Bootstrapping Communications into an Anti-Censorship System

Patrick Lincoln,[1] Ian Mason,[1] Phillip Porras,[1] Vinod Yegneswaran,[1] Zachary Weinberg,[2] Jeroen Massar,[3] William Simpson,[3] Paul Vixie,[3] and Dan Boneh[4]

[1]SRI International [2]Carnegie Mellon University / SRI [3]Internet Systems Consortium [4]Stanford University

Abstract

Adversary-resistant communication bootstrapping is a fundamental problem faced by many circumvention (anti-censorship) systems such as Tor. Censoring regimes actively harvest and block published Tor entry points and bridge nodes. More recently, some countries have resorted to reactive (follow-up) probing of the destination hosts of outbound encrypted traffic to identify unpublished Tor nodes. We present the design of a new architecture for bypassing censorship, called DEFIANCE, that extends Tor with resilience to both active harvesting and network scanning attacks. The first goal is accomplished using the DEFIANCE Rendezvous Protocol (RP), and the second is achieved using a novel handshake that we call Address-Change Signaling (ACS). We describe prototype implementations of both components, discuss the limits of our architecture, and evaluate what it would take for a determined adversary to defeat our system. While we develop our prototype components over Tor, their design can be easily extended to other circumvention systems.

1 Introduction

Although originally designed as an anonymity system, Tor is quietly gaining prominence as a tool for bypassing Internet censorship. Tor uses onion routing to guarantee traffic anonymity; that is, Tor interposes an intermediate set of three relays with corresponding *onion* layers of encryption. Each relay learns only of its previous and next hops, and no relay learns both the origin and destination [8]. As Tor was designed to be an anonymity system and not a circumvention system, adversaries blocking *all* use of Tor is outside the scope of its threat model. Hence, Tor traffic is often subjected to indiscriminate (wholesale) blocking by censorship regimes.

There are two common techniques an adversary might adopt for blocking use of Tor within their network. The first and most common form of attack involves entry point blacklisting. For example, the list of Tor entry node IP addresses is publicly available and is trivially blocked today by many countries practicing adversarial filtering. To counter this attack, Tor introduces the concept of bridge nodes: entry nodes that are not listed in the main Tor directory [4]. To obtain a bridge address, users either visit a website or send an email from a gmail address to an auto-responder (`bridges@torproject.org`). However, it is not difficult for a determined, well-funded adversary (such as a nation-state) to harvest a list of bridge IP addresses by the same means. Currently, Tor bridge addresses distributed in the above forms are all inaccessible from China. This bootstrapping problem, that we try to address using rendezvous strategies described in this paper, is a limitation that is fundamental to all circumvention systems.

The second form of blocking involves the use of protocol signatures that distinguish Tor encrypted Transport Layer Security (TLS) packets from other applications that use the TLS protocol. Such tactics have been successfully employed by censoring countries on multiple occasions [5, 16, 18]. While the Tor developers have been able to modify the software each time to remove the distinguishing signature, this is an arms race. This has also stimulated the development of proactive defenses in the form of pluggable transports that broadly add dress such tactics, such as Obfsproxy [6] which obscures all cleartext patterns in Tor traffic.

Additionally, adversaries may use both active and reactive network probing techniques to uncover and block unpublished entry points. As an example of a reactive probing attack, recently the Chinese Great FireWall was observed conducting two follow-up probes for each outbound port TCP/443 connection. While the target of the first set of probes with garbage binary data was unknown, the second set specifically targeted Tor by performing an SSL negotiation, an SSL renegotiation and successfully building a one-hop Tor circuit [21]. Such attacks motivate the need for some form of access control before clients can validate the existence of a Tor bridge and obtain access to Tor services.

