Characterizing the Security of Endogenous and Exogenous Desktop Application Network Flows

Matthew R. McNiece[1,2], Ruidan Li[2], and Bradley Reaves[1]

[1] North Carolina State University {mrmcniec,bgreaves}@ncsu.edu
[2] Cisco Systems, Inc. {ruidali}@cisco.com

Abstract. Most desktop applications use the network, and insecure communications can have a significant impact on the application, the system, the user, and the enterprise. Understanding *at scale* whether desktop application use the network securely is a challenge because the application provenance of a given network packet is rarely available at centralized collection points. In this paper, we collect flow data from 39,758 MacOS devices on an enterprise network to study the network behaviors of individual applications. We collect flows *locally* on-device and can definitively identify the application responsible for every flow. We also develop techniques to distinguish "endogenous" flows common to most executions of a program from "exogenous" flows likely caused by unique inputs. We find that popular MacOS applications are in fact using the network securely, with 95.62% of the applications we study using HTTPS. Notably, we observe security sensitive-services (including certificate management and mobile device management) do not use ports associated with secure communications. Our study provides important insights for users, device and network administrators, and researchers interested in secure communication.

1 Introduction

Most desktop applications make connections over the network to pull content, check for a license, or save a resource. While encrypted communications were once rare, Google recently reported that 95% of Chrome connection on Mac platforms use HTTPS [1]. However, studies on TLS adoption have not accounted for non-browser applications — the vast majority of networked software. Measuring per-application network use requires a vantage point that can map network traffic to the originating application. At a small scale, tools such as personal firewalls can let an individual determine what sorts of connections applications on their device are making, but are limited to a single device. Network-layer telemetry tools like middleboxes can provide a higher-level view of desktop flows but lose definitive context of the application responsible for the flow.

In this paper, we bridge these two viewpoints, bringing the local context of observing an application making a connection but providing visibility across 39,758 MacOS devices in an enterprise setting. At scale, we run into challenges when examining the traffic from even a single application across many hosts,

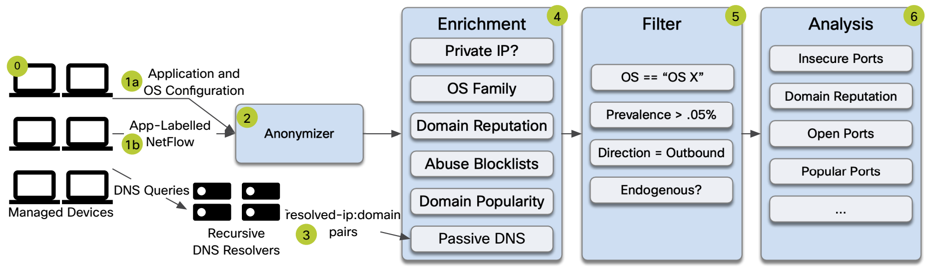

Fig. 1: Telemetry collected from a population of 39,758 MacOS devices contains application-labelled netflow that is anonymized and enriched for further analysis.

as some connections are made on behalf of the user, while others are part of what an application does natively. For example, a word processor may check for updates while simultaneously loading embedded content in a particular document. The software developer is only responsible for one of these network uses, so differentiating these two classes of behaviors is critical to properly evaluate the security behavior of the application. Our perspective is interesting because we are able to study a large population of hosts, with a vantage point into each host in the population that can attribute traffic back to a specific process. Studying this as a population of hosts allows us to reasonably and confidently make claims about an application's behavior, while in individual studies we could observe point data but would be limited to describing an instance of an application. These measurements and insights can provide valuable context for administrators and incident responders to understand expected behaviors of applications on managed devices and for application developers to better understand the holistic connections their applications are making. Our work makes the following contributions:

- We perform the first large scale study of application network behavior on desktop applications
- We demonstrate and evaluate a technique to differentiate user-triggered and application-endogenous behavior
- We examine listening ports, reputation of over 282,000 domains, and over *three billion* connections to evaluate the attack surface of 143 desktop applications
- We investigate popular applications such as Microsoft Office and MacOS daemons that do not entirely use secure communication channels

2 Methods

figure 1 shows a high-level view of our phased methodology and analysis pipeline. Our goal is to combine our data sources while maintaining the anonymity of any users, use accurate matching techniques, and deal with the challenges of combining billion-row datasets. We begin in **Phase 0** with a population of around 39,000 IT-managed devices, all of which are desktops or laptops. These devices belong to a Fortune 500 company and we consider these to be representative of a typical enterprise managed device. We are interested in two types of collected telemetry: application and OS configuration **(1a)** and application-labelled flow data **(1b)**. Application and OS configuration contains information about currently installed processes on the system, OS version information, and more. Application-labelled flow data contains metadata about connections made on behalf of some application on the device. In their raw form, both of these data sources contain fields that include potential human-identifying information such as usernames, machine names, and MAC addresses. **Phase 2** removes or encrypts all human identifiers. **Phase 3** consists of a data source that has passively collected DNS queries generated by all devices on the network. **Phase 4** transforms and enriches the now anonymized data feeds from **1a** and **1b**. Flow data is first enriched with fields indicating the OS family of the device that generated the traffic, and if the source and destination IPs are each private or public. Then, the flow data is joined with the `(domain,resolved_ip)` pairs observed from the passive DNS data, recovering the domains associated with each flow. Finally, using the recovered domain we check against blocklists, popularity, and reputation sources and look for any matches at the second-level domain.

