Facade: High-Throughput, Deniable Censorship Circumvention Using Web Search

Ben Jones Sam Burnett Nick Feamster
Sean Donovan Sarthak Grover Sathya Gunasekaran Karim Habak
Georgia Institute of Technology

Abstract

Censorship circumvention systems that use HTTP as cover traffic make tradeoffs between deniability and performance by offering either deniability at the expense of performance (*e.g.*, Infranet) or performance at the expense of deniability (*e.g.*, StegoTorus). These systems do so because HTTP is typically very asymmetric, with very little capacity to carry covert data in each HTTP GET request; higher throughput channels achieve performance by generating sequences of HTTP GET requests that do not mimic normal user traffic patterns. Fortunately, the emergence of new web services makes it increasingly common for any individual HTTP GET requests to contain more entropy. For example, site-specific search services create GET requests that contain sequences of search terms that can encode more bits than a single deniable HTTP request otherwise would. In this paper, we design a new encoding technique that uses web search terms to encode hidden messages in an upstream channel for censorship circumvention; implement the encoding technique in a system that resists fingerprinting attacks; and compare the security and performance of Facade to existing censorship circumvention systems that use HTTP as cover traffic.

1 Introduction

Censorship restricts the autonomy of citizens and may even be considered a violation of human rights [9]; it is also becoming increasingly pervasive. The Open Net Initiative reports that 38 of 74 countries tested showed signs of Internet censorship [11] of political, religious, or social content. Researchers have developed circumvention tools to help users, but most of these tools would not work if the censor blocked all traffic except HTTP. To enable free speech and empower citizens in light of the ubiquity of HTTP communications, researchers have designed various covert channels that operate over HTTP [4,13]. These systems must operate in the face of a censor who may seek to either detect that a user is requesting censored content or disrupt the communications between a client and a censored destination.

Systems that build covert channels into HTTP must make design choices concerning the tradeoff between *deniability* and *performance*. Deniability gives a user the ability to disavow the use of a particular system; typically, systems attain deniability by encoding data inside cover traffic that masquerades as innocuous traffic; this traffic must both conform to standard protocol behavior to resist fingerprinting attacks [5] and reflect the traffic patterns that would be generated by a "normal" user [1,4]. Existing systems provide covert channels that either offer performance at the expense of deniability (*e.g.*, StegoTorus [13]) or deniability at the expense of performance (*e.g.*, Infranet [4], Collage [1]). In particular, these channels lack good ways to communicate data from client to server and thus offer very poor upstream performance. Many of these systems are also not resistant to "fingerprinting attacks", whereby a censor can detect an attempt to emulate a protocol by its unorthodox responses to error conditions [5].

In this paper, we develop a covert communication channel in HTTP that provides the deniability of previous circumvention systems such as Infranet and Collage but with significantly better upstream throughput. We take advantage of the growing ubiquity of Web search to encode more information into upstream HTTP GET requests, in the form of search queries. Infranet and Collage achieve deniability by encoding upstream messages in a sequence of Web requests whose pattern matches that of a normal web user, yet the encoding of these messages relies on using individual Web requests or tasks that map to symbols, both of which require several Web requests to convey a single covert message. (For example, a visible HTTP request in Infranet maps to a symbol that is $\lg(k)$ bits, where k is the number of links on a webpage.)

In contrast, our new encoding takes advantage of the fact that many pages have site-specific search (*e.g.*, Wikipedia), and that user searches generate HTTP GET requests that contain these search terms. HTTP GET requests containing search terms are good cover traffic because they are higher entropy than individual link clicks; thus, they make it possible to encode more data than existing schemes. Because the set of possible search terms provides much more entropy than a selection from a set of possible links, any individual visible HTTP request thus encodes a symbol with many more bits. As in Infranet, these symbols can be mapped either directly to bits that encode a hidden message or as an index into a dictionary of known URLs. To make the visible request stream resemble normal user activity, we can also insert GET requests that represent "dummy clicks" that do not represent any hidden information.

