Conscript Your Friends into Larger Anonymity Sets with JavaScript

Henry Corrigan-Gibbs*
Stanford University
henrycg@stanford.edu

Bryan Ford
Yale University
bryan.ford@yale.edu

ABSTRACT

We present the design and prototype implementation of ConScript, a framework for using JavaScript to encourage casual Web users to participate in an anonymous communication system. When a Web user visits a cooperative Web site, the site serves a JavaScript application that instructs the browser to create and submit "dummy" messages into the anonymity system. Users who want to send non-dummy messages through the anonymity system use a browser plug-in to replace these dummy messages with real messages. Creating such *conscripted anonymity sets* can increase the anonymity set size available to users of remailer, e-voting, and verifiable shuffle-style anonymity systems. We outline ConScript's architecture, we address a number of potential attacks against ConScript, and we discuss the ethical issues related to deploying such a system. Our implementation results demonstrate the practicality of ConScript: a workstation running our prototype ConScript JavaScript client generates a dummy message for a mix-net in 81 milliseconds and it generates a dummy message for a DoS-resistant DC-net in 156 milliseconds.

Categories and Subject Descriptors

K.4.1 [**Computers and Society**]: Public Policy Issues—*privacy*; C.2.0 [**Computer-Communication Networks**]: General—*security and protection*

Keywords

anonymity; conscripted; traffic analysis; dummy messages

1. INTRODUCTION

Although anonymity systems based on verifiable shuffles [27] and delayed message forwarding [8] may offer strong privacy guarantees, the high end-to-end latency that these systems impose makes them relatively unpopular. Thus, users of Mixmaster [14], Mixminion [8], and other such systems may enjoy strong anonymity, but only amongst a small number of users. In many real-world situations, being an anonymous within a small set of users is almost as bad as having no anonymity at all, especially since surveillance agencies may give extra scrutiny to an anonymity system's encrypted traffic flows [17]. In contrast, low-latency anonymity systems such as Tor [9], have relatively large user bases but provide no protection against ISP-level adversaries [16]. A whistleblower trying to "leak" documents anonymously is left to choose between unpopular anonymity systems with relatively strong security properties and more popular systems which are vulnerable to low-cost traffic-analysis attacks.

To help increase the size of anonymity sets in strong anonymity systems—and thus to make these systems more useful in practice—we propose forming *conscripted anonymity sets* using JavaScript. Our framework, called ConScript, is compatible with a number of anonymity systems, so we describe the high-level ideas in the context of a generic anonymity system. Later on, we discuss how to apply the generic framework to popular anonymity systems.

Like AdLeaks [22], an independently developed architecture focusing on document leaking, ConScript leverages JavaScript programs served by Web servers to increase anonymity set size. In contrast with AdLeaks, ConScript is compatible with existing anonymity systems, ConScript offers some protection against active attacks by malicious insiders, and ConScript avoids the problem of message collisions and the need for error-correcting codes in AdLeaks.

In ConScript, cooperative Web servers host a JavaScript application containing the anonymity system's client code. Whenever a user browses to a cooperating Web site, the JavaScript application instructs the user's *browser* to function as a client of the anonymity system. The browser performs the encryption and processing necessary to create a "dummy" client message, then it submits the message to the underlying anonymity system via the `XMLHttpRequest` mechanism. To avoid enlisting users against their will, the Web server may obtain the explicit consent of Web users before running the ConScript JavaScript.

Actual users of the anonymity system, who want to send messages anonymously through the system, use a browser plug-in to intercept the JavaScript client's dummy message and replace it with a real message for the anonymity system. If these "real" messages are indistinguishable from the conscripted user's "dummy" messages, the effective number of participants in the anonymity system (from the perspective

*Work conducted while author was a staff member at Yale University.

WPES'13, November 4, 2013, Berlin, Germany.
ACM 978-1-4503-2485-4/13/11.
http://dx.doi.org/10.1145/2517840.2517866.