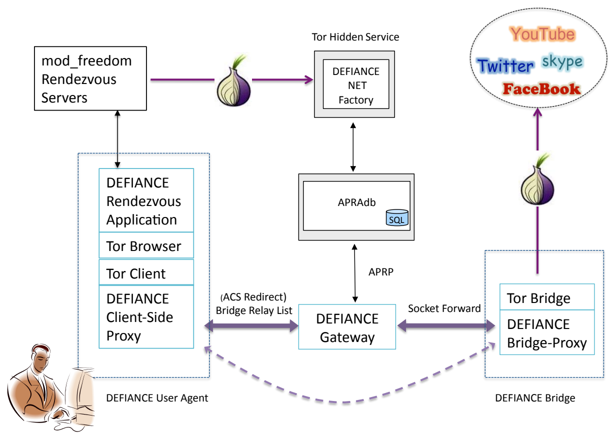

Figure 1: DEFIANCE Architecture Overview

Our goal is to extend Tor with a tunneling service that is resilient to active and passive filtering. To address the aforementioned problems, we introduce the concepts of DEFIANCE *Rendezvous Protocols (RP)*, *Address Pools (AP)*, and *Address-Change Signaling (ACS)*. The DEFIANCE rendezvous process assigns IP address contacts from the Address Pool to a user (or small group of users). ACS specifies that these contacts must be used in a particular way and in a particular order or these bridge relays either will not respond to the contact or will return an innocuous response. Below, we describe our architecture and these concepts in greater detail.

2 Architecture

The communications bootstrap protocols described in this paper are part of a broader system, called DEFIANCE. Its goal is to make Tor resistant to all common filtering attacks. Figure 1 illustrates the overall DEFIANCE framework. The DEFIANCE User Agent running on the user's machine is a Tor client augmented with support for DEFIANCE protocols. Rendezvous servers reveal DEFIANCE *Gateway* addresses via the DEFIANCE Rendezvous Protocol. Gateways respond to Address-Change Signaling by granting access to a DEFIANCE *Bridge*. Bridges provide connectivity to the existing Tor network and thence to the unfiltered Internet.

DEFIANCE Rendezvous Protocol. The DEFIANCE Rendezvous Protocol specifies a series of network interactions and local calculations that ultimately reveal to each DEFIANCE user one of the many addresses of a DEFIANCE Gateway. Rendezvous is designed to be straightforward for honest users, but unacceptably expensive for an adversary who seeks to learn all of a gateway's addresses; it involves both proof-of-life and proof-of-computational-work challenges. The design anticipates that the tasks involved will periodically have to be made more difficult as adversaries gain sophistication. A user who successfully carries out rendezvous will learn a *Network Entry Ticket* (NET) that tells them how to perform Address-Change Signaling and gain access to a set of bridges.

In this paper, we describe a simple instantiation of a rendezvous service that we refer to as `mod_freedom`, an Apache module run by volunteer webservers. Rendezvous servers periodically probe the DEFIANCE NET Factory (a Tor hidden service) to obtain network entry tickets. In practice, multiple realizations of rendezvous services might be spawned and co-exist to increase the level of effort on the censor.

Address Pools. Address Pools are diverse blocks of IPv4 or IPv6 addresses, operated by agencies that already possess such blocks and wish to assist in circumvention. Address Pools frustrate address blocking by sheer size and diversity: any address in a pool can be (perhaps only briefly) the address of a DEFIANCE Gateway. Each Gateway handles a local or regional address pool consisting of many hundreds of small IP address blocks (with 4 to 256 addresses per block). A centralized service, the Address Pool Registration Administrative database (APRAdb), maintains a registry of active addresses, keeps track of current usage of addresses within each pool, and coordinates the gateways with the NET Factory. DEFIANCE User Agents and `mod_freedom` servers never communicate directly with the APRAdb.

Address-Change Signaling. Address-Change Signaling (ACS) prevents adversaries from actively probing for DEFIANCE Gateways. Although all service responses and protocols have a fingerprint, this design attempts to mimic other wide-spread protocols and services, hiding the addresses in plain sight.

This term is similar to traditional frequency-change signaling, but here IP addresses correspond to frequencies. Using the information in a NET, an ACS client makes short-lived connections to DEFIANCE Gateways listening on a sequence of IP addresses. If each connection is timed correctly and transmits the proper authentication, the gateway will reveal values for the next connection. Otherwise, the gateway produces an innocuous cover response.

Ultimately, the gateway reveals contact information for a DEFIANCE Bridge that also has ephemeral addresses provisioned from the Address Pools; this bridge can be used to relay to the Tor network.