We implement this new encoding in a tool called Facade, which tunnels hidden messages (*e.g.*, censored URLs) through a Selenium-driven Chrome browser as a sequence of protocol-compliant HTTP GET requests containing plausible sequences of search terms. Houmansadr *et al.* showed that protocol emulations or even implementations can be fingerprinted by their responses to error conditions [5]. By tunneling requests through Chrome, Facade's requests can resist these fingerprinting attacks. To demonstrate the feasibility of Facade's encoding, we build a message channel on top of OpenSearch [2], a popular search format that allows websites to offer site-specific search. Though Facade could only be deployed on sites with sufficiently high cover search traffic, our analysis of search and click sequences based on a public web search corpus shows that Facade can offer significantly higher throughput than Infranet while achieving comparable deniability properties

The rest of the paper proceeds as follows. Section 2 presents the threat model we consider. Section 3 describes Facade's encoding mechanisms for upstream communication; Section 4 describes the Facade system design and prototype. Section 5 performs a security and performance evaluation, and Section 6 outlines various limitations and avenues for future work. Section 7 surveys related work, and Section 8 concludes.

2 Threat Model

We consider a censor that can mount various passive and active attacks in an attempt to either discover users of Facade or otherwise disrupt communication. A censor may mount a passive attack to determine if a client is running Facade and block the traffic. The censor can monitor any traffic between these two regions and may also attempt to actively disrupt communications. The censor will not, however, disrupt communications if it interferes with legitimate, uncensored communications. We also assume that the attacker can store one HTTP request/response pair and can allow, block, modify, or generate traffic. Passive attacks do not modify existing traffic, and they do not generate additional traffic. Active identification, on the other hand, includes probing and scanning potential Facade clients and servers; for example, an attacker might actively probe clients or modify traffic to determine how clients respond to error conditions. A client that merely mimics a protocol but does not tunnel traffic through it may be vulnerable to active attacks [5].

Facade's threat model is similar to the threat models considered in the design of other circumvention systems, including StegoTorus [13], Infranet [4], and ScrambleSuit [15]. In Facade's threat model, a client in a censored country communicates with a server outside the censored region. The threat model is comparable to (if not more powerful than) that of real-world adversaries, as a case study of the Great Firewall of China (GFW) shows. The GFW retains limited state, only monitors HTTP requests, and actively scans suspicious addresses to determine if they are running Tor [6, 14].

```
<method> /<path>?<query>#<fragment> HTTP/1.1
Host: <host>
Connection: keep-alive
Accept: text/html
User-Agent: Mozilla/5.0
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: <cookie>
```

Figure 1: *An example HTTP request. There are several possible sources of entropy in a request that we can use for covert upstream communication, indicated in italics.*

3 Deniable Upstream Encoding

This section develops techniques for encoding data in HTTP requests. We focus on improving upstream communication because HTTP requests contain few opportunities for encoding covert data and are thus usually a bottleneck, particularly for symmetric communication.

3.1 Design Goals and Considerations

In designing a covert communications channel, we aim to achieve the following design goals:

- *Robust:* The tool should be able to continue operation despite attacks from the censor.
- *Deniable:* It should be difficult to distinguish Facade and normal traffic.
- *Expensive to block:* The channel should be too expensive for the censor to block, either in terms of resources to fingerprint the tool or collateral damage to legitimate traffic.
- *Real-time:* Latency between the client and server should be at most on the order of seconds.

Achieving deniability requires identifying sources of entropy in HTTP requests into which we can embed covert data. Figure 1 shows an example HTTP request. The host, path, query string, fragment, cookie, and body may be modified in various ways to encode data on the upstream channel, only the path, query string, and fragment are deniable. The host and cookie fields must generally remain constant between connections to the same server because changes to them are not deniable. Storing data in the body field is also not robust it is only available in

POST requests and constantly sending POST requests would look suspicious. Although data can be hidden in the cookie field under certain threat models, we assume that the attacker could identify clients that send cookies that the server did not previously set. We also assume that the adversary might enumerate all possible domain names for an IP address, making the host field ineligible for data encoding. By encoding data in only the path and query string, we modify the HTTP request header in a way that is robust to statistical analysis techniques.