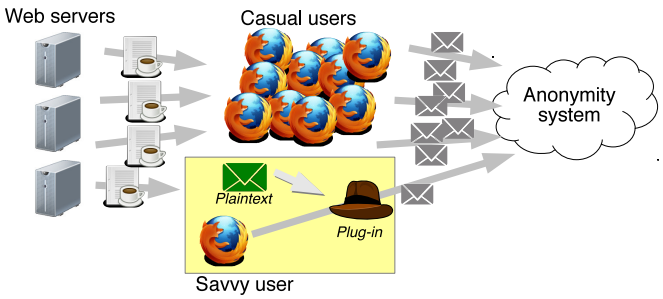

Figure 1: Overview of ConScript's system architecture.

of an adversary) is equal to the number of honest real users *plus* the number of conscripted users. Thus, actual users can hide amongst a much larger set of casual users browsing the Internet. Careful construction of the plug-in protects against arbitrarily malicious Web servers (who try to distinguish real users from conscripted users), eavesdroppers, and adversarial clients.

When paired with a compatible anonymity system, using ConScript *can only increase* the anonymity given to a particular client. In the worst case, the anonymity that a client gets is equal to the total number of real users—i.e., *no worse* than it would have been without using ConScript.

To demonstrate the practicality of ConScript, we have implemented proof-of-concept JavaScript clients for *two different* underlying anonymity systems: a mix-net and a DoS-resistant client/server DC-net [7]. These proof-of-concept applications are not wire-compatible with the corresponding deployed systems—we use them only to approximate the performance of the client application in a deployed system.

Section 2 outlines ConScript's architecture. Section 3 describes how ConScript defends against a number of possible attacks. Section 4 addresses ethical issues related to conscripted anonymity, Section 5 summarizes the results of our implementation and evaluation, Section 6 describes related work, and Section 7 concludes.

2. ARCHITECTURE

2.1 Participants

ConScript's architecture consists of three components: an *underlying anonymity system*, a number of *cooperative Web servers*, and *Web users*. Figure 1 provides a pictorial representation of the interaction of the system's participants.

Anonymity system nodes. At the core of ConScript's architecture is a pre-existing anonymous messaging system. ConScript is, in principle, compatible with a number of existing anonymity systems, but to make the system design concrete, we will first describe how to use ConScript with a mix-net consisting of *single* cascade of timed mixes [4].

A mix cascade consists of a set of M dedicated mix servers, with each server i having a well-known public key pk_i. Over the course of a day (or some other time period), each of N users submits a fixed-length message to a *message pool* hosted by the first mix-server. Users serially encrypt each message m with each of the servers' M public keys:

$$E(pk_1, \ldots E(pk_M, m) \ldots)$$

At the end of the day, the first server shuffles the set of ciphertexts in the message pool, removes duplicate messages, decrypts a layer of encryption, and forwards the messages to the second server. This process continues until the last server holds the N plaintext messages in permuted order. At this point, the last server could, for example, post the anonymized messages on a public bulletin board. If at least one server does *not* collude with the others, then an honest sender's message is anonymous amongst the $h \leq N$ honest senders (provided that we ignore the possibility of active attacks by the mix servers).

An important property of a timed cascade mix-net, and similar systems, is that the anonymity of a single sender *increases monotonically* with the number of other senders who have submitted a message to the system. That is, the anonymity set size of a given message in the *timed* mix-net system *never decreases* when an additional sender adds a message to the message pool.

ConScript takes advantage of the monotonicity of the anonymity in mix-net-style systems. Since adding more users to an anonymity system cannot decrease a user's anonymity, it does not hurt to gather messages from conscripted and unauthenticated senders and insert them into the anonymity system.

Web servers. ConScript requires the cooperation of a number of Web servers, each of which serves a ConScript JavaScript application to its Web clients alongside the Web content it would normally serve. Individuals and organizations interested in supporting Internet privacy or in protecting whistleblowers might agree to embed the ConScript JavaScript application in their Web sites. For example, the Electronic Frontier Foundation, Wikipedia, and the Guardian newspaper might each agree to serve the ConScript client script to users visiting their Web sites.

The JavaScript application, embedded in a normal Web page, contains the client code for the anonymity system in use. For example, if the anonymity system were a mix-net, the JavaScript code would contain everything that a user would need to generate a *dummy* message for the mix-net.

In the case of a mix-net, the JavaScript would contain the public keys of the M servers, a method to generate a dummy plaintext message of the correct length, and the public-key encryption routines for encrypting the plaintext message with each of the M public keys.

All clients' *plaintext* messages in the mix system must begin with a single control bit indicating whether the message is a dummy message (`control_bit=0`) or a real message (`control_bit=1`). In this way, the recipient of a message can determine whether to discard the message as a dummy or process it as a real plaintext. The client-side ConScript JavaScript application encrypts the control bit alongside the clients' plaintext message. (We describe how users send non-dummy messages into the anonymity system later in this section.)

