In the Line of Fire: Risks of DPI-triggered Data Collection

Ariana Mirian
amirian@ucsd.edu
University of California, San Diego
San Diego, USA

Alisha Ukani
aukani@ucsd.edu
University of California, San Diego
San Diego, USA

Ian Foster
ian@dns.coffee
DNS Coffee
San Francisco, USA

Gautam Akiwate
gakiwate@cs.stanford.edu
Stanford University
Palo Alto, USA

Taner Halicioglu
taner@taner.net
Independent
San Diego, USA

Cynthia T. Moore
ctmoore@ucsd.edu
University of California, San Diego
San Diego, USA

Alex C. Snoeren
snoeren@cs.ucsd.edu
University of California, San Diego
San Diego, USA

Geoffrey M. Voelker
voelker@ucsd.edu
University of California, San Diego
San Diego, USA

Stefan Savage
savage@ucsd.edu
University of California, San Diego
San Diego, USA

ABSTRACT

Cybersecurity companies routinely rely on telemetry from inside customer networks to collect intelligence about new online threats. However, the mechanism by which such intelligence is gathered can itself create new security risks. In this paper, we explore one such subtle situation that arises from an intelligence gathering feature present in FireEye's widely-deployed passive deep-packet inspection appliances. In particular, FireEye's systems will report back to the company Web requests containing particular content strings of interest. Based on these reports, the company then schedules independent requests for the same content using distributed Internet proxies. By broadly scanning the Internet using a known trigger string we are able to reverse engineer how these measurements work. We show that these side-effects provide a means to empirically establish which networks and network links are protected by such appliances. Further, we also show how to influence the associatied proxies to issue requests to any URL.

ACM Reference Format:
Ariana Mirian, Alisha Ukani, Ian Foster, Gautam Akiwate, Taner Halicioglu, Cynthia T. Moore, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. 2023. In the Line of Fire: Risks of DPI-triggered Data Collection. In *2023 Cyber Security Experimentation and Test Workshop (CSET 2023), August 7–8, 2023, Marina del Rey, CA, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3607505.3607526

1 INTRODUCTION

Reactive security systems—anti-virus/EDR, firewalls, intrusion prevention systems, email filters, etc.—all face significant challenges. First, they must carefully parse and inspect data streams to identify those containing known (or likely) malicious objects. Second, they must also constantly refresh their intelligence to identify new malicious objects as they appear in the wild. Finally, since each of these activities takes place in an adversarial environment, such systems must anticipate that the data they analyze may be designed to evade or disrupt. Indeed, exploitations of data parsing vulnerabilities in security products have been around for decades, from the 2004 ICQ parsing vulnerability in ISS' RealSecure products [28] to the contemporary 2021 ASProtect unpacking heap overflow in Windows Defender [31]. However, the risks associated with intelligence collection in an adversarial environment are less widely documented or appreciated.

In this paper, we identify and investigate one new such issue: how an intentional feature of FireEye's passive network security products unintentionally exposes those products to manipulation. Our work arises from an inadvertent discovery that FireEye's NX Threat Prevention appliance silently reports back to the company all passively collected Web requests containing particular trigger features. These reports are then used to collect the specified content anonymously by issuing requests from a large, geographically distributed proxy network. The design and implementation of this feature may not have fully anticipated the potential impact of adversarial input, however, which is what we address here. In particular, our paper makes three contributions:

- Identifying and characterizing the network behavior of FireEye's hybrid passive/active intelligence collection;
- Developing a methodology for triggering this behavior and conducting a global measurement to infer the worldwide footprint of FireEye's passive monitoring;
- Characterizing the security risks associated with this behavior including the potential for FireEye to be manipulated into attacking third parties.

Finally, while this work only measures the behavior of a single manufacturer (FireEye), we believe this kind of exposure is a general and under-appreciated design issue for the array of products and

CSET 2023, August 7–8, 2023, Marina del Rey, CA, USA

ACM ISBN 979-8-4007-0788-9/23/08.
https://doi.org/10.1145/3607505.3607526

services that collect and curate threat intelligence from Web access telemetry. Our experience further shows that a product's design issues can cause havoc for an entire orgnaization. As such, we discuss options for mitigating these kinds of issues in such systems.

2 BACKGROUND

In July of 2020, one of us requested a file from a private Web server under our control using a client machine in the same institution. Shortly after, the Web server started receiving external requests for the precise file and path that had just been requested. This was both unexpected and suspicious, because neither the file names or path were advertised to the public, the server was not configured to allow directory listing and, in fact, the entire directory was protected by HTTP basic authentication [12]. This was no fluke. Upon changing the file path name and issuing the request again, external requests again appeared seconds later—now for the new path.

