crawlers are highly unlikely to guess the path correctly. This provides us with high confidence that visitors of the URLs must have originated from the public feeds of the URL scanning services we used.

We used a Caddy server to collect all access information, including timestamps, received headers, and remote IPs. For each visit, we looked up the IP address on a threat intelligence platform, Alienvault OTX API, shortly after data collection [2] to identify whether that IP was already reported by the community as potentially suspicious (e.g., linked to scanning, brute-force, or malware activity). While this is by no means evidence of malicious visitors, it serves as a strong indicator of these visits not coming from researchers like us.

6.2. Honeypage Access Log Analysis

Table 4 shows the high-level findings of our experiment. We define a visit as a set of requests to resources under the same path within five seconds from the same IP address.

The table presents the visit counts broken down by honeylink type and scanning service. Note that we always disregard the first visit after submission, as we assume this originates from the service itself. While we submitted the same number of honeylinks to each service, honeylinks submitted to URLScan and Hybrid Analysis attracted substantially more follow-up traffic than the others. For instance, Hybrid Analysis links with the *entropy-query* pattern received up to 314 visits from 174 distinct IP addresses. Overall, across all link types, Hybrid Analysis submissions were accessed by 459 unique IPs: the largest diversity among all services.

Similarly, for all other services, the total number of unique IPs is larger than that observed for any single honeylink type, indicating that distinct groups of visitors may selectively access different categories of leaked URLs. Further, the fact that several IP addresses are seen repeatedly across visits (i.e., the number of unique IPs is significantly lower than the total observed IPs) indicates that visits are unlikely to originate from a single curious user browsing the service feed. Instead, it points to some automated visits – without allowing us to conclude malicious intent. Overall, we did not observe a systematic preference for URLs that appear more sensitive, suggesting that opportunistic collectors may not yet discriminate based on URL structure.

Further, we find that for each service, more than one-third of the visits originate from an IP flagged as potentially suspicious by the AlienVault OTX API. We observed that 180 distinct IPs visited our honeylinks from at least two services, 65 of which were classified as suspicious. While AlienVault flags do not confirm malicious intent, they suggest that some visitors were previously associated with scanning or reconnaissance activity. Although we cannot draw a definitive conclusion about the reasoning behind these visits, it is worth noting that suspicious actors may already be consolidating sensitive URLs from different URL scanning services.

We also observed differential behavior among the clients used by our visitors, through the User-Agent header. These range from single visits to the provided URL without loading additional resources (e.g., when using curl), to loading of the linked images (like a browser would), through visiting linked resources (like a crawler), and even submitting forms or clicking on buttons (either like a human analyst would or a bot trying to impersonate one).

Considering the 2,710 visits that occurred globally across all of our submitted URLs, 977 were full-asset loads, i.e., they also fetched both embedded images. Of these, 187 show signs of interaction; either by submitting our form,

	Plain	Sensitive-login	Entropy-path	Entropy-query	Total
All IPs					
Anyrun	62 (*26*)	50 (*20*)	53 (*22*)	52 (*21*)	217 (*66*)
Cloudflare Radar	26 (*10*)	20 (*9*)	33 (*16*)	31 (*19*)	110 (*43*)
Joe Sandbox	54 (*25*)	45 (*25*)	47 (*23*)	61 (*31*)	207 (*66*)
Hybrid Analysis	300 (*174*)	283 (*162*)	306 (*160*)	314 (*174*)	1,203 (*459*)
URLQuery	19 (*9*)	20 (*10*)	16 (*7*)	38 (*22*)	93 (*30*)
URLScan	254 (*174*)	199 (*125*)	247 (*179*)	180 (*109*)	880 (*417*)
Total	715 (*418*)	617 (*351*)	702 (*407*)	676 (*376*)	2,710 (*1,081*)
Only suspicious IPs					
Anyrun	33 (*11*)	31 (*9*)	33 (*10*)	28 (*9*)	125 (*27*)
Cloudflare Radar	13 (*4*)	4 (*2*)	8 (*3*)	10 (*6*)	35 (*11*)
Joe Sandbox	19 (*7*)	11 (*6*)	8 (*5*)	20 (*11*)	58 (*20*)
Hybrid Analysis	121 (*63*)	105 (*53*)	121 (*54*)	105 (*52*)	452 (*129*)
URLQuery	14 (*5*)	12 (*3*)	11 (*3*)	11 (*4*)	48 (*7*)
URLScan	82 (*55*)	73 (*37*)	85 (*51*)	63 (*32*)	303 (*110*)
Total	282 (*145*)	236 (*110*)	266 (*126*)	237 (*114*)	1,021 (*304*)

TABLE 4: Access overview for honeypage experiment, showing the number of visits per class of honeylinks and service, along with the number of *unique IPs in parentheses*

clicking a button, or selecting an item from a dropdown which uses JavaScript to navigate the browser. In addition, 79 visits interacted with the QR code, i.e., they visited the URL linked in it, highlighting that they attempt to follow links not embedded in the HTML or rendered in the DOM. Orthogonally, 1,245 visits followed the links provided statically in our page. Since this number exceeds full-asset loads, it suggests that many crawlers operate without rendering the page and instead follow links by directly parsing the raw HTML. This leads us to believe that these are not human visitors, but rather automated visits.

Finally, 79 visits targeted resources that were neither linked statically nor dynamically added. This included attempts to download the folder names as ZIP files (e.g., `/news.zip`), in addition to probes for well-known but non-existent internal paths such as `/.aws/config`. While we cannot definitively determine the intent behind these requests, the patterns closely resemble known reconnaissance behavior, such as scanning for misconfigured backups [27].