Traffic Camouflage. Once the client establishes contact with a DEFIANCE Bridge, its traffic is camouflaged to prevent the adversary from noticing characteristics of the Tor protocol. While a discussion of specific camouflage techniques is outside the scope of this paper, its objective is to steganographically transform the encrypted Tor stream into one of many unencrypted protocols (e.g., HTTP, RTP) in common use on the public Internet. One means to accomplish this objective is through the use of Tor pluggable transports such as Obfsproxy [6], Skypemorph [12] and StegoTorus [20].

3 DEFIANCE Rendezvous

The DEFIANCE *rendezvous protocol* (RP) lets a user in a censored region of the network receive a small amount of information, leading to the DEFIANCE gateways, from servers outside the censored region. Since a censor may control a non-negligible fraction of our clients, it should be difficult for a client to automatically harvest a large number of entry point addresses. Thus an ideal rendezvous scheme would require human involvement (proof-of-life), machine time (proof-of-work), and artificial delays to discover multiple entry points. There is a fundamental trade-off here between system usability and resilience to harvesting attacks.

The puzzles limit the rate at which the censor can learn the identity of new entry points. Our goal is to produce an entry point pool capable of maintaining a positive creation to discovery rate. Let us assume that the adversary commits three 8-hour shifts of 100 humans (round-the-clock) to collect DEFIANCE entry points for blacklist generation. Given a 60-minute average rendezvous point discovery protocol that mixes a series of human and machine challenges, we can anticipate the adversary discovering a minimum of 2400 entry points per day. If we assume that DEFIANCE manages a static entry pool of 10,000 addresses, the adversary could hope to reach closure LOE (level of effort) in about 19 days. More generally, if our pool size is N and the adversary performs K full rounds of the rendezvous protocol (providing that adversary with K random entry points), then a standard balls-and-bins argument shows that the expected number of entry points obtained is

$$E[x] = N - N(1 - \frac{1}{N})^K \tag{1}$$

Hence, performing N full rounds of the above rendezvous protocol will reveal only 63% of our entry points (e.g., given a pool size of 1000 entry points, then 1000 rounds of the rendezvous protocol will yield, on average, only 630 entry points), which has little impact on the system. To discover all of our entry points the censor will need to conduct the full rendezvous protocol ($N \ln N$) times in expectation (known as the coupon collector's bound [17]). We can also employ out-of-band signaling including word-of-mouth social networking to share open entry points among small groups of mutually trusting end users. That is, even if a majority of DEFIANCE entry points are blocked, the user community can share knowledge of the currently available entry points. Thus, it is imperative for the adversary to achieve Closure LOE in order to ensure DEFIANCE is inoperable.

Various rendezvous protocols are being developed today, by the Tor project and others; see [11] for more details. We present here our design for a rendezvous protocol that can be "piggy backed" on existing Web servers. Our system goals are to (*i*) have a large and unpredictable contact surface; (*ii*) minimize deployment and management overhead; (*iii*) be straightforward for end users; and (*iv*) be as robust as possible against adversaries who seek to harvest large numbers of addresses (and block them).

We envision that, at any given time, there would be thousands of operational mod_freedom webservers, many of which are ideally collocated with productive websites. This list of servers would be distributed in pieces through many channels including public websites, social networks and the DEFIANCE software itself. In addition, DEFIANCE users may learn of new mod_freedom servers during the rendezvous process. While it would be arbitrarily difficult for a censor to obtain a complete list of operational mod_freedom servers, our expectation is that it would be even more painful for them to block use of all these websites.

The proposed RP, as currently implemented, consists of three separate components:

1. The DEFANCE User Agent is a client-side desktop application that retrieves and deciphers NET payloads from a `mod_freedom` server.

2. The Apache `mod_freedom` module acts as a middleman, maintaining a supply of NET payloads that it serves in response to valid RP requests. It has no detailed knowledge of NET payloads other than how to replenish its supply, verify a NET factory digital signature, and how to recognize a bona fide NET request. Thus an adversary running a `mod_freedom` server would have little to gain, other than the daily supply of NETs it would intercept.

