Out of Sight, Out of Mind: Detecting Orphaned Web Pages at Internet-Scale

Stijn Pletinckx
TU Delft
S.R.G.Pletinckx@student.tudelft.nl

Kevin Borgolte
Ruhr University Bochum
kevin.borgolte@rub.de

Tobias Fiebig
TU Delft
T.Fiebig@tudelft.nl

Abstract

Security misconfigurations and neglected updates commonly lead to systems being vulnerable. Especially in the context of websites, we often find pages that were *forgotten*, that is, they were left online after they served their purpose and never updated thereafter.

In this paper, we introduce new methodology to detect such forgotten or *orphaned* web pages. We combine historic data from the Internet Archive with active measurements to identify pages no longer reachable via a path from the index page, yet stay accessible through their specific URL. We show the efficacy of our approach and the real-world relevance of orphaned web-pages by applying it to a sample of 100,000 domains from the Tranco Top 1M.

Leveraging our methodology, we find 1,953 pages on 907 unique domains that are *orphaned*, some of which are 20 years old. Analyzing their security posture, we find that these pages are significantly ($p < 0.01$ using χ^2) more likely to be vulnerable to cross-site scripting (XSS) and SQL injection (SQLi) vulnerabilities than maintained pages. In fact, orphaned pages are almost ten times as likely to suffer from XSS (19.3%) than maintained pages from a random Internet crawl (2.0%), and maintained pages of websites with some orphans are almost three times as vulnerable (5.9%). Concerning SQLi, maintained pages on websites with some orphans are almost as vulnerable (9.5%) as orphans (10.8%), and both are significantly more likely to be vulnerable than other maintained pages (2.7%). Overall, we see a clear hierarchy: Orphaned pages are the most vulnerable, followed by maintained pages on websites with orphans, with fully maintained sites being least vulnerable.

We share an open source implementation of our methodology to enable the reproduction and application of our results in practice.

CCS Concepts

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

Keywords

Orphaned resources; web security; measurement;

ACM Reference Format:

Stijn Pletinckx, Kevin Borgolte, and Tobias Fiebig. 2021. Out of Sight, Out of Mind: Detecting Orphaned Web Pages at Internet-Scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3460120.3485367

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.

ACM ISBN 978-1-4503-8454-4/21/11.
https://doi.org/10.1145/3460120.3485367

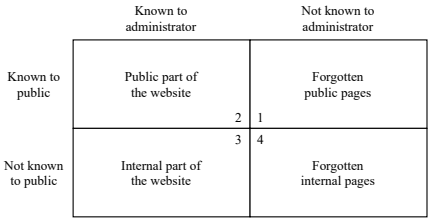


Figure 1: Matrix of known and unknown pages on a website. Columns differentiate between pages that are known, and pages that are not known by the administrator (called "administrator" for simplicity, this can be a team). Rows differentiate between the public's knowledge.

1 Introduction

The World Wide Web is an ever-changing landscape. When operating a website, keeping it updated is imperative for ensuring that it is free of bugs and vulnerabilities. However, keeping a website updated and secure is a cumbersome endeavor that is rarely achieved fully and often not at all [23]. Common causes are delaying or ignoring critical security updates [24] and other security misconfigurations, often rooted in human error [12, 32].

A recent example of a security misconfiguration that led to a compromise is Deloitte's "Test your Hacker IQ" campaign [7]. This advertisement campaign ran in 2015 and made use of a promotional website that remained online long after the actual campaign ended. In November 2020, an IT consultant discovered the old domain and managed to retrieve the database credentials.

This example highlights that administrators can lose track of the state of their website and lead to an old and unmaintained domain exposing data for years. Finding such no-longer-used *domains* is feasible by, for example, tracking certificate transparency logs to identify hosts that stop renewing their certificates but remain reachable, or by using passive DNS traces to identify domains that, over time, receive significantly less traffic. However, these techniques do not allow to identify URLs of abandoned content on a *single* domain but at a different path, like a discontinued web applications hosted at `example.com/web-app` while `example.com` remains actively being maintained and used. In this paper, we aim to shine light on this blind spot and we develop a methodology to identify orphaned URLs of single domains in the wild and at-scale.

We express what it means for pages (and the URL pointing to them) to be orphaned in terms of who knows about them (administrators vs. the public). This leads to four quadrants (Figure 1) that, much like a Johari window [26], characterize the pages of a website as known and unknown to the administrator and the public.

Most websites have pages that are intended for and known to the public. At the same time, some pages are usually not known

to the public or even intended to be accessed by the public, such as administrative interfaces, content management, but also internal pages, like a company's intranet. In general, administrators should "know" about both types of pages. However, they may become unaware of them and then these pages become orphaned. Eventually, these pages will be unmaintained, and, for example, may not receive security updates that address existing vulnerabilities. In our work, we identify these orphaned pages, with a specific focus on Quadrant 1 of Figure 1, that is, pages that were once intended for public access but have since been forgotten by the administrators.

One way how a page can become forgotten and unmaintained is by misconfiguration when removing a web page. Successful removal of a web page requires two steps: first, removing it from the webserver, and, second, making sure no other page on the website links to it. If only the second step is performed, then the page only appears to be removed because it is no longer accessible by navigating on the website.However, since the page itself has not been removed, it remains accessible by navigating to its URL directly. We classify these pages as "orphaned" pages (see Section 2).

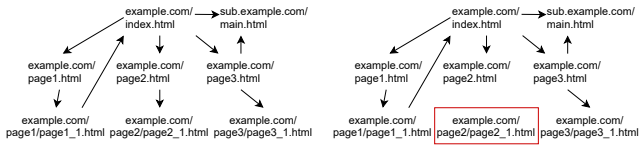
In this paper, we develop new methodology to detect orphaned web pages on a website. We make use of archived data to compare the current sitemap of a domain against its historic versions, and extract unlisted pages that are still accessible. We further scrutinize this list by filtering out purposely archived pages, and perform fingerprint comparisons and copyright checks to validate their unmaintained status. We evaluate our implementation through a large-scale measurement study, and confirm its efficacy for finding orphaned web pages in the wild. To investigate the security impact of orphaned web pages, we compare their security posture to non-orphaned pages on a variety of metrics.

In summary, we make the following contributions:

- We create the first methodology for detecting orphaned web pages on a *single* domain, only using public information.
- We perform the first large-scale detection of orphaned web pages in the wild, and report a lower bound on the prevalence of them. On a sample of 100,000 websites, we observe that at least 1,953 pages, spread over 907 domains, are orphaned, with some of the pages being as old as 20 years.
- We compare the security posture of orphaned pages to a control group of non-orphaned pages, and we find that orphaned pages are more prone to vulnerabilities. For example, we find 19.3% of the orphaned pages processing user input are vulnerable to cross-site scripting (XSS) attacks, which differs substantially to 2.0% of the maintained control group being vulnerable ($p < 0.01$ using χ^2).
- We share an open source implementation of our methodology to be used for further research, and by administrators and security professionals to audit websites.

2 Background

While we are, to the best of our knowledge, the first first to comprehensively study orphaned resources within the context of web pages, the general concept has been studied before (Section 7). Following, we briefly describe the necessary background.



(a) Site referenced **(b) Link removed**

Figure 2: Sitemap example for a website at `example.com`. When `/page2.html` stops linking to `/page2/page2_1.html`, then the latter becomes orphaned.

2.1 Definition of Orphaned Web Pages

We define an orphaned web page based on the sitemap of a website. Figure 2 shows a simplified example of such a sitemap graph: a starting node, called root or index (`example.com/index.html`), with edges (links) to children nodes (pages at `example.com/page1.html`, `example.com/page2.html`, etc. and further descendants. A page becomes orphaned when all links (edges) to a page (node) are removed, and no other page (node) links to that page (see Figure 2b, where the page at `example.com/page2/page2_1.html` has no inbound link). However, although there is no path from the root, the page remains accessible via its URL. That is, a page is orphaned if it cannot be reached through graph traversal from the website's entry points, with a typical entry point being the website's index page.

2.2 Types of Orphaned Web Pages

We can divide orphaned web pages into two categories: unmaintained orphaned pages and maintained orphaned pages.

2.2.1 Unmaintained Orphaned Web Pages Unmaintained orphaned web pages are those where the administrators are no longer aware of their existence (or do not care), and, consequently, do not apply (security) updates to the underlying application(s) providing the page. For example, this can happen if a team runs uses a website for a limited time (e.g., a product promotion or recruitment campaign, as the Deloitte example in Section 1 illustrated) or if the person responsible for maintaing a website leaves the company. With the lack of updates, the orphaned web page might become outdated, possibly making it prone to vulnerabilities over time.

Coming back to the quadrants of Figure 1, we can classify unmaintained orphan pages on the right side (quadrants 1 and 4). In quadrant 1, a user still knows the URL of the orphaned page, and can access it directly. This means that these web pages are detectable as their URL is known by *someone* or recorded *somewhere*, like, for example, in old messaging board post or in archival data.

Pages in quadrant 4 are much more challenging to detect. These pages are orphaned and unmaintained, but they were never intended to be used by the public. This means that their specific URL may never have been referenced on a publicly accessible site, or that the page is hidden behind (simple) authentication. In the former case, the page could theoretically be accessed if one could guess the URL or brute-force it, but this is no small feat and akin to searching for the needle in the haystack, when there might not even be a needle in the first place. Correspondingly, we focus on orphaned pages previously known to the public (quadrant 1), and we consider pages unknown to the public (quadrant 4) out of scope.

2.2.2 Maintained Orphaned Web Pages Web pages may also be orphaned without becoming unmaintained. This can be the case if a link is unintentionally removed. In this case, the orphaned page is unreachable if one only follows links from an entry point on the website. However, the administrator is still aware of the pages and maintains them along the same care as other pages. This is also the case if the orphaned page utilizes the same framework or content management system as the main site.

Maintained orphaned pages correspond to the left quadrants of Figure 1 (quadrants 2 and 3). These pages are generally less of an issue in terms of security vulnerabilities, although they can lead be inconvenient to users or cause confusion.

