Knock and Talk: Investigating Local Network Communications on Websites

Dhruv Kuchhal
dkuchhal@gatech.edu
Georgia Institute of Technology
USA

Frank Li
frankli@gatech.edu
Georgia Institute of Technology
USA

ABSTRACT

Modern webpages are an amalgamation of resources requested from various public Internet services. In principle though, webpages can also request resources from localhost and devices in the LAN, providing a degree of internal network access to external entities. Prior work has demonstrated how this access can be used for supporting web attacks, particularly for profiling and fingerprinting users.

In this paper, we empirically investigate if and how popular websites are interacting with their visitors' localhost and LAN resources, and compare the behavior observed to that from known malicious websites. We crawl and monitor the network requests made by the landing pages of domains in the Tranco top 100K domains as well as 145K websites that are known to be related to malware, phishing, or abuse. For both popular and malicious sites, we detect over 100 sites in each category making requests to internal network destinations, including several highly-ranked sites (within the top 10K). Investigating these sites in-depth, we identify that over 40% of the ones from the top 100k list, do so to conduct host profiling, purportedly for fraud and bot detection. We also uncover cases of legitimate native application communication and likely developer errors. For malicious sites, we do not detect cases of internal network attacks. Rather, we believe that the malicious sites generating local network traffic are compromised or cloned phishing websites, and that the traffic results from the corresponding benign sites. We observe significantly more local activity when on the Windows OS, compared to Linux or Mac OS X, as well as extensive use of WebSockets, which are not bound by the Same-Origin Policy. Ultimately, our exploration provides an empirical grounding on the localhost and LAN activities of websites, revealing both intentional and unintentional behavior.

CCS CONCEPTS

• **Networks** → **Network measurement**; *Network privacy and anonymity*; • **Security and privacy** → **Network security**.

ACM Reference Format:
Dhruv Kuchhal and Frank Li. 2021. Knock and Talk: Investigating Local Network Communications on Websites. In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3487552.3487857

IMC '21, November 2–4, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9129-0/21/11…$15.00
https://doi.org/10.1145/3487552.3487857

1 INTRODUCTION

Websites today stitch together web resources from numerous sources. Beyond its own first-party content, a website fetches various third-party resources such as JavaScript (JS) libraries, CSS style sheets, images and videos, and even other webpages via inline frames. Typically, we think of these third-party resources as hosted by public Internet services, including other web servers, content distribution networks, and API endpoints. However, nothing in modern web browsers prevents a website from also attempting to communicate with internal network services, such as those on the browser's localhost and other devices within the Local Area Network (LAN), the addresses as defined in the RFC1918 [50]. For example, a website could include JS code that sends a request to a service running on a visitor's localhost. As a consequence, a browser affords a degree of internal network access to external entities.

Why would websites need to communicate with local destinations, given the uncertainty and diversity of local devices and network services? One legitimate reason might be for a website to coordinate with an affiliated native application (e.g., visiting a video meeting URL opens the native video conferencing software). However, prior work [4, 23, 31, 39, 40, 54] has identified that in theory, such access could also be used for potentially malicious purposes. These studies have developed proof-of-concept demonstrations of user and LAN device fingerprinting, as well as network attacks on internal services. Particularly with user profiling and tracking, websites and advertisers have escalated to ever more complex and surreptitious methods for gathering user identifiable information [3], and one can plausibly envision the adoption of these internal network based approaches. The proliferation of consumer IoT devices [38] further exposes users to potential security and privacy violations.

In this paper, we empirically evaluate the local network behaviors of both popular and malicious websites, at scale. We crawl and monitor the requests made by landing pages of the Tranco top 100K domains [47], performing two sets of measurements, half a year apart. We similarly crawl and monitor requests made by 145K malicious websites drawn from abuse, malware, and phishing blacklists. Across these websites, we assess if they generate locally-bound traffic, what the nature of the local traffic is, and why they may be communicating with internal network services. As different OSes support varying network services, a website's locally-bound traffic may depend on the underlying host OS. Thus, we also explore how websites' localhost and LAN behavior may differ across three popular desktop OSes: Windows, Ubuntu, and Mac OS X.

In total, we measured landing pages of over 100 benign domains and over 150 malicious websites communicating with localhost services or private network IP addresses. While this population may be relatively small, indicating this behavior is not widespread

(yet) among websites, it is a non-trivial number of sites exhibiting rather unexpected network activity. Furthermore, several of the observed websites are highly ranked with millions of users, including 19 sites ranked within the top 10K. We identify that for many sites, the internal network activity is not uniform across OSes, particularly skewing towards activity exclusively on Windows. Additionally, we note the extensive use of WebSockets for initiating such communication, which bypasses the Same-Origin Policy. These characteristics potentially suggest the intentional targeting of certain platforms.

To understand why these websites may be contacting services hosted on the local network, we manually analyze their behavior. For popular websites, we find that over 40% of them explicitly conduct host profiling. Upon deeper investigation, we determine that this profiling is performed for fraud protection and bot detection, rather than explicit advertising or user tracking purposes (although their approach could be readily adapted for such uses). We also observe a large set of top websites whose local network activity arises due to remnants of website development and testing. These cases should be benign, but highlight a class of web developer errors that should be straightforward to detect and remediate. Finally, we also identify a legitimate scenario for localhost communication where a top website communicates with its affiliated native application. Efforts to protect users from malicious local network traffic generated by a website must preserve these valid use cases.

For malicious sites, we do not find evidence of internal network attacks. Rather, the observed local network traffic mirrors that of top sites. We detect that some phishing sites have cloned their web interfaces from legitimate sites, where the legitimate site itself is conducting host profiling. As a consequence, the phishing site hosts the same JavaScript library as the target site, and thus also generates the same local network traffic. For the other malicious sites, we find that the local network activity reflects the same sorts of web developer errors seen on top sites, indicating that either the attackers made similar errors while developing their attack sites, or that perhaps more likely, these sites are compromised and the local traffic reflects the developer errors exhibited on the original benign sites. Overall, we do not attribute malicious intent to the local network traffic witnessed on these malicious sites.

Ultimately, our study provides the first large-scale measurement of the internal network behaviors of modern websites, uncovering both intentional and unintentional causes of locally-destined traffic. We conclude by discussing the implications of our findings on web security and privacy, as well as potential directions for advancing user online safety.

2 RELATED WORK

Here we summarize the prior work on developing web-based methods for discovering, fingerprinting, and attacking LAN devices, along with prior work empirically evaluating security and privacy behaviors of websites.

2.1 Web-based LAN Attacks

Web-based methods for identifying, profiling, and attacking devices on a user's LAN have existed for years. Over a decade ago, Lam et al. [39] demonstrated how a malicious webpage could scan for and exploit vulnerabilities on LAN devices, potentially propagating worms and profiling users. Grossman and Niedzialkowski [31] similarly presented a web-based method for scanning the local network using the `onerror` JavaScript handlers of image resources. Stamm et al. [54] leveraged this scanning method to hijack the DNS configurations of local routers. More recently, Gallagher [22, 23] refined web-based LAN scanning by using WebSockets and WebWorkers, and Lee et al. [40] did likewise, leveraging HTML5 AppCache to identify cross-origin resource statuses. Acar et al. [4] further built upon these advancements to demonstrate how a webpage can discover and interact with local IoT devices that expose HTTP interfaces, potentially triggering malicious commands as well as fingerprinting the network. Beyond these academic studies, several different web-based network and port scanners have been developed, demonstrating the feasibility of web-based targeting of LAN devices [5, 7, 27, 34, 42, 45, 60].

These works highlight the potential threats that websites pose given internal network access. However, we currently lack awareness of whether websites actually leverage this access in practice. Our study establishes empirical ground on the localhost and LAN network behaviors of modern websites.

2.2 Measurements of Website Security and Privacy Behavior

Numerous empirical studies have evaluated the security and privacy behaviors of websites. Prior work has examined how websites have deployed various security mechanisms. For example, Felt et al. [21] tracked the deployment and configuration of HTTPS across the web. Similarly, Stark et al. [55] evaluated the adoption of certificate transparency on websites. Both Calzavara et al. [9] and Roth et al. [52] examined the content security policies deployed by websites, evaluating their effectiveness in practice.

On the privacy side, existing studies have analyzed the real-world data collection and web tracking mechanisms deployed by websites. As an example, Englehardt et al. [14] developed an open-source web privacy measurement tool and used it to measure different types of tracking on the top 1 million websites. Similarly, Acar et al. [3] uncovered how websites were using surreptitious techniques for profiling and tracking users longitudinally, including through canvas fingerprinting, evercookies, and cookie syncing. Snyder et al. [53] investigated the browser features used by popular websites for advertising and tracking purposes.

As with these prior studies, our work seeks to understand the security and privacy behaviors of websites. Expanding beyond the existing explorations though, we explore how websites interact with localhost and LAN network services, and the security and privacy implications of this behavior.

3 METHODOLOGY

In this section, we describe our method for examining if and how websites interact with users' localhost and LAN resources. We consider two sets of websites, one consisting of popular sites, and the other consisting of known malicious webpages (e.g., phishing sites).

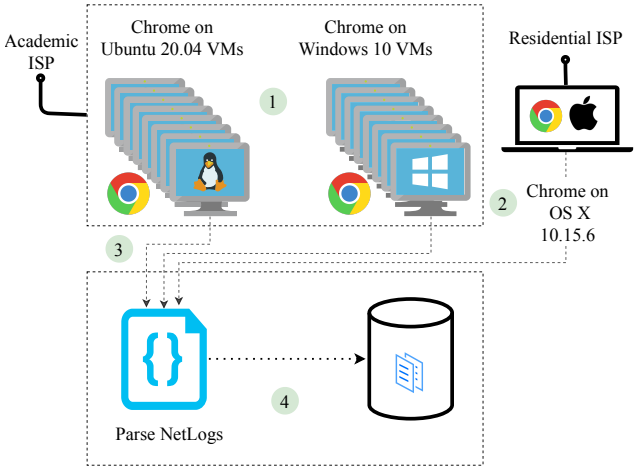


Figure 1: Data collection methodology: 1) We crawl target webpages (i.e., landing pages of top domains or malicious webpages) using full Google Chrome instances running in Linux and Windows VMs on an academic ISP (i.e. Georgia Tech). 2) We similarly conduct a Chrome-based web crawl on a MacBook Air laptop on a residential network (i.e. Comcast) in Atlanta. 3) During page visits, we collect all network events as logged by Chrome's network logging system, before 4) parsing the logs and storing the network events.

3.1 Data Collection

Measurement Population. To study popular websites, we focus on the top 100K domains in the Tranco toplist [47]. This set of domains affords a large-scale exploration of top websites that users visit. We conduct two sets of measurements of the landing pages of the top 100K domains – one using the Tranco snapshot taken on June 3, 2020, and the other taken on March 11, 2021. To evaluate the local network behavior of malicious webpages, we collect ~145K known malicious URLs that were actively listed on one of several blocklists: SURBL (abuse, malware, and phishing sites) [56], Abuse.ch's URLHaus (malware sites) [2] and PhishTank (phishing sites) [46]. As these blocklists often list multiple malicious URLs mapping to the same domain, we only select one malicious URL per domain to increase our measurement's coverage of malicious domains. We monitored and crawled webpages on the blocklists from March – April 2021.