As mentioned earlier, we utilized *CanaryTokens* to mimic secret tokens, to subsequently capture advanced actions and behaviors of visitors to our honeypages. Overall, we observed a total of 33 Canarytokens triggers across the deployed tokens. These included 8 triggered Web requests (via Web bugs), 14 opened decoy documents, and 11 accessed network folders. We did not record any accesses for the other types of deployed Canarytokens. Since we reused the tokens across multiple submitted URLs, we cannot attribute individual triggers to specific honeylinks. However, the accesses originated from a diverse set of IP addresses and geographic locations, suggesting that multiple independent entities engaged with our embedded tokens. These triggers reflect a conservative lower bound on meaningful interaction, indicating that at least some parties actively processed the page contents or opened documents, going beyond basic crawling.

In conclusion, we observe that URLs in public feeds are frequently visited by numerous IP addresses and various behaviors (from simple requests for HTML to interactive crawlers). Our experiment did not reveal an overtly consistent preference for certain URL patterns over others. From potentially suspicious IPs accessing our honeypages to the attempts to download non-existent files, there are indications that some actors flagged as potentially suspicious and exhibiting exploratory behavior may be actively monitoring URL scanning services.

7. Discussion

We now discuss the limitations of our analysis, our responsible disclosure process, insights into the causes of leaked links, and possible countermeasures to the identified problems.

7.1. Limitations

LeakyLinks uses a large language model (LLM) for SPI detection. While this LLM enables automated detection at scale, it also remains susceptible to both false positives and false negatives. As we show in Section 5.3, manual validation on samples yields a precision of 97.0% for pages flagged as containing SPI and 99.5% for pages labeled as non-SPI (with tight 95% Wilson score intervals). Consequently, a small number of false positives and false negatives remains, and the SPI counts in this paper should be interpreted as conservative lower bounds.

LeakyLinks does not simulate user interactions beyond directly loading a page. This constraint, motivated by both ethical considerations and safety concerns, ensures that we do not trigger unintended actions on behalf of users, for instance resetting their password or sending an email. As a result, SPI that requires clicks, closing banners, or other interactive behavior could not be detected.

Another limitation is that we run the LLM locally and offline, which has lower capacity than commercial offerings. We chose this design for ethics and privacy: our analysis processes privacy-sensitive users' artifacts, and we avoid disclosing them to third parties. While this may limit model capacity, it is a deliberate trade-off that prioritizes data protection.

7.2. Responsible Disclosure

Many of our findings reveal critical and sensitive information. As outlined in our ethical considerations (see Appendix A), we carefully designed our experiments and adhered to the principles of responsible disclosure. We defined our disclosure strategy prior to starting our experiments to ensure the ethical handling of our data.

We decided to notify website owners about sensitive URLs discovered by LEAKYLINKS related to their domains. For each affected domain, we send a single email to standard administrative contacts: *webmaster@domain.com*, *security@domain.com*, and *info@domain.com*. If a *security.txt* file was present [15], we follow the specified contact instructions. Our notifications include all findings, even when some may be false positives. We do not manually verify each case to avoid introducing privacy risks or bias. Nonetheless, we consider it our responsibility to inform recipients of any potentially sensitive information that may have been exposed. In addition to notifying website owners, we also decided to inform the URL scanning service providers selected in Section 4, on which we have discovered the exposed URLs. We contact these service providers directly to report observed leaks and offer details that may help improve their visibility settings, filtering, or data retention policies. We do not attempt to contact affected individuals whose data may be exposed (e.g., email addresses in document links or tracking pages), as doing so would be impractical and could violate privacy expectations. Our focus remains on notifying responsible parties in a position to mitigate the exposure.

We already applied the above disclosure strategy to our preliminary pilot experiment, which we ran from October 2024 to April 2025. That is, we focused only on two URL scanning services in our preliminary experiments: URLScan and Cloudflare Radar, and notified the affected website owners of any identified URLs leaking sensitive content. In total, we sent several notifications to website owners, as well as separate disclosures to URLScan and Cloudflare Radar. Notably, after disclosing to Cloudflare, we observed a significant drop in URLs flagged as non-base domains in our main experiment: from around 20.1% in this pilot experiment to merely 1.7% now. We believe the sudden drop of sensitive URLs to be correlated with our disclosure to corresponding services.

At the time of writing, we are in the process of disclosing the findings affecting a broader set of services and domains from the main experiment. Although the results of this later disclosure are not reflected in the current version of the paper, we can already report on the positive feedback received from the pilot study disclosures. Recipients acknowledged the issue, took swift action to reduce exposure, sought further context, and contacted scanning services directly to request takedowns or implement blocking. One recipient shared: *"Personally, it has proved there are some good people out there on the internet. Please keep up the good work."* (quoted with consent).

We also received a detailed reply from URLScan. They outlined existing mitigation options, such as domain-based blocklists, user reporting features, and syntactic filters for URL paths. However, they acknowledged that these measures were insufficient to prevent all leakage. Planned improvements include automated sensitive URL detection methods, similar to what we present in LEAKYLINKS.

We expect similar responses from disclosure for the main experiment and plan to update the paper accordingly.

7.3. Causes for Leaked Links

Naturally, only the respective services can know how URLs were submitted to them. Even if the services offer APIs for programmatic interaction, it is not readily obvious how the URLs were submitted. Nevertheless, our insights into the nature of many links suggest that the links are sent to the services through email providers, which check incoming mail for dangerous links. These may rely on the APIs, but with incorrect settings or by using non-premium versions to scan links in incoming mail.