2.3 Security by Obscurity

In addition to accidentally orphaned pages, whether maintained or not, a page might become orphaned by design, with the intention of "protecting" the page against various attack vectors. Operators may chose to "hide" a page by removing links to it, excluding it from public search (using `robots.txt`) to reduce the attack surface, and only sharing it's URL with a select group of (trusted) people. This may happen for publicly available internal applications, or for pages for which a vulnerability was discovered. Although this might result in less exposure and less traffic to the vulnerable page, security by obscurity is known to be largely ineffective, and the site actually remains vulnerable to exploitation by adversaries. While one could consider these sites as "maintained" orphaned pages, we argue that if they are hidden with the goal of security by obscurity, and administrators are not actually applying other security updates, then they provide additional attack surface (unlike other maintained orphaned pages), and they are actually unmaintained.

2.4 Security Impact of Orphaned Pages

We expect that unmaintained orphaned web pages are more prone to vulnerabilities, due to delayed updates (or not receiving them at all) and pre-dating modern defenses. While the main infrastructure of the website is likely maintained, such as the underlying operating system or webserver (e.g., nginx or Apache), we see no reason to assume that administrators are patching vulnerabilities of orphaned web pages hosted on the infrastructure. This also holds true for server-side applications only used by the orphaned pages, as the administrator expects that the functionality is not reachable anyways, and, thus, they have no incentive to expend time and resources to apply security patches. Naturally, this differs for systems applying automatic updating, though these automatic updates may break over time or are only partially applied (e.g., configuration files are part of software packages on Linux distributions, but they are rarely updated by automatic updates, and misconfigurations and insecure configurations remain even with automatic updates enabled).

Moreover, a website administrator can easily build up a false sense of security. If a technology stack is changed because it used to be vulnerable, the administrator might consider the website safe from specific classes of vulnerabilities. For example, a website might migrate from pages generated dynamically on the server-side to statically-generated pages, with the dynamically-generated pages becoming unmaintained orphaned. The administrator may then think the website is secure from server-side code injection attacks or path traversal vulnerabilities because no code is being executed on the server-side anymore. However, as these pages have only been orphaned and not truly removed, this assumption would be a mistake. Consequently, an unmaintained orphaned web page can become the Achilles' heel of the entire website.

2.5 Threat Model

In this paper, we assume an external attacker with no special access to the target infrastructure or resources exceeding those of an average Internet user (broadband network connection, off-the-shelf PC, no special threat intelligence feeds). Hence, as they lack inside knowledge, if an attacker approaches a target domain, they face the challenge of identifying potentially orphaned domains for further vulnerability scanning [35]. We demonstrate in Section 4.8 that traditional approaches, e.g., google dorks, are not effective at identifying orphaned pages. Hence, we develop a toolchain that utilizes the Internet Archive to detect (potentially) orphaned web page on any domain archived. We describe the design of our framework in Section 3 and demonstrate its efficacy at detecting orphaned pages at scale in Section 4. Please note that if the objective of the attacker is to find a vulnerable orphan page on a specific domain, heuristics we use for the large-scale evaluation can be omitted for a per-domain attack, widening the range of detectable URLs beyond the lower-bound we provide in our study. Finally, we also work on the assumption that such an attacker does not have to be familiar with advanced exploitation techniques, as orphaned websites are—on average—more prone to straight-forward vulnerabilities. We confirm this assumption in Section 5.

3 Methodology

Following, we introduce our methodology for identifying orphaned pages, and detail our large-scale measurement.

3.1 Orphaned Page Identification Methodology

We define orphaned web pages as pages with no ancestry links in the domain's sitemap (see Section 2). That is, when embarking from the root and following links, no path exists to reach the orphaned page. Consequently, this means we cannot rely on any information on the domain/website to identify orphaned pages. For a historic perspective on which sites used to be reachable via the root, we thus leverage the Internet Archive (IA) [2], which provides access to a time-stamped history of websites.

Leveraging this archived data, we learn how a domain used to look like throughout the years, not only in terms of content, but also in terms of its structure, that is, its sitemap. This allows us to compare previous versions of a website to its current version, and see whether any of the paths in its sitemap have been removed.

Correspondingly, we can identify potential candidate orphan pages by investigating pages that were once part of a domain's sitemap, but are not part of it anymore. We can then probe these candidate pages to determine if they are still accessible, and whether their content is different from the last archived version. Using this methodology, we detect pages from quadrant 1 of Figure 1.

A website administrator might advertise (part of) a website's structure in an XML file called `sitemap.xml`. This provides a list of

pages on the domain that a search engine can crawl and index. Unfortunately, not every website makes their `sitemap.xml` available. Some that do might have them excluded from the Internet Archive (see Section 3.2). Therefore, we cannot rely on a pre-constructed sitemap, but instead need to reconstruct it by retrieving the full list of archived pages from the Internet Archive, and using it to determine which URLs might contain potentially orphaned pages.

3.1.1 Gathering Candidate Orphan Pages First, see Figure 3, we utilize the Wayback CDX to retrieve the archived data [19]. It allows us to query a domain, after which the CDX Server returns the list of archived pages for that domain, including subdomains. From this list, we collect two sets: a current set, and a past set. The current set contains all the pages encountered by the crawler in 2020, the past set all pages last encountered before 2020. Because the Internet Archive makes use of crawls, that is, traverses the website from the root or inspects the domain's `sitemap.xml`, we know that each page is reached via a path starting from the root or it is listed in the sitemap. If a page was listed before, but is not present in the 2020 crawls, it has either been removed, was ignored by a later crawl, or got orphaned. We identify orphaned pages by liveness probing after pre-filtering candidates locally (see Section 3.1.4).

We have to account for the fact that the Internet Archive also stores pages that caused redirects, server errors, or client errors during the initial crawl. Given that we are only interested in reachable pages, we query the API to only return web pages that responded with a HTTP status code OK (200) in their original crawl.

3.1.2 Discarding Resource Files We focus our analysis of orphaned pages on actual web pages. However, domains may host additional resources, such as documents, pictures, JavaScript code, stylesheets, etc., which are of no interest to us. Thus, we filter the list of candidate orphan pages by removing URLs ending in a known resource-related file extension (see Appendix B for the full list).

3.1.3 Dynamic URL Detection (DUDe) URLs that are dynamically generated are a common challenge to identifying any kind of resource on the Internet. They can occur when news sites make articles available under unique URLs, but they are only reachable from the index page while they are recent. For example, for archived news articles, some form of (chronological) structure may be used, like `https://example.com/articles/1997/January/name-of-the-article.html`. This results in many URLs sharing a similar prefix with a (possibly generated) suffix. Probing all these URLs would put unnecessary load on webservices. Moreover, it would significantly extend the runtime of our approach, while these URLs are not actually as interesting to us as they are likely not truly unmaintained orphaned, but are likely actually maintained.

Therefore, we use a heuristic to identify the common prefixes of these URLs and remove them from our list before probing for liveness. If a page contains many long links, we try to identify a common prefix based on character frequency. We do so by counting the frequency of each character at an index of the URLs, and generating a prefix based on the most frequent character for each position (in case of a tie, the first one encountered will be used). We then shorten the prefix, one character at a time, until it is general enough (see below), or until we consider it too short to be a valid prefix. We repeat this process until we can detect no more

Algorithm 1 Dynamic URL Detection

```
max_len ← 0
avg_len ← 0
large_links ← {}

for link in page do
    // Check the length, 8 represents len("https://")
    if len(link) > (LT + len(dom_name) + 8) then
        large_links.append(link)
    Find average and max length of all links in sitemap
if len(large_links) ≤ LC then return
for link in large_links do
    for c in link do
        Count character frequency at each position

generated_link ← ""
for i = 0...max_len do
    Append most occurring character at position i to generated_link
prefix ← generated_link[: avg_len]

blocklist ← {}
allowlist ← {}
do
    for link in page do
        if prefix in link then
            blocklist.append(link)
        else
            allowlist.append(link)
    prefix ← prefix[: −1]
while len(blocklist) < PC · len(sitemap)

// Check the length, 8 represents len("https://")
if len(prefix) < (len(domain_name) + 8 + ST) then
    Restart procedure, ignoring links containing prefix
else
    Write out allowlist and blocklist
    Start procedure again on allowlist
```

prefixes. Naturally, this approach limits the number of dynamic orphan pages we can observe. Nevertheless, following our observation on not putting unnecessary strain on networks and systems, we consider it crucial to exclude dynamic URLs.

Algorithm 1 describes our heuristic, it has four parameters:

(1) **Popularity cutoff (PC)**: the percentage of URLs on a domain that need to contain the prefix.
(2) **Short-link threshold (ST)**: the amount of characters a link needs to have to be considered short.
(3) **Long-link threshold (LT)**: the amount of characters a link needs to have to be considered long.
(4) **Long-link cutoff (LC)**: the amount of links on a domain that need to be long for the heuristic to run.

We determine these parameters from a random sample of 1,000 domains, taken from our input data (see Section 3.2). We evaluate the following values: [5%, 10%, 15%, 20%, 25%, 30%] for the popularity cutoff, [5, 10, 15, 20] for the short-link threshold, [20, 25, 30, 35, 40] for the long-link threshold, and [0, 3, 5] for the long-link cutoff. We optimize for the highest percentage of reduction. After evaluating all permutations, we obtain 5% for the popularity cutoff, 15 for the short-link threshold, 20 for the long-link threshold, and 0 for the long-link cutoff, as the optimal input parameters. This configuration reduces the number of URLs per domain by 67%, with a standard deviation of ±30 percentage points and median of 73%.

Optimizing for the highest reduction percentages may filter out unintentionally orphaned web pages by error. We consciously choose this optimization to provide a lower bound on the prevalence of unintentionally orphaned pages in the wild.

