Validating Web Content with Senser

Jordan Wilberding
Georgetown University

Andrew Yates
Georgetown University

Micah Sherr
Georgetown University

Wenchao Zhou
Georgetown University

ABSTRACT

This paper introduces Senser, a system for validating retrieved web content. Senser does not rely on a PKI and operates *even when SSL/TLS is not supported by the web server*. Senser operates as a network of proxies located at different vantage points on the Internet. Clients query a random subset of Senser proxies for compact descriptions of a desired web page, and apply consensus and matching algorithms to the returned results to locally render a "majority" web page. To ensure diverse selections of proxies (and consequently decrease an adversary's ability to manipulate a majority of the proxies' requests), Senser leverages Internet mapping systems that accurately predict AS-level paths between available proxies and the desired web page. We demonstrate using a deployment of Senser on Amazon EC2 that Senser detects and mitigates attempts by adversaries to manipulate web content — even when controlling large collections of autonomous systems — while maintaining reasonable performance overheads.

1. INTRODUCTION

SSL/TLS is the predominant protocol used to protect web content. When used correctly, it provides strong confidentiality and authenticity guarantees. Unfortunately, while SSL/TLS is critically important in securing many web transactions, it is often unavailable and too often is susceptible to implementation weaknesses. To illustrate, Sunshine et al. [24] demonstrate that users often do not heed browsers' certificate warning messages. Additionally, the 2011 DigiNotar incident [26] and academic studies of SSL/TLS deployment [12] bring into focus longheld concerns about the lack of safeguards in the web's public key infrastructure or "PKI" (cf. [8]). Finally, SSL/TLS is not universally available: we find that less than 30% of both Alexa's top 100,000 and top 1,000 websites correctly support SSL/TLS (see Appendix A).

SSL/TLS has been the subject of more than two decades of research and there are numerous proposals for increasing its adoption, mitigating weaknesses of the web's PKI, improving the usability of browsers' certificate warning messages and increasing public awareness of the importance of certificate verification. However, until comprehensive practical solutions are developed *and widely deployed*, it is useful to consider mechanisms for detecting and mitigating the malicious modification of web content in transit that does not require Internet-scale (re)deployments or reconfigurations.

In this paper, we focus on ensuring the *authenticity* of web content. We introduce a system, Senser, that validates retrieved web content *even when SSL/TLS is not supported by the web server*. Senser consists of a set of volunteer-operated *proxies* located at different vantage points on the Internet. Clients query a random subset of these proxies for a desired URL (e.g., a page, stylesheet, image, embedded object, etc.). The proxies fetch the URL and return concise descriptors to clients who may then render a "majority" version of the web resource. Unless an adversary can affect a majority of the chosen proxies' fetches, it cannot undetectably alter the web content. Hence, Senser is appropriate in settings where an adversary may censor/alter non-SSL content in transit.

The architecture of our system is similar to Perspectives [30], which also uses a network of proxies in favor of the web's existing PKI. Perspectives validates a site's certificate by comparing copies of the certificate retrieved from different proxies. The underlying assumption is that an adversary may be able to position himself between the user and the website, but is unlikely to be able to position himself between the website and all of the proxies.

This paper extends this model in two important ways. First, we show that a network of proxies can be used to validate the *content* of web pages. Here, a major challenge is that unlike SSL/TLS fingerprints which are usually consistent regardless of the requestor's location, the content of a web page may change depending upon who is accessing it. As we discuss below, websites often serve localized and/or personalized content, making it non-trivial to form a consensus of a retrieved webpage even in the absence of an adversary. However, by verifying content rather than certificate fingerprints, Senser does not rely on SSL/TLS or the web's PKI and is thus compatible with sites that do not support HTTPS.

Second, we observe that an adversary may be able to influence multiple proxies' views of a web resource if it is advantageously positioned. For example, a malicious or compromised autonomous system (AS) may be on the network path between several of the proxies and the requested web server and can therefore manipulate those proxies' fetches. To mitigate such attacks, we develop client-side proxy selection algorithms that maximize the AS diversity of the paths between the proxies and the web server. Our algorithms take as input compact topological maps of the Internet (sometimes called *Internet atlases* [18]) and the requested URL, and ensure network diversity both for DNS lookups and proxy fetches.

Threat Model. We focus on the problem of discovering (and potentially recovering from) *surreptitious* man-in-the-middle (MitM) attacks against non-HTTPS protected web content.

A particularly interesting class of attackers, and one that we emphasize in this work, is that of a censor who wishes to block or

ACSAC '13 Dec. 9-13, 2013, New Orleans, Louisiana USA

http://dx.doi.org/10.1145/2523649.2523669

modify intercepted web traffic. We conservatively model censors as AS-level adversaries. We distinguish between *blocking* (preventing the user from accessing the requested website), *whole-page alteration* (replacing the true webpage with one chosen by the adversary), and *partial alteration* (selectively modifying sections of a webpage).

We remark that we do not attempt to prevent a censor from *blocking* access to the Senser network; an anti-blocking system (e.g., Tor bridges [27] or Telex [31]) could be used to access the Senser network when faced with such actions. In such a situation Senser still provides an advantage over the anti-blocking system, because the censor has an incentive to participate in the anti-blocking system (e.g., run a Tor exit proxy) to censor content that is accessed through it. Senser mitigates a censor's ability to censor content by participating in the system by comparing content retrieved from multiple Senser proxies that are chosen using an AS-aware proxy selection algorithm. Here, our aim is to detect *whether* (and if so, *how*) particular web pages have been modified by a censor.

Challenges. The Senser architecture presents several technical challenges which we address in this paper:

- **Consensus construction:** Even in the absence of an adversary, websites may offer client-specific content. For example, many websites routinely serve content based on the client's perceived geographic location. In Senser, we relax the requirement that the consensus represent any particular response sent by the web server, and instead attempt to create a "majority version" of that page that contains its core content. In more detail, we represent the web page versions retrieved by the proxies as HTML trees, and perform an efficient tree-matching procedure to find a large common subtree.
- **Bandwidth costs:** The average webpage is estimated to consume at least 320KB of bandwidth [20]. Fetching multiple copies of webpages in their entirety from a set of proxies is thus likely too prohibitive for many clients. We reduce this cost by (i) fetching only concise *summaries* of webpage content and (ii) fetching website content only once using an established consensus.
- **Resistance to AS-level adversaries:** We envision AS-level adversaries who control large segments of the network and may attempt to manipulate web content. To limit the ability of AS-level adversaries, we introduce an offline AS-aware proxy selection algorithm that allows clients to intelligently select proxies such that the paths from the proxies to the destination website are suitably AS-disjoint.
- **Idempotency:** Our architecture requires that a webpage be retrieved multiple times. Senser is therefore ill-suited for web requests that are not idempotent.

The Senser architecture meets the first three challenges, but is incompatible with sites that require HTTP POSTs or use non-effect-free HTTP GETs. We argue, however, that our initial design and implementation are appropriate for a large class of websites: those that serve news stories or other content and do not require readers to authenticate to the site, which are likely targets for content modification due to censorship. Later, we discuss adaptation of Senser to support non-idempotent operations in Section 4.

We evaluate the performance, security, and utility of Senser under both simulation and a testbed deployment. Our results show that Senser is able to render the majority of pages in a usable way while incurring a modest latency overhead for the majority of websites. Our AS-aware proxy selection algorithm is able to reduce the system's failure rate (the proportion of pages that can be undetectably altered by the adversary) by up to 15% by increasing network diversity.