Although simple encodings over the given fields (*e.g.*, direct Base64 encoding of data) are sufficient to evade detection under weaker threat models [13], we target a more powerful adversary and instead exploit sources of entropy from real web services. Web service protocols are ubiquitous, so there is significant legitimate traffic for each protocol, making this covert channel more difficult to distinguish. Additionally, disrupting or blocking access to a Web service might induce significant collateral damage to legitimate users of that service. Censors are typically governments and may not wish to harm local businesses that rely on or make money from web services.

3.2 Search Encoding

We use the OpenSearch Web service protocol as an example of how to build a covert channel in HTTP requests. The OpenSearch interface enables web sites to specify an interface for searching the website [2], thereby enabling users to search the site without navigating to a search engine. The OpenSearch protocol adds search terms to the query string for a template URL. For example, the URL `https://duckduckgo.com/?q=hello` searches `www.duckduckgo.com` for the term "hello." These search terms are the source of entropy that we exploit to increase upstream covert channel throughput. We assume that the censor will not directly block the OpenSearch protocol because doing so would cause significant collateral damage.

We create a deniable covert channel within the OpenSearch protocol using a dictionary encoding of search terms with the same entropy as real search queries. A dictionary encoding relates words to bytes of the covert message. For example, if the dictionary is {"hello", "goodbye"}, then "hello" could represent a 0 and "goodbye" could represent a 1. In the case of the OpenSearch covert channel, the client and server could prenegotiate a dictionary to use with the same entropy as real searches. Depending upon the desired level of security, the client and server could negotiate a client specific dictionary out of band so that the encoded requests perfectly match the entropy of the user's real searches. To transmit data to the server, the client joins these terms with +'s and adds them to the query string of the URL. The exact form of the path and the query string variables depend on the site, but we use a standard structure: `<hostname>/?q=<terms>`. In the given format, we replace hostname with the address of the server and terms with the dictionary encoded search terms. For example, one possible encoding would generate the URL `http://www.example.com/q?=objections+law` to encode the covert text "hello".

3.3 Click Range Mapping Encoding

As our evaluation in Section 5 will show, the search encoding improves on Infranet's upstream performance while maintaining its deniability. However, we can squeeze more information out of the channel and make the protocol seem more realistic by encoding data with links that we click on from the response HTML. For example, if the server returns a page with 8 links and the client selects the 4th link, then the client will communicate a 3 back to the server. If there are k links ordered on a page, then the client can encode $lg(k)$ bits by selecting the link on the page whose order corresponds to the data the client wants to send back to the server. This is Infranet [4]'s main technique for encoding data; by combining search encoding and click range mapping, we can significantly improve the upstream performance that Infranet achieves. Using both techniques also increases deniability because it allows the client to model a real user who will likely switch between searching and selecting results.

4 Facade

We now describe the design of Facade, a prototype system that implements the encoding mechanisms that we described in Section 3.

Figure 2 shows the design of Facade, which is comprised of three layers. The *framing layer* communicates with the tunneled application via SOCKS, manages the authentication of clients to servers, and provides transport and session layer functionality for Facade. Specifically, the framing layer breaks the application's traffic into frames, and handles the transmission and reassembly of these frames. The *encoding layer* accepts frames of data from the framing layer, encrypts them, and encodes the data within HTTP requests and responses. This layer encodes downstream (server to client) and upstream (client to server) traffic differently. The *transmission layer* takes encoded HTTP requests and responses and tunnels the data through existing HTTP tools in a manner that avoids fingerprinting attacks.