We assume that the mix-net uses an IND-CCA2-secure public key encryption scheme [21] to ensure that the encrypted ciphertexts leak no information about the plaintext even under an adaptive chosen ciphertext attack. In particular, the encryption of a dummy message (a string of zeros) and the encryption of a real message (an arbitrary string beginning with a control bit set to one) must be indistinguishable under an adaptive chosen ciphertext attack. A number of standard cryptosystems, including RSA-OAEP, provide IND-CCA2 security under standard hardness assumptions [11]. All IND-CCA2-secure cryptosystems use

randomized encryption routines, so the ciphertexts generated by encrypting two dummy plaintexts (strings of zeros) will be different with overwhelming probability.

The JavaScript application also contains the code to submit the final message to the first mix server using the `XMLHttpRequest` API. This HTTP request will be a *cross-origin request*, since the script issuing the `GET` request was served by the cooperative *Web server*, but the target of the `GET` request is the *mix server*. The "same-origin policy" would normally prevent the Web browser from communicating directly with the mix servers, but the mix nodes can include the `Access-Control-Allow-Origin` header in their HTTP response to allow this sort of cross-origin request [15, 26].

Casual users. Casual users, the first class of ConScript users, are normal Internet users using standard JavaScript-enabled Web browsers with no special browser extensions or modifications.

When a casual user visits the target Web page (hosted by a cooperative Web server as described above), the user's browser will download the page content, which includes the ConScript client JavaScript application.

Once the casual user's browser downloads the ConScript client application, the script will cause the browser to send mix-net-encrypted dummy messages into the anonymity system. In practice, the script might ask for the user's permission before starting to send messages into the anonymity system (see Section 4). After the script begins running, the casual client will not notice any out-of-the-ordinary behavior, except for perhaps a slight drop in browser performance due to the computational burden of generating the mix-net messages.

In this way, the casual user becomes a client of the underlying anonymity system without needing to download a browser extension or install any software tools. Casual users need not know how to generate a public key, install a program, or configure anonymity system client software. Since the casual users' browsers submit genuine client messages into the anonymity system, these casual users are (with caveats explained in Section 3) indistinguishable from real users, from the perspective of an eavesdropping adversary.

Savvy users. Savvy users, the second class of ConScript users, send non-empty messages through the anonymity system, as opposed to the dummy messages that the casual users send. The only difference between a casual user and a savvy user is that every savvy user has a plug-in installed in their Web browser that monitors the browser's outgoing HTTP requests.

To send a message through the anonymity system, the savvy user enters their secret message into the plug-in and then browses to a cooperating Web site. When a savvy user visits a Web page that contains our system's JavaScript client, the plug-in will transparently *intercept* the outgoing message from the browser and will replace the dummy message (generated by the JavaScript client) with a content-carrying message generated by the plug-in.

The effectiveness of ConScript relies on the difficulty of distinguishing savvy users from casual users, so we take a number of steps to prevent side-channel and equivocation attacks that would allow a malicious Web server or an eavesdropper to identify which users have the plug-in installed. Section 3 describes these defenses.

2.2 Trust Assumptions

We say that a participant is "honest" if it executes operations as the system design dictates and if it does not collude with other nodes or an external adversary.

To guarantee that using ConScript does not reduce the level of anonymity provided by an underlying system, we must assume that honest savvy ConScript users can access *at least one* honest ConScript-enabled Web server. This Web server must, in turn, be able to communicate with the underlying anonymity system. If savvy clients cannot access any cooperative Web server, then these clients cannot submit their messages to the underlying anonymity system, and the anonymity set size provided will be smaller than the total number of savvy users.

To guarantee that using ConScript has the potential to *increase* the anonymity set size of the underlying anonymity system, we must additionally assume that at least one honest casual client must be able to communicate with at least one honest Web server. This honest Web server must be able to communicate with the underlying anonymity system.

We make no additional assumptions about user or server behavior, though the underlying anonymity system might require additional trust assumptions over and above those we must make. For example, a mix-net cascade requires that at least one of the mix servers is honest.

2.3 Underlying Anonymity System

ConScript is compatible with any anonymity system that has a certain set of properties, which we enumerate below.