Measurement Setup. As depicted in Figure 1, for both the landing pages of top websites and malicious webpages, we fetch and render (with JavaScript enabled) the target webpage using a full browser instance running within a virtual machine (VM), while monitoring the network requests generated by the website. Our web crawler starts a full Google Chrome (v84) instance with a clean profile (incognito mode) via Chrome's command-line interface[1]. As we are exploring how a website may interact with localhost and LAN resources, which are external to the browser, but may vary depending on the underlying OS, we visit all websites across three popular desktop OSes: Windows 10, Linux (specifically Ubuntu 20.04), and Mac OS X (v10.15.6). We conducted the Windows 10 and Linux web crawls within VMWare VMs, running on a server within Georgia Tech's network. For the web crawl on Mac OS X, we performed the crawl directly on a MacBook Air laptop residing on a residential Comcast network in Atlanta.[2] Throughout our measurements, we disable Chrome's Safe Browsing feature, which blocks network requests to destinations on Google's Safe Browsing blocklist [30]. This configuration is necessary to ensure that Safe Browsing does not interfere with our measurements, such as blocking the browser instance from visiting a malicious page.

Web Telemetry. Across all three OSes, for each target webpage, we load the page and monitor its network activity over a 20 second period, to permit time for execution of JS code and loading of dynamic resources. To select this delay threshold, we randomly sampled 100 websites (landing pages of domains from the top 100k list), and noted the time it takes to complete loading the page – we found that more than 98% of all requests (to any resource, local or otherwise) were made within the first 15 seconds, with the majority being made within the first 5 seconds. Prior research also suggests that users often spend 10–20 seconds on a web page before navigating elsewhere [43]. Therefore, to measure the most common effect of these websites on users, we chose a delay threshold of 20 seconds. Since our analysis of the first crawl (in Section 4.2) suggested that this threshold suffices in detecting the majority of localhost and LAN activity, we continued with the threshold in the second crawl. For gathering network telemetry generated by a webpage while it loads, we record the network logs generated by Google Chrome's network logging system [16], which comprehensively logs all network events (i.e., any network requests sent and responses received) on Chrome's network stack. Among other data, this telemetry includes the following for each event:

- `time`: specifies the timestamp for the network event.
- `type`: specifies the type of network event as defined by Chrome. For example, `URL_REQUEST` events indicate GET/POST requests.
- `source`: specifies the entity that generated the event. When a new network request is initiated, it is assigned a new source ID (in serial order). Subsequent dependent events (e.g., responses) are assigned the same source ID, allowing the events within a network flow to be logically grouped together.
- `phase`: specifies the phase of the network event (either `BEGIN`, `END`, or `NONE`).

We parse and store the network logs in a database for efficient querying. From this telemetry, we can comprehensively observe whenever a webpage initiates a request destined for localhost or an IP address in the private IP address range, what precisely that destination is (e.g., the full URL), and what entity generated this request (the Chrome browser itself also generates network traffic, which we filter out based on the network event source). Additionally, we also consider websites that redirect to a local destination, since in theory, websites can send a request to a local resource, even if they can never receive the response.

[1]We avoid using more popular crawling tools, such as Puppeteer [28] and Selenium [29], as a website can detect their presence and change its behavior – potentially contaminating our measurement.

[2]As Mac OS X is licensed only for use on Apple hardware, we conduct our Mac-based crawls directly on a Mac laptop.

Type of Crawl	OS	# successful loads	# failed loads	Error Type				
				NAME_NOT_RESOLVED	*CONN_REFUSED*	*CONN_RESET*	*CERT_CN_INVALID*	*Others*
Top 100K: 2020	Windows	89744 (89.7%)	10256 (10.3%)	9179 (89.4%)	355 (3.5%)	248 (2.4%)	236 (2.3%)	238 (2.3%)
Top 100K: 2021		91765 (91.7%)	8235 (8.2%)	7287 (88.5%)	239 (3.0%)	230 (2.8%)	251 (3.1%)	228 (2.8%)
Top 100K: 2020	Mac	89819 (89.9%)	10181 (10.1%)	9001 (88.4%)	345 (3.8%)	193 (1.9%)	226 (2.2%)	416 (4%)
	Linux	90175 (90.2%)	9825 (9.8%)	8612 (87.6%)	335 (3.4%)	247 (2.5%)	235 (2.4%)	396 (4%)
Top 100K: 2021		91719 (91.7%)	8281 (8.3%)	7309 (88.3%)	272 (3.2%)	126 (1.5%)	248 (3%)	326 (3.9%)
Malicious	Windows	100317 (68.6%)	45864 (31.4%)	40715 (88.7%)	1475 (3.2%)	530 (1.1%)	1341 (2.9%)	1803 (3.9%)
Malicious	Mac	103154 (70.6%)	43027 (29.4%)	37310 (86.7%)	1488 (3.4%)	523 (1.2%)	1314 (3%)	2392 (5.5%)
Malicious	Linux	106078 (72.6%)	40103 (27.4%)	34723 (86.5%)	1346 (3.3%)	521 (1.3%)	1313 (3.2%)	2200 (5.5%)

Table 1: Web crawl statistics. We successfully load ~90% of domain landing pages when crawling the top 100K domains, and ~70% of the 146K malicious webpages. Nearly ~90% of the failures were due to DNS resolution errors (`NAME_NOT_RESOLVED`). The other top errors included connection refusals (`CONN_REFUSED`) and resets (`CONN_RESET`), as well as HTTPS certificate misconfigurations (e.g., `CERT_CN_INVALID`).

Throughout our crawl, before visiting a webpage, we first check for network connectivity by pinging Google's DNS server (`8.8.8.8`). This ensures that we crawl a site only when the measurement infrastructure has Internet connectivity, and thus we can differentiate between website load failures and network issues on our end.

Ethics. As we conduct a large-scale web measurement, we take care to minimize the load induced on the websites we visited. For each OS, we visit each site only once, and we start measurements on each OS at different times. Thus, we visit each site up to three times without concurrent visits, which should be a manageable load for websites.

3.2 Data Characterization

Our final datasets consist of 11 TBs of telemetry data collected from the following measurements:

(1) Tranco Top 100K domains (as of June 3, 2020) on all three OSes, as measured from July 24 to September 25, 2020.
(2) Tranco Top 100K domains (as of March 11, 2021) on Windows and Linux[3], measured from March 22 to May 1, 2021.
(3) 146K known malicious webpages on all three OSes, measured from March 22 to May 1, 2021.

We observe a ~75% overlap between the two Tranco toplists' snapshots we used, indicating that the majority of top domains in our dataset were measured twice half a year apart. In total, we successfully loaded and gathered telemetry from ~90% of domains in both sets of Tranco toplist measurements. This success rate is commensurate with that observed by the creators of the Tranco toplist [47]. For the measurement of malicious webpages, we successfully gathered telemetry from ~70% of webpages. We further manually investigated a random sample of webpages that were not successfully crawled and confirmed that they remained inaccessible, giving us confidence that our collected data should comprehensively cover the accessible websites.

Table 1 shows our crawl success rate statistics, including the top errors when failing to load a domain's landing page. Nearly 90% of errors in all three crawls were due to DNS name resolution failures. Manually examining a random sample of these DNS errors, we observed that the domains indeed lacked name resolutions, but we could identify subdomains (e.g., through search engines) for which name resolutions existed. We note that in many cases, the domains are CDNs or API endpoints. Recall that our periodic connectivity test did not reveal any network outages on our end, indicating that these errors did not arise due to network connectivity issues with our measurement setup. We also explored the rank of the domains that we failed to crawl and observed that they were evenly distributed among the top 100K, indicating the errors do not skew towards high or low-ranked domains.

[3]We encountered logistical issues preventing us from conducting the second Tranco toplist measurement on Mac OS X, related to our need to execute the crawl on a bare-metal environment.

3.3 Limitations

The data collected in this study affords a large-scale evaluation of website-generated network traffic. However, there are several important limitations to our methodology.

- Our measurements are conducted using Google Chrome and three desktop OSes. These browsers and OSes are widely available and popular, however, websites may behave differently on other browsers and OSes (such as mobile OSes). Future work can extend our methodology to evaluate these other platforms.
- We only fetch the landing pages of top domains, and our crawler does not simulate user interactions with the webpages. As a consequence, our data does not capture the behavior of internal website pages, nor activity triggered by user actions. Thus, the number of domains for which we identify local network activity is a lower bound of the true number of websites exhibiting such behavior. We leave a broader exploration of these dimensions for future work.
- We monitor a webpage's behavior for 20 seconds. This threshold was determined based on an experiment we performed, as described in Section 3.1, as well as prior research on user browsing behavior [43]. Our subsequent analysis (in Section 4.2) suggests that this threshold suffices in detecting the vast majority of website localhost and LAN activity. However, we would fail to detect such behavior on websites where the activity commences beyond our 20 second threshold.

Domain Type (Malicious Category)	# Sites	Data Sources (% Contribution)	Crawl Success Rate			Activity Detected (# Sites)					
						Localhost			LAN		
			W	L	M	W	L	M	W	L	M
Malware	103541	Abuse.ch (99%), SURBL (1%)	61%	65%	65%	72	83	75	8	7	7
Abuse	24958	SURBL (100%)	95%	97%	93%	0	0	0	1	1	1
Phishing	16426	PhishTank (85%), SURBL (15%)	73%	76%	69%	25	41	9	0	0	0

Table 2: Summary of localhost and LAN requests found for malicious webpages. Operating Systems (OS) are Windows (W), Linux (L) and Mac (M).

- Due to logistical constraints, our web crawl on Mac OS X was conducted on Comcast's residential network, whereas our Windows and Linux web crawls were performed on Georgia Tech's network. In theory, crawl results could differ due to varying network configurations or network-dependent (including a network's geo-location) website behavior. However, we did not identify significant network-based discrepancies in our web crawl telemetry – neither during our manual investigations of websites nor during our data analysis.

4 FINDINGS

Here, we analyze our datasets on network activity generated by websites, to answer three research questions about the activity destined to a visitor's local network (i.e. localhost or LAN).

- **RQ1:** Which websites are generating local network traffic?
- **RQ2:** What are the characteristics of local network traffic?
- **RQ3:** Why are websites making local network requests?

For localhost activity, we detect requests generated by websites that are destined for either the localhost domain or loopback IP addresses (i.e., `127.0.0.1` for IPv4 and `::1` for IPv6). To identify LAN activity, we identify requests made by websites to IP addresses in the IANA-reserved private IP address ranges for IPv4 and IPv6 [50]. We did not observe any localhost or LAN network traffic over IPv6 though, so subsequent discussion is exclusively for IPv4.

For landing pages of Tranco top 100K domains, we first present the results from the dataset measured in 2020, and then compare our findings to those from the 2021 top 100K measurements. As mentioned in Section 3.2, domains in the Tranco snapshots used for the two measurements overlap extensively, and many findings remained consistent. Thus, for the 2021 top 100K measurements, we focus on highlighting the changes observed. We additionally characterize our observations about malicious webpages in comparison to the top sites.

4.1 RQ1: Which Websites are Generating Local Network Traffic?

During our first measurement of sites from Tranco top 100K in 2020, we found a total of 107 sites making localhost requests (as listed in Tables 5 and 11) and 9 sites generating LAN requests (as shown in Table 6), with no overlap between the two sets of sites. These sites account for 0.12% of the ones successfully crawled, indicating that local network activity on websites is not currently widespread. However, such activity does occur on a non-trivial set of websites.