In **Phase 5** we apply filters to the enriched flow data and configuration data. We restrict both datasets to only MacOS devices and only consider traffic from applications installed on more than 5% of the device population. We further restrict flows to outbound connections (i.e., a device talking to some remote server). Finally, we make a determination if a particular connection for an application is common across installs of that application, and restrict analysis to those that we believe to be common. We performed the data processing and analysis on a large Spark[2] cluster. In total, our pipeline and analysis took approximately 4,200 CPU hours and 37,000 GB memory hours.

2.1 Data Collection and Characterization

Application Labelled Flows (Phase 1b): Our primary dataset is telemetry from Cisco AnyConnect VPN's Network Visibility Module (NVM) [3]. This tool records all network traffic from a host and critically, the process associated with the traffic. The records generated by NVM include the source and destination IP address, source and destination port, flow size and duration, as well as the name and SHA256 hash of the process binary associated with the flow. We use one day's telemetry collected from NVM on a large enterprise network in September 2020, which is about 320 GB of compressed JSON. This contains records from 39,758 hosts, 143 unique applications, and 3,211,451,385 total flows. This dataset contains only network telemetry, so an application that generates no network traffic is not represented. There were two challenges with using this data. First, flows are bidirectional but do not indicate if the flow originated on the laptop or a remote server. We address this in Subsection 2.3. Second, flows frequently do not have a domain name associated with the destination, so we use the observed passive DNS data to recover domains from IP addresses.

Application and OS Configuration (Phase 1a): We use OSQuery [4] to learn about the state of a device, its installed applications, and recent network activity. Our dataset contains the query results from 35,678 managed endpoints from a single day in September 2020. [3]

Passive DNS (Phase 3): At each recursive DNS resolver on the network, there is a passive collector that records logs of DNS queries and responses. Because these collectors are internal to the network, observing a query/response means that a device inside the network made a query for some specific domain that had a specific IP in the response. All devices in our population are configured to use these DNS resolvers. We use logs collected from the same 24 hour window as the application-labelled flows. This consists of 9.5 billion query/response pairs, approximately 115 Gb of compressed Parquet [5]. We use this data to perform reverse DNS lookups to recover the domains of an observed IP in the flow metadata.

Additional Sources of Enrichment (Phase 4): We use the Snort IP Blocklist [6] and a paid commercial feed of domains associated with spam campaigns as sources of maliciousness. We use the Umbrella Top 1 Million domains list [7] as a proxy for goodness and the Umbrella Investigate Risk Score [8] as a source of domain reputation.

2.2 Data Preparation and Preprocessing

Anonymization (Phase 2): In raw form, the flow data, configuration data contain human-identifying information. The datasets that the research team had access to have had all human-identifying fields (e.g., usernames, MAC addresses) removed with the exception of the machine name. The machine name was encrypted with a key the research team did not have access to; this field was pseudo anonymized so there remained a way to track flows associated with the same device and calculate the number of unique devices in the population that share some trait. Appendix A contains a full discussion of our data ethics.

Passive DNS (Phase 3): We used passively observed DNS data to generate "lookups" that we could use to recover a domain name from an IP address. We use `A,AAAA` records and collected pairs of `rdata,rrname` observed responses, these two fields contain no human identifiable data and thus bypasses the anonymization step. From this key-value pair, we can enrich observed IP addresses from the flow data with all observed domains that resolved to that IP.

Application and OS Configuration (Phase 1a): We used a single query [9] to obtain a snapshot of all active listening ports on the system and the process that owns them.

Application Labelled Flows (Phase 1b): We perform three lightweight operations on the data before it proceeds to the fusing step: remove any records that are `NULL`, add a label with the OS family (Windows or MacOS), and use RFC 1918[10] to label each source and destination address as "Private" or "Public".

[3]This is fewer hosts than are in the flows dataset, but certainly large enough to be a representative sample. OSQuery data was not available for every host that NVM was installed on.

