

A Survey of Online Anonymity

Steve Webb

Mini-Project 2 – “Anonymity Issues in Peer-to-Peer Systems”
Advisor: Ling Liu
CS7001

1 Introduction

Over the past few years, online communication has emerged as an increasingly popular means by which individuals communicate and obtain information. Using e-mail, web browsing, newsgroups, and a host of others, individuals are quickly becoming connected to other individuals in new and exciting ways. However, as the world becomes more and more connected by computer networks, the freedoms possessed by individuals offline must also be provided online. Specifically, in order to facilitate the free exchange of ideas online, participants must be given online anonymity to protect their privacy.

Examples abound that illustrate the importance of anonymity in online communication. In fact, many scenarios in offline communication have an analogous situation online. Some examples of these scenarios include anonymous correspondence with support groups, anonymous shopping (e-commerce), anonymous whistle blowing, and anonymous political opposition [12]. Each of these scenarios poses a unique anonymity issue, but they all emphasize the importance of allowing individuals to communicate freely without fear of persecution.

Currently, online communication provides little to no anonymity. Eavesdroppers can easily identify who a user is communicating with, the frequency and duration of their communications, and the length of the messages being sent and received by the user. Encryption is often used to combat this situation, but encryption only obfuscates the content of a user's communication; it does nothing to hide the identity of the user [24]. Thus, special techniques must be employed to provide participants in online communication with the anonymity they possess offline.

This paper presents a survey of current techniques for providing anonymity for online communications. The remainder of this paper is organized in the following manner. Section 2 explains proxies and how they can be used to provide sender anonymity. Section 3 introduces

source rewriting systems, showing how they provide stronger anonymity than proxies at the cost of lower performance. Section 4 summarizes the conclusions.

2 Proxies

In traditional network computing, a client directly accesses a server to accomplish its task. However, over the past few years, a new type of system has emerged in the world of networking. This new system, called a proxy server, resides in between the traditional client and server, creating a new client-proxy-server networking architecture. Proxy servers (proxies) provide a number of performance benefits for networking, but in the context of anonymity, they serve as middlemen for network communications. Thus, in a client-proxy-server network, if Cartman (the client) wants to communicate with Stan (the server), he must first communicate with Pip (the proxy). Pip is then responsible for relaying Cartman's messages to Stan. As a result of this interaction, Stan is unaware of Cartman's existence because all of Cartman's traffic appears to originate from Pip. This situation illustrates how proxies can be used to provide a very simple form of sender anonymity. As long as the sender's identity is not revealed in the messages being sent, only the proxy knows about the sender's existence.

The main advantages offered by proxies are simplicity and ease of use. In most cases, a proxy can be added to a client-server environment with a minimal amount of effort. However, despite their simplicity, proxies possess a number of disadvantages and are susceptible to a wide range of attacks. The biggest weakness inherent in proxies derives from their biggest strength: their centralization of information. Since proxies act as the middlemen in communications between clients and servers, they have knowledge of all communication taking place in the system. Thus, a passive attacker could easily monitor this communication by gaining access to the proxy. Additionally, if an attacker was unable to access the proxy, information could still be obtained by sniffing the proxy's incoming link or through statistical analysis. If an attacker is

able to insert a network sniffer (i.e. a traffic logger) between the client and the proxy, the destination of the client's messages will be revealed and anonymity will be nonexistent. However, this attack can be thwarted by using encryption between the client and server. Then, if the attacker sniffs the traffic, it will be unintelligible, maintaining the client's anonymity. Despite the added protection offered by encryption, proxies are still vulnerable to statistical analysis. Since proxies simply forward messages they receive, an attacker could easily correlate incoming messages with their corresponding outgoing messages, once again eliminating the client's anonymity. Finally, proxies represent a single point of failure for anonymous communication between clients and servers. Thus, if the proxy fails or is shutdown by an attacker, all anonymous communication is effectively halted [20, 6, 12].

The following protocols illustrate how proxies can be used to incorporate sender anonymity into online communications. Each protocol also demonstrates the advantages and disadvantages of using proxies for online anonymity.