Next, we illustrate how our algorithm removes dynamic URLs for an example domain, see the set of URLs in Listing 1. Here, we

```
https://example.com/articles/1997/January/article1.html
https://example.com/articles/1997/February/article1.html
https://example.com/articles/2002/May/article1.html
https://example.com/articles/2002/May/article2.html
https://example.com/contact.html
https://example.com/login.html
```

Listing 1: URL list before Dynamic URL Detection.

```
https://example.com/contact.html
https://example.com/login.html
```

Listing 2: URL list after Dynamic URL Detection.

first check for long links. We continue by counting the character frequency at each position, followed by generating a URL with at each position the most encountered character of that position. In our example, this becomes `https://example.com/articles/1997/May/articlei.htmlhtmll`. We then shorten the URL to the average URL size among the URLs on the domain, which we use as a prefix. This prefix URL is `https://example.com/articles/1997/May/articlei`. Since no URL matches this prefix, we shorten it one character at a time until at least 5% match the prefix, or until the prefix is too short. In our example 5% corresponds to 0.45 pages, which is rounded up to 1 page. In practice, this will be a (much) higher threshold as websites typically have more than nine pages.

We shorten the prefix until it becomes `https://example.com/articles/1997/`, which is present in three URLs. We then remove the URLs containing the prefix and repeat the process on the remaining URLs. The second generated prefix is `https://example.com/articles/2002/May/arti`, which does not require further shortening as it is matched by three of our URLs. These URLs are removed, after which the algorithm stops since no further long links are present in our set, resulting in three URLs remaining (see Listing 2).

3.1.4 Probing Liveliness of Candidate Orphan Pages After filtering the list of potential orphan pages with DUDe, we analyze the pages by probing them. This allows us to exclude pages that were actually taken down after their links have been removed. We perform a HTTP HEAD request to retrieve the HTTP status code of the web page. We discard pages producing error responses (4xx and 5xx status codes), as well as any page that does not return the HTTP status code OK (200).

3.1.5 Open Source Implementation We provide a dockerized open source implementation of our methodology at https://github.com/OrphanDetection/orphan-detection (anonmymized for submission). It implements our methodology for running it against a single website (see Figure 3), making large-scale studies embarassingly parallel and enabling them to scale horizontally across machines easily.

The open source nature of our implementation also allows security professionals and researchers to tailor our methodology to their needs. For our large-scale measurement study, we chose cutoff and threshold parameters that led to a high reduction rate to obtain a lower bound with our measurements. However, when used on a single domain, security professionals may prefer a more lenient Dynamic URL Detection, or even omit this module from the procedure entirely, to apply additional context-aware filters on the list of potential orphans. We expand on use cases of our technique later (see Section 6).

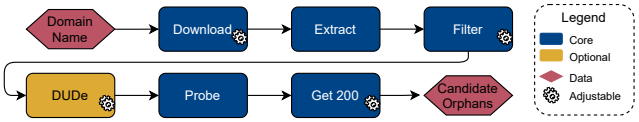


Figure 3: Overview of our implementation.

3.2 Large-Scale Dataset

We evaluate the efficacy of our method through large-scale measurements, detecting orphaned pages in the wild. Following, we discuss the data set that we used for our study, and report on the timeline and implementation of our measurements. We deploy our methodology on 20 servers within our university network, and split our study into two phases: downloading and processing.

3.2.1 Input Data Sets We rely on two different data sets for our measurements: First, a random sample of 100,000 domains from the Tranco Domain List, and, second, the archived data for these domains from the Internet Archive (see Section 3.1.1).

Tranco Domain List: Research has shown that traditional Top 1M lists are less reliable than initially thought [22, 28, 29]. They are not stable, contain unresponsive websites, and are vulnerable to manipulations, such as promoting one's domain. Additionally, most of them actually disagree on the exact list and order of domains, with one study showing a meager 2.4% overlap among the agreed order of the four Top 1M lists [22].

To counter the existing issues with Top 1M sets, we use the Tranco list [22]. It averages the ranks of domains, as listed by Alexa, Cisco Umbrella, Majestic, and Quantcast. In turn, the list is more resilient to manipulations and exhibits less fluctuations.

For this study, we use the Tranco list from 14 December 2020.[1] We evaluate our methodology on a subset of the Tranco list, consisting of a random sample of 100,000 domains taken from the top 500,000 ranked domains. The exact list of domains is available at our open source repository.[2]

Internet Archive: To detect orphan pages, we make use of the Internet Archive, which is an online digital library, archiving and providing access to various resources including websites [2] (see Section 3.1.1). We query the Internet Archive's Wayback Machine using the CDX API [19]. It stores a URL key, timestamp, full URL, media type (MIME), status code received during crawl, a message digest, and the length. However, we only retrieve the timestamp and full URL, and do this only for pages that returned a status code of OK (200) upon their initial crawl (see Section 3; this makes our approach lightwight in terms of archive data).

3.2.2 Downloading Phase For the downloading phase, we take our input domains and retrieve the archive file for each domain. We retrieved the archives between December 16, 2020 and December 20, 2020, with ten downloads in parallel on each of our servers, with a delay of one minute between each batch. We do so to reduce the load on the web archive in consultation with the Internet Archive.

When fetching the archive data for a domain, we encountered four different issues: (1) the server throws an error, (2) the server is temporarily offline, (3) the server sends back 0 bytes, and (4) we hit

[1] Available at https://tranco-list.eu/list/5Q3N.
[2] Available at https://github.com/OrphanDetection/orphan-detection.

the rate limit. On each issue, we move the corresponding domain to the back of our queue and attempt to retrieve it again later. We repeat this process until we retrieved all archival data, or are left with cases only returning 0 bytes or an error (for which we then assume the data is not available in the Internet Archive).

Overall, we retrieved sitemaps for 96,537 domains from our sample of 100,000 domains (with the remaining domains not being indexed), for a total of 6,092,214,431 URLs, that is, on average 60,922 pages per domain (median: 6,955, standard deviation: 126,094).

3.2.3 Processing Phase We then apply our methodology on the pages from the 96,537 domain archives. Table 1 shows an overview of timeframe and results of our large-scale measurement, which we discuss next. Filtering the pages on our list of file extensions, we first obtain a total of 4,033,539,860 potential orphan pages. After removing dynamic URLS with DUDe, we reduce the list to 924,190,351 URLs (a reduction of 77%).

We then probe the remaining URLs to determine which pages still return a status code of OK (200). To avoid overloading the domains with our probes, we shuffle the list of URLs and split them over our deployment. This ensures that, on average, we did not exceed 87 requests per domain per hour. Moreover, our probes used a customized User-Agent that allowed administrators to easily opt-out from our study at any time (see also our ethics discussion, Section 6.2).[3] We probed candidate orphaned pages from January 1, 2021 to January 29, 2021. After probing and removing pages that did not respond with a HTTP status code OK (200), we are left with 36,442,679 candidate pages (a reduction of 96%), see Table 1.

3.2.4 Identifying Custom Error Pages Unfortunately, not every unavailable web page returns a status code of Not Found (404), or Redirect (301/302). Instead they might return a custom error page stating the page was not found, while the HTTP status code is actually OK (200). In fact, while probing pages, we regularly encounter discarded pages (i.e., web pages displaying a standard HTML stating a similar message to "This page no longer exists") returning a status code of OK (200). To remove these false positives at scale, we retrieve the size of each page and remove pages from the same domain with a similar size (e.g., within 5 bytes of difference). Leveraging size-based filtering, we remove pages that should have returned an error response, but responded with status code OK, and we reduce the set to 1,821,682 URLs (a reduction of 95%).

We may remove some pages with genuine content that is similar among multiple pages, such as login portals. However, in this paper, we aim to determine a lower bound on the prevalence of orphaned pages, and, thus, we believe that omitting some pages and duplicates is an acceptable trade-off (see also Section 6.3).

3.2.5 Removing Invalid Pages Finally, we discard 564,619 invalid pages because of a bad file encoding or because we actually received a non-HTML file, leading to 1,257,063 pages (a reduction of 30%) that we need to analyze in more depth.

4 Analysis

Here, we analyze and report on the results of our measurement study, and evaluate the validity of our candidate orphan pages.

[3]During probing, we received two requests from administrators to remove their website from our study. Correspondingly, we excluded some links from probing.

Table 1: Summary of our large-scale measurement.

Step	Timeframe	Result
Downloading archive data	December 16, 2020 – December 20, 2020	96,537 Domains
Filtering file extensions	December 20, 2020	4,033,539,860 Pages
Dynamic URL Detection	December 29, 2020	924,190,351 Pages
Probing and extracting pages with HTTP 200 response	January 1, 2021 – January 29, 2021	36,442,679 Pages
Size-based filtering	February 14, 2021	1,821,682 Pages
Removing Invalid Pages	February 23, 2021	1,257,063 Pages

4.1 Data Set Overview

A summary of the data set from our large-scale measurement can be found Table 1. After processing the archive data, filtering them, removing dynamic URLs, probing, filtering based on size, and removing invalid pages, we are left with 1,257,063 candidate orphan pages that we need to analyze in more depth (see Section 3.2.3).

4.2 Archive Data Analysis

After collecting the archive data for the domains in our data set, we can investigate how domains evolve over time. Figure 4a shows the mean amount of pages per domain for each year between 2000 and 2020. Here, we see a clear increasing trend throughout the years, suggesting that websites grow. To account for bias inflicted by relatively young websites, since these would have zero pages before their first appearance online, we also show the growth for all domains that had at least one page archived in 2000. We observe that accounting for age still shows an increasing trend for the number of pages on a domain. This is also true for the median (see Figure 4b). Additionally, Figure 4c shows a boxplot for the amount of pages per domain per year. Apart from confirming the increase in the number of pages per domain over time, the latter two graphs also show the high variance in pages per domain. Especially from the boxplots, we can detect a significant difference between the first quartile and the third quartile.