Our implementation of Senser is released as open-source software and is available at https://security.cs.georgetown.edu/senser/.

2. RELATED WORK

Multiple vantage points. Most similar to Senser are approaches for verifying digital certificates using a set of external verifiers. Wendlandt et al. [30] address the problem of "trust on first use" (TOFU) authentication by verifying that a server's public key remains the same when observed from servers at different locations. Their approach aims to improve usability weaknesses in certificate verification [24] by relying on trusted authenticator nodes rather than on a public key infrastructure. Similarly, Senser does not utilize PKIs and uses different perspectives to validate web pages. However, unlike the approach by Wendlandt et al., Senser verifies a webpage's *content* rather than its certificates. It is also applicable for websites that do not use SSL/TLS.

The Snakes on a Tor Exit Scanner (SoaT) [23] scans Tor [7] exit relays to detect misbehavior. SoaT operates by comparing hashes of web content retrieved from the Tor network with that retrieved from direct (non-anonymized) communication. Noting that false positives may be introduced due to personalized web content, SoaT also retrieves the web content from a second network location to identify the personalized content and reduce its false positive rate, at the cost of increased false negative rates. Senser provides a more comprehensive solution that supports fine-granularity content rendering when adversaries selectively modify portions of web pages.

The recently released Filter Bubble [32] extension for Chrome redirects Google search queries to a set of distributed nodes, informing the user of how search results differ based on geographic region. Like Senser, Filter Bubble detects personalized content, but is applicable only to Google. Similarly, Netalyzr [17] uses a set of distributed nodes to determine whether an ISP is actively blocking, limiting, or giving preferential treatment to certain services.

CensMon [22] aims to provide a similar outcome to Senser by routing queries through several geographically dispersed agents and analyzing the results of the DNS lookup and returned HTML to see how different nodes are receiving different information. Their HTML difference detection is based on MD5 and provides no reconstruction. Further, they do not take network paths into account.

Tree alignment and Merkle Hash Trees. Merkle Hash Trees (MHTs) have been applied to authenticate queries performed by untrusted third parties [5, 6, 10]. To do so, an owner computes a hash tree of the data to be queried, distributes the hash tree to all potential clients, and distributes the data to third parties. The hash tree can then be used by clients to verify query results returned by third parties. Bayardo and Sorensen [1] use a similar method to authenticate the correctness of HTTP 200 and 404 responses. Senser also uses MHTs to concisely describe web pages, but does not rely on a single tree to perform verification. Instead, we apply a tree alignment algorithm to construct a consensus tree and render a "majority" webpage.

While we use a simple and efficient tree alignment algorithm that employs breadth first search, others have explored tree alignment algorithms that yield results closer to the optimal outcome at the expense of performance. These approaches generalize the string alignment problem to trees and solve it using dynamic programming. As with the string alignment problem, the algorithms consider the cost of operations such as node insertion, deletion, and replacement. Carrillo and Lipman [3] present an optimal mul-

tiple tree alignment that runs in exponential time. The pairwise tree alignment problem can be solved in $O(|T_1| \times |T_2| \times h_1 \times h_2)$ time, where $|T_i|$ is the size of tree i and h_i is the height of tree i [25]. Wang et al. [28] improve upon this algorithm to solve the problem in $O(|T_1| \times |T_2| \times \min(h_1, l_1) \times \min(h_2, l_2))$ time, where l_i is the number of leaves in tree i. Followup work [4] applies the center star approximation algorithm [11] for multiple string alignment in order to approximately align multiple HTML trees.

Censorship resistance. Information slicing [16] divides scrambled messages into pieces, and sends the pieces along disjoint paths in a P2P overlay network to reduce the chance of the complete message (i.e., all pieces) being intercepted by an attacker. Senser similarly leverages the use of disjoint network paths to reduce the impact of a malicious AS that censors web content.

There are also a number of proposed anti-blocking systems that use steganography and/or covert channels to bypass blocking efforts. Tor *bridges* [27] are Tor relays that are not publicly listed by Tor directory servers, making them more difficult for an adversary to discover and block. Infranet [9] encodes requests using a sequence of innocuous-looking webpage requests and hides responses within JPEG images. Collage [2] also uses steganography, but uses photo sharing and other user-generated content sites as "drop boxes" for conveying hidden messages. Similarly, in CensorSpoofer [29], clients embed requested URLs using steganographic techniques in email messages and receive responses via IP packets with spoofed source addresses. More recently, a number of *decoy routing* solutions have been introduced [13, 15, 31]. Decoy routers intercept SSL/TLS streams addressed to an unfiltered destination. The routers — which must not be subject to censorship and must be positioned between the sender and the addressed destination — decipher the hidden destination that is embedded in the SSL/TLS exchange (typically, in the handshake) and redirect the SSL communication to the hidden destination. Recent work has shown that adversaries can adjust their routes to enumerate decoy routers and defeat the anti-censorship measure [21].

All of the above anonymity systems implicitly assume that the privacy network's egress points can correctly deliver requests and return the correct responses. Senser can help mitigate potential "last mile" attacks in which the adversary modifies content as it leaves or re-enters the anonymity network. By itself, Senser does not (and is not intended) to bypass censorship efforts. Rather, our focus is to identify *when* an adversary (such as a censor) modifies a web page element and, when such attacks take place, *how* the page has been modified.

3. SYSTEM DESIGN

This section describes in greater detail the Senser system. We begin by presenting an overview of the Senser architecture (Section 3.1). We then discuss the major functionalities of Senser, including (i) the construction of a concise description of a requested URL which we call its *summary* (Section 3.2), (ii) the formation of a *consensus* that represents the majority of the proxies' interpretations of the requested URL (Section 3.3), and (iii) a proxy selection algorithm that reduces the influence of one or more malicious autonomous systems in the network (Section 3.4). We discuss the handling of several practical deployment issues in Section 4.

3.1 System Overview

Senser defends against potential manipulation of retrieved web contents by strategically relaying a user's request for a target webpage to *multiple* proxies, with the hope that a majority of these proxies will individually and correctly retrieve the webpage.

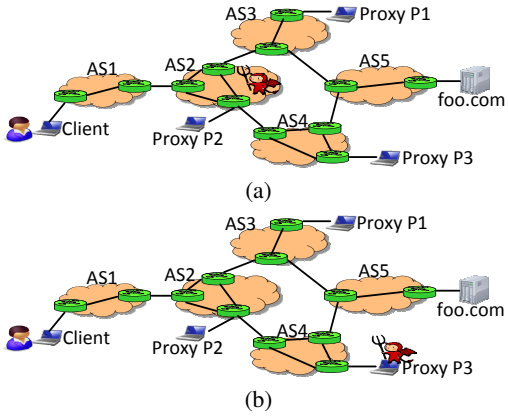


Figure 1: Senser validates retrieved web contents in various attack scenarios. *(a)* An adversary in AS2 modifies (e.g., censors) the web content of foo.com. Proxy P1 and Proxy P3 take routes AS3 → AS5 and AS4 → AS5 respectively, both of which avoid the modification by AS2. The client leverages these proxies to circumvent the attack. *(b)* Proxy P3 is compromised and censors foo.com. The client retrieves copies of the requested URL from P1, P2, and P3, and constructs a "majority opinion" response.