Our initial thought was that we were the victim of a data breach and either the Web server itself or one of the switches on the path was compromised. Over the course of a day, working with our institution's incident response team, we eliminated these possibilities and eventually—using measurements we describe later—associated the behavior exclusively with requests being passively monitored by a FireEye NX Threat Prevention appliance deployed on a key perimeter link. A series of email and phone exchanges with FireEye's representatives ultimately confirmed that we had inadvertently stumbled onto a threat intelligence collection feature of their product line. We share this experience to show how these design issues can be used maliciously to map and target customers of the product.

2.1 FAUDE and intelligence collection

This behavior is associated with a feature called FireEye Advanced URL Detection Engine (FAUDE). FAUDE is a component in a suite of features designed to identify and block malicious URLs, including URL rewriting and message clawback in the e-mail context, and a predictive machine learning model similar to Ma *et al.* [16] for classifying potentially malicious URLs for further analysis by offline site content analysis tools. The architecture and the details of the classification strategy are roughly described by Joshi *et al.* [14]. While FireEye's documentation seems to suggest that this feature is associated with their EX (email protection) products, our experience indicates that, at very least, the NX product is being used as a sensor to support that feature as well.

Key to their approach is a classifier that identifies suspicious URLs in real time for further evaluation by a cloud-based service (in the NX context these are unencrypted URLs observed over passively monitored links). This design has two ramifications: first, that there is outbound telemetry about such URLs from the product to the FireEye cloud[1] and second, that the FireEye cloud service then visits those URLs to enable further analysis.[2]

With this information, we realized that we had inadvertently named one of our files in a way that it matched certain lexical features in FireEye's classifier.[3] When we fetched our file over the local network, our HTTP GET request traversed a link that was monitored by one of our institution's FireEye NX appliances, and telemetry about the file was sent to FireEye's cloud-hosted FAUDE service. In turn, this service scheduled a set of HTTP GET requests to fetch a copy of the same content (all of which failed—and then retried—due to our server's HTTP authentication requirement). The source of these fetches was distributed because FireEye, like most companies collecting threat intelligence, must be careful that their data-collection infrastructure is not "fingerprinted" and explicitly blocked by adversaries. Thus, FireEye employs a collection of proxies used to obfuscate their origin.

While our experience was accidental, it would be a mistake to consider the list of such trigger features to be a well-protected secret. A motivated adversary might gain access to a FireEye appliance (or firmware binary) to reverse engineer the classifier, or simply replicate its approximate training approach to identify likely features to test. Indeed, by immediately fetching content that triggers its classifier, FireEye provides an oracle, making such a feature search far easier.[4]

Finally, in the course of our initial investigation, we noted one other facet of this behavior that is critical to this study: that FireEye implicitly trusts the content of the HTTP `Host` header. As per RFC 2616, each HTTP 1.1 request header includes a `Host` field which "MUST represent the naming authority of the origin server or gateway given by the original URL" [11]. The purpose of this field is to support virtual hosting (*i.e.*, in which a single IP is used to support multiple Web servers, using the `Host` field to disambiguate the site-specific content). In the course of our experiments, it became clear that FireEye's collection infrastructure treats the content of the captured `Host` header as the canonical server name for the URL to be fetched. Thus, if the `Host` header is spoofed, FireEye will still follow it. Concretely, if a FireEye appliance triggers on `http://foo.com/trigger.txt`, but the `Host` header specifies `bar.com`, then FireEye's collection proxies issue a request for `http://bar.com/trigger.txt`.

3 METHODOLOGY

While we theorized that this behavior was strictly a byproduct of FireEye's passive network monitoring, one of the confounding factors was the existence of end-system monitoring of unknown provenance (many of our servers employ EDR products—from both FireEye and other vendors). To this end, we devised a technique that leverages hop-limited probes to send triggering content along the path to the server without ever reaching the server itself. Responses elicited by such a probe can be unambiguously attributed to an entity observing the packets in transit. Moreover, by manipulating the initial IP time-to-live (TTL) on probe packets, we were able to iteratively test successive hops on the forward path to further isolate

[1] This is implicitly clear in at least one FireEye presentation which states that "Advanced URL Defense requires two-way threat intel subscription."

[2] It appears that this feature and its network-visible side effects are not well understood, as our institution's incident response team was entirely unaware of it and our FireEye representative initially insisted that there was no FireEye feature that could account for the behavior we were observing. Upon further consultation they identified FAUDE as the source.

[3] The trigger term is related to a popular financial institution, suggesting that the classifier was monitoring for phishing pages.

[4] For this work, we did not have a physical FireEye appliance available to us for reverse engineering, and instead choose to treat the FireEye product as a black box and focused on using our single known trigger term to explore its behavior when triggered.