2.1 Anon.penet.fi

The anon.penet.fi remailer (commonly referred to as Penet) was a type 0 remailer that provided its users with a minimal amount of sender anonymity for e-mail communication [16, 17, 12, 11]. Penet acted as a simple proxy server that provided a mapping service between real e-mail addresses and addresses it created. Initially, if Cartman wanted to use the service, he provided his real e-mail address (e.g. cartman@comedycentral.com), and Penet generated an anonymous e-mail address (e.g. an123@anon.penet.fi) for him to use. Penet then stored the mapping between these two addresses in its local database. To e-mail Stan, Cartman used his new anonymous e-mail address. When Stan responded to Cartman's anonymous e-mail address, Penet looked up Cartman's real e-mail address in its database and forwarded Stan's message to the real address.

Although the anon.penet.fi remailer was extremely popular (500,000 users at its peak), it was eventually shut down due to an onslaught of subpoenas requesting real user e-mails [17, 12, 11]. Penet's demise is an excellent example of the disadvantages and risks involved with using centralized proxies to provide anonymity. Penet's users were completely reliant upon it to provide anonymity, and when it was shutdown, anonymous communication was halted for those users.

2.2 Anonymizer©

Anonymizer© (<http://www.anonymizer.com>) is a proxy for accessing web sites anonymously; consequently, it provides users with a minimal amount of server anonymity for web browsing. As Martin claims in [16] and [17], the Anonymizer© service is a "Penet-style remailer for HTTP traffic." However, unlike Penet, which was a completely free service, Anonymizer© offers a subscription service in addition to a free service. If Cartman wants to use this anonymous web proxy to view a webpage, he simply connects to the webpage via the proxy. For example, assume Cartman wants to access Stan's webpage, but he doesn't want Stan to know about it. To accomplish this feat, Cartman visits <http://anon.free.anonymizer.com/http://www.stan.com> (where <http://www.stan.com> is Stan's webpage). This action will allow Cartman to view Stan's page, but from Stan's perspective, the Anonymizer© server is accessing the webpage. Thus, Stan is oblivious to the fact that Cartman is viewing his webpage.

Anonymizer© is a very simple and efficient system, but it suffers from the same problems as all other centralized proxies. Unlike Penet, Anonymizer© is still running, but its users are just as vulnerable to exploitation as Penet's users were. If law enforcement officials requested sensitive information from Anonymizer©, they would be obligated to provide it, thus eliminating the anonymity possessed by its users. Additionally, the free version of Anonymizer©

does not protect against the sniffing attacks discussed above (i.e. no encryption is used between the user and the proxy).

2.3 Janus and the Lucent Personalized Web Assistant (LPWA)

Janus is a web proxy designed to provide “anonymous personalized web browsing” to its users. In order to accomplish this goal, Janus tackles the problem of name translation [7]. The name translation problem is a byproduct of personalized browsing. Many of the websites that Cartman visits require him to create an account which results in the creation of a new username and password for each. Thus, in order for Cartman to access these sites, he must remember all of his usernames and passwords. Moreover, most of the sites require Cartman to reveal a valid e-mail address so that they can verify his account and send him personalized updates. Janus alleviates this problem by automatically creating an aliased username, password, and e-mail address for each of the sites Cartman accesses. To allow Janus to generate this information, Cartman provides Janus with his real e-mail address and a secret (i.e. a password) once per Janus session [7]. Then, when Cartman accesses a site, Janus uses these two pieces of information along with the site’s domain name as inputs into the Janus function. Given these three inputs, the Janus function produces an aliased username, password, and e-mail address. For more information about the construction and operation of the Janus function, please consult [7].

Once Cartman has given Janus his e-mail address and a secret, he no longer has to create new usernames and passwords for the various sites that he visits. Instead, he can use the aliased usernames and passwords generated by Janus. In order to access this aliased information, Cartman inputs escape strings which Janus replaces with aliased information (e.g. “\U” for username, “\P” for password, and “\@” for e-mail address) [7]. Once Cartman ends his session with Janus, all information about that session is deleted. Thus, Janus provides more security than other proxies because it does not store its users’ information after they are done using the service. If Cartman wants to use the same aliased usernames and passwords during his next Janus session,

he must provide Janus with the same e-mail address and secret. The Janus function will then generate the same aliased information, and Cartman will be able to benefit from personalized browsing without having to lose his anonymity.