2.3 Enrichment (Phase 4)

We fuse together application-labeled flows with the observed `ip:domain` pairs from pDNS to recover domains from the flow metadata. While we were not able to match exact pDNS queries to hosts due to a lack of consistent identifiers, the pDNS data was collected from the same network in the same 24-hour period. When we observe the same IP on the same network within a time window we have high confidence that it resolved to that domain. In cases where a single IP resolves to multiple domains, we report all matched domains.[4] Enriching the flow data with the DNS lookups requires a join between 3.2 billion flows and 2.9 billion `ip:domain` pairs. Some IPs resolve to orders of magnitude more domains (e.g., AWS, CloudFlare) which causes a skewed join. Skewed joins are painful for many distributed compute systems, including ours, it took us multiple iterations and about 1,000 CPU hours to make this single join work.

Traffic Direction: NVM records symmetrical flow metadata, but does not indicate if the origin of the connection was remote or local, so it is unclear if the source IP was the local device or the remote server. We did not use ports as an indication of which end was the remote (e.g., if destination port was `HTTP/80`) because if we assumed well-known or registered ports were always the remote, that would bias any research questions involving port usage. We did not have access to an authoritative record of the IP address allocated to a given device, which could have helped bypass this challenge.

We used RFC 1918 to build a simple heuristic to determine traffic direction. We then label each flow as Internal, Outbound, Inbound or NAT, for more information see Appendix B. We hypothesize that most traffic should be Outbound as most applications follow a client-server pattern where an application initiates connections to some Internet-facing service. We then looked at the distribution of traffic and found that 81.9% of traffic was Outbound, which supports that hypothesis. Internal consisted of 2.86% of traffic, Inbound 15.19% and NAT 0.05%. Outbound represents more than 80% of the traffic and we can best reason about it, so we restrict our analysis to this type only. We revisit inbound traffic using OSQuery in Subsection 3.5.

2.4 Filtering (Phase 5)

We restricted our dataset in three ways. First, we restricted our analysis to only MacOS applications, as we had more devices running MacOS in the population. Second, we only consider applications with observed traffic from more than 5% $1,987$ of devices. This excludes rare applications (e.g., Steam) as they would not be indicative of the applications installed on a typical enterprise device. Finally, we removed traffic associated with `*.acme.TLD`, the domain of the enterprise where data was collected. In addition to this being a non-negotiable requirement from the data provider, this allows our results to be more generalizable. `*.acme.TLD`

[4] This overapproximates possible domains, risking misclassifying an IP as disreputable in our analysis. Because our results do not identify any endogenous domain as disreputable, this concern is moot.

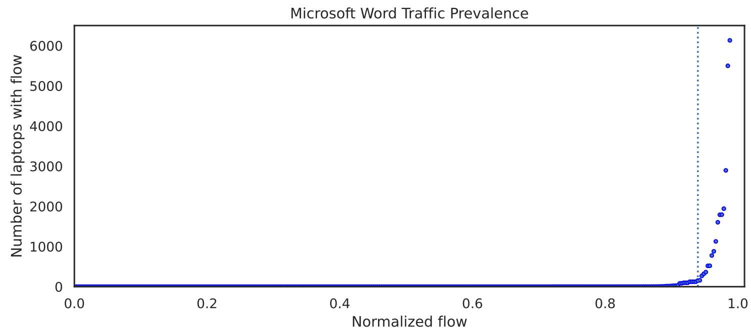

Fig. 2: Each dot represents a unique connection `(port,2LD)` made by an Microsoft Word. Most connections (239 of 323 total) appear on a single device, however a small number appear on many more. We apply our endogenous heuristic to draw a line at a prevalence of 159 devices, which includes the most prevalent 6% flows.

is likely to appear endogenous when using our techniques and we would like to isolate this from actual endogenous traffic. Many users visiting the ACME homepage would make that domain appear to be part of endogenous traffic for a browser. This lets us focus on application behavior in general and less on behavior within a specific domain or IT configuration. In practice, this removed 3.2% of flows overall, or 1.2% of flows at the 2LD.

Application Endogenous Traffic: Throughout this work, we considered that a user can cause an application to make connections to destinations that the developer had no knowledge of. For example, consider two connections from Microsoft Word, the first to `mastercard.com` on port 443 across 97 unique devices, and the second to `office.com` on port 443 across 5,498 unique devices. Intuitively, these are two different types of connections. It is likely that Word connects to `office.com` because of code that the application developer wrote, while it's also more likely that a user action triggered the network connection to `mastercard.com`.

More generically, consider an application installed and used by thousands of independent users and the popularity of individual network connections made by that application. If the same connections are made by many different users, either these users are all using the application in the same way or the application itself has a common behavior using the network. As these connections become more and more popular it is more likely that it is application behavior. Alternatively, if an extremely common user behavior triggers a network request, if it is popular enough it may as well be owned by the application. For example, if hitting "save" in Word triggers a network connection, that may as well be considered application behavior because it is ubiquitous across users.