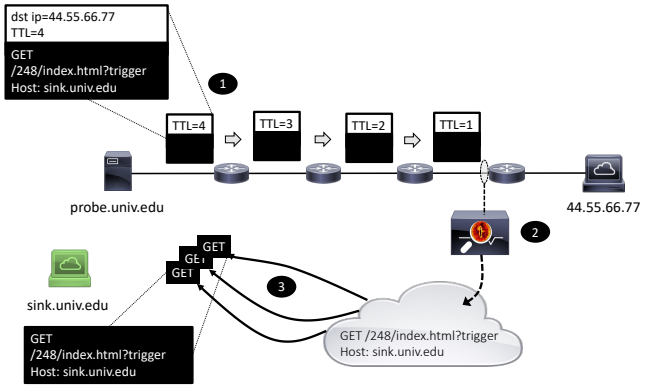

Figure 1: We use a probe server to transmit carefully crafted, TTL-limited HTTP GET requests through suspected FireEye devices to elicit proxy responses to a separate sink server.

the particular link being monitored. Indeed, this was precisely the approach we used to identify the link being monitored at our institution (an inference later confirmed by IT staff).

3.1 Conducting a probe

Figure 1 depicts the apparatus we constructed to test for the presence of FireEye devices: a "probe" server that issues HTTP GET requests on port 80 towards a target host and a "sink" HTTP server that logs all requests received on any port. We deployed the probe server on a special network whose packets were routed around our institution's FireEye appliances, located in the United States. The sink server was located on a different network and both servers used globally routable addresses and had working reverse DNS name resolution. Moreover, we only ran probes on IPv4 addresses, as we did not see evidence that IPv6 addresses were responsive to the same method.

To probe a target host, the probe server establishes a TCP connection[5] to the target IP address on port 80. The probe server then transmits an HTTP 1.1 GET request of the following form:

```
GET /nonce/index.html?trigger HTTP/1.1
Host: sink.univ.edu
```

As shown, the `Host` header in the request specifies the DNS name of our sink server. The request URL incorporates the trigger keyword as a query parameter while the path contains a nonce ensuring that any request to the sink server can be uniquely attributed to a particular probe[6].

The GET request is sent along the open TCP connection to the target (as shown in Step ❶ in Figure 1), but with an initial IP time-to-live (TTL) value that ensures the packet will be dropped by the network at least one hop before the target server (measurement of the forward path length is performed separately via `traceroute`, as described below). Once the request is sent, the probe server closes the TCP connection to the target server by transmitting a `RST` segment (with the standard TTL value). If an attempt to send a probe failed, we did not attempt to re-send a probe in the same scan.

After sending a probe to target T with nonce N, we examine the logs at the sink server. If a request for content with nonce N appears (*i.e.*, from a proxy as shown in Step ❸) then we conclude that the path between our probe server and target T is monitored by some FireEye device that triggered the response (depicted as Step ❷ in Figure 1).[7] By decrementing the initial TTL in subsequent request probes one can explore if this monitoring is in: the stub network hosting the target, a transit network between our probe server and T, or elsewhere in the path.

3.2 Global measurement

Building on our basic probing mechanism, we conducted a global scan for the presence of FireEye monitoring. Starting with roughly 60M IP addresses identified by Censys (on July 22, 2020) as offering service on port 80 [7], we then subsampled to mitigate our impact on the network (and FireEye). Reasoning that network monitoring appliances are typically deployed to protect networks rather than individual hosts, we randomly selected a single representative IP from each /24 (typically the smallest externally routable prefix) to avoid sacrificing significant scanning fidelity. This aggregation results in a target set of ~3M IP addresses.

For each of these target IPs, we performed a series of scans. In each scan, the probe server first performs a `traceroute` to each IP address on the list. This traceroute is used to calculate the forward path hop count, n. We perform five probes towards the target using an initial TTL of $n-1$ (expiring at the hop just before the target, as described previously). Using this procedure we scanned the entire set of ~3M target IPs sequentially on July 24, 2020. We repeated the entire process (including `traceroute`) two additional times, on July 29 and July 31 (we delayed the second and third scan to reduce any additional load we may have caused). Multiple scans allow us to account for instability in the network path between the probe server and each target IP address and helps to mitigate the impact of packet loss on our measurements.

From these initial scans we identified a subset of roughly 50K IPs that were responsive (*i.e.*, that a probe directed towards the Web server at that IP address produced a subsequent proxy request to our sink server for the associated URL).[8] We then performed another series of scans to explore, at finer granularity, where the FireEye monitors were located on the paths. Concretely, we performed a series of scans with ever-decreasing initial TTLs in the probe packets (*i.e.*, $n-2$, $n-3$, etc.). The purpose of these scans was to determine the hop count at which probes directed towards a previously responsive /24 stop triggering FireEye response activity and, thus, infer which link on the path is being monitored.

[5] Our experiments suggest that FireEye devices are stateful and will not trigger on content sent absent an active connection.

[6] Note that we utilize a single trigger keyword in our study. Our objective was not to iterate over potential keywords, but rather to note the subsequent behavior from a triggering keyword.