Anonymity set size is monotonic w.r.t. users. The anonymity provided to a particular user of the anonymity system must *increase monotonically* with the number of total users of the system. If the anonymity of a particular user can *decrease* when the system has more (potentially adversarial) users, then conscripting many users into the anonymity system might actually *hurt* the anonymity of the system overall.

Simulatable traffic streams. It must be possible to simulate the behavior of a real user such that the behavior of the simulated user and the real user are indistinguishable from the perspective of an adversary. For example, the mix-net client simulator generates an onion encryption of a string of zeros using an IND-CCA2-secure cryptosystem.

Easy to identify malformed messages. The anonymity system should be able to identify and reject malformed user messages (to prevent a malicious user from disrupting communication).

Messages do not depend on the set of active users. To submit a message to the anonymity system, a user should not need to know the identities of the system's other users. A traditional DC-net [5], for example, would not be suitable because it requires every user to share a secret with every other user of the system. In contrast, the client/server DC-net we use only requires users to know the public keys of the system's servers [7].

2.4 Compatible and Incompatible Anonymity Systems

We now briefly describe which anonymity systems are compatible with ConScript.

Yes: Timed Cascade Mixes and Verifiable Shuffles. Timed cascade mixes (introduced in Section 2.1) and verifiable shuffles [3, 19] satisfy all properties necessary to be compatible with ConScript.

Probably Yes: Anonymous Remailers. Any anonymous remailer using fixed-length messages which has the monotonic anonymity property described above is compatible with ConScript. Not all anonymous remailers exhibit the monotonic anonymity property, however, and determining whether or not a particular remailer system has this property is not necessarily straightforward in general. To create a dummy message for an anonymous remailer, the client JavaScript application follows a process very similar to the process used to create a dummy message for the mix net (described earlier in this section). We discuss other design issues related to using remailers in Section 5.

Probably No: Tor. It is not clear whether any anonymity gain would result from conscripting users into the Tor network. If a Tor user picks a route through the Tor network with an adversarial first and last node, that user's anonymity is compromised. If some fraction of the relays in the Tor network are dishonest, adding more clients to the Tor network *does not* change the probability that a new circuit selected through the network will begin and end at an adversary-controlled relay [25]. Conscripting users into the Tor network might actually *harm* real users, since the conscripted users would consume Tor's scarce network resources with dummy messages, leaving less network bandwidth for real users' messages.

2.5 Effective Anonymity Set Size

Consider a deployment of ConScript that has j (honest) savvy users and k (honest) casual users, whose messages reach the anonymity system servers. ConScript provides a level of anonymity equivalent to the level that would be offered by the underlying system when run with j real users plus k users who submit "dummy" messages into the system. In the simple cascading mix-net we described earlier, the effective anonymity set size would be $j + k$.

3. ATTACKS AND DEFENSES

In this section, we consider possible attacks against ConScript, leaving attacks against the underlying anonymity system out of scope.

3.1 Malformed JavaScript

An adversarial Web server might modify the JavaScript it sends to the client in an attempt to distinguish savvy users (who have the conscripted anonymity set browser plug-in installed) from casual users (who do not have the plug-in). The extended version of this paper [6] shows a sample of malicious JavaScript that a malicious Web server might send to try to distinguish savvy users from casual users.

To defeat these attacks, the browser plug-in must *only* replace the dummy message with the savvy user's real message when the JavaScript on the relevant page *exactly matches* the JavaScript that the plug-in expects to see. To perform this check, the browser plug-in must have a copy of the JavaScript code that it expects the Web server to send. If the Web server sends JavaScript code that does not match the expected code, the plug-in should simply run the served JavaScript as a casual user would.

The requirement that the plug-in have a copy of the Web server's expected code is somewhat burdensome—it restricts the type of content that the Web server can serve alongside the ConScript JavaScript application. A modern Web page often has tens of linked scripts, `iframe`s, Flash movies, and other dynamic content but a malicious Web server could exploit any of these objects running alongside the JavaScript client to mount a distinguishing attack. Preventing the distinguishing attack requires the plug-in to have a copy of *all* content on the page (except for static text and images). In this way, the plug-in can detect when the server has served malicious or incorrect JavaScript to the user.