We distinguish these types of traffic into two categories, *application endogenous* versus user triggered, or *exogenous*. Endogenous is a term borrowed from biology that indicates that something grows or originates from within an organism. This differs from something exogenous that grows or originates from outside an organism. The tie here is that such endogenous traffic originates from within an application's native behavior, not external influences.

Given the collected metadata, we observe that *application endogenous* traffic will appear on more devices than user triggered traffic. Second, we observe that if, for each application, we plot each flow against the number of unique hosts it appeared on, a fairly regular pattern occurs, as seen in figure 2. Our intuition is that this is following a Pareto distribution. However, different applications will have different distribution parameters, complicating a simple decision criteria. To solve this, we identify the point on the observed distribution where the curve grows steeply, i.e., the "elbow" of the plot [11]. We then consider any types of flows that fall at or higher than the elbow to be endogenous to the application and flows that fall below to be user triggered. We acknowledge that there are some confounds with this approach, namely if there is some extremely popular destination, e.g. `google.com`, it may be indistinguishable from an endogenous behavior due to its ubiquity. We restrict most of our analysis to these endogenous connections, and while there are almost certainly interesting things happening in the exogenous flows, our focus is on application native behavior which endogenous traffic better represents. We further explore this approach and cases where it succeeds and fails in Subsection 3.1.

3 Analysis

We observed traffic from 39,758 unique MacOS hosts over a 24-hour period in September 2020. Each host in this population is a user-facing endpoint, such as a desktop or laptop. We observed 143 unique applications installed on more than 5% of the population ($\geq$ 1,987 hosts). These applications generated 3,211,451,385 total connections. After recovering domains with pDNS, we observed 282,715 unique domains, 61,607 (21.8%) of which were unique second-level domains, e.g., `google.com`. The typical endpoint in our population produces a large number of unique flows with a median 23,642 connections, where a unique flow is defined by tuple `(application,destinationport,domain)`.

3.1 How many of an application's connections are endogenous?

We applied the "elbow" filtering techniques described in subsection 2.4 and examined how traffic changed before and after this filtering. First, when considering the number of flows per application, defined by tuple `(application,destination-port,domainat2LD)`, before filtering there were median 17.50 unique flows, after there were median 3.0. This reduction in types of traffic suggests that many applications have few types of endogenous traffic. The maximum number of unique flows before filtering was 70,175 and after 2,309. When considering the maximum case, the application here is Google Chrome, and our elbow approach struggles to reduce the endogenous traffic to a manageable set, although it does reduce by 96.7%. Chrome and other browsers have orders of magnitude more apparent endogenous connections than other applications. We explore browsers' behavior with elbow filtering in Subsection 3.6. We see similar trends between number of

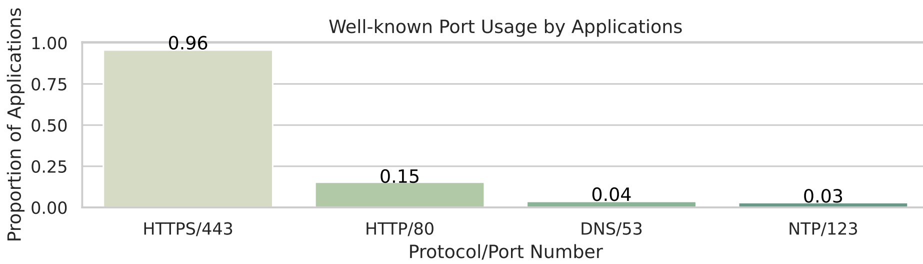

Fig. 3: Well-known ports used as part of application's endogenous traffic. 96% of applications use HTTPS in endogenous connections.

flows and number of second-level domains. The median number of second-level domains connected to per application was 15.00, after filtering it was 3.0. In the maximum case for Google Chrome, before there were 50,691 unique 2LDs, after 2,288. Lastly, we consider the destination ports used per application. In the median case an application used 2 ports before filtering, after there was 1. In the maximum case, there were 9,427 ports connected to by an application, afterwards there were 9. In this case, Google Chrome was the maximum before, but after it filtered down to 4 ports; VMware-Nat was the maximum after filtering.

By considering only traffic that is endogenous to an application, we have a tractable dataset to consider within each applications as most applications do not have many types of endogenous traffic. This is not a tautologically true statement by construction. There are cases (namely browsers) that do have many types of endogenous traffic. Within browsers, there are types of traffic (e.g., email) that are common but are not endogenous behavior. This technique does separate those types of connections from those less common, such as checking niche websites.
Takeaway: There are few connections made by applications that are common across hosts, so differentiating endogenous versus exogenous connections can drastically reduce the dataset size.