Senser uses the same mechanism to validate HTML content and embedded objects (pictures, videos, etc.). We assume that one or more adversaries may perform blocking, whole-page alteration, and/or partial alteration attacks against some fraction of the proxies. In addition, the web server may intentionally serve inconsistent contents to different proxies. If a majority of the proxies are able to retrieve an accurate version of the targeted URL, then the user can construct a "consensus" version from the returned results that more accurately reflects the URL's actual contents.

System model. Figure 1 presents an overview of the Senser system. Senser consists of a client running on the user's machine, and a pool of proxies distributed across the Internet. The client communicates with the proxies using a secure communication channel (e.g., through SSL/TLS). As with the Perspectives system [30], we assume that the list of proxies and their public keys is preloaded with the Senser client software (or securely retrieved using a trusted directory) and that no PKI is required to authenticate the proxies. Finally, we assume that a subset of the proxies may be disconnected or compromised by an adversary.

Consider the example scenarios shown in Figure 1 in which the adversary modifies the content of a requested web page. In Figure 1a, the client uses three proxies: P1, P2, and P3. Here, AS2 modifies responses from foo.com, causing the client to receive a modified webpage from P2 (since the path from P2 to foo.com traverses AS2). On the other hand, proxies P1 and P3 will be able to retrieve the unmodified webpage, as their routes to the web server avoid AS2. Senser leverages the responses received at these proxies to circumvent the attack.

Similarly, Figure 1b shows an attack scenario due a compromised proxy. The use of multiple proxies enables the client to form a consensus and construct an accurate view of the requested URL.

We explain this process in more detail below.

System execution. Upon receiving a URL request from the user, Senser takes the following steps to generate a response:

- The client forwards the URL request to a selected subset of proxies via secure communication. Note that proxies located in different ASes may still share part of their routes to the web server. A malicious AS may therefore affect the responses received by different proxies. To mitigate the po-

```
<html>
<title>IP Lookup</title>
<body>Your IP is <b>10.0.0.1</b></body>
</html>
```

Figure 2: Example HTML document.

tential damage due to a malicious AS, we deploy a *proxy selection* module to maximize the diversity of the AS-level paths from the chosen proxies to the website.

- Each selected proxy sends an HTTP request to the web server, and passes the received HTTP response to the *summary construction* module. The module crafts a concise summary of the requested object (e.g., HTML page) and returns the summary to the client, again via secure communication. To produce compact summaries, we construct Merkle Hash Trees (MHTs) over retrieved web content, because HTML has a hierarchical structure that can be treated as a tree.
- The client compares the summaries collected from the proxies to identify inconsistencies, and uses a *consensus construction* module to resolve the inconsistencies. The *consensus version* contains the elements agreed upon by a majority of the proxies (for example, the elements present in a majority of the summaries of HTML documents). Rather than doing a fuzzy comparison, we require elements to match exactly since even the change of a single word can significantly alter the meaning of a webpage.
- Finally, the Senser client retrieves the consensus version's content from one or more proxies. The retrieved contents are compared against the reported summaries for consistency (i.e., to ensure that the retrieved content hashes to the correct value in the consensus version MHT), and the result is returned to the browser for rendering.

3.2 Summary Construction

Upon receiving a client's request for a particular URL, the proxy forwards that request to the requested web server. In the case of HTML documents, the retrieved webpage is normalized using an HTML parser (e.g., Jsoup[1]). The proxy then creates an MHT of the retrieved content.

We first describe the case in which the client requests an HTML document: The MHT is constructed using the (normalized) HTML's structure such that each node in the MHT corresponds to either an HTML tag (e.g., <body>) or the text inside a tag (e.g., the content in <title>content</title>). Basing the MHT's structure on the structure of the HTML document may create imbalanced trees, but the construction allows us to quickly identify the structural and contextual differences across multiple MHTs (see Section 3.3).

Importantly, the MHTs we use differ from traditional MHTs in which each internal node contains only a hash over its children and pointers to those children. In Senser, an internal node consists of (i) a hash over the corresponding HTML tag name and, if applicable, its attributes (the *tag-hash*), (ii) a hash of the node and all of its children (the *full-hash*), and (iii) pointers to its children. A leaf node sets its tag-hash as NULL and its full-hash based on a hash over its corresponding content (which could be a string, a URL, a picture, or some other object type).

To illustrate, consider the example HTML document in Figure 2 which shows the IP address of the visiting user. Figure 3 shows the corresponding MHT-based summary of the example HTML document. Note that the root of the MHT contains the tag-hash ("TH") generated from the tag html and the full-hash ("FH") generated from the tag-hash and the full-hash of its two children (the title element and the body element). The MHT-based summary is recursively defined in a top-down fashion until leaf nodes are reached.

[1] http://jsoup.org/

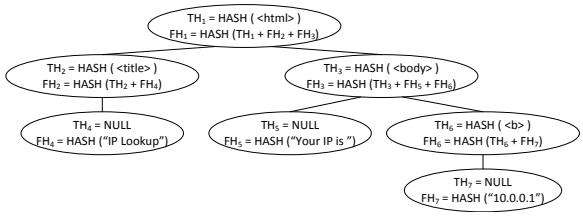


Figure 3: The corresponding MHT-based summary of the example HTML document depicted in Figure 2.

Algorithm 1 Consensus Construction

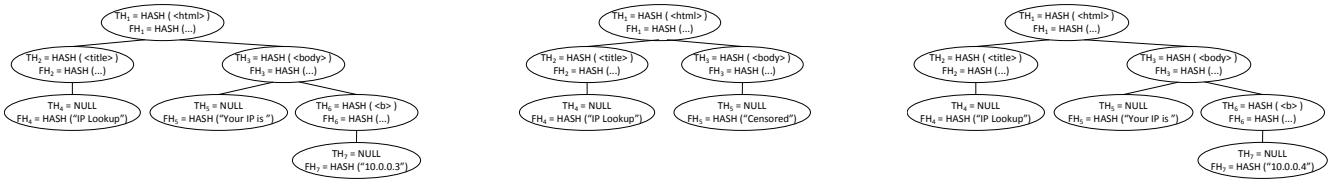
```
1: proc consensus(nNodes, MHT_roots[])
2: root ← an empty leaf node
3: nodes_queue ← an empty queue
4: nodes_queue.push_back(root)
5: candidates_queue ← an empty queue
6: candidates_queue.push_back(MHT_roots)
7: while nodes_queue ≠ empty do
8:     currentNode ← nodes_queue.pop_front()
9:     candidates ← candidates_queue.pop_front()
10:    for all node in candidates do
11:        count_fullhash[node->fullhash]++
12:        count_taghash[node->taghash]++
13:    end for
14:    if maxF ← MAX(count_fullhash) > nNodes/2 then
15:        find node, s.t. count_fullhash[node.fullhash]=maxF
16:        currentNode ← clone(node)
17:    else
18:        if maxT ← MAX(count_taghash) > nNodes/2 then
19:            find taghash, s.t. count_taghash[taghash]=maxT
20:            currentNode.taghash ← taghash
21:            numChildren ← MAX(node.numChildren), for all node where
               node.taghash=taghash
22:            for 0 < i < numChildren do
23:                newNode ← an empty leaf node
24:                currentNode.addChild(newNode)
25:                nodes_queue.push_back(newNode)
26:                candidates.push_back(node.child[i])
27:                candidates_queue.push_back(candidates)
28:            end for
29:        else
30:            currentNode ← a node marked NON-CONSENSUS
31:        end if
32:    end if
33: end while
34: return root
```

We remark that the hash tree sent to the client does not contain the HTML corresponding to the hashes, which significantly reduces Senser's communication overhead. The actual content is retrieved by the client once a consensus has been reached.