[7] The absence of such a response is not conclusive (*i.e.*, packets may be lost and we do not fully understand the queuing behavior of FireEye's proxy network), but one can repeat probes to increase confidence.

[8] Note that these results are a lower bound on the number of FireEye-monitored /24 networks. It is entirely likely that there are FireEye customers whose networks are protected by firewalls and our probes will not even reach a monitored link. Moreover, we do not know if this behavior is limited to certain configuration options or software versions. Finally, there are a range of situations where a single FireEye device will monitor a large address range (*e.g.*, a /16), so one should not assume that each monitored /24 represents a distinct device.

Probed IP Addresses		Responsive IP Addresses	
ASN Name	% of IPs (#)	ASN Name	% of IPs (#)
COMCAST-7922	4.07% (135810)	SKB-ASSKBroadbandCoLtd	48.99% (24903)
AMAZON-02	2.97% (99152)	KIXS-AS-KRKoreaTelecom	35.90% (18247)
KIXS-AS-KRKoreaTelecom	2.82% (94140)	HWCSNETHuaweiCloudServicedatacenter	1.77% (901)
DTAGInternetserviceprovideroperations	2.38% (79549)	UCSD	0.52% (265)
ATT-INTERNET4	2.25% (75267)	UCLA	0.52% (223)
AMAZON-AES	1.57% (52453)	VA-TECH-AS	0.44% (207)
FranceTelecom-Orange	1.40% (46776)	CHINANET-IDC-BJ-AP	0.37% (187)
OCNNTTCommunicationsCorporation	1.31% (43740)	BIZLAND-SD	0.33% (169)
ASN-IBSNAZ	1.21% (40474)	ICNDP-AS-KRNamincheonBrodcastingCo.	0.31% (160)
UninetS.A.deC.V.	1.20% (39999)	WSU-AS	0.31% (158)

Table 1: Top-ten AS names of the probed (left) and responsive (right) IP addresses in our global scan.

ASN Category	% of ASes (#)
Computer and Information Technology	43.63% (315)
Education and Research	18.98% (137)
Government and Public Administration	5.96% (43)
Finance and Insurance	5.96% (43)
Service	5.96% (43)
Community Groups and Nonprofits	3.74% (27)
Retail Stores, Wholesale, and E-commerce	3.60% (26)
Manufacturing	2.77% (20)
Media, Publishing, and Broadcasting	2.22% (16)
Construction and Real Estate	1.39% (10)

Table 2: Top-ten AS categories according to ASdb by unique ASN.

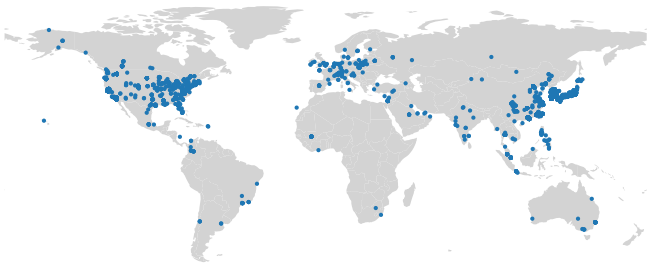

Figure 2: Location of probed IPs that elicited a response.

4 FIREEYE MONITORING FOOTPRINT

Of the original 3,340,474 IP addresses scanned, 50,830 were *responsive* (*i.e.*, led to a proxy request for the probed content). Using the NetAcuity service [19], Figure 2 maps these responsive IPs to their approximate geographical location, showing strong concentrations in North America, Europe and coastal Asia. Using CAIDA's Prefix-to-AS dataset [6], we map these responsive IPs to 771 unique responsive ASNs. Table 1 shows both the top-10 ASes originating IP address in our probe data set and the top-10 most responsive ASes (*i.e.*, which originate responsive IPs). As shown, these two distributions are highly dissimilar. The AS distribution of probed IPs is comparatively flat with the top-10 ASes originating less than 20% of all IPs probed, whereas the top-10 responsive ASes originate almost 90% of all responsive IPs. In particular, we note that two large Korean telecommunications companies account for over 80% of all responsive IPs.[9] Most ASes containing responsive IPs, however, are stub ASes representing individual organizations (*e.g.*, UCLA, BIZLAND-SD, etc.)

We further classify the ASes containing responsive IPs using the Stanford ASdb dataset [34] to assign an organizational category to each AS. Table 2 presents the most common categorizations of the responsive ASes (covering almost 95% of all ASes). Unsurprisingly, Computer Information and Technology is the largest category, reflecting hosting, cloud, and ISP organizations, followed by Education and Research, Government, Finance and Service (together representing almost over 80% of all responsive ASes). As an alternate characterization method, we evaluated historical PTR records from OpenIntel [24] to map responsive IP addresses to DNS names. Of our 50K responsive IPs, we identified 29,232 historical PTR records for 8,582 unique IPs (a given IP can have multiple PTR records). These PTR records map to 860 registered domains, of which at least 52 domains are associated with large educational institutions, 41 are major United States Government agencies, and more than 24 are commercial enterprises—all consistent with the AS-level analysis.[10] This characterization represents a lower bound since PTR mappings are not uniformly available and may be particularly ineffective in cloud deployments.