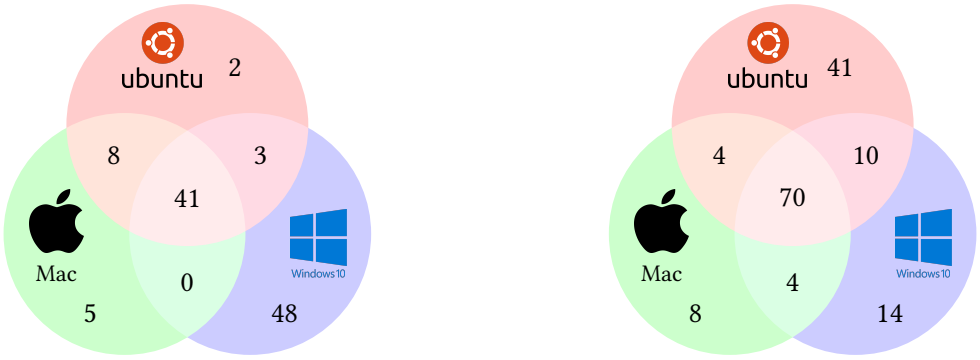
Similarly, in our 2021 top 100K measurements, we observed 82 sites making localhost requests (as listed in Table 7) and 8 sites generating LAN requests (as shown in Table 10), with again no overlap between the two sets of sites. Out of the 82 sites generating localhost requests, 19 sites were crawled in 2020 but were not observed as generating such traffic, whereas 21 sites were not crawled in 2020 (as their domain was not listed in the top 100K). The remaining sites exhibited the same behavior in both measurements. For the sites generating LAN requests, only one site was found performing LAN requests in both 2020 and 2021, while the others from 2020 stopped in 2021.

In our measurement of malicious webpages, we found a total of 151 sites making localhost requests (as listed in Table 8), and 9 sites sending LAN requests (as shown in Table 9). A summary of the malicious crawl statistics can be found in Table 2.

Behavior Across OSes. We observe that the set of websites generating localhost traffic is not uniform across OSes. As shown in Figure 2a, during the 2020 top 100K crawl, we found 92 sites (86%) generating localhost requests when on Windows, compared to 54 sites (50%) for both Linux and Mac. Note though that the set of 54 sites for Linux and Mac are not equivalent, with 5 sites generating activity on Mac but not Linux, and 5 other sites active for Linux but not Mac. Overall, only 41 sites (38%) behaved equivalently on all three OSes. While few sites produced localhost traffic exclusively on Mac (5 sites) and Linux (2 sites), 48 sites (45%) did so on Windows 10, which suggests a degree of targeting towards Windows users (explored further in Section 4.3).

In our 2021 top 100K measurements, we observed that behavior on Windows and Linux was similar, with 47 sites behaving equivalently on both OSes. For malicious webpages, as seen in Figure 2b, we also find that localhost traffic is not observed uniformly across OSes, although here, Linux elicited localhost behavior on more webpages. Similarly, we found that LAN activity on websites is also not consistent across OSes (although the population sizes are small enough to prevent identifying clear OS skew).

Website Rankings. We also investigate whether the population of websites that make local requests skews based on the sites' popularity. In Figure 3, we plot the CDFs of ranks for domains whose landing pages were found to be actively generating traffic to localhost in our 2020 top 100K measurements, across the three OSes. We observe fairly linear CDF curves, indicating that our detected domains are spread uniformly throughout the top 100K domains. Notably, there are highly-ranked sites (with millions of users) that display localhost behavior, such as those listed in Table 3. In our 2021 top 100K measurements, we see a similar distribution (as visible in Figure 9 in the Appendix). As listed in Tables 6 and 10, ranks of domains whose landing pages were found making requests to LAN destinations are also similarly distributed across the top 100K, with the highest site ranked at 4381 in 2020 and 4847 in 2021. None



(a) For the crawl of Tranco top 100K websites in 2020.

(b) For the crawl of malicious websites.

Figure 2: The overlap between localhost network requests generated between OSes.

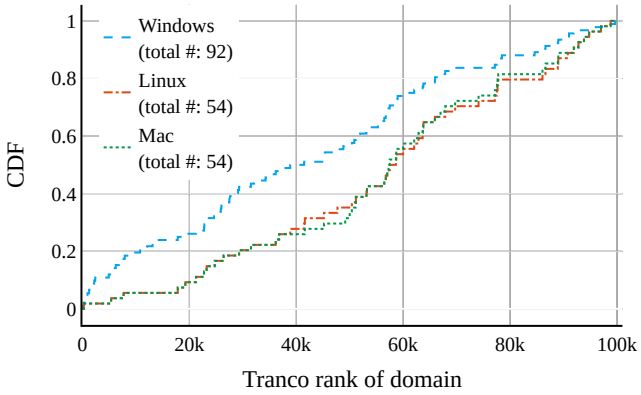


Figure 3: CDFs of domain ranks for Tranco top 100K landing pages generating localhost traffic, across OSes, for our 2020 measurements.

Rank (↓)	Windows	Rank (↓)	Linux and Mac
104	ebay.com	243	hola.org
243	hola.org	5369	faceit.com
429	ebay.de	7699	zakupki.gov.ru*
536	ebay.co.uk	17826	rkn.gov.ru*
932	ebay.com.au	19243	cruze...sulvirtual.com.br*
1250	fidelity.com	21245	wowreality.info
1288	citi.com*	22729	smartcatdesign.net
1843	ebay.it	23218	cponline.pw*
2200	ebay.fr	24739	gamezone.com
2394	ebay.ca	26399	filemail.com

Table 3: Top 10 domains whose landing pages were making localhost requests in our 2020 top 100K measurements (Linux and Mac are identical). Starred domains were not observed generating such traffic in our 2021 top 100K measurements.

of the domains belonging to malicious webpages were found in the Tranco top 100k.

4.2 RQ2: What are the Characteristics of the Local Network Traffic?

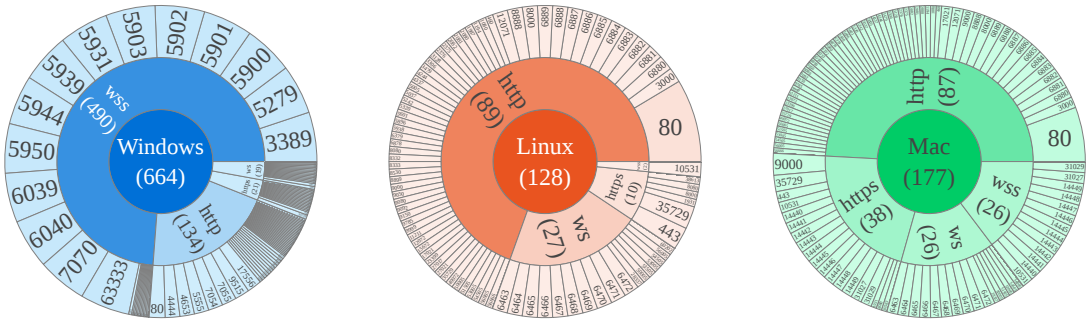
Here we consider what protocols and ports are used by websites during a page load, to make requests to the local network.

Protocols and Ports. Figure 4a depicts the protocols and ports used in the localhost requests we observed in our 2020 top 100K measurements, across the three OSes. (Data on individual sites is listed in Tables 5 and 11.) We observe that the majority (about 60%) of localhost requests on Windows were made using the Secured WebSockets (WSS) protocol, to a set of 14 ports. The use of WebSockets is interesting as it is not bound by the Same-Origin Policy, potentially allowing bidirectional network communication. In comparison, only a quarter of requests on Windows were sent over HTTP, with an eighth sent over HTTPS (primarily to the default HTTP(S) ports). Linux and Mac exhibited the opposite pattern, with 86% of requests sent over HTTP and HTTPS (primarily to the default ports) for both OSes.

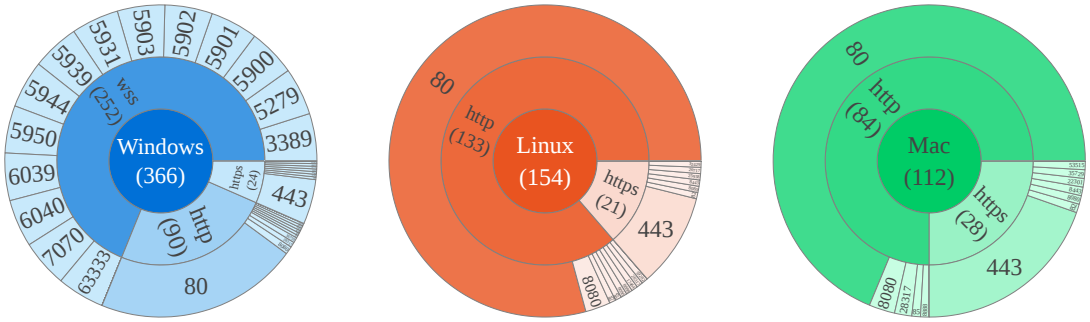
In the 2021 top 100K crawl, as well as the crawl of malicious webpages (shown in Figures 8 and 4b of the Appendix, respectively), we see largely a subset of the ports and protocols observed with the 2020 top 100K crawl. We see fewer ports and protocols in our later measurements due to some changes in the type of local network activity exhibited by websites, as examined further in Section 4.3.

For LAN traffic, as shown in Tables 6 and 10, all requests from top 100K sites (in both the 2020 and 2021 crawls) were made to either HTTP or HTTPS, using the standard ports. This held true for most malicious webpages (as seen in Table 9), except one site requesting HTTP on port 1080.

Request Timing. Figure 5a depicts the CDF of the time between when the landing page is successfully fetched and when we begin observing localhost requests, for our 2020 top 100K measurements. We observe that for Linux and Mac, over half of the localhost requests are initiated within 5 seconds after the landing page is fetched. For Windows, the median gap is 10 seconds. The maximum delay is 14 seconds for Mac and 17 seconds for both Linux and Windows. Similarly, Figure 5b shows the time delay between when the landing page is fetched and when we begin observing LAN requests, for our 2020 top 100K measurements. Here, the median



(a) For the crawl of Tranco top 100K websites in 2020.



(b) For the crawl of malicious websites.

Figure 4: Distribution of website network requests to localhost. The center of each diagram indicates the OS and the total number of requests observed for websites on that OS. The first ring divides these requests by the network protocol/scheme. The outer ring specifies the localhost port number requested.

delay for all three OSes is within 5 seconds. The maximum delay is 5 seconds on Windows, 15 seconds on Mac, and 16 seconds on Linux. These delays reveal that in many cases, the localhost and LAN traffic generate by websites do not immediately manifest.

Our observations remain roughly consistent over our 2021 top 100K crawl and our crawl of malicious webpages (as shown in Figures 6 and 7 in the Appendix).

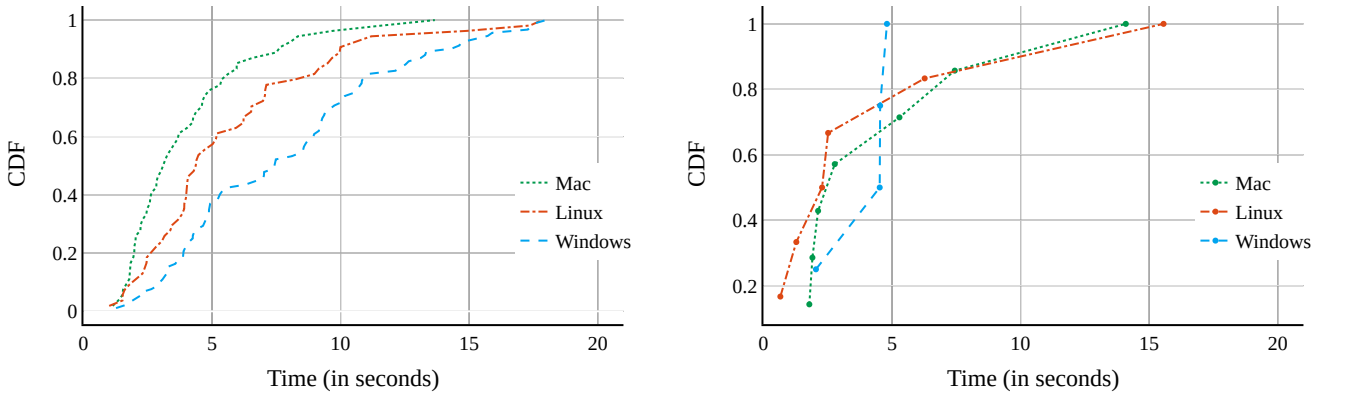
4.3 RQ3: Why are Websites Making Local Network Requests?