3.2 What ports do applications use to communicate?

Building on results from subsection 3.1, we know that in the median case, an application uses few ports, both before and after looking only at endogenous traffic. We find that most applications only use well-known ports (below 1024); in endogenous traffic, there are only 4 ports used by more than 2 applications: HTTPS, HTTP, DNS, and NTP. figure 3 shows that 95.62% of all applications we consider use HTTPS in their endogenous traffic, while 15.33% use HTTP. These are not mutually exclusive; an application that uses both HTTP and HTTPS is counted in both categories. Some common ports (`SMTP`) are not included here as they only had connections to `*.acme.TLD` which were excluded. We acknowledge that having a connection on a port does not guarantee the intended protocol is used, we assume that this is the case but did not have access to PCAPs to validate this assumption.

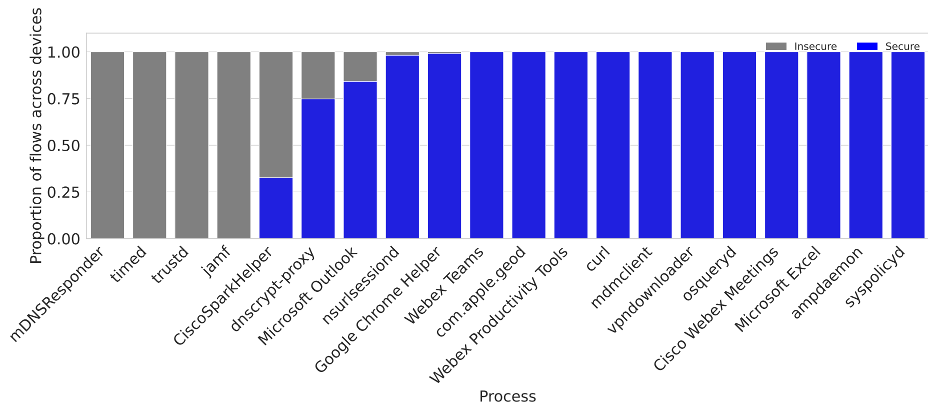

Fig. 4: Distribution between secure and insecure ports for 20 most popular applications. Insecure connections made by applications are further explored in subsection 3.3

Takeaway: Our unique perspective reinforces the common wisdom that HTTP/S has become the main communication protocol for virtually all traffic.

3.3 What applications are not using secure ports?

We are looking only at "well known" ($< 1,024$) ports. We assume that if there is traffic on a well known port, that traffic is using the protocol expected on that port, but we do not have ground truth for this. While there is not an authorative list of "secure" ports, we define the following ports as secure: `SSH/22,HTTPS/443,NTTPS/563,LDAPS/636,IEEE-MMS-SSL/695,SILC/706,FTPS/-989,FTPS/990,TELNET-SSL/992,IMAPS/993,POP3S/995`. We then define an insecure port as a port not being in that set. Some exceptions here that we did not account for are ports designed to be secure over plaintext, such as Kerberos, or ports where the secure component is $> 1,024$. In figure 4 we consider all traffic for an application across all devices and plot the distribution between insecure and secure ports. There are applications that appear to transmit sensitive data or contain some security feature that communicate over insecure channels, such as `trustd` and `jamf`. Next, we explore the connections made by these applications in more detail. **Takeaway**: While most endogenous traffic is secure by default, there are notable instances of prominent applications featuring insecure communication channels.

TrustD: `trustd` is a system daemon on MacOS that manages and updates the system trust store, including checking for certificate revocation. In our study of insecure ports we found that trustD uses HTTP/80 for all of its communications. This gave us pause, as it is a system-level process that is responsible for a critical security function, thus we expected it to use TLS. Upon further investigation, we discovered that this process talks to a limited set of domains. These domains appear to be either authenticated Apple services that we were not able to determine their exact use and purpose (`pancake.apple.com` and `mesu.apple.co-m`) or OCSP checks from major CAs (`ocsp.digicert.com,ocsp.apple.com`, etc.). OCSP is a certificate revocation protocol that uses a signed request and response. Although the data is authenticated, the sender is not, which exposes a potentially exploitable attack surface if an attacker could craft a malicious OCSP or certificate that the service attempts to load. **Takeaway**: `trustd` gained some attention in the news and our findings align with what other researchers[12] found and further explains why OCSP must happen over HTTP.

Microsoft Office: Microsoft Excel and Outlook are the only office applications that break the "top 20" to appear on figure 4, but we will examine all the Office products (Excel, Outlook, Powerpoint, Word, OneNote, AutoUpdate). Of those, Outlook is the only product to have any endogenous traffic on an insecure port. Outlook's has connections to 164 unique 2LD with port `HTTP/80`. Marketo, a marketing automation company, appears to own 153 of these domains. These Marketo domains appeared on the same number of hosts, we suspect this is because they are all hosted on the same IP and the pDNS domain recovery matched all of them. These domains appeared on $< 2\%$ of hosts that were running Outlook so we suspect the elbow heuristic failed for this case. Of the remaining 11 domains, 3 are additional marketing services e.g. `sendgrid.net`, 5 are CDNs or IaaS providers including `amazonaws.com,akamaiedge.net`, 1 is `office.com`, and the remaining 2 are HR service providers. `office.com` appears both with connections on `HTTP/80` and on `HTTPS/443` suggesting that not all the connections Outlook makes back to Microsoft use TLS.