3.2 Selective Denial of Service

A malicious Web server could try to distinguish savvy users from casual users by selectively denying service to users of to the anonymity system, in an attack analogous to the *trickle attack* against mix-nets [23]. For example, if the attacker wants to determine if a particular user is a savvy user, the Web server could serve incorrect JavaScript to all users except a particular target user and a set of users that the attacker controls. If the anonymity system outputs a real message when fed messages from only the target user as input, then the malicious Web server both learns that the target user is a savvy user *and* the attacker learns the content of the savvy user's message.

One technique to prevent such attacks is to maximize the number of cooperating Web servers serving the JavaScript application. Every savvy client could visit a number of Web servers (instead of just one) to ensure that a single malicious Web server cannot block communication between users and the anonymity system. Other techniques for preventing the trickle attack could also apply here [23].

3.3 Dangers of Cryptography in the Browser

The application that generates casual clients' dummy messages must implement public-key cryptography algorithms in JavaScript. (In contrast, the savvy clients' messages are constructed by the browser plug-in, which presumably can access standard cryptography libraries.)

The Web browser environment is not the ideal place to run cryptographic software: many browsers do not offer a source of cryptographic randomness, it is difficult to prevent side-channel attacks in the browser—perhaps mounted by a script running in another tab—and client-side cryptography libraries are less mature than their server-side counterparts. Even so, these limitations may not be fatal. In the worst case, a flaw in or a side-channel attack against the client-side cryptography library will allow an adversary to distinguish the savvy from the casual clients but such an attack will *not* allow the adversary to read the savvy clients' messages or to otherwise violate the anonymity of the underlying system.

Since the savvy clients' messages are generated using the browser plug-in, and since modern Web browsers allow plug-ins to execute native code through the Netscape Plugin Application Programming Interface (NPAPI), the savvy clients' messages will be encrypted using cryptographic routines provided by conventional cryptography libraries (e.g., GPG). Thus, in the worst case of an adversary who can distinguish all casual clients from savvy clients, a savvy client will be still be anonymous amongst the set of savvy clients, all of whom use the plug-in to encrypted their messages.

3.4 Other Attacks

The extended version of this paper [6] describes how a ConScript deployment could defend against a number of other possible attacks.

4. ETHICAL ISSUES

Up to this point, we have considered only the technical questions related to the deployment of ConScript, but we have not addressed the equally important ethical issues that deployment of such a system would raise. The fundamental question is whether it is ethical to "conscript" an unsuspecting Web user into participating in an anonymity system without the user's consent. Instead of trying to resolve this ethical question here, we will outline three possible deployment scenarios of ConScript (with varying levels of "conscription") and we will make an ethical argument for each.

User opt out. One possible way to deploy ConScript would be to require the cooperating Web server to display a conscripted anonymity "badge" on any Web page that serves the ConScript JavaScript client. Web users could "opt out" of participation in the ConScript by clicking the badge. FlashProxies [10], a system for using Web browsers for censorship circumvention, takes this approach.

A utilitarian argument in favor of an "opt-out" approach is that the total social benefit of ConScript is much greater than the total social cost of conscription to the unsuspecting Web users. This argument would be most persuasive in areas where the probable risk to a conscripted Web user is low but where the social benefit of anonymous communication is high. For example, in a country with a judicial system that would not imprison a conscripted Web user just for being conscripted, and with an invasive Internet surveillance regime, an "opt-out" policy might be the most ethical one.

User opt in. Another possible deployment strategy would be to require the explicit consent of Web users before conscripting their browsers into the anonymity system. For example, a pop-up window appearing after the Web page loads could explain the anonymity system to the user, including the potential effects on the user's bandwidth and power usage, and then ask whether the user wants to participate.

An ethical argument in favor of an "opt-in" deployment strategy is that a Web user should have the choice of whether or not to participate in ConScript, especially if participation could consume the user's bandwidth or drain the battery on the user's mobile device. Giving the user a choice to opt in to the network allows those users who want to participate the option to do so, but does not force participation in an anonymity system on those who do not. If users could be imprisoned without trial for being suspected of using an anonymity system, then an opt-in strategy might be the most ethical one.

Unethical even with opt in. Yet another ethical position is that deploying conscripted anonymity is not ethical under any circumstances, even when using an opt-in mechanism. One argument supporting this position is that the risks of being conscripted into an anonymity system might not be clear to a novice Web user, even after the user is presented with a description of the conscripted anonymity system. If the user does not have the technical understanding to make an informed decision about the risks of opting in to the system, then it might not *ever* be ethical to offer users the option to participate in ConScript.