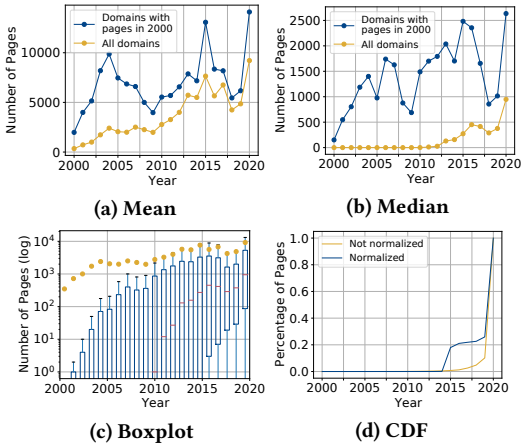
We described earlier that the 2020 sitemaps function as the bases for gathering the candidate orphan pages (see Section 3). We now take a look at how, and specifically when, the 2020 sitemaps came to be. Figure 4d depicts two Cumulative Distribution Functions (CDFs) of when a page of a 2020 sitemap was first seen in the archive data. The non-normalized CDF is calculated by summing the pages across all domains for each year and constructing a CDF of the sum, the normalized CDF is calculated by taken the average of the CDF percentages for each year across all domains. Both are only calculated for pages that were last seen in the archive in 2020.

Interestingly, both CDFs suggests that old pages indeed get removed from a domain eventually. However, the other plots of Figure 4 also indicate that sitemaps are growing over the years. This then means that the older a page is, the less likely it is meant to be online. We will leverage this observation later on in our analysis.

Note that the trends of the graphs rely on the Internet Archive's operations throughout the years. For example, sitemap growth is naturally influenced by the Internet Archive's crawl scale.

4.3 Page Similarity

We identified a first parameter that can provide an indication of whether a page is unintentionally orphaned: its last seen date. Albeit not sufficient on its own, there is another intuitive indicator of



(a) Mean (b) Median (c) Boxplot (d) CDF

Figure 4: Analysis of the archive data. (a) and (b) show the number of pages on a domain, (c) shows it as boxplots (yellow markers represent the mean). (d) depicts a normalized and non-normalized CDF of which percentages of pages from the current sitemap were present in a specific year.

whether a page was intentionally orphaned: its similarity to other pages of the same domain. Thus, to check whether a page has actually become unmaintained orphaned since its last listing on the Internet Archive, we compare the fingerprint of a page's current version to its last archived version.

Detecting (near-)duplicates among web pages and other types of documents has been extensively studied before. Henzinger [18] compared the two state of the art approaches for near-duplicate detection of web-pages (Broder et al. [8] and Charikar [9]), and concluded that Charikar's simhash performs better, which also has a small memory footprint [27]. Thus, we base our similarity detection on it. Our comparison works by first creating a fingerprint of the two pages and comparing them.

Creating Fingerprints using simhash: We create the fingerprint of a web page by first removing all HTML tags and extracting only the content of the page. We then divide the extracted content into n-grams of $n = 8$. For each n-gram, we construct a 64-bit hash using the FNV-1a hashing algorithm. To construct the fingerprint, we sum each hash over the same index in the following way: If the hash of the n-gram has the value 1 at index i, we add 1 to index i in the fingerprint. If the hash has the value 0, subtract 1 from index i in the fingerprint. For each index in the fingerprint, convert the value to 1 if its positive, and to 0 if its negative. The result is a 64-bit fingerprint of the web page.

Comparing Fingerprints using Hamming Distance: For each URL in our data set, we retrieve its current version and its latest archived version, and we create fingerprints for both pages. We then compare the 64-bit fingerprints using the Hamming distance, which is the number of bits that need to be changed for the two fingerprints to produce two equal strings. The inverse gives us the similarity.

Figure 5 shows the distribution of the similarity scores of the candidate orphaned pages. We observe a clear Gaussian distribution around 0.5 with an increase at 1. Indeed, it follows our expectations that, although pages change over the years (e.g., their design), core content often remains unchanged. The left tail of the curve

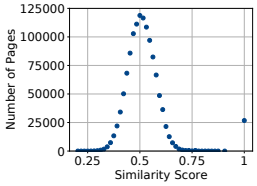


Figure 5: Distribution of the calculated similarity scores.

represents pages that have changed more drastically throughout the years, while the right tail of the curve shows the pages that have remained unchanged, with a spike at 1 for the pages that appear to have not changed at all. The latter are of high interest to us, as they might be unmaintained orphaned pages.

4.4 Orphan Likelihood Score

Leveraging the two parameters that can provide an indication of whether a page is intentionally or unintentionally orphaned, its last seen date and its similarity score, we can derive a metric for the likelihood of a page being orphaned. As a starting point, we take a domain's sitemap from 2020 and we subtract the year in which a page was last encountered from 2020 to get the page's age. The oldest page in our data set is from 1997, and, hence, the maximum age a page in our data set can have is 23 years. The similarity score is normalized to [0-1], and we can easily scale it to [0-23]. As such, we combine the parameters to the orphan score $O = \frac{23Sw_s + Aw_a}{23}$. Here, S is the similarity score, A is the age (calculated by subtracting the last-seen year from 2020), and w_s and w_a are weights of the similarity score and the age. We normalize to [0-1].

We evaluate different values between 0 and 1 in increments of 0.1 for w_s and w_a. Figure 6 depicts the results. With each increment the distribution of the orphan score slowly excludes a set of links (on the right side of the graph) until it converges to the normal distribution of Figure 5 at $w_s = 1$ and $w_a = 0$. The graph becomes split in two: a normal distribution on the left (low orphan score), and a set of outliers on the right of the bell (higher than average orphan score), allowing us to study the set of links that got "detached" from the greater chunk in the graph.

We use a weight allocation of $w_s = 0.9$ and $w_a = 0.1$, along with a cutoff of 0.9. Meaning, we discard all pages with an orphan score lower than 0.9. This reduces our set of pages to 26,756 URLs.

4.5 Classifying the Type of Pages

Following, we classify the identified 26,756 URLs and determine whether they are truly orphaned. We look for specific indicators, in the source code or content, that provides insight into the orphan status. We divide them up in three classes: likely orphaned, not orphaned, and uncertain. With likely orphaned, we identify clear indicators that the page is orphaned, such as outdated copyright statements or incomplete/boilerplate code. Second, we label a page as non-orphaned when we detect that the copyright on the page is at least from 2020. Note, that this will mislabel pages with a dynamically generated copyright, which is in line with our goal of providing a lower bound. Finally, we label a page as uncertain when there is no a clear indicator for it to be likely orphaned or for the page to be non-orphaned. Our indicators are:

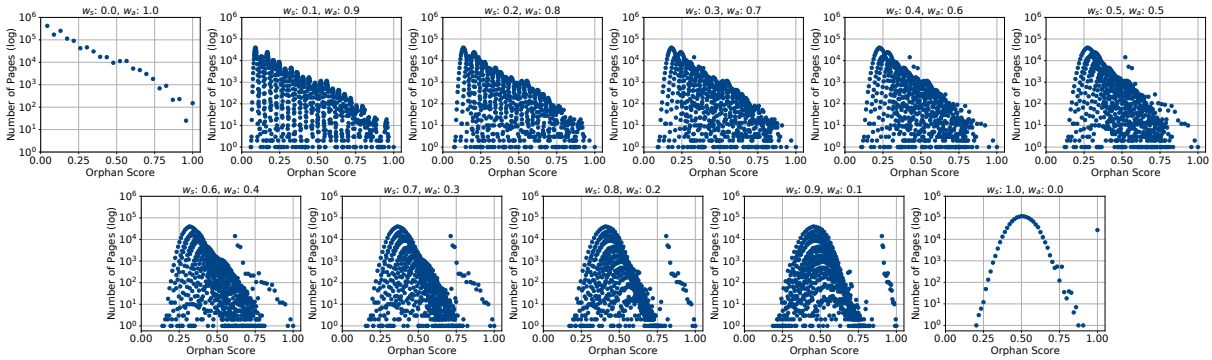


Figure 6: Empirical evaluation of the weights in the Orphan Score formula. We see that a higher weight for the similarity score leads to the exclusion of a specific set of links. Chosing the extremes ($w_a = 1$ or $w_s = 1$) leads to a more discrete clustering of the orphan score, thereby increasing the per-score-max.

4.5.1 Copyright Statements Copyright statements, often at the bottom of a page, can give insight into when a page was last updated. To capture these statements, we scan for the word "copyright" (case insensitive) and the copyright symbol (©). If present, we examine the 50 characters before and after the copyright, and try to detect a date. If the year is 2020 or 2021, we assume the page is up-to-date. If the year is older, we label it as likely orphaned.

4.5.2 Boilerplate Code Websites often use "boilerplate code" to speed up the development process. They can function as a template complying with the general setup of the website, or as a test page to confirm successful functionality or deployment. This boilerplate code, when rendered, may display an empty page, or a generic (small) set of words. The code itself, however, may be filled with boilerplate HTML tags or JavaScript code blocks, assuring compliance with the general template of the website.

Boilerplate code that is not further developed with content can be an indication of an orphaned web page, presumably because there was intention of deploying an endpoint, which was then later not further pursued, although without removing the page.

We identify those pages as boilerplate that result in an empty page or one with less than 5 words after we strip all HTML-tags and script blocks from the source.

4.5.3 Error Pages We already removed pages that respond with a status code OK, yet display an error message (see Section 3.2). However, such pages might still be present in our data because on our heuristic to filter at-scale might not be perfectly accurate. Therefore, on this pre-filtered data set, we now utilize a more expensive and more accurate additional analysis step. To detect these pages, we look for common phrases, such as "Page Not Found." The full list of key-phrases can be found in Appendix A.

4.5.4 Following Redirects and Loading Frames Earlier, we removed all pages that responded with a status code different than OK (200). While effective at-scale, this approach misses specific redirects, for example in JavaScript or HTML meta tags. We detect these redirects by comparing them against a list of known in-page redirect techniques (see Appendix A). Similarly, we can use a list of HTML tags used for loading frames to detect these types of pages. Albeit pages making use of redirects or frames can be classified as such, they might still be unmaintained or forgotten.

For both types, we attempt to detect the resource referenced by the redirect or frame. In case multiple frames are loaded, we only analyze the first one we encounter. Once resolved, we apply the same classification as we did initially. For redirects, we might encounter a loop, that is a page redirects to another redirecting page, possibly resulting in an infinite redirect loop. Like modern browsers, we limit the number of consecutive redirects. In our case, we follow a maximum of 20 redirects.