Device Management (jamf): jamf is a tool used for IT device and policy management for MacOS and typically runs with root or sub-root privlege. jamf has a single insecure connection, on `HTTP/80` to `akamaiedge.net`, which largely means that it could be connecting and pulling down anything hosted by Akamai, a large CDN.

CiscoSparkHelper: CiscoSparkHelper has 7 endogenous flows and uses 3 ports, 444, 5004, 33434. On both port 5004 and 33434, there are 3 connections each, two to `(wbx2.com` and `webex.com)` which appear to be domains owned by Cisco WebEx; the third connection is to `amazonaws.com`. For port 444 we see a single connection to `wbx2.com`. Port 444 official use is for SNPP, while 5004 and 33434 are used for VoIP audio/video calls. 5004 and 33434 were not included in our "secure" ports as they are not $< 1,024$.

3.4 Are applications communicating with reputable domains?

Next, we are interested evaluating the reputation of domains that applications connect to using proxies for danger and safeness. We focus on 2LDs, and only those that are part of the endogenous set of traffic for an application. We use four proxies for danger: if a domain resolved to an IT-managed DNS blackhole, if a domain appeared in a Snort IP blocklist [6], if a domain appeared in a commercial domain abuse feed, or if a domain appeared as part of DGA for a handful of well known campaigns (Mirai, Zeus, Cryptolocker). We use two proxies for safeness, the reputation score of a domain from OpenDNS Umbrella [8], and the position of a domain on the Umbrella Top 1M [7]. We found no domains that matched any of our proxies for danger, and no domains that flagged as malicious in OpenDNS Umbrella. In all 143 applications, we found that at least 50% of 2LDs per application are in the Umbrella Top 1M Popular Domains, and 15 applications had 100% of their 2LDs in the Top 1M.

Takeaway: Domain reputation services are correctly evaluating endogenous traffic as benign, which is not surprising and would have been interesting if they were not. "Popular" sites lists from Alexa, OpenDNS, etc. can be used by incident responders and network administrators to help determine if a particular site is worth further investigation. However, there's a large portion of endogenous traffic in popular enterprise application that do not appear on these lists. This could be a limitation of these popularity lists where a more focused list of common traffic for a specific application would be useful.

3.5 What is the attack surface that applications expose through open ports?

In this experiment, instead of using the application-labelled flow data, we use configuration information generated through OSQuery. The query-pack for "listening-ports" provides information about all listening ports assigned to a process. This is a better way of identifying listening ports than using the flow data since we have an authoritative snapshot of the ports used, and can avoid using heuristics to determine traffic direction. We find on average an application exposes 11.1 listening ports per host, with a median of 1. A few outliers cause this skew, these applications are listening on many different ports: Docker (254 unique ports), dnscrypt-proxy (67), mDNSResponder (28) CiscoSparkHelper (28), Safari (16), vmware-natd (14), Microsoft Teams Helper (11), Spotify (9) zoom.us (4) and Dropbox (3). Docker is, by far, the application that exposes the most listening ports, which in retrospect is logical. Developers can run containers on their laptop using Docker, and can configure the ports that the container listens to on the host OS. Interestingly there are also applications that are exposing more ports than the median that we did not expect to have such a network presence, such as: BetterTouchTool (2), HP Device Monitor (3), [VisualStudio] Code Helper (Renderer) (4). We were unable to confidently attribute these connections back to the domains where they originated because a lack of consistent identifiers between OsQuery and NVM. **Takeaway**: Applications that expose listening ports add exposed attack surface that deserves further investigation. There have been cases of applications running webservers that they use to communicate with their web-clients that lead to vulnerabilities [13,14,15]. We leave determining what applications are doing with these listening ports for future work.

3.6 How is endogenous traffic represented in browsers?

As mentioned in Subsection 3.1, our approach for determining what traffic is endogenous for an application struggles to narrow down traffic observed from browsers. We suspect that this is because browsers are entirely user-triggered applications and there are likely large similarities in browsing destinations between users. In our dataset, we observed traffic for three browsers: Google Chrome, Mozilla Firefox, and Apple Safari. Browsing behavior for users between browsers is fairly similar as common tasks (email, social media, news) are not functions exclusive to any one browser. Across all three browsers, there are 2,671 unique connections (domain at 2LD, port), 1,720 (64%) of those are common across at least two browsers, and 1,335 (50%) are common across all three. Looking at endogenous connections exclusive to a single browser, Chrome has 593 unique connections, Firefox has 187 connections, and Safari has 171. This mimics the rough pattern of popularity of each of these browsers within our device population, so we do not make any conclusions about one of these browsers having a smaller set of endogenous behavior. **Takeaway**: Browsers have fundamentally different types of traffic from other desktop applications, but their traffic is similar to each other.