The above summary construction technique applies to both well-structured documents (e.g., HTML and XML objects) and binary objects. The latter case applies, for example, when the user requests an image or video file. Here, the summary consists of a single (leaf) node with a tag-hash of NULL and a full-hash that is the hash of the file's contents.

3.3 Consensus Construction

To construct a consensus among the MHT-based summaries returned by the proxies, the client simultaneously performs a breadth first search (BFS) on each of the MHTs. Algorithm 1 presents the pseudo-code of the consensus construction procedure. The algorithm takes as input the MHTs from the proxies and outputs the consensus tree. While there exist other algorithms that allow con-



(a) Summary at P1 (b) Summary at P2 (c) Summary at P3

Figure 4: An example scenario of consensus construction. The three returned summaries are shown in Figure (a) - (c), where Figure (a) and (c) correspond to cases where the original document (Figure 2) is received without manipulation; Figure (b) corresponds to a censorship case. The full-hashes of the internal tree nodes are omitted for brevity.

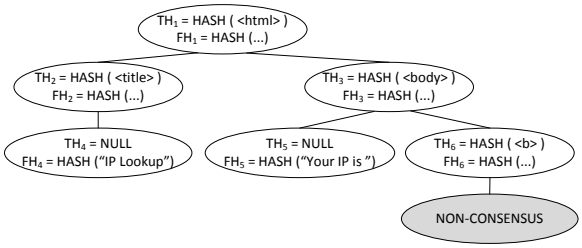


Figure 5: The consensus result for the example scenario shown in Figure 4. The consensus version accurately reflects the original document, with the exception of the profile-based content.

tent comparison based on metrics such as edit distance (see Section 2), they usually impose a significant overhead that is too expensive for online processing. We adopt a simple and efficient tree alignment that employs breadth first search.

The algorithm starts from the roots of the MHTs, and traverses recursively through the MHTs in a top-down fashion. For each tree node, we compute the majority consensus for the full-hashes and tag-hashes (lines 10-13): if a majority of the proxies agree on the same full-hash, which indicates that a majority consensus has been reached for the complete subtree rooted by that tree node, then the whole subtree is copied into the final consensus tree (lines 14-16); otherwise, if the corresponding tree nodes in a majority of the summaries have the same tag-hash, we heuristically assume that these tree nodes correspond to the same fragment in the HTML but disagree on the contents, in which case, that tree node is copied into the final consensus tree (lines 18-20), and the BFS algorithm will construct the consensus version of the corresponding subtree when the children nodes are visited (lines 21-28). If neither a tree node's full-hash nor its tag-hash are present in a majority of the MHTs, no consensus can be drawn, and the node is marked as NON-CONSENSUS (lines 29-31).

To illustrate, Figure 4 presents an example scenario of consensus construction in which the client wants to construct a consensus version based on the summaries returned by proxies P1, P2, and P3. Figure 4(a)-(c) corresponds to the summaries returned by the three proxies, where P1 and P3 managed to retrieve the original webpage (shown in Figure 2) with slight variations according to their current IP addresses, and P2 received a partially altered version.

Figure 5 shows the result of executing the consensus construction algorithm (Algorithm 1). The three roots share the same tag-hash, yet their full-hashes differ from each other. Therefore, the root of the consensus MHT is assigned a tag-hash of html, and the BFS construction continues to the children nodes. For the left branch, the three summaries have the identical full-hash, indicating a consensus on the complete title element. In contrast, for the right branch, a majority consensus (2 out of 3) is reached for the text "Your IP is", but no consensus can be drawn for the IP address part, as this "profile-based" content varies amongst the proxies.

3.4 Proxy Selection

The quality of the final consensus relies on the summaries returned from the selected proxies. To achieve reasonable performance, a Senser client needs to select a relatively small number of proxies from the pool of all available proxies. Rather than selecting the proxies at random, we discuss in this section methods for proxy selection that mitigate the potential damage caused by a malicious AS. Conceptually, this is achieved by maximizing the diversity of the AS-level paths from the proxies to the destination web server.

DNS consensus. To determine the AS-level paths, we need to know the IP address of the destination web server. This task is more complex than it appears: we cannot rely on the DNS resolution results retrieved locally, since doing so would create a single point-of-failure (i.e., an adversary can poison the DNS server or hijack the DNS request/response to point the client to a fake destination).

To resolve this issue, our approach takes into account the DNS resolution results at the proxies collectively. Based on the assumption that a majority of the proxies are benign and are not subject to a man-in-the-middle attack, the client forwards DNS requests through secure communication channels to q randomly selected proxies to perform the DNS resolution on its behalf. If $\lfloor q/2 \rfloor + 1$ proxies return the same IP address, then that IP is accepted.

One source of complication comes from the use of a reverse proxy, at companies such as Google, for performance optimization and load balancing purposes. These companies deploy a large number of servers with different IP addresses to handle the requests for popular web content. However, given the nature of reverse proxies, these IP addresses often reside within the same AS, and hence users would be likely to observe the same AS-level path when accessing the web content from these IP addresses (though the content may be hosted on different servers). If $\lfloor q/2 \rfloor + 1$ proxies return IP addresses located in the same AS (as locally determined by the client using a compact database such as GeoMind), then the client accepts an IP address chosen randomly amongst the addresses in that AS.

The use of content distribution networks (CDNs) such as Akamai further complicates the problem. Here, a given URL may be mapped to multiple IP addresses that correspond to cache servers distributed over the world.

If the proxy selection process cannot reach a consensus on the IP address of the destination website, it falls back to selecting the proxies uniformly at random. Otherwise, if an IP consensus can be reached, it then adopts the AS-disjoint proxy selection algorithm, described next.

AS-disjoint routes. After it has determined the IP address for the destination, the client next needs to determine the paths which the available proxies take to reach the destination. This is done by

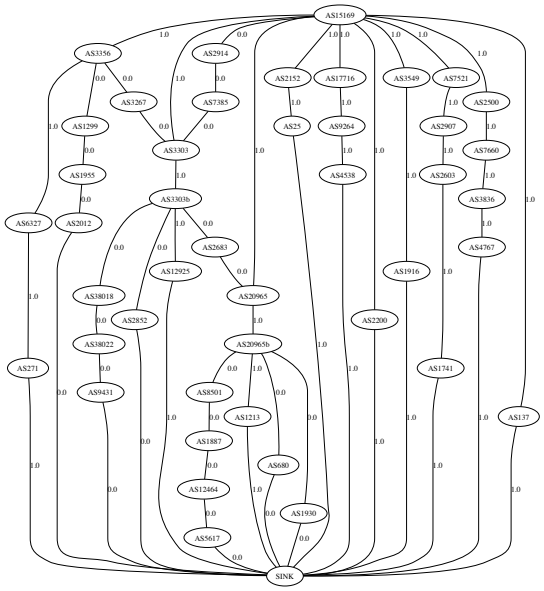


Figure 6: The AS-level disjoint paths for the example scenario. The selected endpoint ASes include AS271, AS12925, AS1213, AS25, AS4538, AS2200, AS1916, AS1741, AS4767, and AS137.

utilizing an Internet mapping service such as iPlane Nano [18] that infers the AS-level path between two arbitrary IP addresses without performing active Internet measurements[2].