For example, URLs may be leaked indirectly in enterprise environments through large-scale integrations. For instance, services like URLScan offer built-in integrations with security orchestration platforms (SOARs) and endpoint detection tools (EDRs) [37]. Organizations can automatically submit URLs extracted from emails or user activity for analysis, typically as part of phishing defense or incident response workflows. In such setups, every inbound email or suspicious URL an employee clicks can trigger a scan, often without user awareness. While these workflows improve security, misconfigured settings (e.g., public scan visibility) can unintentionally expose sensitive URLs at scale.

In particular, we found many cases where the links either had a substring like *mail* or *click* in them or pointed to domains known to be used during initial email filtering (such as safelinks.protection.outlook.com). Naturally, it may also be that a user themselves submitted the links to the service manually, so we cannot offer a definitive conclusion to the observed phenomenon.

7.4. Countermeasures

Prior blog posts on URL-scanning data leaks have already put forward recommendations for service operators and website owners, such as expiring passwords reset links quickly, redacting email addresses on unsubscribe pages, requiring additional information (e.g., a ZIP code) before revealing full delivery details, and avoiding API keys in URLs in favor of HTTP headers [6, 38]. Our measurement confirms that these issues arise at scale and motivates additional countermeasures from the scanning services as well.

From a purely technical perspective, mitigating the reported issues is relatively straightforward, and several viable avenues for improvement are available. As a radical approach, URL scanning services could make the visibility of submitted URLs private by default, e.g., by restricting access to the scan results exclusively to the original submitter. However, such a measure would likely affect the underlying business model of these services.

A more targeted alternative could involve limiting the visibility of URLs likely to contain sensitive information. Scanning services can employ a variety of techniques to identify such sensitive URLs. For instance, URLScan noted in our exchange that they use generic filters to exclude URLs containing keywords like "unsubscribe" in their path from being scanned. However, our experiments demonstrated that these filters are not consistently effective.

A more robust and comprehensive analysis pipeline, such as the one proposed in LeakyLinks, could automatically flag URLs whose rendered content contains SPI and treat them as private by default. In our snapshot, even if all URLs that our extrapolation suggests contain SPI (5,475 out of 2,286,501 unique URLs) were hidden from public feeds, this would reduce visible volume by only about 0.24%, leaving more than 99% of URLs available for threat research, debugging, and other benign uses. Such a policy would therefore impose minimal cost on the utility or business model of scanning services, while substantially reducing the exposure of sensitive content to adversaries monitoring public feeds.

8. Related Work

Security and privacy experts from the industry have discussed specific threats associated with URL scanning services in the recent past. Motivated by the unintended information leakage of private GitHub repositories by URLScan in 2022 [9], Bräunlein [6] presented preliminary evidence that scanned URLs could expose additional sensitive endpoints such as Dropbox file transfer links, account creation links, password reset links, etc. Concurrently, Tinder Security Labs [34] also performed similar investigations. More recently, Vin01 [40] demonstrated that this issue extends across three URL scanning services due to their inherent design choices — public indexing of scanned URLs. This work raises awareness of the risks associated with URL scanning services, however, it does not provide a comprehensive account of the threat landscape as it is based on manual dorks that are used to extract sensitive links pointing to selected services. In this study, we develop an automated analysis pipeline that does not rely on known vulnerability patterns, and we report on a large-scale measurement encompassing hundreds of thousands of URLs from six archives, thus providing the most systematic study of the issue to date. Our pipeline can be used not just as a measurement tool, but also to identify unanticipated SPI leakage without relying on dork enumeration, thus becoming a potentially valuable tool for URL scanning services as well.

We are not aware of any published papers on the security and privacy risks of URL scanners, but the dangers of other public data sources have been investigated by the community. An independent investigation reported over 12k live session secrets spanning across 2.7M Web pages, publicly accessible through the Common Crawl archive [35]. On similar lines, El Yadmani et al. [14] also investigated the misconfigurations among publicly accessible cloud buckets and detected over 215 instances where sensitive credentials were exposed to the Web. Another study by Matic et al. [24] detected sensitive URLs on the Web by leveraging a corpus of URLs from the *Common Crawl* project. They performed ML-driven classification of URLs based on their content, combined with pre-defined keywords and categories from the *Curlie.org project*, with an accuracy of 90%.

Other related work focuses on analyzing PII leakage across the Web. Senol et al. [32] investigated the prevalence of credentials leak through login forms on Top 100,000 websites and reported that over 1.8k websites in the EU and 2.9k in the USA include analytics and tracker domains that exfiltrate the sensitive PIIs from these forms. Cui et al. [12] analyzed the Web forms on 11.5k websites. They compared their data collection strategy to their privacy policies and reported an apparent inconsistency between the data collection practices and the privacy policy disclosures among websites. Reaves et al. [29] performed a large-scale analysis on the security posture of publicly accessible SMS gateways and raised concerns over the plethora of sensitive PIIs that they expose, including but not limited to financial details, emails, and password reset links. Kaspereit et al. [20] developed *LanDscAPe* to analyze the security misconfigurations within LDAP servers on the Internet. They report a series of issues on these servers from their analysis, where 4.9k of them exposed different classes of PIIs, including passwords.

Although these studies reported PII leakage across different services, no academic research has investigated the risks associated with URL scanning services, even in light of their increasing popularity. Building on this line of work, we focus on SPI, i.e., the non-public subset of PII defined in Section 3.2, and close this gap by performing large-scale measurements to determine the extent of SPI leakage among the most popular URL scanning services.