4.6 Removing One Domain Name

When analyzing our remaining 26,756 pages we notice one domain name evading DUDe as it was spread over multiple TLDs, biasing our results with a little over 20,000 entries. We missed these cases as the pages are spread over multiple TLDs (`.my`, `.hk`, `.tw`, `.net`, `.ie`, `.ae`, and `.id`), and our heuristic functions on a fully-qualified domain name basis.

Remarkably, the TLDs of the domain are primarily Asia-oriented, with the exception of `.net` and `.ie`. We manually filter out the pages using this SLD to have a more diverse set of domains that gives a better depiction of the accuracy of our methods. Doing so leaves us with 5,914 pages.

4.7 Analyzing Types of Pages

For all links left in our set, we download their latest version as of March 14, 2021 and analyze them for the type of pages (redirects, boilerplate, etc.). Figure 7 shows a Sankey diagram of the results. On the left we see our initial classification and on the right after following redirects and loading frames.

The most common pages we encounter are in-page redirects via HTML or JavaScript (2,093 pages). Although redirecting can be a mechanism to divert traffic from orphaned pages, our results indicate that many of these redirects tend to be misconfigured (45%), of which the majority seems to cause an endless loop of redirects (65% of the misconfigured redirects). From the redirecting links that end up at a page with a copyright statement, we see around an equal amount with a recent timestamp and without (both 17%).

Second most common, we see the boilerplate pages (1,612 initial pages, 1,752 after redirects and frames). We believe these are forgotten. We assume there was initial intent to deploy content,

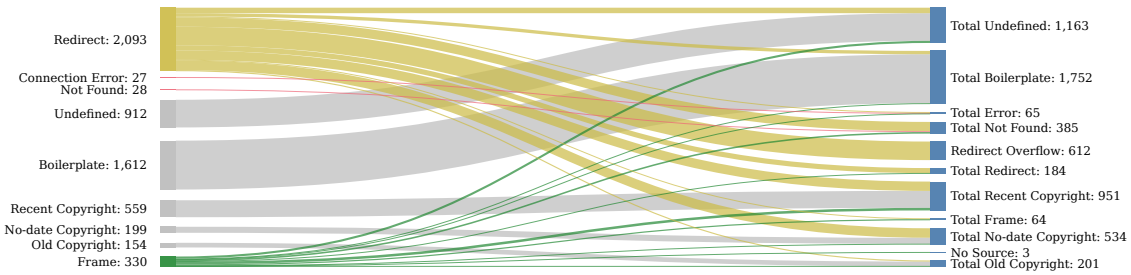


Figure 7: Sankey diagram of the detected page types. On the left we have the initial classification. The flows in the diagram depict the result of following redirects and loading frames, which leads to the results of our classification shown on the right.

Table 2: Summary of the most important types of pages. We observe that our method is capable of detecting web pages that are likely to be orphaned, and we can say that 951 pages are in fact not orphaned. We also see a large group of pages for which the determination is not clear.

Type	Count	Orphan Status	Total
Old Copyright	201	Likely Orphaned	1,953
Boilerplate	1,752		
Copyright without date	534	Uncertain	1,706
Undefined	1,163		
Recent Copyright	951	Not Orphaned	951

but it was not further pursued. This could be due to a myriad of reasons, such as simply forgetting about it or refactoring.

Particularly interesting are the pages that contain a copyright statement. In total, we found 1,686 such statements, with the majority of the pages actually having an up-to-date timestamp (56%). While these might be dynamically generated, we consider these pages maintained in the spirit of identifying a lower bound. The ones that do not have an up-to-date timestamp are divided into two groups: statements with a timestamp (12%), and statements without any timestamp (32%). When a timestamp is present, we assume the page was last maintained during that year. Figure 8 depicts the distribution of the years extracted from copyright statements. These pages are likely orphaned, while the copyright statements without a timestamp could be orphaned, but might not.

Table 2 summarizes the types of pages and their orphaned status. For at least 1,953 URLs, the pages appear forgotten, unmaintained, and therefore likely orphaned. These pages are spread over 907 domains, with one domain having as many as 142 likely orphaned pages. For 1,706 pages, the decision is not as clear. While we could not detect any clear sign of being maintained, we also do not know if they are not. Lastly, we identify 951 pages which appear maintained and not orphaned.

4.8 Google Visibility of Orphaned URLs

We also make use of Google's Programmable Search Engine [15] to determine how many of our pages are indexed by Google. We construct a simple query that searches for "`site:<url>`" and count the number of results returned by Google. If no results are returned, we assume the page is not indexed. This gives insights into how "easily" reachable these pages are on the web.

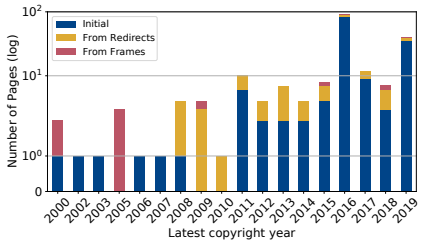


Figure 8: Extracted copyright years from copyright statements on pages that are not considered to be up-to-date. The majority of the pages were last touched at some point in the past 5-10 years, although we also observe pages which appear to be unmaintained for much longer.

We perform the lookups between March 10, 2021 and March 12, 2021 (over 3 days due to rate limiting) on all 5,914 URLs. We find that only 16% of the pages were indexed by Google. This shows that many of our (likely) orphaned web pages are not (easily) reachable through the web, and hence difficult to find without making use of archived databases, highlighting the need for our technique.

We observe no correlation between any of the identified types and the indexed status of a page, neither between any of the orphaned statuses and Google's indexing. Only pages for which we detected an old copyright statement we see that the more recent a page, the more chance of it being indexed. However, we expect this is just following the general distribution from Figure 8, or that they will soon be removed by Google.

5 Security Evaluation

Next, we investigate if orphaned pages actually have any security implications. That is, we try to answer the question: Are orphaned pages (on average) more vulnerable than non-orphaned pages?

5.1 Methodology

Following, we assess the vulnerability of orphaned pages compared to non-orphaned pages with two control groups.

5.1.1 Control Groups First, we compare orphaned pages to a control group from the general population. Specifically, we compare the orphaned pages to recent non-orphaned pages sampled at random from our data set. These random recent non-orphaned pages were all crawled by the Internet Archive in 2020. This means that our first control group (Control 1) is a sample similar to a random

Internet crawl, and provides insight into whether the security posture of orphaned pages differs from the general Internet.

Second, we instead compare orphaned pages to non-orphaned pages from the same website. That is, we investigate the hypothesis that all pages on websites that have orphaned pages in the first place, including the non-orphaned pages on these websites, are more likely to suffer from vulnerabilities than the general Internet. This follows from the intuition that websites with orphaned pages are likely not maintained properly (or they would not have orphaned pages), and thus they might also be less likely to deploy other measures to improve security, resulting in being overall more vulnerable than the general Internet. Our second control group (Control 2) consists of one random non-orphaned page for each website with an orphaned page. Again, all pages are recent, that is, they were part of the Internet Archive in 2020. Correspondingly, with our second control group, we can test whether the amount of vulnerabilities is "internally consistent" for websites with orphaned pages, that is, whether they have (in general) a lower/higher security level than the general Internet.

5.1.2 Security Analysis We analyze the security posture of all three groups (Orphaned, Control 1, and Control 2) by scanning them with Wapiti [31] for cross-site scripting (XSS) vulnerabilities, SQL injection (SQLi) vulnerabilities, and their security configuration.

Wapiti is a black-box web-application vulnerability scanner that we chose because it is actively maintained and it allows us to limit the scope of the scan to the *specific* URL (including all query parameters, that is, only modifying existing query parameter values, nor removing them, nor adding new ones; e.g., for an orphaned page with a form that is posted to the same URL without any query parameter modifications) and page (allowing query parameters to be added, and removed; e.g., adding a new pagination parameter `?page=3`, or submitting a form that sets the parameter `?form=submit`). These scope restrictions to URL and page are particularly important to prevent accidental scope-creep beyond the orphaned pages we detected, that is, scanning, identifying, and reporting vulnerabilities in other non-orphaned pages that are reachable from the orphaned pages themselves by following links. Moreover, Wapiti provides a Proof of Concept (PoC) for each found vulnerability, which aids us in verifying the reliability of our scans and assess the accuracy of our findings.

We investigate XSS and SQLi vulnerabilities because they pose significant risks to website operators and website users. SQLi attacks allow adversaries to execute SQL queries on a web application's database, which can lead to sensitive user information being leaked (e.g., personally identifiable information (PII)), privileges being escalated, or access control being bypassed (e.g., SQLi for a login form). Similarly, cross-site scripting (XSS) attacks can allow attackers to inject and execute arbitrary JavaScript code within the user's browsers in the scope of the website, which can allow the attacker to steal user data and login tokens; worse, depending on involved URLs, domain, and scopes for cookies, an orphaned page might have access to cookies set by non-orphaned pages of a website. Last, we study their security configuration to get insights into how security is being handled. See Section 6.2 for a discussion of the ethics of our security analysis, that is, how we minimized harm and performed responsible disclosure.

5.1.3 Requirements Minimization We also minimize our data set to not scan pages that cannot be vulnerable because they do not satisfy the requirements for the specific vulnerability. This is important from an ethical perspective, to reduce network load and to not cause website operators to expend resources to serve requests that have no purpose.Notably, XSS and SQLi vulnerabilities require that the website processes user input in some form. Hence, for XSS and SQLi, we limit us to pages allowing user input.

To establish whether a page allows user input, we err on the side of caution and include all pages having *any* indicator that they *might* process input. We consider the presence of any query parameter and the presence of `form` and `input` tags within the HTML code of a page as an indicator of user input. When only including such pages, we find 636 Orphaned pages, 997 pages from Control 1, and 1007 pages from Control 2 that might allow for user input. Please note that this filter was applied after group selection.