Framing. The framing layer provides an interface between applications which send hidden messages; and the encoding layer, which encodes each message as a sequence of web search requests. It also manages authentication. Facade frames each hidden message from the application in a four-byte header. The header has a two-byte sequence number, a one-byte session ID, and a

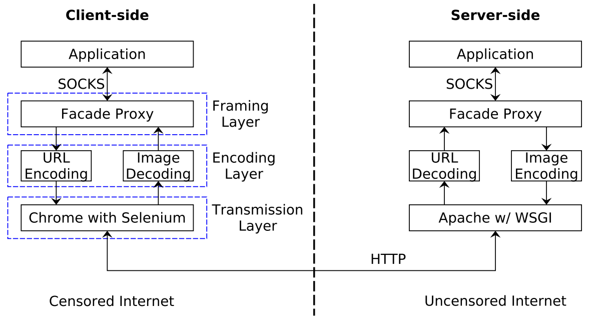

Figure 2: *Facade uses multiple layers to encode and send data.*

one-byte field for individual flags. The sequence number allows application data units from the application to be fragmented into smaller units. The MORE flag allows the server to indicate to the client that the server has more data to send; this mechanism is useful because the server cannot send HTTP responses without first being prompted with HTTP requests from the client. The SYN flag signals that the client wishes to authenticate a new session. Facade relies on the application's transport layer (above the framing layer) to provide the reliability guarantees that a particular application requires. Facade uses the ScrambleSuit session authentication protocol [15], which relies on a shared secret to authenticate users before using an extension of the Uniform Diffie-Hellman handshake to establish a session key.

Encoding. The encoding layer uses the methods described in Section 3 to hide data within HTTP requests and responses. After receiving data to send from the framing layer, the encoding layer encrypts the data, encodes it, and passes it to the transmission layer. To receive data, the server side reverses this process.

Transmission. To defend against fingerprinting [5], Facade must tunnel the encoded data through software that uses a real HTTP implementation. To generate the requests that the encoding layer produces, Facade uses Selenium's [10] ChromeDriver, which is a testing framework that integrates Chrome's automation functionality with Python bindings. The server side uses Apache, a widely deployed web server, with WSGI for integration with Python. Because both of these applications are widely used, the censor cannot block all traffic using those implementations and they provide the requisite fingerprinting resistance. By integrating with existing tools, Facade also makes it easier for web site operators to deploy our tool, thereby increasing adoption and making Facade more difficult to block.

5 Evaluation

We analyze Facade's robustness and deniability in the face of both passive and active attacks from the censor. We then analyze Facade's performance and compare it to other systems that design covert channels based on HTTP, including StegoTorus [13] and Infranet [4].

5.1 Security Evaluation

We enumerate a variety of passive and active attacks that a censor could mount against Facade under our threat model (Section 2) and argue that Facade is robust against these attacks.

Passive attacks. Facade resists passive attacks that aim to either identify Facade traffic or determine that a particular user is sending Facade traffic.

- *Request pattern analysis.* To prevent the censor from blocking traffic based upon a regular expression, Facade's encoding layer uses dictionary encodings to generate URLs based on real search terms. Because Facade's transmission layer generates traffic using a Chrome browser, Facade's traffic is normal HTTP traffic.
- *Blacklisting and filtering.* A censor could block traffic using IP or port blacklists. Deploying Facade on legitimate web servers would make it difficult for a censor to deploy a blacklist that blocks Facade, presuming that the censor does not want to also block access to the legitimate site and the legitimate site has sufficient cover search traffic. The censor cannot block port 80, and our threat model assumes that the censor is not willing to block arbitrary web traffic without ensuring that the web server is running Facade. Because Facade is designed to be indistinguishable from real HTTP traffic and resistant to active fingerprinting attacks, discovering and blacklisting Facade servers would be difficult. Even if the censor could determine that if a server were running Facade, we could move Facade servers to Amazon AWS and assume that the censor is unwilling to block all AWS services.
- *Entropy-based traffic analysis.* The censor could block Facade if the entropy of a sequence of search results differed from that of a normal user's search query sequence. A censor could also observe that the pattern of searches and "clicks" did not match that of a normal user. To defend against this type of analysis, Facade generates search request sequences that have similar entropy to existing search queries; to defend against analysis that looks for deviations in click sequences on search results, Facade can insert false clicks that match normal request patterns and use those clicks to perform range mapping (as described in Section 3.3). Depending upon the level of security the user wants, Facade

could also be tailored with a user specific dictionary or click sequence.