9. Conclusion

URL scanning services aim to provide users with security and privacy protection by detecting malicious content, such as malware or phishing pages. In our work, we investigated the security and privacy *harms* which originate from such services through the public exposure of scanned URLs. With the help of LeakyLinks, a fully automated pipeline for analyzing URL scanning services' feeds regarding potential leaking URLs, we conducted an in-depth analysis of the live feeds of six primary URL scanning services.

Our analysis highlights that LEAKYLINKS has high precision in identifying leaking URLs, which enabled us to comprehensively study the ecosystem of URL scanning services by analyzing over 300k URLs. Doing so, LEAKYLINKS identified 4,417 URLs exposing SPI with 97% precision. Considering our design choice to avoid false positives and our experiments with the precision for negative labels, this number represents a conservative lower bound of the problem in the wild. Our honeylink experiments further highlighted that URLs accessible through the public feeds of URL scanning services are visited by automated, sometimes highly interactive, bots, underscoring the possibility that actors may already be exploiting leaked links.

These findings highlight a systemic problem in the design and deployment of URL scanning services, where well-intentioned automation can inadvertently compromise user privacy and security. Addressing these risks requires technical safeguards, careful business considerations, and broader awareness among developers, service providers, and end users.

10. Acknowledgment

We thank our anonymous shepherd and the reviewers for their valuable feedback.

11. Availability

LEAKYLINKS is fully open source and is publicly available at https://github.com/cispa/leakylinks

References

[1] *ANY.RUN: Interactive Malware Analysis Sandbox*. URL: https://app.any.run/.

[2] AT&T Cybersecurity. *AlienVault Open Threat Exchange (OTX)*. URL: https://otx.alienvault.com.

[3] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. "Qwen technical report". In: *arXiv preprint arXiv:2309.16609* (2023).

[4] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. "The Menlo Report". In: *IEEE Security & Privacy* 2 (2012). DOI: 10.1109/MSP.2012.52.

[5] Bolster AI. *How URL Scanners Can Mitigate the Risks of Malicious Websites*. 2024. URL: https://bolster.ai/glossary/how-url-scanners-can-mitigate-the-risks-of-malicious-websites.

[6] Fabian Bräunlein. *urlscan.io's SOAR Spot: Chatty Security Tools Leaking Private Data*. 2022. URL: https://positive.security/blog/urlscan-data-leaks.

[7] Lawrence D Brown, T Tony Cai, and Anirban DasGupta. "Interval estimation for a binomial proportion". In: *Statistical science* 2 (2001). DOI: 10.1214/ss/1009213286.

[8] Euijin Choo, Mohamed Nabeel, Doowon Kim, Ravindu De Silva, Ting Yu, and Issa Khalil. "A Large Scale Study and Classification of VirusTotal Reports on Phishing and Malware URLs". In: *ACM on Measurement and Analysis of Computing Systems* (2024). DOI: 10.1145/3673660.3655042.

[9] cillian64. *Tell HN: GitHub Leaked Names of Private Repos with Pages*. 2022. URL: https://news.ycombinator.com/item?id=30348980.

[10] *Cloudflare Radar: Internet traffic and trends*. URL: https://radar.cloudflare.com.

[11] *Cloudflare's URL Scanner, new features, and the story of how we built it*. 2024. URL: https://blog.cloudflare.com/building-urlscanner.

[12] Hao Cui, Rahmadi Trimananda, and Athina Markopoulou. "Understanding Privacy Norms through Web Forms". In: *PETS*. 2025. DOI: 10.56553/popets-2025-0002.

[13] Brittany Day. *Email Security Intelligence - Understanding Malicious URL Protection - and Why You Need It to Secure Your Email*. 2021. URL: https://guardiandigital.com/resources/blog/understanding-malicious-url-protection-and-why-you-need-it-to-secure-your-email.

[14] Soufian El Yadmani, Olga Gadyatskaya, and Yury Zhauniarovich. "The File That Contained the Keys Has Been Removed: An Empirical Analysis of Secret Leaks in Cloud Buckets and Responsible Disclosure Outcomes". In: *IEEE S&P*. 2025. DOI: 10.1109/SP61157.2025.00009.

[15] Edwin Foudil and Yakov Shafranovich. *A File Format to Aid in Security Vulnerability Disclosure*. RFC 9116. Internet Society, 2022.

[16] John Franks, Phillip Hallam-Baker, Jeffrey L. Hostetler, Scott Lawrence, Paul J. Leach, Ari Luotonen, and Lawrence Stewart. *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617. Internet Society, 1999.

[17] Gil Friedrich. *Email Security: URL Scanning beyond the Basics*. 2021. URL: https://cybersecurityventures.com/email-security-url-scanning-beyond-the-basics/.

[18] *Hybrid Analysis: Free malware analysis and threat intelligence sandbox*. URL: https://hybrid-analysis.com/.

[19] *Joe Sandbox Cloud: Deep malware and phishing analysis sandbox*. URL: https://www.joesecurity.org/joe-sandbox-cloud.

[20] Jonas Kaspereit, Gurur Öndarö, Gustavo Luvizotto Cesar, Simon Ebbers, Fabian Ising, Christoph Saatjohann, Mattijs Jonker, Ralph Holz, and Sebastian Schinzel. "LanDscAPe: Exploring LDAP Weaknesses and Data Leaks at Internet Scale". In: *USENIX Security*. 2024.

[21] Brian Kondracki, Johnny So, and Nick Nikiforakis. "Uninvited guests: Analyzing the identity and behavior of certificate transparency bots". In: *USENIX Security*. 2022.