We select the "best" proxies according to the constructed AS-level graph. Generally speaking, we consider two properties: *disjointedness* and *randomness*. We want disjointedness in order to make it more difficult for a single adversary to affect the quorum by controlling a single AS. If the paths are disjoint, they have to control an AS on at least $\lceil k/2 \rceil$ paths to create their quorum. However, randomness is also important: if an adversary can deterministically calculate which proxies will be selected, it can carefully position rogue proxies in a manner that significantly increases the probability that its proxies will be selected. Section 4.4 discusses such a potential attack and its countermeasures in greater detail.

To find vertex-disjoint paths, we adopt the max-flow algorithm: we split each vertex with multiple incoming and outgoing edges in two — an in-node and an out-node — where all incoming edges are connected to the in-node, all outgoing edges are connected to the out-node, and the in-node and out-node are connected by exactly one edge. We then set all edges to have a weight of 1.0, and run the max-flow algorithm on the constructed graph. The max-flow result corresponds to the maximal number of vertex-disjoint paths, based on which we can select the proxies that maximize the path diversity. If the number of disjoint path is smaller than the desired number of proxies, we iteratively relax the weights of the bottleneck edges in the graph in order to find the "more disjoint" paths.

The max-flow algorithm produces disjoint AS-level paths. To complete the proxy selection and add randomness to the selections, a random proxy is picked from each selected endpoint AS.

An example scenario. To illustrate the proxy selection process, we consider a webpage request to http://google.com. Google has deployed reverse proxies, making it difficult to reach a consensus on the DNS resolution. However, all of the resolved IP addresses reside within the same subnet and belong to the same AS. The client randomly selects an IP from these resolved IP addresses, and then determines the AS-level paths from each individual proxy to that IP using the Internet mapping service. For example, for the proxy with IP address 169.229.50.15, our iPlane-based database returns the AS path AS25 → AS2152 → AS15169.

Figure 6 depicts the constructed graph based on the paths from the available proxies. (Note that both AS3303 and AS20965 are each split into two vertices that are connected by a single edge. This reflects the above constraint that each AS should be included in at most one AS-disjoint path.) The numbers associated with the edges show the execution result of the max-flow algorithm. The graph has a max-flow of 10, indicating that it can accommodate at most 10 AS-disjoint paths. The client selects proxies that are located in endpoint ASes that connect to the sink with edges labeled 1. If more proxies are needed, the edges can be relaxed to accommodate more paths (which are no longer necessarily disjoint). For instance, if the edge between AS3303 and AS3303b is relaxed, a proxy located at AS9431 can then be added to the selection.

4. PRACTICAL ISSUES

We next discuss potential limitations of the Senser architecture (Section 4.1) and optimizations (Sections 4.2 through 4.4).

4.1 HTTP POST Requests and Cookies

Senser does not currently support POSTs or GETs that cause side-effects (the latter of which violate HTTP GET semantics, but are unfortunately not uncommon on the web). The incompatibilities arise from Senser's dependence on *multiple* vantage points to verify web content. A single non-idempotent request's are amplified because each proxy causes a state change on the server.

For many websites, these multiple and near-identical requests may be caught by the service and considered only once. (Such techniques are often applied, for example, to prevent credit card transactions from being issued multiple times.) Since the web currently has no standard protocol in place for handling concurrent and identical requests, we conservatively disable support for HTTP POSTs in Senser, and (perhaps optimistically) assume that GET requests are idempotent and side-effect-free.

We note that while our incompatibility with HTTP POSTs and GETs that are not effect-free makes Senser inappropriate for many websites, the use of non-idempotent operations over unprotected HTTP (i.e., without SSL) is ill-advised in many cases (in particular, for site logins). Preventing such capabilities may be disruptive to the user, but it may also protect the user.

As a future enhancement, Senser may be amended to offer some support for HTTP POSTs by relaying POSTs requests through a single, randomly chosen proxy. Alternatively, clients may apply a secure reputation system (cf. [14]) to rate proxies, and relay POSTs through a proxy with a sufficiently high reputation score.

We similarly have limited support for cookies. Since many sites rely on cookies to manage state and present a cohesive user experience, Senser adopts the cookies returned by a randomly selected proxy, and forwards those cookies in subsequent HTTP requests.

4.2 Incremental Consensus Construction

Senser forwards HTTP requests to multiple proxies, providing the opportunity to construct the consensus incrementally as responses arrive. Rather than waiting for all the summaries to be returned, the client checks whether a top-level MHT consensus (out of *all* the selected proxies; not out of only the proxies from which a response has been received) has already been reached whenever a new summary is received. If a consensus can be reached, the client can conclude on that consensus without waiting for the remaining

[2]Due to the current unavailability of iPlane Nano, we use the iPlane [19] service to query AS-level paths between all available proxies and the Alexa top sites, and record the results in a concise Internet "map" that is loaded with the Senser client software.

summaries. If there is no consensus at the top level of the hash tree, Senser waits for all summaries before constructing a "majority version" of the webpage. (This is because a "majority version" of the webpage is created given any input, so it is difficult to determine when the consensus construction algorithm can be run without ill effect before all the summaries have been received.)

DNS lookup requests are handled in a similar fashion: When possible, the client chooses an IP to satisfy the lookup before replies from all proxies have been received.

The repeated invocation of consensus construction consumes more computation resources. However, the additional computation overhead is overshadowed by its benefits: given the long-tail distribution of different proxies' latencies, it effectively improves the time-to-first-byte latency by eliminating the effects of slow, corrupt, and/or failed proxies.

4.3 Caching

Senser also benefits from the adoption of multiple caches. When the same URL is requested, the use of caching saves considerable overhead at several stages of Senser's operation. In particular, the consensus IP of the web server that hosts a given URL as well as the routes from each proxy to the resolved IP address are unlikely to change frequently, and therefore benefit from client-side caching. In addition, it is typical in web browsing that multiple visits to the same website occur within a short period of time (e.g., clicking links on a portal webpage such as yahoo.com for reading about related topics). In such scenarios, the client can additionally cache proxy selection results to further reduce latency overheads. Senser does not cache a webpage or the MHT constructed from that page. Instead, the webpage is fetched each time it is requested by a client.

We evaluate the performance benefits of incremental consensus construction and caching in Section 5.3.

4.4 Resistance to Knowledgeable Attackers

An adversary may attempt to game the proxy selection algorithm presented in Section 3.4 by strategically placing malicious proxies to maximize their chances of being selected for a targeted website. More specifically, the attacker exploits instances in which there are few AS-disjoint paths from the honest proxies to a targeted website. If an attacker is able to add corrupt proxies to the network that are AS-disjoint from the existing proxies w.r.t. the targeted site, then the adversary increases the likelihood that the corrupt proxies will be disproportionately chosen for that site. That is, the proxy selection algorithm will choose these proxies with high probability to improve the route diversity, and as a result, the adversary gains the control of a significant portion of the selected proxies.

To counter such an attack, we introduce additional randomness into the proxy selection algorithm. Instead of definitively selecting the proxies that maximize AS-level path diversity, we conduct a *weighted* randomized selection from the proxies. This approach strikes a balance between randomness (to counter the exploit presented above) and path diversity (to limit the potential damage caused by a malicious AS). Here, we introduce a parameter, $\alpha \in [0, 1]$, that determines the level of randomness used to select proxies. If k is the number of desired proxies, the modified algorithm selects $\lfloor \alpha k \rfloor$ proxies using the AS-disjoint algorithm from Section 3.4 and $\lceil k(1-\alpha) \rceil$ proxies uniformly at random from the remaining (unchosen) proxies. We explore the performance-security tradeoffs of selecting α in Section 5.4.