5 FIREEYE PROXY NETWORK

In addition to mapping the organizations deploying FireEye IDSes, we also briefly analyze the proxy network that issues queries in response to our probes. Our aim is to understand how often responses arrive, their frequency, and whether these requests can be used in a malicious manner (e.g. denial-of-service, or DDoS). In our analysis we consider each individual GET request arriving at our sink server containing the trigger keyword and a nonce. It is frequently the case that multiple requests include the same nonce, often from disparate source IPs. We observed 568 unique source IPs (hereafter "proxy") which collectively issued 235,393 requests to our sink server during the period of our study.

[9] In these two instances, our probes are responsive over a large fraction of /24s in each AS and, frequently, at greater "hop" distance from the destination IP, suggesting an intentionally more comprehensive deployment.

[10] We omit the particular organizations identified due to potential privacy concerns and security risks.

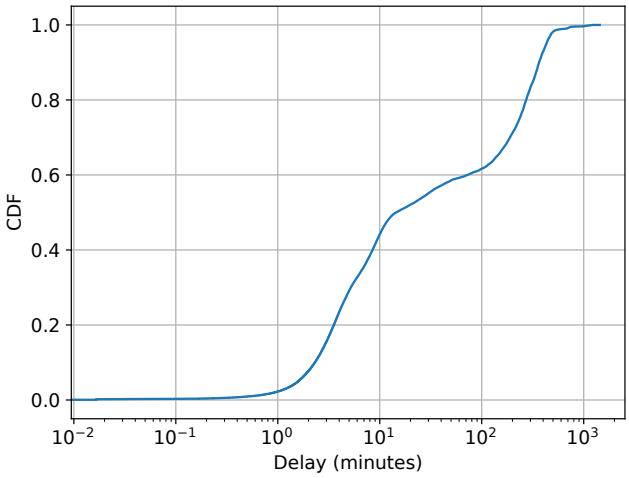

Figure 3: Delay from probe issuance to first response.

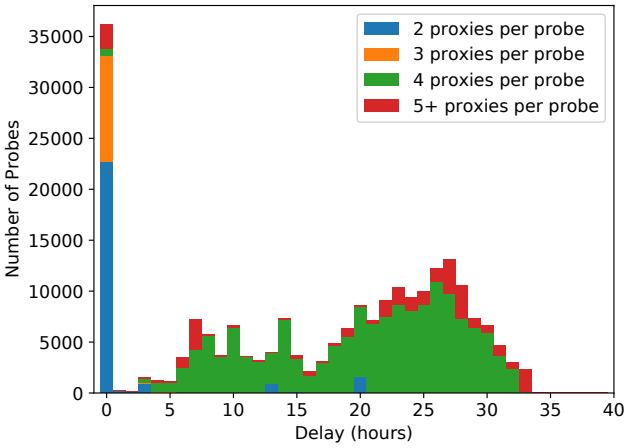

Figure 4: Time between the first and last proxy response broken down by number of responses received.

These requests are generally issued promptly—perhaps to ensure that transient content is captured before it disappears. Figure 3 captures this "reaction time" by plotting the CDF of delay between our probe issuance and receipt of the first request containing the corresponding nonce at our sink server. The median time between a probe and its first proxy response is 14 minutes (*i.e.*, making this behavior a practical oracle), but occasionally it takes almost an entire day to receive a response.

Moreover, multiple requests arriving in response to each probe is the norm, with a median of four requests.[11] It is generally the case that these requests are from distinct proxies (only 32% of probes generate multiple requests from the same proxy IP). Moreover, subsequent requests are generally spread out over time as shown in Figure 4, perhaps to limit network load or validate that the (potentially) offending content is still present.

Finally, we note that seven proxies in particular are orders-of-magnitude more likely to be the first proxies to respond to a probe than the others. They are also the most frequently seen regardless of arrival order and together respond to 96.4% of all probes. Six of these are located in ASN 16509, an Amazon AWS network, and the last is in ASN 4766 (Korea Telecom). We surmise that FireEye employs a small group of machines dedicated to issuing the first probes to suspicious URLs, but then outsources subsequent requests to a proxy network with a globally diverse IP footprint (perhaps to avoid blocking due to fingerprinting). The static nature of these frequently appearing proxies thus provides another oracle, as any content fetched by one of these IP addresses can be inferred to contain terms of interest to FireEye devices.