3. The NET factory constructs and supplies NET payloads to `mod_freedom` servers. Since `mod_freedom` webservers could be run by adversaries, a potential concern is that they could launch distributed denial of service attacks against the NET factory. Hence, we implement the NET factory as a Tor hidden service to make it less conspicious. The NET factory also functions as a *Private Key Generator*, PKG, for the *identity-based encryption* used in the RP. It can throttle NET supply to disreputable `mod_freedom` servers, as well as reward reputable ones. The NET factory is also the only component that deals directly with the APRAdb.

3.1 An Apache Module for DEFIANCE Rendezvous

The Apache module, `mod_freedom`, hooks into the `ErrorDocument` handler; that is, the mechanism for generating HTTP "404 Not Found" error responses. If it detects a special pattern in a GET request that would otherwise have produced an error, it sends back an encrypted NET, that we call a *NET payload* and describe below. Note, while we trap "404 Not Found" errors, the actual responses are encrypted NETs sent back as "200 OK" messages. The random-looking encrypted strings that are part of both the request and response must be steganographically encoded to resemble legitimate requests and responses (a discussion of steganography strategies is outside the scope of this paper). An alternative design choice is to simply use the cookie and set-cookie fields in the HTTP header.

Modules may be added to any compatible Apache server without rebuilding the entire server; this facilitates deployment. By inserting our protocol into what would otherwise have been error messages, we ensure that our module does not interfere with the normal operation of the hosting website. By building on the most popular web server in use today, our rendezvous protocol can be deployed on a very large number of hosts, rendering it infeasible to block all rendezvous service.

3.2 DEFIANCE Rendezvous Requests and Responses

The DEFIANCE User Agent uses a `mod_freedom` server's URL to construct a GET request to that server for an image, with a message to `mod_freedom` embedded in the URL. This message contains the actual request and a symmetric key with which to encrypt the reply, all encrypted using the public key (which is simply the server's name) of the server using the Boneh-Franklin IBE crypto-algorithm [1]. The choice of IBE here is important, because it eliminates the need for a method of distributing `mod_freedom` public keys.

The DEFIANCE User Agent is preconfigured with public keys for particular sites that implement `mod_freedom`. Using these keys, it crafts a GET request for an image, with a message to `mod_freedom` embedded in the URL. This URL is designed to appear relatively natural (although full of hexadecimal numbers), but be unlikely to correspond to real content. It contains a request for a NET payload, and a key with which to encrypt the reply, all encrypted with the appropriate public key.

Upon receiving this request, `mod_freedom` does reply with an image; however, steganographically embedded in this image is a NET payload, encrypted using the short-lived key that was embedded in the request. The user agent extracts the NET from the payload and guides the user through the remainder of the rendezvous process.

3.3 Armored NET Payloads

The adversary can potentially harvest NETs either directly from the NET factory by masquerading as a `mod_freedom` server, or by connecting to lots of rendezvous servers. We impose rate limits and credential checks at both levels, but it is still plausible that the adversary can build up a fairly large list in a short time. Therefore, NETs are encapsulated in *payloads* that require significant human and computational effort to decode. Therefore, an adversary who has obtained many of them should not be able to decode them faster than we can change gateway addresses.

Figure 2 illustrates the structure of a NET payload. Each layer (nested box in the diagram) requires the user, or the user-agent, to carry out some task in order to decrypt the next layer. Currently, there is a fixed sequence of three layers surrounding the NET payload.

The outermost layer is just the transport encryption done by `mod_freedom`, and not really considered part of the payload. To remove this layer, the user-agent must reverse the steganography and decrypt the "image" revealing the actual NET payload. The first actual payload layer contains the digital signature of the NET factory and must be verified to ensure that the payload was not served by a malicious `mod_freedom` server. The second layer is "proof of life:" presently, a CAPTCHA-style image of a decryption key that the user must read and type in. Finally, the third layer is "proof of work:" a computational puzzle that can be decrypted only with a small (but nontrivial) amount of CPU time. Under all these layers is the NET provisioning (described in section 4), and potentially an updated list of `mod_freedom` servers for the client to use next time.