4 Related Work

Host-based anomaly detection has been a staple of security research since the early 2000s [26] and numerous works have explored addressing anomalous activity on a host [40,41,22]. Anomaly detection has been well explored at the network level with tools like Snort [36] and NetFlow based techniques [43,29]. Identifying applications and operating systems through observing network traffic and reconnaissance has also been well explored [34,25,42], including single packet fingerprints of operating systems [38,37]. There are also enterprise software offerings that can identify the application generating network traffic and measure aspects of flows [20,21,16,31]. Researchers have also used TLS fingerprints as a method of identifying software [17], provided techniques to impersonate common software to circumvent censorship [27], and shown how parrot-based circumvention can fail [30]. By contrast, we act as an omniscient passive observer of traffic and study the connections made by applications focusing on the common connections rather than the anomalous.

Internet-scale measurement has been conducted to look at HTTPS and TLS implementation and weakness [24,39], building features to measure DNS [19,32], or scraping the internet to measure the use of cryptographic libraries [35]. With regard to work in desktop application security, in 2006 Bellissimo et al. studied the updated mechanisms of popular desktop applications [18]. In 2012, Georgiev et al. found that the implementation of SSL certification validation was fundamentally broken in many widely used libraries [28]. In contrast, our work does not examine TLS implementation but instead measures apparent TLS usage. In 2017 Dormann published a blog about the consequences of insecure software updates [23], and a year later, Microsoft was still distributing software over HTTP [33]. To the best of our knowledge, there has not been a prior large-scale, extensive study of the security posture of desktop applications communication channels.

References

1. https://transparencyreport.google.com/https/overview
2. https://spark.apache.org/
3. https://www.cisco.com/c/en/us/td/docs/security/vpn_client/anyconnect/anyconnect47/administration/guide/b_AnyConnect_Administrator_Guide_4-7/nvm.html
4. https://osquery.io/
5. https://parquet.apache.org/
6. https://snort.org/downloads/ip-block-list
7. https://umbrella.cisco.com/blog/cisco-umbrella-1-million
8. https://docs.umbrella.com/investigate-api/docs/security-information-for-a-domain-1
9. https://github.com/osquery/osquery/blob/master/packs/incident-response.conf#L211
10. https://tools.ietf.org/html/rfc1918
11. https://github.com/georg-un/kneebow
12. https://blog.jacopo.io/en/post/apple-ocsp/
13. https://nvd.nist.gov/vuln/detail/CVE-2019-13450
14. https://nvd.nist.gov/vuln/detail/CVE-2019-13449
15. https://nvd.nist.gov/vuln/detail/CVE-2019-15006
16. Anderson, B., McGrew, D.: Identifying encrypted malware traffic with contextual flow data. In: Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security. pp. 35–46. AISec '16, ACM, New York, NY, USA (2016)
17. Anderson, B., McGrew, D.: Tls beyond the browser: Combining end host and network data to understand application behavior. In: Proceedings of the Internet Measurement Conference. p. 379–392. IMC '19, Association for Computing Machinery, New York, NY, USA (2019)
18. Bellissimo, A., Burgess, J., Fu, K.: Secure Software Updates: Disappointments and New Challenges. HotSec pp. 37–43 (2006)
19. Chen, Y., Antonakakis, M., Perdisci, R., Nadji, Y., Dagon, D., Lee, W.: DNS Noise: Measuring the pervasiveness of disposable domains in modern DNS traffic. Proceedings - 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014 pp. 598–609 (2014)
20. Cisco Systems, Inc.: Cisco Security Analytics White Paper (2018), https://www.cisco.com/c/dam/en/us/products/collateral/security/stealthwatch/white-paper-c11-740605.pdf
21. Cisco Systems, Inc.: Cisco Encrypted Traffic Analytics - White Paper (2019), https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf
22. Denning, D.E.: An intrusion-detection model. In: 1986 IEEE Symposium on Security and Privacy. pp. 118–118 (1986)
23. Dormann, W.: The Consequences of Insecure Software Updates (2017), https://insights.sei.cmu.edu/cert/2017/06/the-consequences-of-insecure-software-updates.html
24. Durumeric, Z., Ma, Z., Springall, D., Barnes, R., Sullivan, N., Bursztein, E., Bailey, M., Halderman, J.A., Paxson, V.: The Security Impact of HTTPS Interception (2017)
25. Durumeric, Z., Wustrow, E., Halderman, J.A.: Zmap: Fast internet-wide scanning and its security applications. In: 22nd USENIX Security Symposium (USENIX Security 13). pp. 605–620. USENIX Association, Washington, D.C. (Aug 2013)

26. Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T.: A sense of self for Unix processes. pp. 120–128. Institute of Electrical and Electronics Engineers (IEEE) (Dec 2002)
27. Frolov, S., Wustrow, E.: The use of TLS in Censorship Circumvention. Proceedings of The Network and Distributed System Security Symposium (2019)
28. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: validating ssl certificates in non-browser software. In: ACM Conference on Computer and Communications Security. pp. 38–49 (2012)
29. Hofstede, R., Bartoš, V., Sperotto, A., Pras, A.: Towards real-time intrusion detection for NetFlow and IPFIX. 2013 9th International Conference on Network and Service Management pp. 227–234 (2013)
30. Houmansadr, A., Brubaker, C., Shmatikov, V.: The parrot is dead: Observing unobservable network communications. Proceedings - IEEE Symposium on Security and Privacy pp. 65–79 (2013)
31. Kleopa, C., Judge, C.: Snort - OpenAppID (2015), `https://www.snort.org/documents/openappid-detection-webinar`
32. Kountouras, A., Kintis, P., Lever, C., Chen, Y., Nadji, Y., Dagon, D., Antonakakis, M., Joffe, R.: Enabling Network Security Through Active DNS Datasets. Research in Attacks, Intrusions and Defenses pp. 1–21 (2016)
33. Leonhard, W.: Microsoft is distributing security patches through insecure HTTP links — Computerworld (2018), `https://www.computerworld.com/article/3256304/microsoft-is-distributing-security-patches-through-insecure-http-links.html`
34. Lyon, G.F.: Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure, USA (2009)
35. Nemec, M., Klinec, D., Svenda, P., Sekan, P., Matyas, V.: Measuring Popularity of Cryptographic Libraries in Internet-Wide Scans pp. 162–175 (2017)
36. Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proceedings of the 13th USENIX Conference on System Administration. p. 229–238. LISA '99, USENIX Association, USA (1999)
37. Shamsi, Z., Cline, D.B.H., Loguinov, D.: Faulds : A Non-Parametric Iterative Classifier for Internet-Wide OS Fingerprinting. ACM Conference on Computer and Communications Security pp. 971–982 (2017)
38. Shamsi, Z., Nandwani, A., Leonard, D., Loguinov, D.: Hershel: Single-Packet OS Fingerprinting. IEEE/ACM Transactions on Networking **24**(4), 2196–2209 (2016)
39. Springall, D., Durumeric, Z., Halderman, J.A.: Measuring the Security Harm of TLS Crypto Shortcuts pp. 33–47 (2016)
40. Wagner, D., Soto, P.: Mimicry attacks on host-based intrusion detection systems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. p. 255–264. CCS '02, Association for Computing Machinery, New York, NY, USA (2002)
41. Yeung, D.Y., Ding, Y.: Host-based intrusion detection using dynamic and static behavioral models. Pattern Recognition **36**(1), 229–243 (2003)
42. Zalewski, M.: P0F V3: Passive Fingerprinter, 2012 (2012), `http://lcamtuf.coredump.cx/p0f3/README`
43. Zhenqi, W., Xinyu, W.: NetFlow based intrusion detection system. Proceedings - 2008 International Conference on MultiMedia and Information Technology, MMIT 2008 pp. 825–828 (2008)

A Data Ethics

While working on this project, we followed all institutional procedures from all affiliated institutions. Our IRB reviewed our proposal and datasets and determined that this was not human subjects research. All human and machine identifiers in our dataset have been removed and replaced with encrypted versions that are encrypted with a key that the research team does not have access to. All telemetry was collected through existing monitoring infrastructure that has strict ACLs. Furthermore, all telemetry was collected from corporate managed and owned devices where users are made aware that that the devices are monitored for security and compliance. Throughout our analysis we focus on the network behavior of applications not individual users. Any individual user's data could be excluded from our dataset without impact to our findings. We made no attempt to find evidence of sensitive actions or non-work-related activity (video games, streaming video, social media, etc.) The focus of our research is on the network behavior of applications, not of the individuals using the applications.

B RFC 1918

RFC 1918[10] describes and reserves 3 IP ranges for private use only, we used this to label each source IP and destination IP as "private" or "public". If an IP is "private" then it is not on the Internet, and is instead on some internal/private network. After labeling each flow, there are four possible combinations:

– Private Source to Private Destination (Internal) - Neither end is an Internet facing IP, communication to internal services
– Private Source to Public Destination (Outbound) - Destination is an Internet facing IP, likely an outbound connection
– Public Source to Private Destination (Inbound) - Destination is not an Internet facing IP, so is either a connection from a NAT device to an internal service, or an inbound connection from a public service to a device
– Public Source to Private Destination (NAT) - Both ends have an Internet facing IP, but one must be local device with a NAT IP though we can't tell which