[22] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. “Quantifying the Carbon Emissions of Machine Learning”. In: *arXiv preprint arXiv:1910.09700* (2019).
[23] Yupei Liu, Yuqi Jia, Jinyuan Jia, and Neil Zhenqiang Gong. “Evaluating LLM-based Personal Information Extraction and Countermeasures”. In: *USENIX Security*. 2025.
[24] Srdjan Matic, Costas Iordanou, Georgios Smaragdakis, and Nikolaos Laoutaris. “Identifying Sensitive URLs at Web-Scale”. In: *ACM IMC*. 2020. DOI: 10.1145/3419394.3423653.
[25] Ali Mustafa. *LeakyLinks Source Code*. URL: https://github.com/cispa/leakylinks.
[26] Charlie Osborne. *URLScan API Unwittingly Leaks Sensitive URLs, Data*. 2022. URL: https://portswigger.net/daily-swig/urlscan-io-api-unwittingly-leaks-sensitive-urls-data.
[27] OWASP Foundation. *Test File Extensions Handling for Sensitive Information (WSTG-CONF-03)*.
[28] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. “Opening the Blackbox of VirusTotal: Analyzing Online Phishing Scan Engines”. In: *ACM IMC*. 2019. DOI: 10.1145/3355369.3355585.
[29] Bradley Reaves, Nolen Scaife, Dave Tian, Logan Blue, Patrick Traynor, and Kevin RB Butler. “Sending out an SMS: Characterizing the Security of the SMS Ecosystem with Public Gateways”. In: *IEEE S&P*. 2016. DOI: 10.1109/SP.2016.28.
[30] Shachar Roitman, Ohad Benyamin Maimon, and William Gamazo. *Effective Phishing Campaign Targeting European Companies and Organizations*. URL: https://unit42.paloaltonetworks.com/european-phishing-campaign/.
[31] Sebastian Roth, Stefano Calzavara, Moritz Wilhelm, Alvise Rabitti, and Ben Stock. “The Security Lottery: Measuring Client-Side Web Security Inconsistencies”. In: *USENIX Security*. 2022.
[32] Asuman Senol, Gunes Acar, Mathias Humbert, and Frederik Zuiderveen Borgesius. “Leaky Forms: A Study of Email and Password Exfiltration Before Form Submission”. In: *USENIX Security*. 2022.
[33] Thinkst Canary. *Canary Tokens*. URL: https://canarytokens.org.
[34] Tinder Security Labs. *How to Categorize and Prevent Risks of Sensitive Links in Urlscan*. 2022. URL: https://medium.com/tinder/how-to-categorize-and-prevent-risks-of-sensitive-links-in-urlscan-d6cd0a58b0da.
[35] Truffle Security. *Research Finds 12,000 ‘Live’ API Keys and Passwords in DeepSeek’s Training Data*. 2025. URL: https://trufflesecurity.com/blog/research-finds-12-000-live-api-keys-and-passwords-in-deepseek-s-training-data.
[36] *urlquery.net: URL and domain scanning service*. URL: https://urlquery.net/.
[37] *URLScan API Integrations*. 2025. URL: https://urlscan.io/docs/integrations/.
[38] *URLScan Scan Visibility Best Practices*. 2022. URL: https://urlscan.io/blog/2022/07/27/scan-visibility-best-practices/.
[39] *urlscan.io: A sandbox for the web*. URL: https://urlscan.io.
[40] Vin01. *You Can Not Simply Publicly Access Private Secure Links, Can You?* URL: https://vin01.github.io/piptagole/security-tools/soar/urlscan/hybrid-analysis/data-leaks/urlscan.io/cloudflare-radar%22/2024/03/07/url-database-leaks-private-urls.html.
[41] VirusTotal. *VirusTotal: An online service for analyzing files and URLs using multiple antivirus engines and threat blacklists*. URL: https://www.virustotal.com.
[42] Jack Zalesskiy. *Threat Intelligence Feeds are powered by approximately 14,000 daily public sessions from over 300,000 security researchers*. 2023. URL: https://any.run/cybersecurity-blog/threat-intelligence-feeds.

Appendix A. Ethics Considerations

We expand our discussion on the ethical considerations for this study and detail on each stakeholder, their risk, and their benefits.

Affected Individuals. The general public, or affected individuals, use online services and expect privacy, yet their data may be leaked via Web archives or URL scanners. Furthermore, leaked authentication tokens could allow account interaction in the context of the affected individual. Our project risks analyzing sensitive information and interacting with sessions without consent and unintentionally creating a dataset of leaked data. To mitigate this, our crawler never performs any interaction on a webpage other than visiting the link. We also minimize manual analysis, never identify individuals, and commit to a strict self-signed ethical commitment. Data is securely stored on our secure servers, accessible only for authorized team members, and encrypted after the experiments. Our research highlights privacy risks (that are potentially already exploited by bad actors, see 6.2), hopefully motivating deletions and improvements of safeguards. Lastly, our honeypage testing does not impact individuals, as these pages are never publicly indexed or advertised.

Website Owners. Website owners, such as Google or Dropbox, store sensitive user data and reputational and regulatory interests in privacy. Their sites may be unknowingly archived or scanned, posing a risk of data leaks. To prevent reputational harm, we decided not to name specific companies but to categorize Web services instead. This research provides valuable insights for website owners, raising awareness of data leakage and providing recommendations on how to improve configurations to enhance privacy. Additionally, we disclosed the leakages to them. For our honeypage experiment, we only used our own domains, except for *canary-*

tokens.org, which aligns with its intended purpose, posing no expected negative consequences for website owners.