The payload format is extensible, allowing additional layers or changes of the tasks in the future, should this be necessary to maintain harvesting resistance.

4 Address-Change Signaling

Address-Change Signaling (ACS) securely inhibits an active scan by network censors seeking to locate, identify, and/or block DEFIANCE resources. To avert scanning and avoid blocking, it is necessary that the IP addresses used by servers, proxies, and tunnels vary frequently. An inventory of IP address assets are rotated in and out of use continuously.

Each DEFIANCE user must communicate with several IP addresses in a specific sequence. This ensures that a given incoming connection came from somebody possessing the NET provisioning details and knowing the expected protocol to access each address. A network censor having only the suspicion that some block of IP addresses are used for censorship bypass (but not possessing complete NET provisioning details) will find only normal commodity servers or empty space.

NET provisioning is provided to the user via the Rendezvous Protocol. These provisioning details allow the DEFIANCE User Agent to perform a multi-step address-

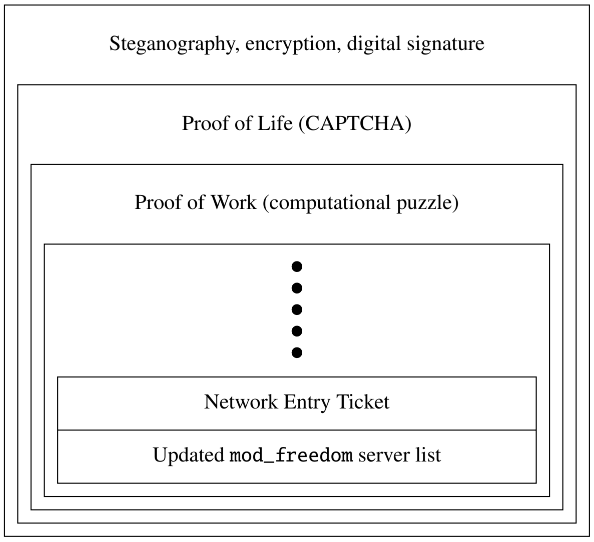
Steganography, encryption, digital signature
Proof of Life (CAPTCHA)
Proof of Work (computational puzzle)
Network Entry Ticket
Updated mod_freedom server list

Figure 2: Structure of NET Payloads delivered by mod_freedom servers

```
{
   "initial" : "192.0.2.11",
   "redirect" : "192.0.2.22",
   "wait" : 95,
   "window" : 112,
   "passphrase" : "OUST COAT FOAL MUG BEAK TOTE"
}
```

Figure 3: Provisioning information in JSON format.

change signaling "dance". Upon successful completion of the dance steps, the user application is communicating with an otherwise hidden DEFIANCE Bridge.

Details of the Address Pool Registration Protocol (APRP) for management and synchronization are beyond the scope of this paper. Only relevant details of NET provisioning are described, together with steps the DEFIANCE User Agent must complete.

4.1 ACS NET Provisioning

Each user (or small group of users) is provided with a pair of contact addresses, a pair of timing values in seconds, and a secret passphrase. Figure 3 shows an example, formatted in JSON [3].

Although there is no expiration field, the validity of this contact information is limited in time. The distribution mechanism may also request a delay for activating the contact addresses, or a longer period of validity. By default, these contacts will be available immediately, and valid for 24 hours.

Both (*initial* and *redirect*) contacts are standard string forms for IP addresses. These addresses should not be Domain Names for the actual servers, as this could leave an audit trail, and/or permit an adversary to modify DNS values in transit.

URL scheme names are not included in the contact strings. When using the address, the chosen URL scheme name must be appropriate to the expected contact. The default implementation uses "http://" and "https://" for the Initial Contact and Redirect Contact, respectively. In the future, the Redirect scheme could also be "http://" with a suitable camouflage protocol that provides confidentiality, mutual authentication, and Perfect Forward Secrecy.

Timing (*wait* and *window*) delays serve as a "security by obscurity" safeguard against detection. The values determine timing between contacts, allow sufficient time for contact to occur, and provide a time limit on replays. These timing parameters are different for each user (or small group of users) sharing the contact address pair.