While the volume and rate of proxy responses do not indicate DDoS potential, we note that our measurements were incredibly conservative in order to reduce load on both FireEye and the networks we were probing. It is entirely possible that a targeted attacker could flood a network with proxy responses, but we do not test the full capability of this hypothesis.

[11]For reasons unknown, there is a long tail with as many as 29 requests being generated to a single probe.

6 RELATED WORK

The complex parsing tasks inherent in security products, combined with their privileged access has made them an attractive target for attackers. Indeed, parsing vulnerabilities in antivirus scanners and network intrusion detection appliances alike have been widely documented [1, 15, 20, 22, 23, 27–29, 31–33]. It is also well understood that attackers may leverage limitations in their detection model [9] or ambiguities in their vantage point [13, 26] to evade detection. Finally, literature documents the reality that modern security products make extensive use of real-time telemetry to deliver intelligence from customers [3, 30]. Since telemetry can include both behavior and content, there can be significant privacy risks if it is exposed to third parties [8, 17]. However, the adversarial risks associated with *how* such telemetry is gathered and acted upon are not well understood.

Closest to our work, Bethencourt *et al.* focused on techniques for exploiting public network security oracles to identify, via scanning, the existence of commercial honeypot and blackhole monitors [2]. However, the similarity is limited to the notion of empirically detecting otherwise covert network monitoring via side effects and does not involve passive monitoring devices or telemetry. Durumeric *et al.*'s more contemporary detection of TLS middleboxes [10] also involves the use of an implicit oracle (TLS protocol impossibility results), but is otherwise distinct. We also note additional works that utilize a limited TTL methodology similar to ours, mostly in censorship-related measurements [4, 5, 25].

7 ETHICS

Our measurement study involved worldwide scanning of open Web servers to identify the presence of on-path FireEye passive monitoring devices. The subject of this measurement was the set of IP addresses that had previously responded to past Censys network scans at port 80. The measurement community has long held that simply connecting to Web sites in this manner imposes no significant harm. Moreover, because of our $n-1$ TTL methodology, the trigger string that we have used to test is not delivered to the Web

server and thus does not even generate an error log. We explicitly close each connection with a RST packet and thus we impose little connection state burden on the target IPs. As well, our scans were widely distributed and low rate—imparting only a handful of packets to each target /24 network during each scan. Finally, our sink host Web server provided content indicating that this was part of a measurement study and provided a contact e-mail address if it caused any concerns or problems.[12]

Due to the security implications of this work, we also have an ethical disclosure responsibility. We disclosed this issue to FireEye—both verbally over the phone and in writing—in mid-July of 2020 as part of our incidental discovery of the behavior. This included the potential for this behavior to be misunderstood by customers (indeed, our first explanation for the external requests for private files was a data breach and the problem was only isolated to FireEye after a lengthy incident response effort at our institution), the risks associated with blindly following the Host header and the potential for security implications (such as DDoS) on third parties as a result. Finally, we have made a point not to publicly document the IP addresses used by FireEye's proxy network or a comprehensive list of the institutions protected by FireEye to minimize the possibility that this information could be used for abusive purposes.

8 DISCUSSION

Responsive data collection of the kind FireEye employs incurs a range of security risks, summarized briefly here:

Enables reconnaissance. As we have demonstrated, FireEye's reactive behavior provides an implicit oracle, revealing the presence of its monitoring. Such information (*i.e.*, which sites and links are protected by FireEye and which are not) can be valuable information for an attacker. Indeed, by mechanically controlling the TTL it is possible to focus the resolution down to a single link, which we were able to demonstrate manually on individual networks.

Allows laundering of attacks. Because FAUDE treats the captured Host header as canonical, it is possible to drive the FireEye proxy hosts to issue arbitrary GET requests to any host (recall that the trigger string can be embedded as a parameter or other off-path URL component). While, in principle, GET requests are meant to be idempotent, the reality is that a range of services encode side effects in GETs. Moreover, a number of Web vulnerabilities can be encoded entirely in a GET request (*e.g.*, the Apache Struts exploit implicated in the Equifax data breach [21]).[13] Thus, an attacker could use this vulnerability to coerce FireEye to attack third parties while disassociating themselves from the attack.

Potentially enables DDoS. In spite of our highly conservative scanning, we were able to drive over 100 proxy requests per second to our sink server via our probes.[14] We see no clear evidence of a rate-limit for the FAUDE cloud service that addresses the overall per-destination IP request rate. This suggests that an attacker, with some fine tuning and further testing, might drive FireEye to commit a distributed denial-of-service (DDoS) attack via its proxy network with a modest number of targeted scans—amplified further by requesting large objects that actually reside at the target. Moreover, because the system treats the Host header as canonical, such an attack can be accomplished without the attacker ever sending traffic directly to the target.