Active attacks. Facade also resists censors who may actively probe the client or server or otherwise actively modify Facade in an attempt to disrupt communication.

- *Reordering, replaying, dropping, or disrupting HTTP messages.* A censor could disrupt the communications channel by manipulating the visible HTTP traffic. However, its options for doing so are limited presuming its unwillingness to disrupt legitimate HTTP transactions. Because modifying the order of search terms or the terms themselves would modify the semantic content of a query, we assume that the censor is unwilling to change the search string. Should this become a significant attack from the censor, an erasure code may also offer additional security at the cost of performance.
- *Server discovery via session initiation probing.* The censor cannot detect Facade servers with active scanning that attempts to initiate Facade sessions because Facade uses a pre-shared secret for authentication. Facade uses ScrambleSuit's [15] key exchange and inherits the properties of that authentication mechanism (and is thus robust against preplay and replay attacks).
- *HTTP fingerprinting.* The censor cannot fingerprint Facade's HTTP implementation (*i.e.*, using a "parrot" attack [5]) because Facade uses a popular browser and web server; therefore, all messages that the censor observes are protocol-compliant, and both endpoints respond to active probing as a normal client or server.

5.2 Performance Evaluation

In this section, we quantify the highest capacity Facade can encode data at and maintain deniablity comparable to existing circumvention systems (*e.g.*, Infranet). We also analyze the tradeoff between performance and deniability as clients choose to encode data in search queries vs. click range mapping. Finally, we compare the performance of Facade to two existing real-time HTTP covert channels: StegoTorus and Infranet.

Entropy of Facade encodings. Using standard entropy calculations, we can quantify the maximum amount of information that Facade can encode in a single HTTP request and still remain deniable. Using the AOL search corpus [8], we calculated that the average length of a search query was 17.42 bytes and that the average information content of per byte, or the entropy, was 4.48 bits. Thus, a search query can encode 78.04 bits of information while maintaining deniability.

In contrast, click range mapping encodes $\lg k$ bits per URL, which is three bits with $k = 8$ from Infranet; k could be set to a larger value, but $k = 8$ was the value used in the Infranet evaluation. Because these entropies differ by so much and better deniability requires a balance between click range mapping and search encoding, Facade faces a tradeoff between performance and deniability; there is an inverse relationship between the fraction of messages encoded with click-based range mapping and Facade's throughput. Although Facade would perform best when only using the search encoding, such an encoding would not model normal user behavior and might trigger detection of Facade. On the other hand, it is not necessary to encode all traffic with click range mapping because users do some searching while browsing.

Protocol	Encoded Bits per URL	Deniability
Facade	78.04	statistical deniability
Infranet	3	statistical deniability
Stego-Torus	12,000	N/A

Table 1: *Facade communicates more information than Infranet, but StegoTorus significantly outperforms Facade.*

Comparison to existing tools. Although Infranet and StegoTorus both achieve reasonable throughput, Facade achieves better performance than Infranet for comparable levels of deniability (StegoTorus, on the other hand, does not provide statistical deniability). Infranet used range mapping to encode data, encoding $lg(k)$ bits per URL with $k = 8$. Infranet boosted throughput by proportionally weighting URLs according to their likelihood and correlated these more probable visible URLs with more probable covert URLs (arithmetic coding). Unfortunately, this optimization only works when the probability distribution over hidden messages (*i.e.*, the set of URLs that a client might request) is known in advance; this distribution is incredibly difficult to model in general.