The *passphrase* uses the [RFC2289] One-Time Password System dictionary. This provides a 64-bit secret key with an additional 2 bits of checksum. The passphrase prevents adversaries from indirect discovery of the ephemeral Bridge Relay(s). An interloper who can intercept or modify a TLS connection will fail the final verification.

4.2 ACS "Dance" Steps

A DEFIANCE User Agent begins by contacting the *initial* address using HTTP. A time-dependent HTTP Set-Cookie is returned to the user agent, along with innocuous content.

The user agent delays for *wait* seconds, then contacts the *redirect* address using HTTPS. The time-dependent verification key used for the ciphersuite is generated from the *passphrase* string and the HTTPS Hello random (timed) values. The encrypted response contains a set of bridge relays, each having an expiration, an identity, a secret, a TCP port, and an IP address.

At each step of this dance, the APRAdb maintains state about the potential dance-in-progress. Connections coming to the wrong address, or outside the allowed *wait* and *window* delays, or using the wrong verification key, are answered innocuously (or not answered at all).

4.3 ACS Implementation

ACS is comprised of several separate components: the aforementioned APRAdb, one or more NET factories, multiple regional DEFIANCE Gateway daemons (dgw), and a corresponding lightweight Apache module (mod_dgw).

A redundant set of central APRAdb servers maintain a database of the Address Pool(s) used by widely distributed regional DEFIANCE Gateways. PostgreSQL is used for caching of contacts and outstanding listeners in both the APRAdb and dgw.

Each NET Factory and DEFIANCE Gateway (dgw) communicate with the APRAdb using the Address Pool Registration Protocol (APRP). Consolidating the DEFIANCE Gateway logic inside a separate daemon, the dgw process could be run on another machine or serve multiple webservers with variable content from the same machine via different IP addresses. Separating this logic also allows restart of each component without affecting the operation of the others, and avoids having to debug a fully running Apache instance along with the complexities of its multi-processing modules.

We use a custom Apache2 module (`mod_dgw`) to intercept requests to the `dgw` process. Our objective is to make the client-facing server as similar to a normal webserver as possible. The webserver loaded with `mod_dgw` module will perform special handling on ACS requests while the rest of the system is identical to a normal webserver setup. `mod_dgw` installs itself at the head of the Apache internal module handler list. If `mod_dgw` sees an anticipated request, it forwards details of the request to its `dgw`. `dgw` returns further instructions to `mod_dgw`: remain silent, prepare innocuous content, or return specific content.

5 Discussion

The framework as described above is in its infancy. There are many areas for additions and improvements.

Using `mod_freedom` as a rendezvous strategy has several strengths. It is a minimally invasive extension to an existing web server, and should have minimal performance impact. It is easy to install, and requires almost no day-to-day maintenance. It also provides ample techniques to prevent the harvesting of NETs. It does not require mutual trust between DEFIANCE and web server operators. An adversarial web server running `mod_freedom` will be able to harvest only their daily allotment of NET payloads, and will still have to go through the time-consuming process of *proof of life* and *proof of work* to unpack them. Consequently, simply partitioning the NETs according to the supplied web servers allows for the reputation tracking of `mod_freedom` servers themselves.

Some other obvious extensions include mechanisms to associate *reputations* with users and tailor our responses, such as the level of difficulty to unwrap, according to a particular reputation. We can also deliver more than just a NET; for example, for users with a high reputation we could also deliver an additional list of servers running `mod_freedom`.

6 Related Work

Open proxies such as DynaWeb [9] and Ultrasurf [19] that are commonly used for circumvention provide no anonymity guarantees to end users. Proxies also have publicly advertised and relatively stable IP addresses, so it is easy to block them in an address filter. To address this, Köpsell and Hillig proposed these covert proxies as an add-on to their AN.ON service [15]. The Tor Project calls them "bridge relays" and has deployed them extensively [4, 7]. Browser-hosted proxies [11] aim to make so many proxies available that it would be hopeless for a censor to block them all; there is still a global directory, but its entries are highly dynamic and the service is piggy-backed on a cloud-storage service that is so widely used that the censor will hesitate to block it. Keyspace hopping [10] takes a similar approach to the proxy discovery problem, where each proxy only responds to a subset of clients, that possess certain unforgeable information, at any given time. While keyspace hopping considers the client's IP network to be this identifier, our identifier is the NET that is obtained through rendezvous.