To understand why websites may be generating local traffic, here we first manually investigate each website in our 2020 top 100K measurements, identifying which elements of the webpage are originating the localhost and LAN requests in an attempt to determine the underlying reason. We subsequently also compare the observations with those from the 2021 top 100K crawl as well as the crawl of malicious webpages.

Through our exploration of the 107 websites found making localhost requests in our 2020 top 100K measurements, we identify 4 main classes of behaviors: 1) 36 websites were scanning localhost ports for fraud detection, 2) 10 websites were conducting bot detection, 3) 12 websites were found communicating with native applications, and 4) 44 websites request localhost resources likely due to developer error. For the 5 remaining websites, we could not ascertain the cause of the localhost requests (discussed further in Appendix C). The characterization of their domains is listed in Tables 5 and 11.

Port (↓)	Service/App	Used for
3389	Windows Remote Desktop	Fraud Detection
4444	Malware: CrackDown, Prosiak, Swift Remote	Bot Detection
4653	Malware: Cero	
5555	Malware: ServeMe	
5279	Unknown	Fraud Detection
5900-03	Remote Framebuffer (e.g., VNC)	
5931	AMMYY Remote Control	
5939	TeamViewer	
5944	Unknown (likely VNC)	
5950	Cisco Remote Expert Manager	
6039-40	X Window System	
63333	Tripp Lite PowerAlert UPS	
7054-55	QuickTime Streaming Server	Bot Detection
7070	AnyDesk Remote Desktop	Fraud Detection
9515	Malware: W32.Loxbot.A	Bot Detection
17556	Microsoft Edge WebDriver	

Table 4: Common services or malware that operate on the localhost ports scanned for fraud and bot detection purposes.



(a) Requests to services hosted on localhost

(b) Requests to services hosted on LAN devices

Figure 5: CDFs of the time delays between when the landing page is fetched and when we observed the first local network request, across OSes, for our 2020 top 100K measurements.

Among the 9 LAN-requesting websites in our 2020 top 100K measurements, listed in Table 6, we believe that 6 websites are doing so due to developer error. For the remaining three (which all request the HTTP root directory), we were unable to clearly identify an underlying reason (also discussed in Appendix C).

We observe a strict subset of these classes of local network behavior for our 2021 top 100K crawl and the crawl of malicious webpages. Thus in this section, as we discuss each behavior class, we first discuss results from our 2020 top 100K crawl and then draw comparisons between the different datasets.

4.3.1 Fraud Detection. As shown in Table 5, for the first set of rows where the localhost network activity reason is *Fraud Detection*, we encountered 36 websites in our 2020 top 100K crawl that make the same set of WebSocket Secure (WSS) requests to 14 distinct localhost ports, all using the same URL path and operating only on Windows 10.

We determine that these websites are deploying a fraud detection script from LexisNexis called ThreatMetrix [41], which claims to help the websites distinguish between trusted transactions and fraudulent behavior. On each of these websites, we identified that all of the localhost requests were made by a dynamically-generated JavaScript blob object. We observe that these blobs are generated by an external JavaScript resource loaded from either a subdomain (e.g., `regstat.betfair.com` for `betfair.com`) or similar-appearing domain (e.g., `ebay-us.com` for `ebay.com`). The JavaScript blobs collect telemetry from the localhost WSS requests, and upload the results back (in an encrypted format) to the external JavaScript-hosting site (e.g., `ebay-us.com`). Conducting WHOIS lookups on these domains and their IP addresses, we find that these domains all belong to the ThreatMetrix Inc. organization. As advertised on their websites [41], ThreatMetrix provides their customers with protection against fraud using network-based heuristics. This purpose aligns with our observation that all but one (i.e., `commoncause.org`) of the sites deploying ThreatMetrix are e-commerce websites.

We observe that the localhost ports scanned by ThreatMetrix are primarily the standard ports for various remote desktop software for Windows, as shown in Table 4. Ports were mapped to an application based on information from the IANA's Service Name and Transport Protocol Port Number Registry [35] and the SANS Internet Storm Center's TCP/UDP Port Activity Database [10]. We hypothesize that ThreatMetrix is attempting to determine if the website visitor's host machine (focusing specifically on Windows hosts) may be under remote control, potentially correlating with fraudulent activity. As the ThreatMetrix script uses the WebSocket protocol for initiating requests, which is not bound by the Same-Origin Policy and thus permits reading data from the request responses, the script may also be gathering more extensive information about the network services active on each port (e.g., server version and configuration).

We briefly note that as our methodology only examines website landing pages. ThreatMetrix may be more broadly deployed on the internal pages of other websites. Indeed, a recent blog post [1] identified several websites using ThreatMetrix specifically on login pages. We leave the exploration of internal website pages to future work. In comparison to our study, which conducts a large-scale comprehensive and systematic investigation of website localhost and LAN activity, the blog post's investigation was an ad-hoc and manual one that considered a few websites suspected to be using ThreatMetrix. The set of landing pages we identify as using ThreatMetrix is a superset of those found in the blog post.

In our 2021 top 100K measurements, we continue to observe this class of behavior for popular websites. We do note changes in the websites deploying ThreatMetrix though (as indicated in Tables 5 and 7), as some websites stopped (e.g., `citibank.com`) while some other previously crawled sites began exhibiting its fraud detection localhost traffic (e.g., `cibc.com`). Interestingly, we also recorded a set of phishing sites making localhost requests matching with ThreatMetrix's behavior (as seen in Table 8). While investigating these phishing pages, we identify that they impersonate legitimate websites using ThreatMetrix. We believe that the attackers have cloned the web interface code of the legitimate impersonated websites (e.g., `customer-ebay.com` cloning `ebay.com`), inadvertently also copying over the ThreatMetrix JavaScript library (resulting in the phishing page generating localhost requests). Prior work has documented similar cloning behavior for phishing websites [57].

Reason	Rank (↓)	Domain	Protocol	Ports	Paths	OS		
						W	L	M
Fraud Detection	105-45156	ebay.{at, ca, ch, co.uk, com, com.au, com.my, com.sg, de, es, fr, ie, in, it, nl, ph, pl, us}	wss	3389, 5279, 5900-03, 5931, 5939, 5944, 5950, 6039-40, 63333, 7070	/	✓		
	1251	fidelity.com				✓		
	1289-7907	citi{.com, bank.com, bankonline.com}*				✓		
	5680	marktplaats.nl*				✓		
	7441	betfair.com				✓		
	13119-57251	tiaa{.org, -cref.org}*				✓		
	13901	2dehands.be*				✓		
	25990	santanderbank.com				✓		
	29104	ameriprise.com				✓		
	34251	commoncause.org*				✓		
	45228	ctfs.com*				✓		
	50853	2ememain.be*				✓		
	90641	highlow.net				✓		
	97182	metagenics.com				✓		
Bot Detection	8608	sbi.co.in*	http	4444, 4653, 5555, 7054-55, 9515, 17556	/	✓		
	25881	cnes.fr*				✓		
	27491	din.de*				✓		
	32114	csob.cz*				✓		
	48803	anaf.ro*				✓		
	55267	data.gov.in*				✓		
	55852	allegiantair.com*				✓		
	58948	tmdn.org*				✓		
	65955	beuth.de*				✓		
	99638	bank.sbi*				✓		
Native Application	5370	faceit.com	ws	28337	/	✓	✓	✓
	23219	cponline.pw (-)	ws	6463-72	/?v=1	✓	✓	✓
	29301-77550	samsungcard{.com, .co.kr}	wss	10531, 31027, 31029	/	✓	✓	✓
			https	14440-9	/?code=*&dummy=*	✓	✓	✓
	36141	gamehouse.com*	http	12071-72, 17021, 27021	/v1/init.json?api_port=*&query_id=*	✓	✓	✓
	47690	games.lol	ws	60202	/check	✓	✓	
	57008	zylom.com	http	12071, 17021	/v1/init.json?api_port=*&query_id=*	✓	✓	✓
	74089	iwin.com		2080-82	/version?_=*		✓	✓
	77134	screenleap.com (-)		5320	/status, /*/up	✓	✓	✓
	88902	acestream.me (-)		6878	/webui/api/service	✓	✓	✓
	91904	trustdice.win		50005, 51505, 53005, 54505, 56005	/, /socket.io	✓	✓	✓
	98789	runeline.com (-)	ws	6463-72	/?v=1	✓	✓	✓
Unknown	244	hola.org	http	6880-9	/*.json	✓	✓	✓
	21246	wowreality.info		1080, 1194, 2375, 2376, 3000, 3128, 3306, 3479, 4244, 5037, 5242, 5601, 5938, 6379, 8332, 8333, 8530, 9000, 9050, 9150, 9785, 11211, 15672, 23399, 27017	/	✓	✓	✓
	62048	svd-cdn.com		6880-9	/*.json	✓	✓	✓
	78456	usaonlineclassifieds.com*	ws	2687, 26876	/	✓		
	84569	usnetads.com*				✓		

Table 5: Summary of website localhost requests found during the 2020 top 100K crawl. Operating Systems (OS) are Windows (W), Linux (L) and Mac (M). Domains marked with an asterisk (*) did not make localhost requests during the 2021 top 100K crawl.

Rank (↓)	Domain	Protocol	Local IP Address	Port	Paths	OS		
						W	L	M
4381	gsis.gr	http	10.193.31.212	80	/system/files/2020-06/*.png	✓	✓	✓
19523	farsroid.com		10.10.34.35		/		✓	
35262	saddleback.edu	https	10.156.2.50	443	/*.ico			✓
46972	skalvibytte.no	http	10.0.0.200	80	/wordpress/wp-content/uploads/2020/04/*{.jpg,.mp4}	✓	✓	✓
56325	unib.ac.id		192.168.64.160		/wp-content/uploads/2019/10/*.jpg	✓	✓	✓
61554	adnsolutions.com		10.0.20.16		/wp-content/uploads/2018/11/*.jpg			✓
65302	tra97fn35n5brvxki5-sj8x5x34k2t4d67j883fgt.xyz		10.10.34.35		/		✓	
73062	zoom.lk	https	192.168.0.208	443	/wp_011_test_demos/wp-content/uploads/2017/05/*.jpg			✓
91632	1-movies.ir	http	10.10.34.35	80	/	✓	✓	✓

Table 6: Summary of website LAN requests found during the 2020 top 100K measurements. Operating Systems (OS) are Windows (W), Linux (L) and Mac (M).

4.3.2 Bot Detection. From the *Bot Detection* section of Table 5, we see that 10 sites in our 2020 top 100K measurements made HTTP requests to the same set of 7 ports on localhost, using the same URL path and only on Windows 10.

We identify that this activity is initiated by BIG-IP ASM Bot Defense, a bot detection service developed by F5 Inc. [20]. On each site, the localhost requests were initiated by a JS script hosted at `/TSPD`, which is the standard path used by BIG-IP ASM Bot Defense [19, 62]. Note here that this bot defense uses HTTP, as opposed to WSS as done by ThreatMetrix (from Section 4.3.1). Despite the application of the Same-Origin Policy to these HTTP requests, this bot defense is able to deduce whether or not a localhost port is active. The JS script was heavily obfuscated, but we hypothesize that this inference is based on a timing side channel. A request to an active localhost port returns quickly (even if the response cannot be read), while requests to inactive ports time out.