ConScript is compatible with both "opt out" and "opt in" policies, so the decision of which policy to use can be left to the organization deploying such a system. Different policies will be appropriate in different societal contexts. In contexts where the risks to conscripted users are high, using ConScript may not be appropriate at all.

5. IMPLEMENTATION AND EVALUATION

To evaluate the performance of ConScript, we have implemented the JavaScript client applications for two anonymity systems: a timed cascade mix-net and a verifiable client/server DC-net. Our prototypes perform all of the cryptographic operations that a full-featured JavaScript client would perform but they do not yet produce messages that are wire-compatible with the underlying anonymity systems.

To be useful, the ConScript JavaScript client must be able to produce at least one dummy message before the user browses away to another page. One recent survey of Web usage finds that the median time spent on a Web page is 11 seconds [13], so the JavaScript application should generate at least one dummy message every few seconds.

We tested each prototype on four platforms: a modern Linux workstation (Ubuntu 13.04), a Mac laptop (Mac OS 10.6), an iPhone 4 (iOS 6), and a Motorola Android phone (Android 2.2). We tested the JavaScript client on each device using the Chrome 26, Firefox 21, Safari 5, and Opera Mobile 12 browsers, where available.

Mix-net and Remailer. To simulate the casual user workload for a mix-net or anonymous remailer, we created a JavaScript application that encrypts a 256-byte message with five layers of RSA-2048 public key encryption using the OpenPGP.js JavaScript library [20]. Table 1 presents results for the mix-net client. Our results demonstrate that even a CPU-limited mobile device can generate a mix-net dummy message in fewer than 10 seconds, which is less time than the median time spent on a Web page (11 seconds, as explained above). The extended version of this paper [6] sketches a method for distributing the mix servers' public keys to ConScript users.

Verifiable DC-net. To simulate a casual user's workload when using a more computationally intensive anonymity system, we implemented the client functionality for a verifiable DC-net [7, 12] using the Stanford JavaScript cryptography library [24]. Our evaluation uses the NIST P-256 elliptic curve group [18] and requires the client to generate a 32-byte dummy message. The performance results (Table 1) for the verifiable DC-net suggest that the ConScript JavaScript application is arguably practical on faster machines, since both the workstation and laptop were able to generate a dummy message in less than four seconds each. Performance on the CPU-limited iPhone is less impressive—generating a single message took at least 30 seconds.

6. RELATED WORK

The FlashProxy system [10], which was one of the inspirations for this work, uses a JavaScript application to coerce Web users into serving as bridges into the Tor anonymity network [9]. Every additional Web browser that runs the FlashProxy application increases the *access* to the Tor network in regions where Tor relays are blocked. In contrast, every additional Web browser that runs the ConScript JavaScript application increases the *anonymity* available to users

		Mix-net	DC-net
Workstation	Chrome	81	156
Intel W3565 3.20 GHz	Firefox	73	1,781
Laptop	Chrome	133	231
Intel Core 2 Duo 2.53 GHz	Firefox	171	3,062
	Safari	669	3,338
Apple iPhone 4	Chrome	9,009	62,973
Apple A4 (speed unknown)	Safari	7,280	32,972
Motorola Milestone	Opera	†	63,504
ARM Cortex-A8 600 MHz			

† Opera Mobile does not support the `getRandomValues` API required by OpenPGP.js.

Table 1: Time (milliseconds) to generate a ConScript "dummy" user message in JavaScript using different hardware and browser combinations.

of the anonymity system. Anonymity systems have used *dummy messages* in the past to deter traffic analysis attacks [2, 14]. However, these systems use dummy messages only *inside of* the anonymity network and they do not have "dummy users" send messages into the system to increase the effective number of total users of the system.

Bauer [1] describes a system for using a specially crafted banner ads served to an unwitting user's Web browser to create a covert channel between two servers in a mix network. Bauer considers only *passive* adversaries, whereas we consider *active* adversaries that also can monitor all network traffic. Since ConScript considers a stronger threat model, we address a number of security issues in Section 3 that Bauer's work did not consider.