Can create confusion at protected sites. FireEye does not publicize the existence or operation of this feature, nor does it provide a mechanism for users to determine if a request for content is from its proxy network or not. Thus, if a trigger term is inadvertently incorporated in a filename or URL, it may generate external requests to URLs that are private and have never been knowingly shared. From our own experience, such an event can incur a significant investment of incident response time and resources.

As we have mentioned, FireEye is far from unique in actively collecting and mining Web access telemetry. Indeed, we recognize that online data collection is increasingly *required* for most security products—using customers as sensors has become the only effective way to "keep up" with attackers. However, it is important to design such mechanisms to be robust to manipulation and, ideally, sufficiently robust that one can be transparent about their existence and how they operate. Our experience suggests several design principles that, if followed, would reduce risks, such as those we've identified, in practice:

Minimize the oracle. The reconnaissance risk is enabled because every FireEye monitor uses the same set of trigger terms, the trigger terms are non-specific and can be embedded anywhere in the URL (suggesting an n-gram model), requests for matching content are issued promptly, and requests are triggered without validating that content was delivered. Changing one or more of these would have significantly hampered the effectiveness of our measurement survey. A more substantive solution would be to gather suspicious content inline (*i.e.*, as it is being passively monitored by the FireEye device) to eliminate the oracle altogether (although at the price of introducing new privacy risks).

Conservative scheduling of collection requests. In our experiments we received hundreds of thousands of requests from the proxy network, all to the same server. While each was artificially unique, the filenames were identical and none returned content of interest. At a minimum, per destination IP rate-limits would minimize the viability of DDoS attacks. Further, alerting on particular triggers and destination IPs would also provide situational awareness that a data collection capability was being manipulated.

Do not trust the Host header. The HTTP Host header is not reliable and should either be ignored or only used in combination with the destination IP address to which a monitored request was issued. Either implementation would foreclose the possibility of anonymous DDoS or GET attacks.

Although this list of design practices may seem self-evident in retrospect, it is precisely these kinds of behaviors that may escape scrutiny when the effects are not immediately apparent (i.e., when they are undocumented and occur silently). That lack of transparency, in turn, motivates the need and value of measurements like ours to uncover the existence and scope of such issues in fielded systems.

[12] The contact email did not receive any email for the duration of this study.

[13] This risk was demonstrated in 2018 by Netsparker researchers who showed how Googlebot could be so coerced to apply this vulnerability on a third-party site [18].

[14] Note that the relationship between probe rate to responsive /24 and the subsequent request rate appears complex and is decidedly not a simple linear function.

ACKNOWLEDGMENTS

We thank our anonymous reviewers for their time and invaluable feedback. We would also like to thank our collaborators within the UCSD IT and SDSC organizations for their time and help in initially identifying this behavior.

Funding for this work was provided in part by National Science Foundation grant CNS-2152644, the Irwin Mark and Joan Klein Jacobs Chair in Information and Computer Science, and operational support from the UCSD Center for Networked Systems.

In addition, this material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-2038238. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] S. Alvarez. 2007. AntiVirus (In)Security. https://fahrplan.events.ccc.de/camp/2007/Fahrplan/attachments/1324-AntivirusInSecuritySergioshadownAlvarez.pdf.

[2] J. Bethencourt, J. Franklin, and M. Vernon. 2005. Mapping Internet Sensors With Probe Response Attacks. In *Proceedings of the 14th USENIX Security Symposium (USENIX Security '05)*. USENIX Association, Baltimore, MD, USA, 193–208.

[3] L. Bilge and T. Dumitraş. 2012. Before We Knew It: An Empirical Study of Zero-Day Attacks in the Real World. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 833âĂŞ844.

[4] K. Bock, A. Alaraj, Y. Fax, K. Hurley, E. Wustrow, and D. Levin. 2021. Weaponizing Middleboxes for TCP Reflected Amplification. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3345–3361. https://www.usenix.org/conference/usenixsecurity21/presentation/bock

[5] K. Bock, P. Bharadwaj, J. Singh, and D. Levin. 2021. Your Censor is My Censor: Weaponizing Censorship Infrastructure for Availability Attacks. In *2021 IEEE Security and Privacy Workshops (SPW)*. 398–409. https://doi.org/10.1109/SPW53761.2021.00059

[6] CAIDA 2020. Routeviews Prefix to AS mappings Dataset for IPv4 and IPv6. http://www.caida.org/data/routing/routeviews-prefix2as.xml.

[7] Censys [n. d.]. censys.io, July 22, 2020 dataset.

[8] J. Cox. 2020. Leaked Documents Expose the Secretive Market for Your Web Browsing Data. https://www.vice.com/en/article/qjdkq7/avast-antivirus-sells-user-browsing-data-investigation.

[9] Skylight Cyber. 2019. Cylance, I Kill You! https://skylightcyber.com/2019/07/18/cylance-i-kill-you/.