Investigating further, we discover that 4 out of the 7 ports scanned are notably used by well-known malware (as shown in Table 4). The other ports are for the Microsoft Edge WebDriver, used for browser automation, and the QuickTime Streaming Server, which historically has been heavily exploited [8]. This set of ports suggests that BIG-IP ASM Bot Defense potentially searches for (i) indicators of host compromise, and (ii) the presence of browser automation.

We find that BIG-IP ASM documentation indicates that this product enables user fingerprinting [18]. The F5 Inc. website also claims that they provide their customers, including government agencies, protection against botnets using network-based heuristics [17]. Aligned with this description, the sites we observed deploying this bot detection script belonged to government-related entities (e.g., central banks and websites hosting government data).

Interestingly, we do not observe sites making bot detection requests during our 2021 top 100K crawl as well as the crawl of malicious webpages. Upon inspecting the websites that originally generated such requests in the 2020 top 100K crawl, we indeed confirm that these websites no longer host the BIG-IP ASM Bot Defense JS script (although we are uncertain of the impetus for this change).

4.3.3 Native Applications. In our 2020 top 100K crawl, we identified 12 websites which were communicating with native applications associated with the website itself, as listed under *Native Applications* in Table 5. Except for one site, all of these sites behaved uniformly across OSes. These websites, as detailed in Appendix A, are e-commerce[4], gaming, online communication, and media sites.

Given the close relationship between the website and the native app (e.g., a gaming website contacting its associated gaming client), we assume benign intent here, although we note that the website may be detecting if a visitor has already installed the associated application, and adjusting its behavior (e.g., advertising) accordingly. Additionally, this class of localhost communication highlights a legitimate use case that should be preserved if attempting to defend against malicious forms of web-based localhost traffic.

In addition to these 12 websites, we found another 14 websites in our 2021 top 100K measurements similarly attempting to communicate with their native apps (as listed in Table 7). Only one website was found no longer making these localhost requests in our second crawl. For malicious webpages, we identified one similar case with `elilaifs.cn`. Inspecting the webpage indicates that it imports the JS library of Thunder [63], a download manager, which attempts to communicate with its native application.

4.3.4 Developer Errors. Beyond the intentional locally-bound traffic that websites make, we find that a large class of local resource requests likely arise due to a developer error. For our 2020 top 100K measurements, we believe this situation occurred on 44 (out of 107) websites requesting localhost resources and 6 (out of 9) sites requesting LAN resources, indicating that it is relatively widespread among our observed sites. Table 11 in the Appendix and Table 6 characterize these websites in detail.

In these cases, we observe localhost and LAN requests for various files (e.g., images, videos, JavaScript libraries) that were likely hosted at a local server during web development. For example, a number of websites request a URL path that contains `/wp-content/uploads/`, indicating the requested resource is a file uploaded to the local WordPress installation. Upon manual investigation, we

[4]The e-commerce sites communicates with a client-side security application.

Reason	Rank	Domain	Protocol	Ports	Paths	OS	
						W	L
Fraud Detection	2912	cibc.com	wss	3389, 5279, 5900-03, 5931, 5939, 5944, 5950, 6039-40, 63333, 7070	/	✓	
	8173	betfair.com (+)				✓	
	10679	highlow.com				✓	
	28370	moneybookers.com				✓	
	31170	ebay.com.hk				✓	
	64012	marks.com				✓	
Native App	592	iqiyi.com	http	16422-23	/get_client_ver?*	✓	✓
	7664	qy.net				✓	✓
	10966	qiyi.com				✓	✓
	12350	iqiyipic.com				✓	✓
	15581	ppstream.com				✓	✓
	34989	ppsimg.com (+)				✓	✓
	44280	soliqservis.uz (+)	wss	64443	/service/cryptapi	✓	✓
	75083	nfstar.net (+)	http	28317, 36759	/get_thunder_version/	✓	✓
	80108	9ekk.com (+)				✓	✓
	87274	somode.com (+)				✓	✓
	82814	mcgeeandco.com (+)	https	4000	/socket.io/?	✓	✓
	86605	71.am (+)	http	16422-23	/get_client_ver?*	✓	✓
	94270	didox.uz (+)	wss	64443	/service/cryptapi	✓	✓
	96284	gnway.com (+)	ws	38681-87	/	✓	
Developer Errors	5154	phonearena.com	http	1500	/floor-domains	✓	✓
	5331	madmimi.com		5555	/2.1.2/sockjs.min.js	✓	
	14951	nursingworld.org		80	~4af7b9/globalassets/images/*.jpg	✓	
	21280	ums.ac.id			/ums-baru/wp-content/*	✓	✓
	25940	zee.co.ao (+)			/industrialwp/wp-content/*	✓	✓
	37323	raovatnailsalon.com (+)	https	443	/raovatnailsalon/wp-content/*	✓	✓
	42107	panduit.com	http	4502	/apps/panduit/clientlibs/*.js	✓	
	45497	internetworld.de	https	443	/	✓	✓
	47861	mcknights.com		9988	/livereload.js		✓
	50650	san-servis.com	http	80	/vina/vina_febris/images/*	✓	✓
	54756	postfallsonthego.com (+)			/magazon/magazon-wp/wp-content/uploads/*	✓	✓
	55755	wealthcareportal.com (+)			/NonExistentImage48762.gif	✓	✓
	55477	lited.com		110066	/getversionjpg?hash=*	✓	
	68872	workpermit.com	https	6081	/news-ticker.json	✓	✓
	75989	ethiopianreporterjobs.co (+)		443	/wp-content/uploads/*	✓	✓
	77974	macroaxis.com (+)	http	8080	/img/icons/search.png	✓	✓
	83256	adfontesmedia.com (+)		8888	/adfontesmedia/wp-content/uploads/*	✓	✓
	84378	charityvillage.com (+)			/core/js/api/web-rules	✓	✓
	90632	showfx.ro (+)	https	443	/wordpress/x-street/wp-content/*	✓	✓
	98402	xaydungtrangtrinoithat.com (+)			/wp-content/uploads/*	✓	✓

Table 7: Summary of new website localhost requests seen in the 2021 top 100K crawl. (+) represents a domain that was not present in the 2020 top 100K list snapshot (and thus was not previously crawled). Operating Systems (OS) are Windows (W), Linux (L) and Mac (M).

did not find any evidence suggesting intentional localhost or LAN communication across these websites, and we believe that these resource requests are unintentional and remnants of when the websites were being developed and tested. Appendix B explores the different occurrences across websites in more detail.

Similar cases were found in both the 2021 top 100K crawl and our crawl of malicious webpages. For malicious pages, we believe that either attackers made such developer errors in their own attack sites, or more likely, the attackers compromised a benign website exhibiting these errors. In total, we attribute more than 90% of the localhost activity on malicious webpages to this behavior class.

While presumably benign, these local network requests indicate that some functionality on the website is incorrect, as the webpage attempts to load a resource that does not actually exist. We also

note that the requests can reveal aspects of the website development process, such as a particular vulnerability testing JavaScript framework used by `rkn.gov.ru`. Ultimately, these requests reflect a class of web developer errors that can impact the functionality of websites but should be easy to identify and remedy, such as through searching for localhost or private IP address requests in the website code base or examining the requests made when loading the website.

5 DISCUSSION

We synthesize our results and their security/privacy implications.

5.1 Website Anti-Abuse

From our measurements, we discover that a number of websites leverage web-based localhost scanning to defend themselves against abuse. These sites are primarily e-commerce and government-related and include highly-ranked sites for organizations such as eBay and Fidelity Investments. However, the websites do not conduct the abuse detection themselves, but rather (as uncovered in Sections 4.3.1 and 4.3.2) they rely on third-party services (i.e., ThreatMetrix and BIG-IP ASM). A 2018 ThreatMetrix brochure [32] claimed to protect $170 billion worth of e-commerce transactions in a year, preventing $19.5 billion in fraud. They advertised an extensive database of 1.4 billion unique online identities collected from their clients operating in the e-commerce space. Given the prevalence of online abuse across various platforms beyond just e-commerce, such as on social media and communications websites, we may observe an expansion of web-based localhost scanning for anti-abuse on other sites.

Our analysis in Sections 4.3.1 and 4.3.2 identified that these anti-abuse measures focus on detecting indicators of host compromise or remote control. We are unable to assess the reliability of these signals for anti-abuse. However, we hypothesize that attackers could evade this detection with relative ease by modifying the ports they operate on. For example, attackers could configure a remote control server on a bot to run on a non-standard port, and malware could dynamically adjust the ports it listens on. This evolution would likely result in an arms race that seems tilted in the attacker's favor due to information imbalance. As any web-based localhost scanning activity would be visible to website visitors as well as attackers, attackers can directly observe and counter the anti-abuse strategies deployed. In contrast, attackers lack direct visibility into the operations of server-side anti-abuse mechanisms. Thus, we question whether web-based localhost scanning for fraud and bot detection will remain viable long-term.

5.2 Web Tracking

We did not uncover explicit evidence of user tracking through web-based localhost or LAN scanning, a positive finding regarding web privacy. However, we did observe clear host profiling as part of fraud and bot detection, which can naturally be extended for user fingerprinting and tracking (as demonstrated by existing proof-of-concept techniques [4, 39]). Similar to other web fingerprinting methods that have been used in practice, such as detecting the browser extensions installed [58], web trackers could distinguish users based on their localhost network services and LAN devices. Prior work [3] uncovered how websites often deploy surreptitious methods for web tracking. With ongoing efforts to curtail the amount of data that websites can obtain on users, including Google's Privacy Sandbox project [48], web trackers may be forced to resort to novel tracking mechanisms such as those based on local network scanning.

Further incentivizing this new tracking strategy, we expect that the amount of information that can be derived from web-based localhost and LAN scanning is likely expanding. On user machines, numerous modern native applications (such as those discussed in Section 4.3.3) host localhost network services, and their diversity and popularity have grown (particularly gaming and video conferencing clients during the COVID-19 pandemic [15, 59]). Similarly, user home networks contain growing numbers of devices [38], particularly IoT devices with exposed network interfaces that can be readily detected [4]. Thus, web-based local scanning should provide significant amounts of information about users, which also serves as high entropy features for fingerprinting and tracking them. We discuss defending against such website behavior next.

5.3 Defending Against Malicious Web-Based Local Traffic

We did not discover any websites generating malicious attack traffic to localhost or LAN network services, as hypothesized by prior work [4, 39, 54]. However, our measurements were of popular websites where one might not expect large-scale malicious activity. It is possible that attacker-controlled websites may launch web-based local network attacks in practice, and we leave such exploration for future work (as discussed in Section 6). Nonetheless, browsers should ideally protect users from both malicious attacks and web tracking based on locally bound network traffic.

For website LAN traffic, we did not find any legitimate use cases, with the majority of websites likely exhibiting developer error (as found in Section 4.3.4). However, the prior demonstrations of web-based LAN attacks [4, 39, 54] revealed the threat of giving websites the ability to initiate LAN-destined traffic. Thus, the browser could protect LAN devices from both malicious attacks and web tracking through outright blocking all network requests destined for private IP addresses (which incidentally would also protect against DNS rebinding attacks where external DNS names resolve to internal IP addresses [36, 37]).