Furthermore, albeit first released in 1995, JavaScript only became widely adopted with more available libraries, like the prominent jQuery library, first released in late 2006 and still being widely used today, and the use of AJAX. In turn, encountering it, and therefore XSS vulnerabilities, in pages from before 2007 is unlikely, which is why we limit our XSS tests to pages newer than 2007.

5.2 Results

Following, we present the results of our security scans. Please see Table 3 and Table 4 for an overview of our results.

5.2.1 Cross-Site Scripting (XSS) Limiting our security scan for XSS vulnerabilities to the exact URL, we find that 128 orphan pages suffer from XSS vulnerabilities (20.1%). This stands in stark contrast to the general Internet (Control 1) with 17 vulnerable pages (1.7%) and also the non-orphaned pages on the same websites themselves (Control 2) with 64 pages (6.3%) vulnerable to XSS, with both differences being statistically significant at $p < 0.01$ in χ^2.

Broadening the scope to the page, that is, allowing query parameters to be added and removed, these numbers only change slightly: 123 orphaned pages (19.3%) are vulnerable to XSS, compared to 20 pages (2.0%) of the general Internet sample (Control 1) and 59 pages (5.9%) for the non-orphaned pages of the websites (Control 2), statistically significant at $p < 0.01$ in χ^2.

Naturally, one would expect to find more vulnerable pages when broadening the scope from URL to page. However, this is not the case for orphaned pages (128 pages and 123 pages respectively) and for Control 2 (64 pages, and 59 pages respectively). The reason is simple: We first performed our URL scope scans, then notified the operators in a first round about the vulnerabilities we found, and then proceeded by broadening the scope of our scans to page. Correspondingly, to understand the differences between both scans, we need to analyze the set differences between our scans and manually investigate. We found that our page-scoped scan discovered one additional orphaned page with a XSS vulnerability, while it could not reproduce six vulnerabilities on orphaned pages because they were already remedied by the affected parties after our first notification. Results in the other two control groups are similar: We found nine new XSS vulnerabilities for our general sample (Control 1), and nine new XSS vulnerabilities for the same-website non-orphaned pages (Control 2).

To assess the accuracy of our findings, we also manually verified a random sample of the XSS PoCs by Wapiti. Unfortunately, our manual verification took place after notifications were already sent out to the operators, which means that our true positive rate is a lower bound, and the actual number of real vulnerabilities we identified might be higher. In our manual verification of a random sample of 25% of the found XSS per group (105 total, 10 Control 1, 32 Control 2, 63 Orphaned), we found the Wapiti provided PoCs to be true positives in 92.4% (97 total, 6 Control 1, 28 Control 2, 63 Orphan) of the cases. However, for the four samples from Control 1 and one sample from Control 2, the affected pages have been removed, now delivering Forbidden (403) or not Found (404) errors, suggesting that they were addressed by the operators. This leaves only three potential false positives in Control 2.

A noteworthy vulnerable orphan page we encountered is an online sports betting site `example.bet`. For this website, an old login page has remained active after the website migrated to a new login mechanism at `example.bet/login`. This endpoint is vulnerable to a simple query parameter XSS of the form "`/login?ref="><ScRi Pt>alert('VULNERABLE')</sCrIpT>`." We also verified that it is no longer the current login page, that is, it is indeed orphaned.

Overall, we find that orphaned pages are more likely to be vulnerable to XSS than a general sample of pages (Control 1) and non-orphaned pages on websites that have orphans (Control 2), statistically significant at $p < 0.01$ in χ^2. Interestingly, websites that have orphaned pages (Control 2) also have a worse security posture than pages for which we identified no orphans (Control 1), also statistically significant at $p < 0.01$ in χ^2.

5.2.2 SQL Injections (SQLi) Considering SQLi vulnerabilities, we find 25 orphan pages (3.9%), 22 general pages (1.7%, Control 1, difference significant at $p < 0.05$ in χ^2), and 81 non-orphaned pages for websites with orphans (8.0%, Control 2, significant at $p < 0.01$ in χ^2) are vulnerable to SQLi when scoping our scan to the exact URL. This is interesting: Orphan pages themselves are only slightly more vulnerable than general pages, but non-orphaned pages on website with orphans are almost five times as likely to be vulnerable to SQLi than general pages (significant at $p < 0.01$ in χ^2).

Broadening our measurement scope from the URL to the page, however, these results change. Here, we see a significant increase to 69 vulnerable orphan pages (10.8%), and slight increases to 27 general pages (2.7%, Control 1, significant at $p < 0.05$ in χ^2) as well as 96 pages for Control 2 (9.5%, $p \approx 0.39$ in χ^2). This highlights that restricting scans to the exact URL is insufficient, for example, because of SQLi on some parameters that were not part of the original URL, such as a pagination parameter. Similar to our XSS scans, our page-scoped scans were performed after our first round of notifications and some vulnerabilities have been remedied before our page-scoped scan. Overall, by broadening the scope of our scan, we found 53 new vulnerable orphan pages, 18 new vulnerable pages for our general sample (Control 1), and 43 new vulnerable non-orphaned pages for websites with orphans (Control 2).

We manually validated the PoCs to assess our accuracy for SQLi vulnerabilities. We find that 122 (38.1%) pages can be readily verified to be vulnerable with the PoCs. A further 73 (22.8%) pages have been taken down by their operators since, possibly in response to our notifications. Sixty-five (65, 20.3%) pages adopted other mitigations after our notification, leaving 60 (18.8%) PoCs that require manual in-depth analysis. Unfortunately, we have been unable to assess whether they are truly false positives: They are blind SQLi that rely on the `sleep` function, and, while the pages do behave differently for the generated PoCs and the original URL, we did not find any correlation to the value provided to the `sleep` function. For some cases sampled at random, we could determine that the `sleep` function was unsupported by the databases (e.g., MSSQL), resulting in the database query failing, the page load never completing, and no response being sent to the user. Accordingly, while we believe that these cases are true positive injections due to their behavioral differences, we refrained from further manual in-depth analysis to not cause undue harm.

Table 3: Number of Cross-Site Scripting (XSS) and SQL Injection (SQLi) vulnerabilities among pages that allow for user input. p-values are relative to the Orphan group in χ^2.

Scope	Vuln.	Control 1 n=997	Control 2 n=1007	Orphaned n=636	p_{c1}	p_{c2}
URL	XSS	17 (1.7%)	64 (6.3%)	128 (20.1%)	<0.01	<0.01
	SQLi	22 (2.2%)	81 (8.0%)	25 (3.9%)	<0.05	<0.01
Page	XSS	20 (2.0%)	59 (5.9%)	123 (19.3%)	<0.01	<0.01
	SQLi	27 (2.7%)	96 (9.5%)	69 (10.8%)	<0.01	≈0.39

A noteworthy SQLi example is an orphaned page of a government organization that is responsible for entrance exams for education in medical professions, which still provides access to old instances of applicant handling systems via `example.med/index.php/applicants/applicants-2008/apps08prospective?tmpl=index\&print=1\&page=1`. Here, the `tmpl` parameter is vulnerable to SQLi, such as, `1+or+sleep(10)#`. After our notifications, this page was removed without providing a reply back to us.

Overall, for SQLi, we find orphaned pages to be significantly more likely to be vulnerable than a random sample on the web (Control 1), statistically significant at $p < 0.05$ in χ^2 for our URL-scoped scan and at $p < 0.01$ in χ^2 for our page-scoped scan. Moreover, for the broader page-scoped scan, there is no longer a significant difference between pages that are orphaned and non-orphaned for websites that have orphans (Control 2). In fact, all pages on websites with orphans have a comparable share of SQLi vulnerabilities (9.5% and 10.8% respectively), which is also significantly higher than that of general pages (2.7%) with $p < 0.01$ in χ^2.

5.2.3 Security Configuration Issues Finally, in addition to XSS and SQLi, we also assess if sites set a Content Security Policy (CSP) or HTTP Security Headers, as well as if they use the Secure or HttpOnly flags for cookies. We only report URL-scoped results as the scan scope does not affect these issues. Although we encounter a similar trend as to our XSS evaluation, the effect sizes (even though significant in χ^2) are too small to allow for conclusive statements. Similarly, we only found a limited number of cases of Open Redirects and Path Traversal vulnerabilities (0 to 7), preventing us from drawing conclusions regarding their differences.

6 Discussion

In this section, we discuss the applied and scientific use cases for our methodology. Furthermore, we discuss ethical implications of our approach, how we addressed these, and the limitations our methodology and measurements have.

Table 4: Comparison of found security configuration issues. We omit some p-values (χ^2) due to small effect size.

Issue	Control 1 n=1875	Control 2 n=1944	Orphaned n=1953	p_{c1}	p_{c2}
No Content Security Policy	1758 (93.8%)	1838 (94.5%)	1896 (97.1%)	<0.01	<0.01
No HTTP Security Headers	1660 (88.6%)	1751 (90.1%)	1859 (95.2%)	<0.01	<0.01
No Secure Flag Cookie	738 (39.4%)	755 (38.8%)	644 (33.0%)	<0.01	<0.01
No HttpOnly Flag Cookie	545 (29.1%)	713 (36.7%)	578 (29.6%)	≈0.72	<0.01
Path Traversal	0 (0%)	7 (<1%)	2 (<1%)	-	-
Open Redirect	1 (<1%)	0 (0%)	1 (<1%)	-	-

6.1 Applications and Future Research

Our methodology holds practical applications for network defense and vulnerability assessments, and potential for future research.

6.1.1 Use by Network Defenders In Section 2, we have shown various scenarios in which a web page can become orphaned. Here, our implementation can assist defenders by automating discovery for outdated page and web resources in general. Instead of (solely) relying on the Internet Archive as a baseline, administrators can also make use of internally stored sitemaps of the website, allowing pages outside of web crawls to be evaluated as well.

Additionally, instead of looking at orphaned pages with an HTTP OK (200) status code, other responses might be interesting, such as status code 50x responses indicating applications that still exist, but, for example, lost access to their backend database. The filtering of file extensions can also be adjusted, or omitted, allowing the detection of other orphaned resources like (exposed) documents.