AdLeaks [22]—a system design published independently while this work was in preparation—uses JavaScript served by online advertising networks to conscript users into participation in an anonymity system. Unlike ConScript, which is general and compatible with a number of existing anonymity system, AdLeaks conscripts users only into AdLeaks' own anonymity system. In addition, the AdLeaks anonymity system is *not* designed to prevent active attacks by dishonest participants in the system, whereas ConScript can protect against such attacks.

7. CONCLUSION

We have presented ConScript, general architecture for conscripted anonymity, we discuss a number of attacks against ConScript (and possible defenses), we consider ethical issues of deploying such a system, and we implement and evaluate a proof-of-concept prototype on a variety of devices. ConScript can increase the user base of formally analyzable, but unpopular, anonymity systems, which allows the few security-sensitive users of these systems to hide amongst a larger group of casual Internet users.

Acknowledgements

We would like to thank David Fifield, David Wolinsky, and the anonymous reviewers for their helpful comments on the draft. This material is based upon work supported by the Defense Advanced Research Agency (DARPA) and SPAWAR Systems Center Pacific, Contract No. N66001-11-C-4018.

8. REFERENCES

[1] M. Bauer. New covert channels in HTTP: adding unwitting Web browsers to anonymity sets. In *WPES*, pages 72–78, 2003.

[2] O. Berthold and H. Langos. Dummy traffic against long term intersection attacks. In *PET*, 2002.

[3] J. Brickell and V. Shmatikov. Efficient anonymity-preserving data collection. In *KDD*, pages 76–85, Aug. 2006.

[4] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, Feb. 1981.

[5] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, pages 65–75, Jan. 1988.

[6] H. Corrigan-Gibbs and B. Ford. Conscript your friends into larger anonymity sets with JavaScript (extended version). `http://arxiv.org/abs/1309.0958`, Sept. 2013.

[7] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively accountable anonymous messaging in Verdict. In *USENIX Security*, Aug. 2013.

[8] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III anonymous remailer protocol. In *IEEE SP*, pages 2–15, May 2003.

[9] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *USENIX Security*, Aug. 2004.

[10] D. Fifield, N. Hardison, J. Ellithorpe, E. Stark, D. Boneh, R. Dingledine, and P. Porras. Evading censorship with browser-based proxies. In *12th PETS*, pages 239–258, 2012.

[11] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In *CRYPTO*, pages 260–274. Springer, 2001.

[12] P. Golle and A. Juels. Dining cryptographers revisited. *Eurocrypt*, pages 456–473, May 2004.

[13] R. Kumar and A. Tomkins. A characterization of online browsing behavior. In *WWW*, pages 561–570, 2010.

[14] U. Moeller and L. Cottrell. Mixmaster protocol: Version 2. `http://freehaven.net/anonbib/cache/mixmaster-spec.txt`, Jan. 2003.

[15] Mozilla Developer Network. HTTP access control (CORS). `https://developer.mozilla.org/en-US/docs/HTTP/Access_control_CORS`.

[16] S. J. Murdoch and P. Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In *PETS*, pages 167–183, 2007.

[17] E. Nakashima, B. Gellman, and G. Miller. New documents reveal parameters of NSA's secret surveillance programs. June 2013. Washington Post.

[18] National Institute of Standards and Technology. FIPS PUB 186-3: Digital Signature Standard (DSS), 2009.

[19] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *CCS*, pages 116–125, Nov. 2001.

[20] OpenPGP.js. `http://openpgpjs.github.io/`.

[21] C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, pages 433–444, 1991.

[22] V. Roth, B. Güldenring, E. Rieffel, S. Dietrich, and L. Ries. A secure submission system for online whistleblowing platforms. In *FC*, Apr. 2013.

[23] A. Serjantov, R. Dingledine, and P. Syverson. From a trickle to a flood: Active attacks on several mix types. *Information Hiding*, pages 36–52, 2003.

[24] Stanford Javascript crypto library - ECC branch. `https://github.com/bitwiseshiftleft/sjcl/tree/ecc`.

[25] P. Syverson. Why I'm not an entropist. In *17th International Workshop on Security Protocols*, Apr. 2009.

[26] Cross-origin resource sharing. `http://www.w3.org/TR/2013/CR-cors-20130129/`, Jan. 2013.

[27] D. I. Wolinsky, H. Corrigan-Gibbs, A. Johnson, and B. Ford. Dissent in numbers: Making strong anonymity scale. In *OSDI*, pages 179–192, Oct. 2012.