The situation is more challenging for websites initiating localhost traffic though, as we observe at least one legitimate use case of website localhost communication with associated native applications (from Section 4.3.3). To preserve this use case, browsers should not outright block localhost requests from websites. One potential path forward is for the browser to alert the user before initiating such requests, similar to existing browser permission prompts. This human-in-the-loop approach could prevent unauthorized localhost communication. We also note that, as shown in Table 5, the websites that do contact native applications communicate with relatively few ports (at most 10). We expect this observation to hold broadly, as intuitively, a native application should use a limited number of ports to expose their network services. Thus, the browser could inhibit large-scale localhost scanning by limiting the number of localhost ports that a web origin interacts with. An additional proposal is

for native applications to provide some standardized specification of which domains are permitted to initiate communication, similar to the Manufacturer Usage Descriptions (MUD) proposed for specifying device network behavior [33].

5.4 Ongoing Mitigation Efforts

The browser security community is actively working on mitigating the unintentional exposure of local network resources to browsers. In March 2021, the Web Platform Incubator Community Group (WICG) [61] proposed modifications to the Fetch API [13], where resources loaded from a public IP space are allowed to fetch resources from private/local IP spaces only if: 1) the public resource was loaded over a secure channel (i.e., `https` or `wss`), and 2) a CORS preflight request to the local network origin is successful [51]. These CORS preflight requests should include the *Access-Control-Request-Private-Network=true* header, and browsers should only permit access to the local network resource if the same header is in the responses.

We believe this proposal is a promising step in the right direction, as such an opt-in model for access to a user's local network resources would support legitimate use cases (e.g., native application communication), while barring other unintentional exposure. Ultimately, the real-world security and privacy impact of such a proposal depends on its implementation and adoption. We hope that our findings help inform and drive the further development of such proposals.

5.5 Developer Errors

A large number of websites were found making local network requests likely due to developer error. As discussed in Section 4.3.4, these requests reflect broken functionality on the sites as well as potential information leakage on the web development and testing process. Therefore, we recommend that web developers check for such local network behavior through either analyzing the website codebase or examining network traffic generated by the website during testing. We note that while assessing a website during testing, different user-agents should be evaluated, as we observed different behavior across OSes even for developer errors (likely due to the error occurring in OS-specific portions of the website code). We expect that these errors should be easy to remedy, as the request destination can be updated to the correct public server, or the traffic-generating code can be removed altogether.

6 CONCLUSION

In this study, we conducted a large-scale empirical investigation of if, how, and why modern websites interact with network services on a browser's localhost and LAN. Crawling both the landing pages of the Tranco top 100K domains [47] as well as 145K malware, phishing, and abuse websites, using Google Chrome on three popular desktop OSes (Windows, Linux, and Mac), we uncover hundreds of websites generating requests to internal network destinations, including highly-ranked sites within the top 10K. We find that local network activity was not uniform across OSes, with the most activity found on Windows. We also observed extensive use of WebSockets for locally-destined connections, which notably is not bound by the Same-Origin Policy.

By investigating the detected top websites in detail, we identified four common causes for the local traffic: fraud detection, bot detection, native application communication, and developer error. Notably, the fraud and bot detection techniques conducted host profiling. While we did not uncover explicit user tracking, the adaptation of these host profiling techniques for such purposes may be on the horizon. Defending against such malicious web-based local traffic will require preserving the legitimate use case of native application communication. Meanwhile, the class of developer errors that we exposed, affects website functionality but should be easy for web developers to discover and remedy.

For malicious websites, we did not detect internal network attacks. Rather, 90% of malicious webpages exhibited local traffic matching the developer errors observed on top websites, suggesting that either attackers make the same mistakes when implementing their attack websites, or perhaps more likely, the attackers compromised benign websites exhibiting these errors. We also found an interesting case of phishing webpages cloning the web interfaces of legitimate websites that generate local traffic for fraud detection, thus inadvertently inheriting the same local network behavior.

Ultimately, our work establishes empirical grounding on the intentional and unintentional localhost and LAN network activities of popular websites. Future work can expand upon our initial investigation. As discussed in Section 3.3, our examination of websites only considered the landing page, but we expect that internal sites, such as account creation or login pages also generate localhost and LAN network traffic. Additionally, our measurement methodology (as detailed in Section 3) focused on a desktop environment. Subsequent studies can build upon our work by evaluating websites on mobile platforms and inspecting website internal pages.

Our study also uncovered host profiling purportedly for fraud and bot detection. These techniques can be studied in more depth to characterize their effectiveness in practice and assess the potential risk that they are adapted for user tracking purposes. Finally, browser mechanisms can be developed to detect and prohibit malicious local network requests from websites while preserving legitimate use cases, thus better protecting user security and privacy on the web.

REFERENCES

[1] Lawrence Abrams. 2020. List of well-known web sites that port scan their visitors. https://www.bleepingcomputer.com/news/security/list-of-well-known-web-sites-that-port-scan-their-visitors/.

[2] Abuse.ch. 2021. URLhaus. https://urlhaus.abuse.ch/.

[3] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *ACM Conference on Computer and Communications Security (CCS)*. 674–689.

[4] Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. 2018. Web-based attacks to discover and control local IoT devices. In *Workshop on IoT Security and Privacy*. 29–35.

[5] andlabs.org. 2010. Port Scanning with HTML5 and JS-Recon. http://blog.andlabs.org/2010/12/port-scanning-with-html5-and-js-recon.html.

[6] BlueJeans. 2020. BlueJeans, bluejeans.com. https://bluejeans.com/.

[7] Matthew Bryant. 2015. sonar.js: A Framework for Identifying and Launching Exploits against Internal Network Hosts. https://github.com/mandatoryprogrammer/sonar.js.

[8] Christopher Budd. 2016. Urgent Call to Action: Uninstall QuickTime for Windows Today. https://blog.trendmicro.com/urgent-call-action-uninstall-quicktime-windows-today/.

[9] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. 2016. Content security problems? evaluating the effectiveness of content security policy in the wild. In *ACM Conference on Computer and Communications Security (CCS)*. 1365–1375.

[10] SANS Internet Storm Center. 2012. Service Name and Transport Protocol Port Number Registry. (tcp/udp)AttackActivity.
[11] Hancom Secure Co. 2020. AnySign for PC by Hancom Secure Co., hsecure.co.kr. https://hsecure.co.kr.
[12] Discord. 2020. Discord, discord.com. https://discord.com/.
[13] MDN Web Docs. 2021. Fetch API. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.
[14] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *ACM Conference on Computer and Communications Security (CCS)*. 1388–1401.
[15] Adam Epstein. 2020. The pandemic has turned everyone into gamers. https://qz.com/1904276/everyone-is-playing-video-games-during-the-pandemic/.
[16] Matt Menke Eric Roman. 2012. NetLog: Chrome's network logging system. https://www.chromium.org/developers/design-documents/network-stack/netlog.
[17] F5. 2017. F5 iApps: Moving Application Delivery Beyond the Network. https://www.f5.com/services/resources/white-papers/f5-iapps-moving-application-delivery-beyond-the-network.
[18] F5. 2019. K19556739: Overview of BIG-IP ASM client fingerprinting. https://support.f5.com/csp/article/K19556739.
[19] F5. 2020. K33440533: The BIG-IP ASM Bot Defense may erroneously subject clients and web servers to Open Redirection attacks. https://support.f5.com/csp/article/K33440533.
[20] F5. 2020. K42323285: Overview of the unified Bot Defense profile. https://support.f5.com/csp/article/K42323285.
[21] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS adoption on the web. In *USENIX Security Symposium (USENIX Security)*. 1323–1338.
[22] Tom Gallagher. 2016. Port Scanning and WebSockets.
[23] Tom Gallagher. 2016. Security Enhancement for WebSockets to Prevent Private Network Mapping. https://tools.ietf.org/html/draft-gallagher-hybiwebsocketenhancement-00.
[24] GitHub. 2020. LiveReload.js. https://github.com/livereload/livereload-js.
[25] GitHub. 2020. OWASP-Xenotix-XSS-Exploit-Framework. https://github.com/ajinabraham/OWASP-Xenotix-XSS-Exploit-Framework.
[26] GitHub. 2020. SockJS-node, a Node.js server for WebSocket emulation. https://github.com/sockjs/sockjs-node.
[27] GitHub. 2020. XSS Rays. https://github.com/beefproject/beef/wiki/Xss-Rays.
[28] GitHub. 2021. Puppeteer. https://github.com/puppeteer/puppeteer.
[29] GitHub. 2021. Selenium. https://github.com/SeleniumHQ/selenium.
[30] Google. 2021. Safe Browsing. https://safebrowsing.google.com/.
[31] Jeremiah Grossman and T Niedzialkowski. 2006. Hacking Intranet Websites from the Outside: JavaScript Malware Just Got a Lot More Dangerous. In *Blackhat USA*.
[32] RELX Group. 2018. Risk & Business Analytics teach-in. https://www.relx.com/~/media/Files/R/RELX-Group/documents/presentations/risk-teach-in-8Nov18.pdf.
[33] Ayyoob Hamza, Dinesha Ranathunga, Hassan Habibi Gharakheili, Matthew Roughan, and Vijay Sivaraman. 2018. Clear as MUD: generating, validating and applying IoT behavioral profiles. In *Workshop on IoT Security and Privacy*. 8–14.
[34] Taylor Hornby. 2015. Port Scanning Local Network From a Web Browser. https://defuse.ca/in-browser-port-scanning.htm.
[35] Internet Assigned Numbers Authority (IANA). 2012. Service Name and Transport Protocol Port Number Registry. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.
[36] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. 2009. Protecting browsers from DNS rebinding attacks. *ACM Transactions on the Web (TWEB)* 3, 1 (2009), 1–26.
[37] Martin Johns, Sebastian Lekies, and Ben Stock. 2013. Eradicating DNS Rebinding with the Extended Same-origin Policy. In *USENIX Security Symposium (USENIX Security)*. 621–636.
[38] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. 2019. All things considered: an analysis of IoT devices on home networks. In *USENIX Security Symposium (USENIX Security)*.
[39] VT Lam, Spiros Antonatos, Periklis Akritidis, and Kostas G Anagnostakis. 2006. Puppetnets: misusing web browsers as a distributed attack infrastructure. In *ACM Conference on Computer and Communications Security (CCS)*.
[40] Sangho Lee, Hyungsub Kim, and Jong Kim. 2015. Identifying Cross-origin Resource Status Using Application Cache. In *The Network and Distributed System Security Symposium (NDSS)*.
[41] LexisNexis. 2020. LexisNexis ThreatMetrix. https://risk.lexisnexis.com/products/threatmetrix.
[42] myria.de. 2007. JavaScript LAN Scanner. https://www.myria.de/lan-scan/index.php.
[43] Jakob Nielsen. 2011. How Long Do Users Stay on Web Pages? https://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/.
[44] nProtect. 2020. nProtect Online Security, nos.nprotect.com. https://nos.nprotect.com/.
[45] Peppersoft. 2017. Local Network Scanner with JavaScript. http://peppersoft.net/local-network-scanner-javascript/.
[46] PhishTank. 2021. PhishTank. http://phishtank.org/.
[47] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2018. Tranco: A research-oriented top sites ranking hardened against manipulation. In *The Network and Distributed System Security Symposium (NDSS)*.
[48] The Chromium Projects. 2020. The Privacy Sandbox. https://www.chromium.org/Home/chromium-privacy/privacy-sandbox.
[49] Ram Sundara Raman, Adrian Stoll, Jakub Dalek, Reethika Ramesh, Will Scott, and Roya Ensafi. 2020. Measuring the Deployment of Network Censorship Filters at Global Scale. In *The Network and Distributed System Security Symposium (NDSS)*.
[50] Yakov Rekhter, B Moskowitz, Daniel Karrenberg, GJ de Groot, and Eliot Lear. 1996. Rfc1918: Address allocation for Private Internets.
[51] Titouan Rigoudy and Mike West. 2021. Private Network Access - Draft Community Group Report. https://wicg.github.io/private-network-access/.
[52] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, and Ben Stock. 2020. Complex Security Policy? A Longitudinal Analysis of Deployed Content Security Policies.. In *The Network and Distributed System Security Symposium (NDSS)*.
[53] Peter Snyder, Lara Ansari, Cynthia Taylor, and Chris Kanich. 2016. Browser Feature Usage on the Modern Web. In *ACM Internet Measurement Conference (IMC)*.
[54] Sid Stamm, Zulfikar Ramzan, and Markus Jakobsson. 2007. Drive-by pharming. In *International Conference on Information and Communications Security*. Springer, 495–506.
[55] Emily Stark, Ryan Sleevi, Rijad Muminovic, Devon O'Brien, Eran Messeri, Adrienne Porter Felt, Brendan McMillion, and Parisa Tabriz. 2019. Does certificate transparency break the web? Measuring adoption and error rate. In *IEEE Symposium on Security and Privacy (S&P)*. 211–226.
[56] SURBL. 2021. SURBL URI reputation data. http://www.surbl.org/lists.
[57] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, Daniel Margolis, Vern Paxson, and Elie Bursztein. 2017. Data Breaches, Phishing, or Malware? Understanding the Risks of Stolen Credentials. In *ACM Conference on Computer and Communications Security (CCS)*.
[58] Erik Trickel, Oleksii Starov, Alexandros Kapravelos, Nick Nikiforakis, and Adam Doupe. 2019. Everyone is Different: Client-side Diversification for Defending Against Extension Fingerprinting. In *USENIX Security Symposium (USENIX Security)*.
[59] Charlotte Trueman. 2020. Pandemic leads to surge in video conferencing app downloads. https://www.computerworld.com/article/3535800/pandemic-leads-to-surge-in-video-conferencing-app-downloads.html.
[60] Joe Vennix. 2015. lan-js: Probe LAN devices from a web browser. https://github.com/joevennix/lan-js.
[61] W3. 2021. The Web Platform Incubator Community Group (WICG). https://www.w3.org/community/wicg/.
[62] John Wagnon. 2017. Proactive Bot Defense Using BIG-IP ASM. https://devcentral.f5.com/s/articles/proactive-bot-defense-using-big-ip-asm-25685.
[63] Wikipedia. 2021. Xunlei. https://en.wikipedia.org/wiki/Xunlei.
[64] Zoom. 2020. ZOOM US, zoom.us. https://zoom.us/.