Product Owners. Product owners develop tools that interact with Web archives or URL scanners, aiming to enhance security or user experiences. However, poor implementation may cause data leaks. Our study could harm the product's reputation if named. To mitigate this, we responsibly disclosed all our findings to the vendors so that they can fix the issues before publication and benefit from improved security and privacy. The honeypage experiment is expected to have no impact on product owners.

Service Providers. Another group is service providers, such as Web archives and URL scanners, that store and share Web content and potentially expose sensitive user data. Our findings could harm their reputation and lead to legal consequences if they violate privacy laws (e.g., GDPR). In the worst case, these providers might not be able to afford this, potentially leading to the permanent shutdown of their services. To prevent such implications, we need to anonymize their names in our publication. However, anonymizing providers in our publication limits external pressure for improvements, reduces transparency for affected users, and hinders research replication. We believe the risk of lawsuits or shutdowns is low, while the benefits of naming providers and giving them a fair chance, along with responsible disclosure, outweigh the risks. Our honeypage experiment is expected to have minimal impact on their resources as they typically process hundreds of pages per hour.

Service Observers. Service observers monitor archives and scanners, seeking content for malicious (e.g., data brokers) or benign (e.g., researchers) purposes. Our study aims to reduce sensitive data leaks, negatively impacting those seeking such content, an ethically acceptable outcome. Observers may also be affected by a shutdown of the services, which we believe to be unlikely (see above). Furthermore, the honeypages experiment might lead to a slight increase in their resource usage or affect the dataset used by researchers. Since we only submit a very small number of pages as described in Section 6.1, we do not assume a significant negative effect.

Researchers. As a research team, we are also stakeholders, aiming to raise awareness and reduce data leaks. We face potential legal risks that we considered carefully before starting the project, e.g., GDPR Article 89 exempts research from certain privacy restrictions. As mentioned earlier, we could also be impacted by service shutdowns; however, we also benefit from contributing to academia and improving privacy protections. The honeypage experiment poses minimal risk, only affecting our resources.

Appendix B. LLM usage considerations

In accordance with the new LLM Policy at IEEE S&P, we describe and explain the usage of LLM in preparing this paper and within our project.

Throughout the paper writing process, we used ChatGPT and Grammarly LLMs for improving clarity and correctness of our writing, such as for grammar checks, typo correction, and suggested phrasing. At no point did we use an LLM to generate whole paragraphs or sections. All AI-assisted edits were carefully reviewed for accuracy and were manually integrated into the manuscript rather than copied entirely.

In the project itself, we leverage a local instance of LLM in our analysis pipeline. We did so to ensure scalability for processing large volumes of data and, more importantly, the privacy of individuals whose data may be involved (see Appendix A). Since we view the manual review of potentially sensitive content at scale as privacy-invasive, we rely on a *local* LLM to perform the necessary categorization.

While we use an LLM for our pipeline, we reevaluated its role throughout the development of our framework. In our pilot study, a larger part of the pipeline was using LLM-assistance, yet we decided to move toward a more heuristic-based filter later on. This was mainly driven by two reasons: first, the results were not better than those from a heuristic-based approach in these specific parts of the pipeline, and second, we aimed to reduce unnecessary resource consumption and environmental impact when comparably good alternatives existed.

For transparency, we estimated our carbon footprint using the Machine Learning Impact calculator presented in [22]. Experiments were conducted on private infrastructure. In total, 50 hours of computation were performed on eight NVIDIA A100 PCIe 40 GPUs (TDP 250 W). Total emissions are estimated to be 56 kg CO_2eq, which corresponds to driving 176 km in an average ICE car.

Besides the environmental considerations, the use of LLMs can also introduce irreproducibility into results. To minimize this random factor, we configured the model with a zero temperature and a static random seed, i.e., the randomness factor, to ensure a near-deterministic behavior. Repeating the LLM step of our pipeline 10 times on the manually verified sample results in a precision of 97%, indicating a stable behavior. While we do not publish our dataset itself due to its sensitive nature (see Appendix A), we make our code open-source in [25]. We are confident that running the same experiment with similar input data will yield a very similar outcome.

Appendix C. Routing Invariance and Token Distribution

Token length distribution. For each URL where we detected SPI, we measured the length of its *shortest high-entropy token*. Figure 3 shows the percentage of SPI URLs as a function of this minimum token length. The largest share (close to 30%) have a shortest token of length 8, and in total more than half of SPI URLs have a shortest token of length at most 10. This motivates our choice of $L = 8$ as a permissive baseline cutoff, because it captures the large cluster of SPI URLs that rely on short tokens.

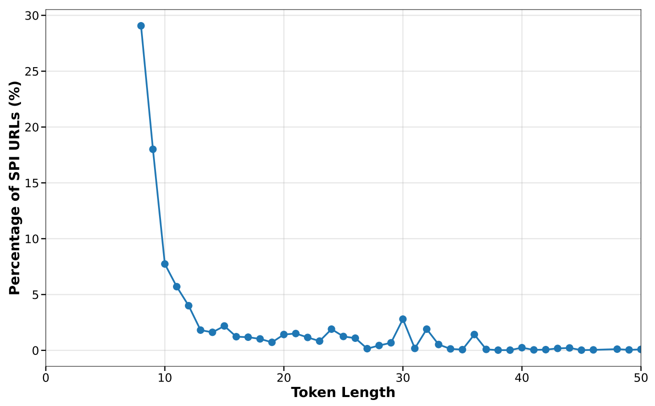


Figure 3: Percentage of SPI URLs versus minimum token length.