Collage [2] is a scheme for steganographically hiding messages within postings on sites that host user-generated content, such as photos, music, and videos. Such strategies are limited in the bandwidth they can support, but might be useful as rendezvous schemes. Telex [22], Decoy Routing [14], and Cirripede [13] take a different approach to address-filtering resistance: TCP streams are covertly "tagged" to request that a router somewhere on the path to the overt destination divert the traffic to a covert alternate destination.

7 Conclusion

The Tor anonymity system is currently subject to entry-point blacklisting and reactive network probing attacks by filtering adversaries. In this paper, we introduced the DEFIANCE framework and described components that address these attacks. Our design is a straightforward extension of Tor and is flexible to accommodate additional Rendezvous Protocols and bridge distribution strategies. We believe that Rendezvous Protocols when combined with large, ephemeral Address Pools offer signficant resilience against entry-point harvesting attacks. In addition, Address-Change Signaling offers potential resilience against active and follow-up probing attacks observed recently. We have prototype implementations of all three components, which we hope to deploy and integrate with Tor client software in the near future.

8 Acknowledgments

We acknowledge helpful comments from Drew Dean, Roger Dingledine, and Andrew Lewman. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center Pacific under Contract No. N66001-11-C-4022. 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 or Space and Naval Warfare Systems Center Pacific. Distribution Statement "A:" Approved for Public Release, Distribution Unlimited.

References

[1] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing, 2003.

[2] S. Burnett, N. Feamster, and S. Vempala. Chipping Away at Censorship Firewalls with User-Generated Content. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[3] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, 2006.

[4] R. Dingledine. Behavior for bridge users, bridge relays, and bridge authorities. Tor Proposal #125, 2007.

[5] R. Dingledine. Iran blocks Tor; Tor releases same-day fix. Tor Project official blog, 2011.

[6] R. Dingledine. Obfsproxy: the next step in the censorship arms race. Tor Project official blog, 2012.

[7] R. Dingledine and N. Mathewson. Design of a blocking-resistant anonymity system. Technical report, The Tor Project, November 2006.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[9] Dynamic Internet Technology Inc. DynaWeb. Proxy service, 2002.

[10] N. Feamster, M. Balazinska, W. Wang, H. Balakrishnan, and D. Karger. Thwarting Web Censorship with Untrusted Messenger Discovery. In *Privacy Enhancing Technologies*, 2013.

[11] D. Fifield, N. Hardison, J. Ellithorpe, E. Stark, R. Dingledine, P. Porras, and D. Boneh. Evading Censorship with Browser-Based Proxies. In *PETS*, 2012.

[12] Hooman Mohajeri Moghaddam and Baiyu Li and Mohammad Derakhshani and Ian Goldberg. Skypemorph: Protocol obfuscation for tor bridges. Technical report, University of Waterloo, 2012.

[13] A. Houmansadr, G. T. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, pages 187–200, 2011.

[14] J. Karlin, D. Ellard, A. Jackson, C. E. Jones, G. Lauer, D. P. Makins, and W. T. Strayer. Decoy routing: Toward unblockable internet communication. In *USENIX Workshop on Free and Open Communications on the Internet*, 2011.

[15] S. Köpsell and U. Hillig. How to Achieve Blocking Resistance for Existing Systems Enabling Anonymous Web Surfing. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 47–58, 2004.

[16] N. Mathewson. Tor and Circumvention: Lessons Learned. Invited talk at the 4th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), 2011.

[17] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge: Cambridge University Press, 1995.

[18] Runa. An update on the censorship in Ethiopia. Tor Project official blog, 2012.

[19] UltraReach Internet Corp. Ultrasurf. Proxy service, 2001.

[20] 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 19th ACM conference on Computer and Communications Security*, 2012.

[21] T. Wilde. Knock Knock Knockin' on Bridges' Doors. Tor Project official blog, 2012.

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