A LOCALHOST COMMUNICATION WITH NATIVE APPLICATIONS

Below we detail the localhost communications that websites initiate with native applications, as discussed in Section 4.3.3.

- *E-commerce*: `samsungcard.co.kr` redirects to `samsungcard.com`, which is an e-commerce website. We found that `samsungcard.com` looks for the presence of INCA Internet's nProtect Online Security [44], an endpoint security solution (which claims to protect the user's transactions with a financial services site against fraud), and communicates with a digital signature software called AnySign [11]. Our telemetry observed `samsungcard.com` requesting localhost ports 14440-9 over HTTP via a JS file that specified "nProtect Online Security" in its header, and requests to other ports over WebSockets, via a different JS file with "AnySign for PC" in its header. Further investigation revealed that AnySign does utilize bidirectional communication with its native application over WebSockets.

The end users first download nProtect Online Security from the financial services' website that employs nProtect (in this case, samsungcard.com) and the website then uses a JS plugin to look for the native application on the user's computer. We note that while the end goal of this localhost communication is similar to the fraud detection goals of ThreatMetrix (in Section 4.3.1), the localhost communication is used to interact with a native application which performs the fraud detection, rather than the communication being done as part of the fraud detection method (as with ThreatMetrix).

- *Gaming*: faceit.com, gamehouse.com, games.lol, zylom.com and trustdice.win are gaming websites that conduct localhost communication to detect their native game on the user's computer. faceit.com and games.lol attempt to communicate with their native client via the WebSockets protocol, whereas the others communicate using HTTP.
- *Communication*: cponline.pw and runeline.com redirect to channel invitation pages of Discord, an IM/VoIP application [12]. These web pages initiate localhost communication with Discord's native client. Similarly, screenleap.com makes localhost requests to its own Screen Share product. Given the recent extensive use of video conference software clients such as Zoom [64] and BlueJeans [6], we expect that we are missing other communication-based websites performing localhost requests, as their activities are not exhibited on landing pages.
- *Media*: acestream.me initiates localhost communication with the Ace Stream client, a live-streaming media platform.

B LOCALHOST AND LAN COMMUNICATION DUE TO DEVELOPER ERROR

As discussed in Section 4.3.4, a large class of localhost and LAN communication arises from the likely remnants of website development and testing, suggesting a developer error. Table 11 characterizes the individual websites that we believe exhibit this situation for localhost resources, and Table 6 does likewise for websites fetching LAN resources. Here, we also discuss these cases in more detail.

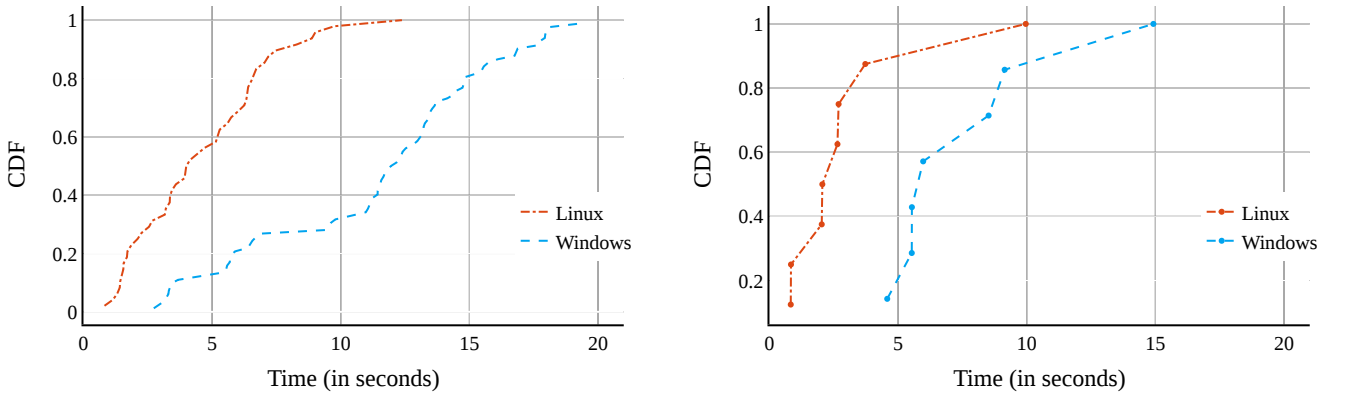
- *Accessing files hosted on a local file server*: As seen in the first set of rows in Table 11 for localhost requests and Table 6 for LAN requests, we find that 57% (25/44) and 67% (6/9) websites request files (e.g., images) from a server on localhost and LAN, respectively, all using HTTP(S). For example, a number of these sites make localhost HTTP requests where the URL path contains /wp-content/uploads, attempting to fetch an uploaded file to a local WordPress installation.

  Interestingly, there are cases where this behavior differs across OSes. For example, sites like walisongo.ac.id, classera.com, gomedici.com, xaipe.edu.cn, zoom.lk, aau.edu.et, saddleback.edu, and adnsolutions.com

  returned different responses when visiting from different OSes. In contrast, farsroid.com, tra...fgt.xyz (abbreviated) and upbasiceduboard.gov.in returned invalid responses for certain OSes. We hypothesize that these discrepancies arise in part because websites may present themselves differently for different OSes (for example, based on the user-agent string), and certain website elements that generate the unintentional resource fetch are not included, when on certain OSes. External network elements, such as the ISP, could also cause this. For example, the IP address that upbasiceduboard.gov.in resolved to when visited via Mac is inactive, whereas when visited via Linux and Windows (on a different ISP), the resolved IP is active.
- *Penetration testing*: rkn.gov.ru fetches the JavaScript file xook.js, which is a file used as part of OWASP's Xenotix Exploit Framework, which web developers use for vulnerability detection and penetration testing [25].
- *LiveReload.js*: Several websites fetch the file livereload.js, which is for the LiveReload.js tool that helps web developers dynamically monitor changes during web development [24].
- *Redirect*: romadecade.org and fincaraiz.com.co redirect to http://127.0.0.1/. We are uncertain of the reason.
- *SocksJS-Node*: We observe five websites initiating localhost HTTP(S) requests where the URL path is /sockjs-node/info. This path is associated with SockJS-node, which is the Node.js server-side component of the SockJS JavaScript library [26]. This library provides a WebSocket-like communication interface between a server and a browser even for legacy browsers without WebSocket support. Thus, we believe that the website developers were using SockJS during development and testing, which involved deploying a localhost SockJS-node server, and this artifact remained in the published website. Interestingly, this activity was observed only when on a Mac.
- *Other local services*: We observe 7 websites initiating localhost HTTP(S) requests likely for interacting with local network services during website development and testing. For example, zakupki.gov.ru updates user activity state at a localhost service.

C UNKNOWN CASES

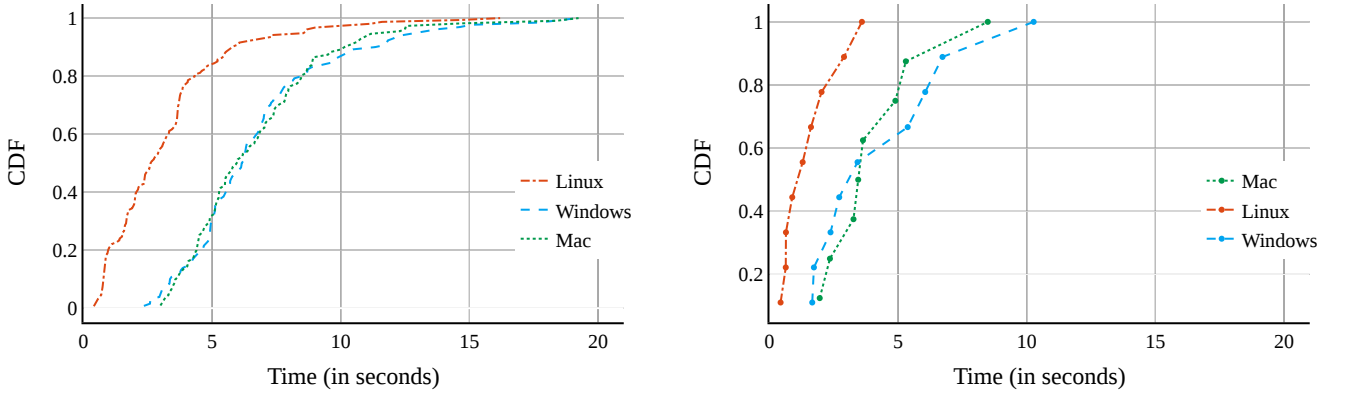
We did not identify plausible explanations for the 5 websites listed under *Unknown* in Table 5. There were 3 websites (farsroid.com, tra...fgt.xyz and 1-movies.ir) generating LAN traffic that we could not confidently classify. These sites return a 403 Forbidden page with an iframe element sourced at http://10.10.34.35:80. Recently, Raman et al. [49] observed such iframe sources occurring during website censorship in Iran. We note that farsroid.com and 1-movies.ir resolve to Iranian IP addresses, but tra...fgt.xyz did not. We suspect that the observe behavior is related to censorship, although we remain uncertain of the exact situation, and are unclear why this particular LAN address is used.