5. EVALUATION

This section evaluates Senser's effectiveness and efficiency. In Section 5.1, we describe our implementation of Senser. We examine the accuracy of constructed consensus pages in Section 5.2 and measure the performance of our implementation in Section 5.3. Using topologies constructed from real-world trace data, we conduct simulation experiments to assess Senser's ability to mitigate censorship in Section 5.4.

5.1 Experimental Setup

Senser consists of two core components: (i) a client-side application that intercepts browser requests and constructs a consensus, and (ii) a proxy that retrieves requested URLs and returns the resulting MHT. We use the JSoup HTML parser to normalize webpages and the GNU Crypto implementation of the Tiger hash function. Tiger was chosen for its fast performance on 64bit architectures – in our benchmarks, it was approximately 50% faster than SHA-2.

Clients run an instance of the Firefox browser that has been configured to forward requests through the local Senser client (which itself acts as an HTTP proxy). To obtain meaningful performance results, we disable both memory and disk caching in Firefox. The Senser client communicates with Senser proxies using a custom HTTP API secured with SSL.

In our experiments, the Senser client and proxies use a 24 thread pool to respond to requests. Firefox was configured to allow up to 24 concurrent HTTP requests.

We deploy 12 Senser proxies on Amazon EC2 in four separate regions (US East, Brazil, Ireland, and Singapore), with three proxies per region. In our experiments, we set $k = 11$. The client utilizes a Verizon FIOS broadband connection in Washington, DC.

5.2 Consensus Construction Accuracy

To minimize latency, we trade off efficiency for optimality: while our consensus construction algorithm is efficient, it does not achieve the optimal solution (which requires exponential time [3]). To determine how often our consensus algorithm is able to construct a usable webpage, we evaluate our algorithm against websites chosen from the Alexa "top 1,000" list, which we will refer to as the *Alexa websites*. We visited the top pages of each of these 1,000 websites and additionally randomly clicked five selected links on each site.

We manually viewed a screenshot of each page that was successfully retrieved and determined whether (i) the page was rendered correctly, (ii) the page was rendered with errors but was still usable, or (iii) the page was not usable. Cases where the page had errors but was still usable were sometimes caused by CSS (Cascading Style Sheets) that varied across different regions. For example, a page that is usable with errors might display the correct content (with the exception of ads) but have excessive whitespace.

In order to better understand what types of websites our consensus construction algorithm supports, we split up the 1,000 websites into their Alexa categories and computed the success rate for each category. The results are shown in Table 1. A majority of websites either render correctly or are usable with errors. The highest success rate was obtained with the World category, which contains many region-specific websites that are unlikely to serve localized content (e.g., chinanews.com, a large state-owned news agency).

5.3 Performance

We measure Senser's performance by recording the time-to-last-byte (TTLB) for each of the 1,000 Alexa websites. For each website, we retrieve the top page, follow five links on the site, and record the average TTLB of the requests. TTLB is obtained using the Selenium WebDriver[3] Firefox add-on and includes the amount of time taken to fully load and render each page, including all resources directly linked to on a page as well as any resources indi-

[3] http://seleniumhq.org/

Table 1: Consensus construction accuracy, by website category.

Category	% of tested sites	Failures	Usable with errors	Correct
All	100.0%	39.3%	12.0%	48.8%
Uncategorized	33.7%	38.3%	11.0%	52.2%
World	32.4%	27.7%	12.2%	59.5%
Computers	10.3%	56.0%	11.3%	32.3%
Regional	6.2%	48.2%	11.7%	39.1%
Business	2.4%	46.9%	7.1%	45.1%
Shopping	2.3%	57.7%	10.8%	29.7%
Sports	2.2%	30.8%	23.1%	46.2%
Reference	1.5%	62.2%	8.1%	29.7%
Home	1.4%	52.9%	20.6%	23.5%
News	1.3%	43.5%	6.5%	50.0%
Society	1.3%	45.0%	18.3%	41.7%
Games	1.0%	58.7%	23.9%	13.0%
Recreation	1.0%	71.4%	6.1%	22.4%
Adult	0.8%	47.2%	36.1%	16.7%
Arts	0.8%	39.5%	13.2%	47.4%
Kids and Teens	0.8%	36.1%	13.9%	50.0%
Science	0.6%	46.7%	10.0%	43.3%
Health	0.1%	100.0%	0.0%	0.0%

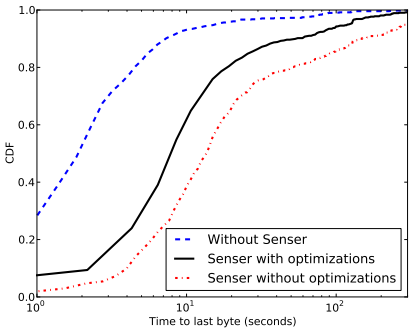


Figure 7: Cumulative distribution of time-to-last-byte, with and without performance optimizations.

rectly loaded with Javascript. We additionally measure the effects of our optimizations (described in Sections 4.2 and 4.3).

Figure 7 shows the cumulative distribution of the resulting average TTLBs when loading sites with Senser, without Senser, and with Senser when optimizations were disabled. Without optimizations the majority of websites took under 14 seconds to load, incurring a 7-fold increase in the median TTLB when compared against directly accessing a page. As shown in the Figure, Senser's overhead can be reduced considerably by enabling the optimizations. With optimizations enabled, the majority of websites can be loaded in under 8 seconds. Websites hosted in distant geographic regions took the longest to load in all cases, as would be expected.

Microbenchmarks. To better understand Senser's performance costs, we measure the average run time of each of the system's components on the top 50 Alexa websites.

Figure 8 shows the processes that contribute to the time it takes to access a website with Senser due to the DNS and MHT consensus procedures. The Figure's key identifies each process, with *Other* collectively representing the time for serializing and deserializing the MHT from the proxies to the client, and the time for choosing the random nodes. The average total time taken is less than the sum of the time taken by the *DNS Consensus* and *MHT Consensus* steps because DNS lookups can often be handled by the cache.

Network communication makes up the majority of the time it takes to load a page: *DNS Consensus* is the time taken to receive DNS lookup results from the proxies, *Receive MHT* is the time taken to receive MHTs from the proxies, and *Fetch Page* is the time

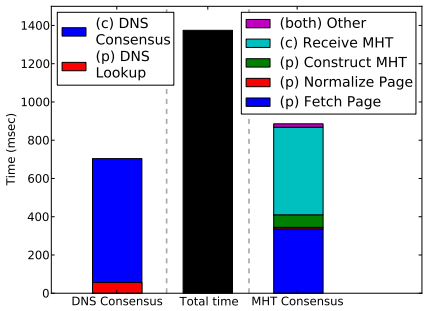


Figure 8: Average time taken by various operations. The notation *(c)* indicates an operation happens on the client and *(p)* indicates an operation happens on the proxies.

taken by the proxies to receive the destination webpage. This is not unexpected given the diverse locations of our proxies.