The Lucent Personalized Web Assistant (LPWA) essentially provides the same functionality as Janus with a few additions. LPWA is an HTTP proxy that uses the Janus function to create unique aliased usernames and passwords [8]. Cartman uses LPWA in the same manner he uses Janus, and he is also able to use the same escape strings to input account information into the websites he visits. LPWA also provides an aliased e-mail address for each website Cartman visits. Therefore, if Cartman's aliased username at `http://www.stan.com` is *X*, his aliased e-mail address for that site will be `X@lpwa.com`. Since LPWA knows Cartman's real e-mail address (he gave it to the Janus function when his session began), LPWA is able to send all e-mail sent to `X@lpwa.com` to Cartman's real e-mail address. When LPWA forwards e-mail to Cartman's real e-mail address, LPWA also includes the alias e-mail address in the CC field of the e-mail. This allows Cartman to benefit from personalized browsing, but it also gives Cartman the ability to block all e-mail sent to the aliased e-mail address (Cartman can set a local mail filter for his alias account) [8].

3 Source Rewriting

Nodes within source rewriting systems typically provide anonymity in a number of ways. First, they change the source field of the messages they receive to their own address, helping to anonymize the previous node in the network. This technique is often combined with chaining to increase anonymity and protect against colluding nodes. Chaining is the process of sending messages through many different nodes. Thus, if a message is routed through *n* nodes, all *n* nodes must collude in order to verify the sender of a message [20]. To combat against statistical analysis, source rewriting systems use fixed-length messages so that each message is

indistinguishable from any other message. Additionally, nodes in source rewriting systems reorder the messages they receive to help prevent timing correlations between incoming and outgoing messages. Along with the reordering of messages, dummy traffic (i.e. meaningless messages) is also inserted into the network to help thwart statistical analysis by attackers [9].

Source rewriting systems offer a number of benefits over centralized proxies. Through the use of source field alteration, chaining, fixed-length messages, message reordering, and dummy traffic insertion, these systems are able to provide a stronger level of anonymity than proxies [20]. However, this additional anonymity comes at a cost as each of these operations requires the consumption of resources. The process of changing the source field of a message adds latency to the communication because it prevents nodes from forwarding messages immediately. Chaining requires nodes to use additional bandwidth as they forward messages to other nodes, and additional latency is added to the communication as messages are forced to traverse a number of nodes to reach their destination. Fixed-length messages require nodes to pad or fragment their messages so that they fit within the specified length. These operations require computational resources, and the resulting messages waste bandwidth. If a message is significantly smaller than the specified fixed-length, the extra padding represents wasted data that must be sent across the network. If a message is larger than the specified fixed-length, the message must be fragmented into fixed-length chunks. These fragmented messages waste bandwidth in two ways. First, each fragmented message must carry header information that is similar to the original. This header information is necessary to route the messages, but it is also wasteful. Additionally, unless the original message is perfectly divisible by the specified fixed-length, the last chunk will have to be padded, resulting in more wasted data. As nodes reorder messages, additional latency is added to the communication because nodes are no longer forwarding messages as soon as they arrive. Finally, although dummy traffic is necessary to help combat statistical analysis by attackers, it is a significant waste of bandwidth.

The following protocols showcase a variety of source rewriting systems. Each protocol implements the source rewriting techniques in a slightly different manner, and as a result, each protocol has its own advantages and disadvantages.

3.1 Mixes

A mix is essentially a proxy with added functionality [2]. Similar to proxies, mixes forward messages from one node to another. However, unlike proxies, mixes perform source rewriting operations (described above) on incoming messages before sending them towards their destination. These operations are executed to eliminate any correspondence between messages entering the mix and messages exiting the mix [2]. Mixes also rely on public key cryptography to add an additional layer of security to their communications. Thus, all messages sent to a mix are encrypted with that mix's public key so that no one other than the mix can decipher the contents of the message. Upon receiving a message, a mix performs a number of operations. First, the mix decrypts the message using its private key. This decryption will reveal another message and the address to which the mix should forward that message. After the mix obtains these two pieces of information, it checks if it already contains the message in its current batch of messages. If the mix already contains the message, it discards the message to prevent statistical analysis.