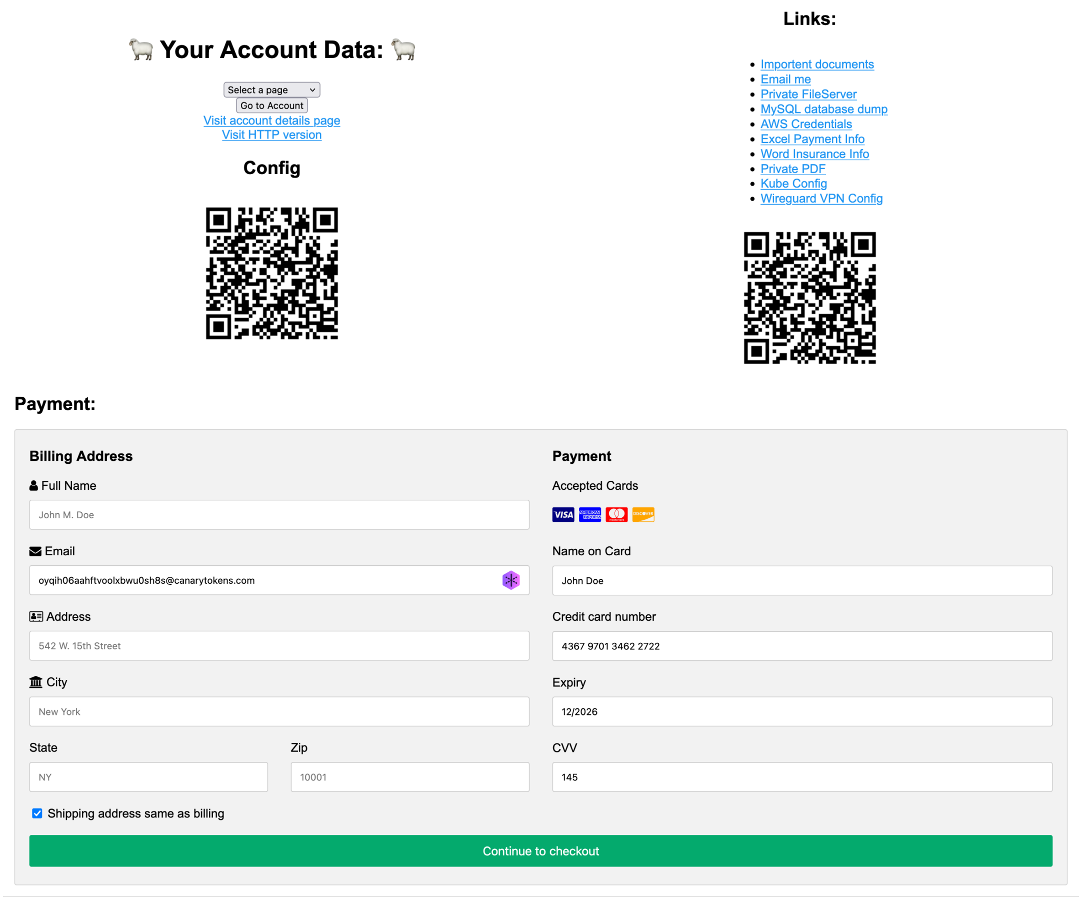
Routing invariance. Our detector sends each SPI URL to one of two branches: a *token* branch (when a high-entropy path/query token is visible) and a *no-token* branch (otherwise). Routing depends on two tokenization parameters: the minimum token length L and the per-character entropy threshold H. To check that our reported SPI total is not an artifact of a single choice of (L, H), we re-ran the detector on the same $4{,}417$ positives for

$$L \in \{6, 8, 10, 12, 16, 20, 24, 28, 32\},$$
$$H \in \{1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0\}.$$

and compared each run to the baseline $(L, H) = (8, 2.0)$. At the baseline, $4{,}387$ URLs are handled by the token branch and 30 by the no-token branch (all $4{,}417$ counted). For nearby settings (e.g., $L = 10$ or 12 at $H = 2.0$, or $L = 8, H = 2.5$), the final SPI count changes by only about 0.3%–3.9%. For moderately stricter parameters, the SPI count drops by roughly 8%–12%, and for very strict settings by up to about 28%–38%, because these thresholds start excluding the short high-entropy tokens that Figure 3 shows are common and route many SPI URLs to the no-token branch where they would be missed. We therefore treat such strict configurations as stress tests rather than realistic operating points, and keep the baseline deliberately permissive to avoid hiding real exposures.

Appendix D.
Honeypage Screenshot

This section presents the screenshot of our honeypage (Figure 4) which we slightly modified for presentation in the paper. We remark that the honeypage was designed to attract automated visit from crawlers and it is clearly not intended to phish humans.



🐑 Your Account Data: 🐑

Visit account details page
Visit HTTP version

Config

Links:

- Importent documents
- Email me
- Private FileServer
- MySQL database dump
- AWS Credentials
- Excel Payment Info
- Word Insurance Info
- Private PDF
- Kube Config
- Wireguard VPN Config

Payment:

Billing Address

Full Name: John M. Doe

Email: oyqih06aahftvoolxbwu0sh8s@canarytokens.com

Address: 542 W. 15th Street

City: New York

State: NY

Zip: 10001

[✓] Shipping address same as billing

Payment

Accepted Cards

Name on Card: John Doe

Credit card number: 4367 9701 3462 2722

Expiry: 12/2026

CVV: 145

Figure 4: Screenshot of our honeypage intended for automated scanners.