6.1.2 Use by Red Teams Security professionals, like pentesters, can also benefit from using our technique. Common tools like DirBuster or Gobuster can detect (unintentionally) accessible pages on a domain, but rely on the use of wordlists and enumeration. Our technique is complementary: it provides a targeted-search on a domain. It identifies specific end points potentially not included in common word-lists and provides an indication of pages that are not maintained anymore, meaning they are likely more vulnerable. Similarly, Google Dorks [33] are commonly used for identifying orphaned or hidden sites. However, as we have shown in Section 4.8, our approach largely covers pages *not* indexed by Google.

6.1.3 Future Research Our technique provides the first baseline for studying the prevalence of orphaned web-pages on the web. Further research should refine our technique to also provide an upper bound of orphaned resources. Similarly, we do not investigate the impact of other resources, such as images, videos, PDF files, etc. Such (forgotten) resources can potentially expose private data, especially when encountering PDF or Excel (XLS) files that are not meant to be accessible online (anymore).

While our technique for detecting orphaned web page is effective, we believe different approaches in detecting these pages are worthy of exploring. For example, old blogs, forum posts, or Twitter tweets, can potentially contain orphaned URLs. Extracting those from social media platforms could be more robust against the limitations (crawl heuristics and depth) of web archives. In addition, other historic information sources may be valuable in identifying orphaned non-web resources, that is, forgotten servers, virtual machines, and services. Specifically, we believe that Certificate Transparency (CT) logs and passive DNS data sets offer an opportunity to identify further types of orphaned resources.

Finally, orphaned web pages are a form of misconfigurations (see Section 7). Surveying website administrators after they used our technique can also yield interesting insights into the prevalence (on a per-domain basis) and root-causes of orphaned resources.

6.2 Ethics

Our work uses active measurements. As such, we are following established best practices as outlined in the Menlo report [4, 13]. We took precautions to limit the operational impact of our work on the web archive in coordination with the operators, and on the measured sites by limiting and randomizing the number of requests a site receives (see Section 3.2.3). We also included contact information in the user agent of all web requests that we made. Two parties opted-out of our research, and we excluded them from it.

Our HREC (Human Research-Subject Ethics Council) does currently not accept applications for ethical evaluations or waivers for research that does not directly involve human subjects. Hence, for the automated security scans we carefully considered the trade-off between the utility of our research vs. the potential harm we cause. As we made sure to only *detect* vulnerabilities and did not exploit them, we consider our approach ethical.

We disclosed all XSS and SQLi vulnerabilities we found on orphan pages or pages in our control groups to the responsible administrators, or to the respective CERT when no direct contact information was available. We were able to identify points of contact for 327 vulnerable pages, with some contacts covering multiple pages (i.e. from the same website). Our email for contacting the respective administrators can be found in Appendix C. We received one response from a national CERT, who did not share further information about how they remediated the vulnerabilities, and one response from an operator, who informed us that the affected site was indeed orphaned, and would be taken offline. Furthermore, there was one response from an operator with an affected site in the non-orphaned control group, who reported to apply mitigations within a month from notification.

6.3 Limitations

Our study has several limitations which are important to note for correctly assess the impact of our results. Specifically, we are trying to establish a *lower bound* for orphaned web-pages. We believe it is important to state a lower bound to prevent an alarmist response and alert fatigue. Our implementation provides an accessible aid for users to assess the severity of this problem on their domain, while simultaneously inviting further research. This means that all heuristics employed in our study (Dynamic URL Detection heuristic and fingerprint comparison) are configured conservatively to reduce the chance of over-matching. However, it also inevitably causes us to miss potentially orphaned web pages, meaning orphaned web resources are like more prevalent.

The same holds for our methods for detecting copyrights, following redirects, and loading frames. Developing more ways of following redirects and frames can provide additional depth to our page type analysis. Similarly, detecting more copyright statements (for example, different copyright standards, or in-code construction) can improve accuracy. Especially the detection of *dynamic* copyright statements can contribute to expanding our results.

Finally, we conducted our study on a sample of 100,000 domains from the top 500,000 entries of the Tranco Top 1M. The prevalence of orphaned pages may differ for less prominent domains.

7 Related Work

Two areas of research are closely related: work on orphaned Internet resources (beyond the web) and use-after-free attacks on them, and work on maintenance and security misconfigurations.

7.1 Orphaned Resources

Khalafut et al. studied the prevalence of orphaned DNS servers on the Internet [20], with orphaned DNS servers being those with a DNS record pointing to them, but with no delegation from a parent zone. Overall, 1.7% of records belong to orphaned DNS servers. Liu et al. [25] demonstrate how through "Dangling DNS Records" domains can be hijacked for malicious purposes. Borgolte et al. [6] investigated how DNS records pointing to cloud IP addresses can lead to domain takeover attacks because IP address use-after-free attacks on cloud infrastructure and they show that these attacks are practical and cost effective to execute on public clouds. Stale `NS` record types have been studied by Alowaisheq et al. [1], who show how they can be exploited via DNS hosting providers, possibly allowing domain hijacking for 628 domains of the Alexa's 1M. Beyond DNS, Gruss et al. [16] show that use-after-free attacks can also apply to email addresses.

Related to the web, Aturban et al. examine the existence of orphaned annotations in Hypothes.is, a tool to annotate websites [3]. An annotation is orphaned when it can no longer be attached to its target page, because the page does not exist anymore, or because the annotated content on the web page has been removed. They found that 27% of annotations are orphaned, and 3% could be reattached to archived versions of the target page by making use of web archives. This indicates the potential of using web archives to obtain orphaned data; a concept we also leverage in this paper.

7.1.1 Orphaned Resources vs. Use-After-Free Related work on orphaned resources focuses on resources that are no longer allocated themselves, but still are delegated from another service. Terminology for this situation ranges from stale, dangling, use-after-free, to orphaned, with the underlying concept being the same. In contrast to these broader concepts, we focus on resources and services that lost their delegation, while continuing to exist, that is, the inverse kind of orphaned resources. In our work, the orphaned resource (web pages) cannot be re-used or hijacked when an attacker takes over an orphaned delegation. Instead, the resource itself becomes the liability, for example, due to its vulnerabilities. Orphaned web pages are, however, still under the original owner's control. Other terms, such as use-after-free, signify that control has been relinquished and the resource can be taken over for exploitation.

7.2 Update Behavior and Misconfigurations

Orphaned web pages may be a form of security misconfigurations, which have been extensively studied in the human factors in computer security community [21]. Dietrich et al. studied the perspective of system administrators on security misconfigurations [12]. We see parallels between the issues they uncovered and potential causes for orphaned pages, such as a lack of documentation (non-updated sitemaps), lack of responsibility (who controls which page), and unclear processes and procedures (unplanned changes of pages). The work of Li et al. also suggests that orphaned pages might be an issue of misconfiguration [24]. Albeit we confirmed the prevalence of orphaned pages, future work needs to confirm if its origin is in misconfigurations.

Security misconfigurations are regularly studied with at-scale measurements. For example, Continella et al. measured misconfigurations around Amazon S3 cloud storage services [10], finding 14% S3 storage buckets being public. Ferrari et al. similarly studied misconfigured NoSQL services [14]. Earlier, Springall et al. measured misconfigured FTP servers, with 8% of 13.8M FTP servers allowing for anonymous access [30]. A study by Bijmans et al. finds over 1M unmaintained MikroTik routers vulnerable to embedding cryptomining code in user traffic [5]. Also DNS(SEC) misconfigurations have been shown to cause unavailability and security issues [11, 17, 34].

These studies underline how misconfigurations are still a frequent cause for security issues, and are occurring in the wild.

8 Conclusion

In this paper, we introduce the concept of orphaned web pages as a security risk and develop a novel methodology to detect them. Using a sample of 100,000 websites, stemming from the 500,00 highest ranked sites in the Tranco Top 1M, we found 1,953 pages that are likely orphaned, spread across 907 domains, with some of these pages being as old as 20 years.

We find that orphaned pages are significantly ($p < 0.01$ using χ^2) more likely to be vulnerable to XSS (19.3%) and SQLi (10.8%) vulnerabilities than maintained pages (2.0% for XSS and 2.7%). Furthermore, maintained pages on sites that host orphans are also more likely to contain XSS (5.9%) and SQLi (9.5%) than maintained pages, leading to a clear hierarchy: Orphaned pages are the most vulnerable, followed by maintained pages on websites with some orphans, and least vulnerable are other maintained pages.

Acknowledgments

We thank the Internet Archive for providing open access to their data, which made our measurements possible.

This material is based on research supported by the European Commission (EC) under grant CyberSecurity4Europe (#830929), the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Dutch Research Council, NWO) under grants RAPID (CS.007) and INTERSECT (NWA.1160.18.301), the Deutsche Forschungsgemeinschaft (German Research Foundation, DFG) under Germany's Excellence Strategy (EXC 2092 CASA 390781972), and the Wiener Wissenschafts-, Forschungs-, und Technologie Fund (Vienna Science and Technology Fund, WWTF) under grant IoTIO (ICT19-056).

Any views, opinions, findings, recommendations, or conclusions contained or expressed herein are those of the authors and do not necessarily reflect the position, official policies, or endorsements, either expressed or implied of their host institutions, the Internet Archive, or those of the EC, NWO, DFG, or WWTF.

References

[1] E. Alowaisheq, S. Tang, Z. Wang, F. Alharbi, X. Liao, and X. Wang. "Zombie Awakening: Stealthy Hijacking of Active Domains through DNS Hosting Referral." In: *Proceedings of the 27th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 11/2020. DOI: 10.1145/3372297.3417864.

[2] I. Archive. *About the Internet Archvie*. URL: https://archive.org/about/ (visited on 04/15/2021).

[3] M. Aturban, M. Nelson, and M. Weigle. "Quantifying Orphaned Annotations in Hypothes.is." In: *Proceedings of the 19th International Conference on Theory and Practice of Digital Libraries (TPDL)*. 09/2015. DOI: 10.1007/978-3-319-24592-8_2.