5.4 Simulation Study

To better understand the effect of malicious ASes on Senser's ability to reach a consensus and resist censorship attempts, we evaluate Senser under simulation using realistic network topologies. To perform the simulation we obtained AS-level graphs of the routes between a set of 18 geographically diverse proxies and the Alexa websites. Each graph was obtained by querying iPlane [19] for the AS-level paths between each proxy and the Alexa websites. The proxies were PlanetLab nodes in the following regions: Brazil, Canada, China, Czech Republic, Finland, France, Germany, Hungary, Ireland, Italy, Japan, Poland, Portugal, Russia, Slovenia, Thailand, United States (east coast), and United States (west coast).

For each of the Alexa websites, we randomly designate $n\%$ of the ASes appearing in the graph as malicious ASes and choose k proxies to use to reach the website. We say that our proxy selection algorithm *failed* if at least half of the routes pass through a malicious AS. (The inverse does not necessarily mean that the attempt to visit the website would have succeeded, since malicious proxies could have prevented a consensus from being reached.) We repeat this process 1,000 times for each website and take the average of their outcomes.

We divide ASes into three groups: top-tier ASes, transit ASes, and endpoint ASes. Top-tier ASes are those that contain at least 5% of all ASes in their customer cone according to CAIDA[4]. Any AS that contains either a proxy or an Alexa website is designated as an endpoint AS. The remaining ASes are designated transit ASes. We vary the types of ASes that are malicious to see how an adversary's capabilities affect Senser's failure rate.

Figure 9 shows Senser's random proxy selection algorithm's failure rate as n, the number of malicious proxies, increases. We vary n to determine how the algorithm responds to different situations. The failure rate is the worst when only top-tier ASes become malicious, as one would expect. It takes over five malicious ASes for the failure rate to reach 50% with the other AS sets. The transit only line never reaches 1.0 because some paths are composed entirely of endpoint and top-tier ASes. Since endpoint ASes are those that appear as an endpoint in any of the graphs, it is possible for an "endpoint AS" to be a transit AS in some graphs.

We compare the random proxy selection algorithm shown in Figure 9 to our AS-disjoint proxy selection algorithm. The latter AS-aware technique performs better in most situations, but does not

[4] `http://as-rank.caida.org/`

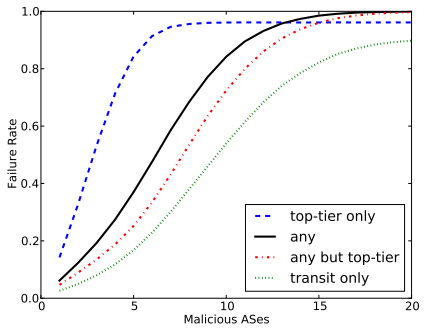


Figure 9: Random path selection simulation (varying n).

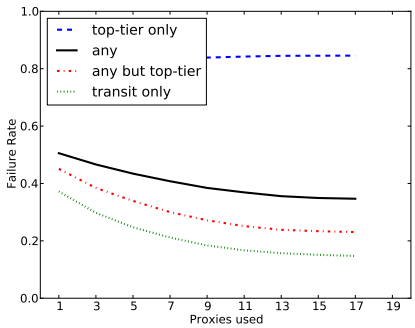


Figure 10: Random path selection simulation (varying k).

Table 2: Failure rate comparison of randomization vs. AS-disjoint proxy selection (varying n).

ASes	top-tier only	any	any but top-tier	transit only
1	0.01 (5%)	0.00 (4%)	0.00 (2%)	0.00 (5%)
5	0.02 (3%)	0.04 (10%)	0.03 (11%)	0.03 (15%)
10	0.00 (0%)	0.05 (6%)	0.07 (10%)	0.07 (12%)
15	0.00 (0%)	0.01 (1%)	0.02 (2%)	0.04 (5%)
20	0.00 (0%)	0.00 (0%)	0.00 (0%)	0.02 (2%)

Table 3: Comparison of randomization vs. AS-disjoint proxy selection (varying k).

Proxies	top-tier only	any	any but top-tier	transit only
1	0.01 (1%)	0.03 (6%)	0.04 (8%)	0.04 (11%)
5	0.02 (2%)	0.05 (11%)	0.05 (14%)	0.05 (19%)
9	0.02 (2%)	0.04 (10%)	0.03 (13%)	0.03 (17%)
13	0.02 (3%)	0.03 (7%)	0.02 (8%)	0.02 (14%)
17	-0.01 (-2%)	-0.01 (-4%)	-0.01 (-4%)	-0.02 (-12%)

when only top-tier ASes can be malicious, because the max-flow based proxy selection algorithm is inclined to select top-tier ASes that have high degrees. We vary the types of ASes that are malicious to observe how effective Senser is against different types of adversaries. Table 2 highlights the difference between the random algorithm's and the AS-disjoint algorithm's failure rates and the AS-disjoint algorithm's percent improvement over the random algorithm. The difference is highest with the "transit only" AS types with the AS-disjoint algorithm performing up to 15% better. This is due to the fact that the AS-disjoint algorithm is able to avoid having a transit AS (unlike a top tier AS that is more difficult to avoid) appear in multiple paths, which would otherwise be shared by paths from multiple proxies if chosen randomly.

To see how the number of nodes affects the failure rate, we set $n = 11$ (which roughly corresponds to a 40% failure rate with the AS-disjoint algorithm when only transit ASes can be malicious) and vary k, the number of proxies used. Figure 10 shows the effect on the randomized algorithm; the effect on the AS-disjoint algorithm is similar and is omitted for brevity. As the number of proxies used increases, the failure rate for most AS types decreases until about 13 or 15 proxies are used. The failure rate increases slighty when only top-tier ASes are malicious, because the additional malicious top-tier ASes have a high probability of routing the additional proxies' traffic. Table 3 shows the difference between the random algorithm's and the AS-disjoint algorithm's failure rate and the AS-disjoint algorithm's percent improvement over the random algorithm. The AS-disjoint algorithm performs worse with only top-tier ASes when $k = 17$, but performs the same or better in all other cases, up to a 19% improvement for only transit ASes.

Weighted randomized selection algorithm. Section 4.4 explains an attack in which a knowledgeable adversary can use the AS-disjoint proxy selection algorithm to increase his or her chances of controlling a majority of the proxies selected. The weighted randomized selection algorithm helps mitigate this attack by choosing $\lfloor \alpha k \rfloor$ proxies using the AS-disjoint proxy selection algorithm and $\lceil k(1-\alpha) \rceil$ proxies using the random proxy selection algorithm. Figure 11 shows how various α values perform when transit and endpoint ASes are malicious ($\alpha = 1$ reduces to the pure AS-disjoint proxy selection algorithm and $\alpha = 0$ reduces to the pure randomized proxy selection algorithm). An α as low as 0.3 shows a clear improvement over the randomized proxy selection algorithm when $k \geq 11$ and the improvement increases as α does. A higher α is not always better, however, because the proxy selection algorithm becomes more predictable as α increases. An α of 0.5 prevents the majority of proxies from being chosen by the potentially vulnerable AS-disjoint proxy selection algorithm while reducing the random algorithm's failure rate for $k \geq 11$.

5.5 Evaluation Summary

We demonstrated the effectiveness and efficiency of Senser through a combination of actual deployment on Amazon EC2 nodes and simulations using a dataset of PlanetLab proxies. In our evaluation using the Alexa top 1,000 websites, Senser was able to accurately detect censorship in many scenarios for a majority of sites.