Appendix E.
Prompt for Screenshot-Based SPI Classification

```
You are a security triage model analyzing a SCREENSHOT IMAGE ONLY. Decide whether the visible
 content is SENSITIVE PERSONAL INFORMATION. That is private information related to a person
that should only be accessible by the owner of that data, not publicly accessible to anyone.

FIRST: Identify the page type. If this is clearly general/public content (news, blog, article
, forum, product page, marketing, documentation, legal disclosure, privacy policy, bot block
page, captcha, phishing block), return false. Dating profiles, user profiles, and social
media profiles are typically public and should return false unless they contain private
information that should not be accessible to anyone.

SECOND: If you think you see something sensitive, STOP and check:
- Can I quote an exact visible substring from the image that proves sensitivity?
- Is this DISPLAYED personal data vs. a form merely ASKING for data (labels, instructions,
examples)?
```

```
- Is this a PLACEHOLDER vs REAL user data? (See placeholder rules below)
- What is the page type?
- Could this be an unsubscribe page, staff directory, or public contact info?
- Is this a social media profile page? Public profiles are NOT sensitive.
- Is this a user profile page showing only username, user ID (UID), member ID, or similar
public identifiers? These are NOT sensitive - user profiles with only public identifiers
should return false.
If you have ANY doubt, return false. Prefer false negatives over false positives. Use only
what is visible; do not infer.

CRITICAL RULES:
- Form FIELD LABELS and INSTRUCTIONS are NOT data. Labels like "First Name", "Last Name", "
Email", or any instructional text are NOT sensitive - they are telling the user what to enter
, not showing actual data.
- Generic placeholder patterns in ANY form field (even if styled normally) are NOT sensitive.
 These include Email placeholders: ANY email containing "example.com", "test.com", "demo.com
", "sample.com".
- Any data appearing alongside form labels/instructions in a way that suggests it's an
example or template is a placeholder.
- Placeholders can appear in NORMAL text styling (not just dimmed gray) - always check if the
 content matches placeholder patterns regardless of visual appearance.
- You must quote at least one exact VISIBLE substring in "quoted_evidence" that proves
sensitivity (email, name+address, order/invoice ID tied to a person, token, etc.).
- Public-facing contact/press/event/RSVP blocks and department/staff directories are not
sensitive even if they show names/office phones.
- Social media/networking profiles showing publicly shared information (names, ages, cities,
bio, posts) are NOT sensitive.
- Do not infer hidden context. Only judge what is visibly shown in the screenshot.
- QR codes, barcodes without any visible personal identifier are not sensitive.
- Redacted/blurred artifacts are not evidence; ignore them for sensitivity.
- Full residential addresses together with a visible person's name count as evidence when
quoted.
- Visible government ID numbers or passport numbers that pass simple sanity (length/format)
checks count as evidence when quoted.
- Full bank account numbers or IBANs, and high-entropy secrets/tokens/API keys shown plainly
count as evidence when quoted.

Sensitive = true only if at least one of these appears (with quoted evidence from the
screenshot):
A) PREFILLED personal contact data (email/phone) in a filled input field within a login,
account, or settings page. The email/phone MUST be in a prefilled form field, NOT just
visible somewhere on the page. These MUST be real user data, NOT placeholders (see
placeholder rules above).
B) Account/settings pages showing real user information (name, email, address, etc.) - NOT
form labels, NOT placeholders, NOT business emails in public contexts.
C) Personal documents/files indicating private ownership shown in a personal/account context.
D) Transaction/account details tied to a person (order/invoice/ticket IDs together with a
person's identifier) in an account/transaction context.
E) Secrets or high-risk tokens in plain text (full payment card numbers that are not test
patterns, bank/IBAN numbers, API keys).
F) Visible personal emails (non-role, non-business, non-placeholder) in unsubscribe blocks or
 message headers (From/To/Cc/Bcc) where the personal address is plainly shown.

YOU MUST OUTPUT ONLY VALID JSON MATCHING THIS EXACT SCHEMA. NO MARKDOWN, NO EXPLANATIONS.

Schema (all fields required):
{
  "primary_intent": string,
  "sensitive": boolean,
  "confidence": float (0.0 to 1.0),
  "spi_types": array of strings,
  "quoted_evidence": array of strings,
  "reasons": array of strings
}
```

Appendix F. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

F.1. Summary

This paper presents the first systematic study of Sensitive Personal Information (SPI) leakage through public URL scanning services. It develops an LLM-based, large-scale, and reusable framework to detect and measure SPI leakage across six services, and uses this framework to characterize the scope and nature of the problem. The paper also incorporates a honeypot deployment to study in-the-wild exploitation related to this form of leakage.

F.2. Scientific Contributions

- Independent confirmation of important results in an area with limited prior research.
- Creation of a new tool to enable future scientific study.
- Advancement on a long-known issue.
- A valuable step forward in an established research field.

F.3. Reasons for Acceptance

1) This paper provides a valuable step forward in an established field. While prior work has examined PII leakage in web content, studying SPI leakage specifically in the context of URL scanning services through measurement and characterization is a meaningful and interesting extension.
2) This paper creates a new tool to enable future science. The large-scale, LLM-based, reusable SPI detection framework provides a foundation for future studies on SPI leakage, including both measurement-oriented work and potential remediation efforts motivated by the findings of this paper.

F.4. Noteworthy Concerns

1) The PC was concerned that the paper offers limited insights into the attacks enabled by this issue beyond measuring SPI leakage in the domain of URL scanners.
2) The PC was concerned that the paper may overestimate the threat posed by SPI leakage and in-the-wild targeted exploitation, given the lack of validation experiments supporting these claims.
3) Multiple reviewers noted the lack of baselines comparing the proposed LLM-based leakage detection approach against more efficient and well-established alternatives.
4) Reviewers also raised concerns about the evaluation methodology, particularly the handling of class imbalance in the dataset, which may make the reported metrics sensitive to the chosen sampling strategy.