[4] M. Bailey, D. Dittrich, E. Kenneally, and D. Maughan. "The Menlo Report." In: *IEEE Security & Privacy* 10.2 (03/2012), pp. 71–75. DOI: 10.1109/MSP.2012.52.

[5] H. L. Bijmans, T. M. Booij, and C. Doerr. "Just the Tip of the Iceberg: Internet-Scale Exploitation of Routers for Cryptojacking." In: *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 10/2018. DOI: 10.1145/3319535.3354230.

[6] K. Borgolte, T. Fiebig, S. Hao, C. Kruegel, and G. Vigna. "Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates." In: *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS)*. 02/2018. DOI: 10.14722/ndss.2018.23327.

[7] G. Born. *Deloitte-Seite 'Test your Hacker IQ' legt Benutzerdaten offen*. 11/06/2020. URL: https://www.borncity.com/blog/2020/11/06/deloitte-seite-test-your-hacker-iq-legt-benutzerdaten-offen/ (visited on 03/31/2021).

[8] A. Broder, S. Glassman, M. Manasse, and G. Zweig. "Syntactic Clustering of the Web." In: *Comput. Networks* 29.8-13 (1997), pp. 1157–1166. DOI: 10.1016/S0169-7552(97)00031-7.

[9] M. Charikar. "Similarity estimation techniques from rounding algorithms." In: *Proceedings of the 34th Symposium on Theory of Computing)*. 05/2002. DOI: 10.1145/509907.509965.

[10] A. Continella, M. Polino, M. Pogliani, and S. Zanero. "There's a Hole in that Bucket!: A Large-scale Analysis of Misconfigured S3 Buckets." In: *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*. 12/2018. DOI: 10.1145/3274694.3274736.

[11] T. Dai, H. Shulman, and M. Waidner. "DNSSEC Misconfigurations in Popular Domains." In: *Proceedings of the 15th International Conference on Cryptology and Network Security*. 11/2016. DOI: 10.1007/978-3-319-48965-0_43.

[12] C. Dietrich, K. Krombholz, K. Borgolte, and T. Fiebig. "Investigating System Operators' Perspective on Security Misconfigurations." In: *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 10/2018. DOI: 10.1145/3243734.3243794.

[13] D. Dittrich and E. Kenneally. *The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research*. Tech. rep. U.S. Department of Homeland Security, 08/2012. URL: https://www.dhs.gov/sites/default/files/publications/CSD-MenloPrinciplesCORE-20120803_1.pdf.

[14] D. Ferrari, M. Carminati, M. Polino, and S. Zanero. "NoSQL Breakdown: A Large-scale Analysis of Misconfigured NoSQL Services." In: *Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC)*. 12/2020. DOI: 10.1145/3427228.3427260.

[15] Google. *Programmable Search Engine*. 11/10/2020. URL: https://developers.google.com/custom-search/docs/overview (visited on 04/13/2021).

[16] D. Gruss, M. Schwarz, M. Wübbeling, S. Guggi, T. Malderle, S. More, and M. Lipp. "Use-After-FreeMail: Generalizing the Use-After-Free Problem and Applying it to Email Services." In: *Proceedings of the 27th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 11/2020. DOI: 10.1145/3196494.3196514.

[17] L. Hendriks, P.-T. de Boer, and A. Pras. "IPv6-specific misconfigurations in the DNS." In: *Proceedings of the 13th International Conference on Network and Service Management (CNSM)*. 11/2017. DOI: 10.23919/CNSM.2017.8256036.

[18] M. R. Henzinger. "Finding near-duplicate web pages: a large-scale evaluation of algorithms." In: *Proceedings of the 29th Conference on Research and Development in Information Retrieval)*. 08/2006. DOI: 10.1145/1148170.1148222.

[19] internetarchive. *Wayback CDX Server API - BETA*. 08/07/2013. URL: https://github.com/internetarchive/wayback/tree/master/wayback-cdx-server (visited on 04/13/2021).

[20] A. Kalafut, M. Gupta, C. Cole, L. Chen, and N. Myers. "An empirical study of orphan DNS servers in the internet." In: *Proceedings of the 10th Internet Measurement Conference (IMC)*. 11/2010. DOI: 10.1145/1879141.1879182.

[21] M. Kaur, M. van Eeten, M. Janssen, K. Borgolte, and T. Fiebig. "Human Factors in Security Research: Lessons Learned from 2008-2018." In: *arXiv preprint arXiv:2103.13287* (2021).

[22] V. Le Pochat, T. van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation." In: *Proceedings of the 26th Network and Distributed System Security Symposium (NDSS)*. 02/2019. DOI: 10.14722/ndss.2019.23386.

[23] F. Li and V. Paxson. "A Large-Scale Empirical Study of Security Patches." In: *login Usenix Mag.* 43.1 (2018).

[24] F. Li, L. Rogers, A. Mathur, N. Malkin, and M. Chetty. "Keepers of the Machines: Examining How System Administrators Manage Software Updates For Multiple Machines." In: *Proceedings of the 15th Symposium On Usable Privacy and Security (SOUPS)*. 08/2019.

[25] D. Liu, S. Hao, and H. Wang. "All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records." In: *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 10/2016. DOI: 10.1145/2976749.2978387.

[26] J. Luft and H. Ingham. "The Johari Window: a graphic model of awareness in interpersonal relations." In: *Human relations training news* 5.9 (1961), pp. 6–7.

[27] G. Manku, A. Jain, and A. Sarma. "Detecting near-duplicates for web crawling." In: *Proceedings of the 16th World Wide Web Conference (WWW)*. 06/2007. DOI: 10.1145/1242572.1242592.

[28] W. Rweyemamu, T. Lauinger, C. Wilson, W. Robertson, and E. Kirda. "Clustering and the Weekend Effect: Recommendations for the Use of Top Domain Lists in Security Research." In: *Proceedings of the 20th Passive and Active Measurement (PAM)*. 03/2019. DOI: 10.1007/978-3-030-15986-3_11.

[29] Q. Scheitle, O. Hohlfeld, J. Gamba, J. Jelten, T. Zimmermann, S. Strowes, and N. Vallina-Rodriguez. "A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists." In: *Proceedings of the 18th Internet Measurement Conference (IMC)*. 11/2018. DOI: 10.1145/3278532.3278574.

[30] D. Springall, Z. Durumeric, and J. A. Halderman. "FTP: The Forgotten Cloud." In: *Proceedings of the 46th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 06/2016. DOI: 10.1109/DSN.2016.52.

[31] N. Surribas. *Wapiti: The Web Application Vulnerability Scanner*. 02/20/2021. URL: https://wapiti.sourceforge.io/ (visited on 04/30/2021).

[32] C. Tiefenau, M. Häring, K. Krombholz, and E. von Zezschwitz. "Security, Availability, and Multiple Information Sources: Exploring Update Behavior of System Administrators." In: *Proceedings of the 16th Symposium On Usable Privacy and Security (SOUPS)*. 08/2020.

[33] F. Toffalini, M. Abbí, D. Carra, and D. Balzarotti. "Google Dorks: Analysis, Creation, and New Defenses." In: *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. 07/2016. DOI: 10.1007/978-3-319-40667-1_13.

[34] N. L. M. Van Adrichem, A. R. Lua, X. Wang, M. Wasif, F. Fatturrahman, and F. A. Kuipers. "DNSSEC Misconfigurations: How Incorrectly Configured Security Leads to Unreachability." In: *Proceedings of the 3rd Joint Intelligence and Security Informatics Conference*. 09/2014. DOI: 10.1109/JISIC.2014.12.

[35] M. Vermeer, J. West, A. Cuevas, S. Niu, N. Christin, M. van Eeten, T. Fiebig, C. Hernandez Ganan, and T. Moore. "SoK: A Framework for Asset Discovery: Systematizing Advances in Network Measurements for Protecting Organizations." In: *Proceedings of the 6th IEEE European Symposium on Security & Privacy (EuroS&P)*. 09/2021.

A List of Keywords for Classifying Types of Web Pages

A.1 Redirects

- `window.location`
- `top.location`
- `http-equiv=``refresh"`
- `Redirect(`
- `Object moved to`

A.2 Page Not Found

- `404 Not Found`
- `404 Error`
- `Page Not Found`

A.3 Frames

- `<iframe`
- `<frame`
- `<frameset`

B List of File Extension Excluded from Probing

- .3g2
- .3gp
- .3gp2
- .3gpp
- .3gpp2
- .ai
- .avi
- .bmp
- .css
- .dib
- .eot
- .eps
- .f4a
- .f4b
- .f4v
- .flv
- .gif
- .heic
- .heif
- .ico
- .ind
- .indd
- .indt
- .j2k
- .jfi
- .jfif
- .jif
- .jp2
- .jpe
- .jpeg
- .jpf
- .jpg
- .jpm
- .jpx
- .js
- .m4a
- .m4b
- .m4p
- .m4r
- .m4v
- .mj2
- .mov
- .mp2
- .mp3
- .mp4
- .mpe
- .mpeg
- .mpg
- .mpv
- .oga
- .ogg
- .ogv
- .ogx
- .pdf
- .png
- .psd
- .qt
- .svg
- .svgz
- .swf
- .tif
- .tiff
- .ttf
- .webm
- .webp
- .wma
- .wmv
- .woff

C Vulnerability Disclosure Email

Dear Sir/Madam,

I am contacting you because you are the closest security contact that could be found for <Affected Domain>. I am a researcher at <INSTITUTION>, and we recently performed a measurement study, using 100,000 domains taken from the Tranco top 1M list, on the prevalence of orphaned web pages in the wild.

During this study, some pages from your domain <DOMAIN> were identified as orphaned. To understand the security implications of these types of pages, we ran a rudimentary vulnerability scan (using Wapiti) on the found orphan pages, including the ones on your domain.

During the scan, potential vulnerabilities were detected on the following pages:

########## XSS
http://....

########## SQLi
http://....

At no point in our study did we try to exploit the vulnerability on your domain, neither did we share this data with any third party. Through this email, we hope to make you aware of this potential vulnerability such that adequate measures from your side can be taken to investigate it.

If you have any further questions, please feel free to reach out.

Kind regards,
<Authors>