[10] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N Sullivan, E. Bursztein, M Bailey, J.A. Halderman, and V. Paxson. 2017. The Security Impact of HTTPS Interception. In *Proceedings the 24th Network and Distributed System Security Symposium (NDSS '17)*. Internet Society, San Diego, CA, USA, 1–14.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. 1999. Hypertext Transfer Protocol (HTTP/1.1): Authentication. https://datatracker.ietf.org/doc/html/rfc2616.

[12] R. Fielding and J. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Authentication. https://datatracker.ietf.org/doc/html/rfc7235.

[13] M. Handley, V. Paxson, and C. Kreibich. 2001. Network Intrusion Detection: Evasion, Traffic Normalization, an End-to-End Protocol Semantics. https://www.usenix.org/conference/10th-usenix-security-symposium/network-intrusion-detection-evasion-traffic-normalization. In *10th USENIX Security Symposium (USENIX Security 01)*. USENIX Association, Washington, D.C.

[14] A. Joshi, L. Lloyd, P. Westin, and S. Seethapathy. 2019. Using Lexical Features for URL Classification — A Machine Learning Approach. In *Proceedings of the Conference on Applied Machine Learning in Information Security (CAMLIS '19)*. Washington, DC, USA, 1–6.

[15] RACK911 Labs. 2020. Exploiting (Almost) Every Antivirus Software. https://rack911labs.ca/research/exploiting-almost-every-antivirus-software/.

[16] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. 2011. Learning to Detect Malicious URLs. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 3 (April 2011), 30:1–30:24.

[17] K. Manson. 2022. NSA Probing Reach of Software From Russia's Kaspersky in US Systems. https://www.bloomberg.com/news/articles/2022-05-10/nsa-probing-kaspersky-s-reach-in-us-after-russian-invasion.

[18] S. Morgenroth. 2008. Using Google bots as an attack vector.

[19] NetAcuity [n. d.]. NetAcuity. https://digitalelement.com/solutions/ip-location-targeting/netacuity.

[20] NIST 2004. CVE-204-0362. https://nvd.nist.gov/vuln/detail/CVE-2004-0362.

[21] NIST 2018. CVE-2018-11776. https://nvd.nist.gov/vuln/detail/CVE-2018-11776.

[22] NIST 2021. CVE-2021-33599. https://nvd.nist.gov/vuln/detail/CVE-2021-33599.

[23] NIST 2022. CVE-2022-20685. https://nvd.nist.gov/vuln/detail/CVE-2022-20685.

[24] OpenIntel 2022. Open Intel. http://www.caida.org/data/routing/routeviews-prefix2as.xml.

[25] P. Pearce, R. Ensafi, F. Li, N. Feamster, and V. Paxson. 2017. Augur: Internet-Wide Detection of Connectivity Disruptions. In *2017 IEEE Symposium on Security and Privacy (SP)*. 427–443. https://doi.org/10.1109/SP.2017.55

[26] T. H. Ptacek and T. N. Newsham. 1998. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. https://apps.dtic.mil/sti/citations/ADA391565.

[27] E. Rey. 2015. Playing With Fire: Attacking the FireEye MPS. https://static.ernw.de/whitepaper/ERNW_Newsletter_51_Playing_With_Fire_signed.pdf.

[28] C. Shannon and D. Moore. 2004. The Spread of the Witty Worm. *IEEE Security and Privacy* 2, 4 (July 2004), 46–50.

[29] E. Shimony. 2020. Anti-Virus Vulnerabilities: Who's Guarding the Watch Tower? https://www.cyberark.com/resources/threat-research-blog/anti-virus-vulnerabilities-who-s-guarding-the-watch-tower.

[30] J. W. Stokes, J. C. Platt, H. J. Wang, J. Faulhaber, J. Keller, M. Marinescu, A. Thomas, and M. Gheorghescu. 2012. Scalable Telemetry Classification for Automated Malware Detection. In *Computer Security – ESORICS 2012*. Springer Berlin Heidelberg, Berlin, Heidelberg, 788–805.

[31] M. Stone. 2021. CVE-2021-1647: Windows Defender mpengine remote code execution. https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2021/CVE-2021-1647.html.

[32] F. Xue. 2008. Attacking Antivirus. BlackHat.

[33] K. Zetter. 2016. Symantec's Woes Expose the Antivirus Industry's Security Gaps. https://www.wired.com/2016/06/symantecs-woes-expose-antivirus-software-security-gaps/.

[34] M. Ziv, L. Izhikevich, K. Ruth, K. Izhikevich, and Z. Durumeric. 2021. ASdb: A System for Classifying Owners of Autonomous Systems. In *Proceedings of the 21st ACM Internet Measurement Conference* (Virtual Event) *(IMC '21)*. Association for Computing Machinery, 703âĂŞ719.