(a) Requests to services hosted on localhost

(b) Requests to services hosted on LAN devices

Figure 6: CDFs of the time delays between when the landing page is fetched and when we observed the first local network request, across OSes, for our 2021 top 100K measurements.



(a) Requests to services hosted on localhost

(b) Requests to services hosted on LAN devices

Figure 7: CDFs of the time delays between when the landing page is fetched and when we observed the first local network request, across OSes, for our malicious webpages measurements.

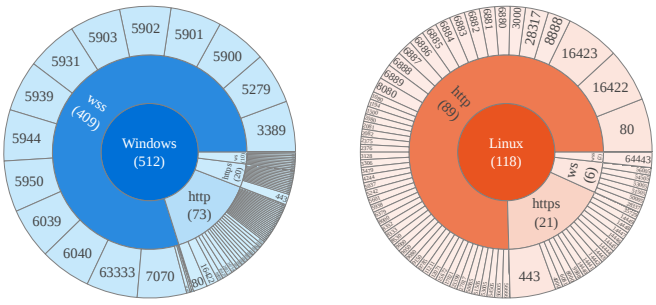


Figure 8: Distribution of website network requests to localhost for the 2021 crawl of top 100K websites. The center of each diagram indicates the OS and the total number of requests observed for websites on that OS. The first ring divides these requests by the network protocol/scheme. The outer ring specifies the localhost port number requested.

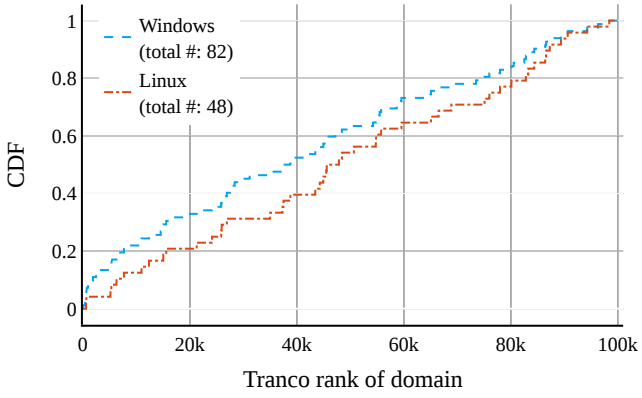


Figure 9: CDFs of domain ranks for our 2021 top 100K measurements of landing pages generating localhost traffic, across OSes.

Malicious Category	Domains	Protocol	Ports	Paths	OS		
					W	L	M
Malware	(79 domains, omitted for brevity)	http(s)	80/443	/*/wp-content/*	✓	✓	✓
	acffiorentina.ru	http	8080	/socket.io/socket.io.js	✓	✓	✓
	elilaifs.cn		28317, 36759	/get_thunder_version	✓	✓	✓
	boatattorney.com	https	35729	/livereload.js	✓		✓
	jdih.purworejokab.go.id	http	80	/website-bphn-bk/*	✓	✓	✓
	metolegal.com			/metolegal/wp-includes/js/*	✓	✓	✓
	ppdb.smp1sbw.sch.id			/ppdbv3/ro-error/*		✓	
	scopesports.net			/scope/xpertspanel/*		✓	
	tonyhealy.co.za			/	✓	✓	✓
	oceanos.com.co			/wp-oceanos/*	✓	✓	✓
Abuse	autorizador5.com.br, classyfashionbd.com, coralive.org, saudiwallcovering.com	http(s)	80/443	/*/wp-content/*	✓	✓	✓
Phishing	ebaybuy.com.buying-item-guest.com, 100-25-26-254.cprapid.com, advancedlearningdynamics.com, smarturl.it, customer-ebay.com, {www.}citibank.gulajawajahe.my.id, o2-billing.org, samarasecrets.com, sic-week.000webhostapp.com, signin01.kauf-eday.de, hotelmontiazzurri.com, mahdistock.com, adesignsovast.com	wss	3389, 5279, 5900-03, 5931, 5939, 5944, 5950, 6039-40, 63333, 7070	/	✓		
	ag4.gartenbau-olching.de, grp02.id.rakutan-co-jpr.buzz	http	80	/	✓	✓	
	ag4.gartenbau-olching.de						✓
	rakuten.* (8 domains), www.ip.rakuten.1ex.info, rakuteni.co.jp.ai12.info, www.ip.rakuten.rbimomro.icu					✓	
	elmagra.net			/dashboard-v1/*	✓	✓	
	etoro-invest.org			/StudentForum//*	✓	✓	✓
	amazon.co.jp.* (12 domains)			/robots.txt		✓	
	survivalhabits.com		44056	/NonExistentImage33090.gif	✓	✓	
	evolution-postepay.com	https	5140	/NonExistentImage19258.gif	✓	✓	
	postepaynuovo.com		62389	/NonExistentImage55353.gif	✓	✓	✓
	sbloccareposte.com	http	44938	/NonExistentImage37362.gif	✓		
	verificapostepay.com	https	49622	/NonExistentImage20705.gif	✓		✓
	aladdinstar.com		8443	/images/*.png	✓	✓	✓

Table 8: Summary of localhost requests found for malicious webpages. OSes are Windows (W), Linux (L) and Mac (M).

Malicious Category	Domain	Protocol	Local IP Address	Port	Paths	OS		
						W	L	M
Malware	test.laitspa.it	http	10.2.70.15	80	/*.css	✓	✓	✓
	wangzonghang.cn		192.168.0.226	1080	/wp-content/themes/*	✓		✓
	{www.}crasar.org		192.168.1.8	80	/crasar/wp-content/themes/*	✓	✓	✓
	mihanpajooh.com		10.10.34.35		/	✓	✓	
	ahs.si	https	192.168.33.10	443	/wp-content/uploads/2019/12/*.png	✓		✓
	fixusgroup.com		172.26.6.230		/wp-content/uploads/2020/02/*.png	✓	✓	✓
	zoom.lk	http	192.168.0.208	80	/wp_011_test_demos/ wp-content/uploads/2017/05/*.jpg	✓	✓	✓
Abuse	001tel.com		172.16.205.110		/usershare/*.js	✓	✓	✓

Table 9: Summary of LAN requests found for malicious webpages. OSes are Windows (W), Linux (L) and Mac (M).

Rank (↓)	Domain	Protocol	Local IP Address	Port	Paths	OS	
						W	L
4847	blogsky.com	http	10.10.34.34	80	/	✓	✓
23723	jollibeedelivery.qa		192.168.8.241	5000	/MyPhone/c2cinfo	✓	✓
47356	unib.ac.id	https	192.168.64.160	443	/wp-content/uploads/2019/10/*.jpg	✓	
61472	bahrain.bh		192.168.110.72		/matomo/*.js	✓	✓
69494	auda.org.au		10.50.1.242	8450	/libraries/slick/slick/*.gif	✓	✓
73274	mre.gov.br		192.168.33.187	443	/modules/mod_acontece/assets/*		✓
95595	haiwaihai.cn	http	172.16.0.4	1117	/UpLoadFile/20160801/*.jpg	✓	✓
96554	techshout.com	https	192.168.0.120	443	/wp_011_gadgets/wp-content/uploads/*	✓	✓

Table 10: Summary of LAN requests found in our 2021 top 100K measurements. OSes are Windows (W) and Linux (L).

Reason	Rank (↓)	Domain	Protocol	Port	Paths	OS		
						W	L	M
Local file server	22730	smartcatdesign.net	http	8888	/wp-content/uploads/2018/06/*.jpg	✓	✓	✓
	36786	uinsby.ac.id		80	/eduma/demo-1/wp-content/uploads/sites/2/2017/11/*.jpg	✓	✓	✓
	38865	upbasiceduboard.gov.in (-)		1987	/TeacherRecruitment2018/images/*.jpg	✓	✓	
	41468	walisongo.ac.id	https	80	/wordpress/wp-content/uploads/2015/07/*.jpg	✓	✓	
	41596	classera.com	http	8080	/wp-content/uploads/2020/04/*.png		✓	✓
	45177	weavesilk.com*		80	/Silk%20Static/*{.mp4, .ogg}	✓	✓	✓
	50390	upsen.net (-)	http	80	/6/10/*.js	✓	✓	✓
	51910	dsb.cn*			.*jpg	✓		
	56450	sin-tech.cn (-)		9999	/admin/kindeditor/attached/image/20191017/*.jpg	✓	✓	✓
	56730	nwolb.com*	https	36762	/*.gif	✓	✓	✓
	57467	cryptopia.co.nz*	http	49972	/*.ico	✓	✓	✓
	63636	weijuju.com* (-)		9092	/image/page/index/*.png	✓	✓	✓
	63770	tdk.gov.tr*		80	/magazon/magazon-wp/wp-content/uploads/2013/02/*.ico	✓	✓	✓
	65915	shqilon.com (-)			/stop/*.html	✓	✓	✓
	66891	aau.edu.et*			/graduation/wp-content/uploads/2020/06/*.png	✓		
	67851	sirrus.com.br			/sitesirrus/wp-content/uploads/2017/07/*.png	✓	✓	✓
	69708	unionbankph.com*		8888	/socket.io/*.js	✓	✓	✓
	77636	qubscribe.com (-)	https	443	/wp-content/uploads/2019/03/*.png		✓	✓
	77761	persian-magento.ir (-)	http	80	/graffito/images/sampledata/*.png	✓	✓	✓
	86045	serymark.com (-)			/sm/wp-content/uploads/2017/06/*.png	✓	✓	✓
	88997	ghana.com (-)	https	8080	/gdc/wp-content/themes/consultix/images/*.png	✓	✓	✓
	92768	gomedici.com	http	3000	/assets/*.png		✓	✓
	93798	xaipe.edu.cn (-)		80	/*.html		✓	✓
	94771	health.com.kh (-)		8899	/newhealth/wp-content/uploads/2018/01/*.png	✓	✓	✓
	96981	urkund.com (-)		4337	/wp-content/uploads/2019/07/*.png		✓	✓
Pen test	17827	rkn.gov.ru (-)	http	5005	/xook.js	✓	✓	✓
LiveReload.js	19244	cruzeirodosulvirtual.com.br*	http	460	/livereload.js	✓	✓	✓
	53124	melissaanddoug.com*	https	35729		✓	✓	✓
	53216	airfind.com*				✓	✓	✓
	58629	hollins.edu				✓	✓	✓
	59978	amitriptylineelavilgha.com (-)	http			✓	✓	✓
Redirect	51142	romadecade.org (-)	http	80	/	✓	✓	✓
	63644	fincaraiz.com.co*				✓		
SocksJS-Node	49144	lyfdose.com	http	9000	/sockjs-node/info?t=*			✓
	49990	klik-mag.com	https					✓
	51101	acedirectory.org						✓
	57249	veteranstodayarchives.com						✓
	66971	smartsearch.me						✓
Other local services	7700	zakupki.gov.ru (-)	https	1931	/record/state	✓	✓	✓
	24740	gamezone.com	http	8000	/setuid	✓	✓	✓
	26400	filemail.com		56666	/	✓	✓	✓
	31518	interbank.pe		9080	/avisos-portal	✓	✓	✓
	58708	fsist.com.br (-)		28337	/getCertificados	✓	✓	✓
	62852	spaceappschallenge.org			/graphql		✓	✓
	90791	fromhomefitness.com (-)	https	8000	/app/getLicenseKey		✓	

Table 11: Summary of website developer errors found. OSes are Windows (W), Linux (L) and Mac (M). Domains marked with an asterisk (*) were in our 2021 top 100K crawl but did not make localhost requests, and domains marked with a minus sign (-) were not present in the 2021 toplist snapshot (and thus were not crawled).