Facade performs significantly better than existing approaches on encrypted data: it encodes ˜26 times as much information as Infranet without a loss in deniability. Unlike Infranet, StegoTorus does not evaluate the entropy of their encoding. However, StegoTorus uses Base64 encoding, so if StegoTorus were to include 2,000 Base64 encoded bytes in each URL, the maximum length many web servers will accept, then the average entropy per URL is $6 \cdot 2,000 = 12,000$ bits. Obviously, StegoTorus can transmit significantly more data for each URL, but, unlike either Facade or Infranet, it offers no statistical deniability if a censor were to analyze the user's request patterns. Table 1 summarizes these results.

6 Discussion

We discuss possibilities for improving the performance and deniability of Facade, as well as improving Facade's resistance to fingerprinting attacks.

Improving performance. Facade's upstream communications channel currently relies on search terms that are embedded in HTTP GET requests. Although such search terms provide a reasonable source of entropy, other web services may offer deniable encoding mechanisms with even higher entropy. For example, a web service that permits sequences of HTTP POST requests may be able to send more bits per symbol with comparable deniability. Additionally, because Facade's encoding layer relies on HTTP requests and responses, sending data from client to server may be slower than necessary if downstream data is "clocked" on the client's HTTP requests. To improve performance, the framing layer might send multiple HTTP responses downstream in parallel.

Better deployability. Facade's transmission layer ensures that Facade is resistant to fingerprinting attacks. Although Facade's current implementation uses Selenium to integrate with Chrome, this integration poses deployment hurdles, especially for tools like Tor who may wish to use Facade as a pluggable transport. In the future, we plan to eliminate these dependencies by integrating Facade with browsers through a JavaScript shim layer.

7 Related Work

We briefly discuss systems that have built covert channels in HTTP. We then survey other methods of obfuscation and communication channels that mimic other protocols.

Covert channels in HTTP. The three systems that are most similar to Facade are StegoTorus [13], Infranet [4], and Collage [1], all of which encode hidden messages (*i.e.*, HTTP requests) in sequences of other visible HTTP traffic. StegoTorus [13] was designed as a pluggable transport for Tor. It fragments Tor traffic into chunks and hides them in HTTP by filling the cookie, path, and query string fields of the URL with base-64 encoded data. StegoTorus is vulnerable to active probing attacks because it does not attempt to generate a deniable request stream, nor does it generate protocol-compliant HTTP traffic. Therefore, although StegoTorus offers higher throughput than other circumvention systems based on HTTP, it is neither deniable nor resistant to fingerprinting attacks.

Infranet [4] encodes hidden messages as sequences of visible HTTP requests that mimic plausible user request sequences, thus providing better deniability than Facade. On the other hand, each request from an Infranet client only transmits a few bytes; using arithmetic encoding based on a known distribution of hidden messages, Infranet can improve the upstream throughput, but Facade can send significantly more bits per request for comparable levels of deniability. Collage [1] is an circumvention tool that also encodes hidden messages using sequences of HTTP requests. Its throughput is similar to Infranet, but it uses user-generated content sites as intermediaries for exchanging hidden messages.

Protocol mimicry. Several systems encode hidden messages within a protocol. For example, SkypeMorph [7] encodes hidden messages as a traffic stream where packet sizes and timings resemble Skype. Although SkypeMorph accurately mirrors Skype's distribution, Houmansadr *et al.* [5] observe that SkypeMorph fails to emulate Skype's UDP control channel and is thus vulnerable to fingerprinting. Format Transforming Encryption (FTE) [3] also emulates protocols by encrypting data to match a regular expression and may produce non-compliant packets. Although mimicry is sufficient for some threat models, Facade defends against a stronger adversary.