In addition, under several likely attack scenarios, our AS-disjoint proxy selection effectively reduced the potential impact of malicious ASes. Our studies also showed that increasing the number of proxies hedges the risk in the presence of malicious ASes.

6. CONCLUSION

This paper introduces a system for validating retrieved web content in the presence of AS-level adversaries. Senser operates by forming a consensus of the requested web content using multiple proxies located at diverse vantage points in the network. By using an AS-aware proxy selection algorithm, Senser achieves good network diversity, and in many instances, prevents even large autonomous systems from undetectably altering requested web content. We validate our approach by using both simulations and a test bed deployment to show our system accurately detects the malicious modification of retrieved web content for a majority of sites in many conservative attacker configurations. Senser's practical approach improves the ability to detect censorship on the web over the current status quo, with no service or software modification required on end host providers. We show this to be essential, as 75% of the Alexa Top 1,000 sites do not provide SSL/TLS.

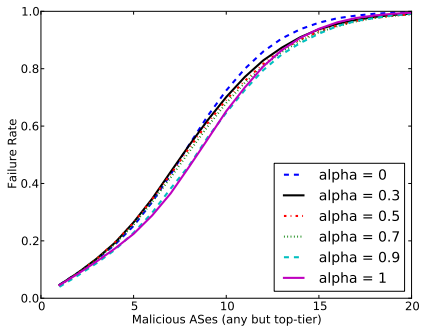


Figure 11: Weighted random. selection simulation (varying n).

Acknowledgments

We thank the anonymous reviewers for their insightful comments and suggestions. This work is partially supported by the National Science Foundation through grants CNS-1149832, CNS-1064986, CNS-1204347, and CNS-1223825. This material is based upon work supported by the Defense Advanced Research Project Agency (DARPA) and Space and Naval Warfare Systems Center Pacific under Contract No. N66001-11-C-4020. 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 Defense Advanced Research Project Agency and Space and Naval Warfare Systems Center Pacific.

References

[1] R. J. Bayardo and J. Sorensen. Merkle Tree Authentication of HTTP Responses. In *World Wide Web Conference (WWW)*, 2005.

[2] S. Burnett, N. Feamster, and S. Vempala. Chipping Away at Censorship Firewalls with User-Generated Content. In *USENIX Security Symposium (USENIX)*, 2010.

[3] H. Carrillo and D. Lipman. The Multiple Sequence Alignment Problem in Biology. *SIAM J. Appl. Math.*, 48(5):1073–1082, Oct. 1988.

[4] C. Chang and S. Lui. IEPAD: Information Extraction Based on Pattern Discovery. In *World Wide Web Conference (WWW)*, 2001.

[5] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine. Flexible Authentication of XML Documents. In *ACM Conference on Computer and Communications Security (CCS)*, 2001.

[6] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic Data Publication over the Internet. *J. Comput. Secur.*, 11(3):291–314, April 2003. ISSN 0926-227X.

[7] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium (USENIX)*, 2004.

[8] C. Ellison and B. Schneier. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Journal of Computer Security*, 16(1):1–7, 2000.

[9] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger. Infranet: Circumventing Web Censorship and Surveillance. In *USENIX Security Symposium (USENIX)*, 2002.

[10] M. Gertz, A. Kwong, C. U. Martel, and G. Nuckolls. Databases that tell the Truth: Authentic Data Publication. *IEEE Data Eng. Bull.*, 27(1):26–33, 2004.

[11] G. H. Gonnet, R. A. Baeza-Yates, and T. Snider. *Information Retrieval. Chapter on New Indices for Text: PAT Trees and PAT Arrays*. Prentice-Hall, Inc., 1992.

[12] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL Landscape: A Thorough Analysis of the X.509 PKI Using Active and Passive Measurements. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2011.

[13] A. Houmansadr, G. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[14] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *World Wide Web Conference (WWW)*, 2003.

[15] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy Routing: Toward Unblockable Internet Communication. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2011.

[16] S. Katti, J. Cohen, and D. Katabi. Information Slicing: Anonymity using Unreliable Overlays. In *USENIX Conference on Networked Systems Design & Implementation (NSDI)*, 2007.

[17] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the Edge Network. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2010.

[18] H. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path Prediction for Peer-to-Peer Applications. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[19] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2006.

[20] S. Ramachandran. Web Metrics: Size and Number of Resources. Available at `https://developers.google.com/speed/articles/web-metrics`.

[21] M. Schuchard, J. Geddes, C. Thompson, and N. Hopper. Routing Around Decoys. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[22] A. Sfakianakis, E. Athanasopoulos, and S. Ioannidis. CensMon: A Web Censorship Monitor. In *USENIX Workshop on Free and Open Communication on the Internet (FOCI)*, 2011.

[23] Snakes on a Tor Exit Scanner. Snakes on a Tor Exit Scanner. `https://gitweb.torproject.org/torflow.git/tree/HEAD:/NetworkScanners/ExitAuthority`.

[24] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *USENIX Security Symposium (USENIX)*, 2009.

[25] K.-C. Tai. The Tree-to-Tree Correction Problem. *J. ACM*, 26(3):422–433, July 1979.

[26] The Associated Press. Hacking in the Netherlands Took Aim at Internet Giants, September 5 2011. Available at `http://www.nytimes.com/2011/09/06/technology/hacking-in-the-netherlands-broadens-in-scope.html`.

[27] Tor Bridges. Tor: Bridges. `https://www.torproject.org/docs/bridges`.

[28] J. T. L. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey. An Algorithm for Finding the Largest Approximately Common Substructures of Two Trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):889–895, Aug. 1998.

[29] Q. Wang, X. Gong, G. T. K. Nguyen, A. Houmansadr, and N. Borisov. CensorSpoofer: Asymmetric Communication using IP Spoofing for Censorship-Resistant Web Browsing. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[30] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style Host Authentication with Multi-path Probing. In *USENIX Annual Technical Conference (USENIX-ATC)*, 2008.

[31] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the Network Infrastructure. In *USENIX Security Symposium (USENIX)*, 2011.

[32] X. Xing. The Filter Bubble Chrome Extension. `http://bobble.gtisc.gatech.edu/`.

A. A SURVEY OF SSL SUPPORT

In order to quantify the level of SSL adoption of popular websites, we tested the ability of Alexa top-100, 1k, 10k, and 100k sites to support TLS/SSL (HTTPS) connections. Table 4 reports our results. "No page found" errors reflect cases in which an HTTPS connection could be established, but the server returned a 404 error. In several cases, there was a 301 or 302 redirect from the HTTPS page to the HTTP page. To err on the side of inclusion, we consider "successes" (i.e., SSL to be supported) as any case in which an SSL certificate existed, even if the certificate was misconfigured (i.e., did not report the correct domain name). The total successes and failures for each Alexa dataset are reported in the Table. As can be seen, a vast majority of the top websites on the web do not offer proper SSL connections.

Table 4: SSL support on the web.

Status	Top 100	Top 1k	Top 10k	Top 100k
Connection refused	42 (42%)	512 (51%)	5679 (57%)	59362 (60%)
No page found	2 (2%)	15 (2%)	122 (1%)	1185 (1%)
Redirect to HTTP	15 (15%)	195 (20%)	1740 (17%)	12305 (12%)
Total Success	41 (41%)	278 (28%)	2459 (25%)	27148 (27%)
Total Failure	59 (59%)	722 (72%)	7541 (75%)	72852 (73%)