Traffic randomization. Censorship circumvention systems aim to generate traffic that evades detection by a censor and although mimicking a protocol is generally considered more deniable than generating random traffic, several systems opt for the latter approach. obfs3 [12] has such a threat model and merely adds another encryption layer on top of Tor, as such, this protocol is still vulnerable to fingerprinting attacks. ScrambleSuit [15] extends this encryption layer with randomized packet sizes, timings, and payloads and active probing resistance. Although randomizing traffic patterns can defeat some forms of DPI, traffic analysis could detect and block these systems.

8 Conclusion

We have presented Facade, a proxy system that hides censored data inside HTTP cover traffic. Facade achieves deniability by encoding messages in visible HTTP requests and responses from a real web browser (Chrome) and whose traffic patterns mimic normal user behavior. In contrast to Infranet, which achieves deniability by encoding messages in sequences of plausible web requests, Facade achieves deniability by encoding messages in plausible query strings. Though Facade is limited to deployment on websites with sufficient cover search traffic, it offers similar deniability with much greater throughput than Infranet.

We have implemented Facade by tunneling hidden messages (*e.g.*, censored URLs) through a Selenium-driven Chrome browser that issues a sequence of protocol-compliant HTTP GET requests containing plausible sequences of search terms. Facade's deniability and performance properties—as well as its lightweight implementation and SOCKS interface—make it well-suited for use in a variety of settings, including as a Tor pluggable transport.

References

[1] S. Burnett, N. Feamster, and S. Vempala. Chipping away at censorship firewalls with user-generated content. In *Presented as part of the 19nd USENIX Security Symposium (USENIX Security 10)*, pages 453–469. USENIX, 2010. (Cited on pages 1 and 6.)

[2] D. Clinton. Opensearch specification. `http://www.opensearch.org/Specifications/OpenSearch/1.1`. (Cited on pages 2 and 3.)

[3] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton. Protocol misdentification made easy with format-transforming encryption. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, pages 61–72. ACM, 2013. (Cited on page 6.)

[4] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. R. Karger. Infranet: Circumventing web censorship and surveillance. In *USENIX Security Symposium*, pages 247–262, 2002. (Cited on pages 1, 2, 3, 4 and 6.)

[5] A. Houmansadr, C. Brubaker, and V. Shmatikov. The parrot is dead: Observing unobservable network communications. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, 2013. (Cited on pages 1, 2, 4, 5 and 6.)

[6] S. Khattak, M. Javed, P. D. Anderson, and V. Paxson. Towards illuminating a censorship monitor's model to facilitate evasion. In *Presented as part of the 3rd USENIX Workshop on Free and Open Communications on the Internet*, Berkeley, CA, 2013. USENIX. (Cited on page 2.)

[7] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 97–108. ACM, 2012. (Cited on page 6.)

[8] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*, InfoScale '06, New York, NY, USA, 2006. ACM. (Cited on page 5.)

[9] M. Rundle and M. Birdling. Filtering and the international system: A question of commitment. In *Access Denied*. MIT Press, 2008. (Cited on page 1.)

[10] Selenium Developers. Seleniumhq browser automation. `http://docs.seleniumhq.org/`. (Cited on page 4.)

[11] The OpenNet Initiative. Filtering data. http://opennet.net/sites/opennet.net/files/ONI_data-20121029.zip, 2012. (Cited on page 1.)

[12] The Tor Project. Tor pluggable transports. `https://www.torproject.org/projects/obfsproxy.html.en`, 2002. (Cited on page 6.)

[13] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 109–120, 2012. (Cited on pages 1, 2, 3, 4 and 6.)

[14] P. Winter and S. Lindskog. How the great firewall of china is blocking tor. In *Presented as part of the 2nd USENIX Workshop on Free and Open Communications on the Internet*, Berkeley, CA, 2012. USENIX. (Cited on page 2.)

[15] P. Winter, T. Pulls, and J. Fuss. Scramblesuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 213–224, New York, NY, USA, 2013. ACM. (Cited on pages 2, 4, 5 and 6.)