Otherwise, the message is placed in the mix's current batch of messages. Each mix maintains a batch of messages to facilitate the reordering of messages. Typically, mixes use one of two methods to determine when to send a batch of messages. One approach is to batch up to n messages. Then, once the current batch contains n messages, the mix sends the n messages in a random order so as to remove the correspondence between a message's arrival and departure. The problem with this approach is evident when a mix receives very little traffic. If n messages never arrive at the mix, the messages will never be sent. To combat this problem, some mixes use time slices to determine when they should forward their current batch of messages. Under the time slice approach, mixes wait for a given amount of time (i.e. a time slice) before sending their

current batch of messages. Then, once a time slice expires, the mix sends the messages in its current batch along with dummy messages (if less than n messages were in the queue when the time slice expired). The following example should clarify these ideas.

When Cartman wants to send a message to Stan, he first encrypts the message with Stan's public key (K_S), creating $K_S(\text{message})$. Then, Cartman decides how many mixes he wants to forward his message through (i.e. the number of links in the mix chain). Assuming Cartman chooses five mixes (e.g. M_1, M_2, M_3, M_4, M_5), he then encrypts his message and Stan's address (A_S) with the public key of the mix closest to Stan (i.e. the last mix in the chain). If the last mix is M_5 , then after this step, the message should look like the following:

$$K_{M_5}(K_S(\text{message}), A_S)$$

Next, Cartman encrypts the message and the address of the last mix with the public key of the mix second closest to Stan (i.e. the second to last mix in the chain). Assuming the second to last mix in the chain is M_4 , after this step, the message should look like the following:

$$K_{M_4}(K_{M_5}(K_S(\text{message}), A_S), A_{M_5})$$

Cartman repeats this process for all of the chosen mixes until the message finally looks like the following:

$$K_{M_1}(K_{M_2}(K_{M_3}(K_{M_4}(K_{M_5}(K_S(\text{message}), A_S), A_{M_5}), A_{M_4}), A_{M_3}), A_{M_2})$$

Once Cartman has completed these cryptographic operations, he performs the necessary padding operations to ensure all of his messages are equal to the specified fixed-length. Then, he sends the message to the first mix in the chain (M_1). When M_1 receives the message, it decrypts the message with its private key ($K_{M_1}^{-1}$), resulting in the following:

$$K_{M_2}(K_{M_3}(K_{M_4}(K_{M_5}(K_S(\text{message}), A_S), A_{M_5}), A_{M_4}), A_{M_3}), A_{M_2}$$

Then, M_1 takes the unencrypted message and performs the necessary padding operations to ensure the message is the same length as the rest of M_1 's messages. After the message is the appropriate size, M_1 checks to make sure the message is unique. If M_1 already has a copy of the

message in its current batch, the duplicate copy is discarded to prevent statistical analysis [2]. If the message is unique, M_1 places it in the current batch of messages. Once this batch is full or the current time slice has expired, M_1 will send this batch of messages (along with any necessary dummy messages) to their next destination. In the case of Cartman's message, it will be forwarded to M_2 . Upon receiving the message, M_2 will perform the same steps taken by M_1 . Eventually, the message will reach M_5 who will finally deliver the message to Stan.

3.2 Crowds

Crowds is a source rewriting system that provides anonymity for web transactions by blending each user's traffic with the traffic of other users [20]. Initially, to execute a transaction, a user must join the crowd of existing members. Then, once the user is a member of the crowd, the user forwards requests to a randomly selected member of the crowd. Upon receiving the user's request, the randomly selected member has two options: send the request to the final destination or forward the request to another randomly selected member. Eventually, the user's request will arrive at its intended destination; however, upon receipt of the request, the receiver will be unable to discern the original sender. The receiver will know which member actually delivered the request, but the receiver will not know if that member originated the request or simply forwarded it [20].

Within the Crowds system, a user is represented by a process called a jondo ("John Doe"). Before a user can use the Crowds system, the user must establish a username and password with a central server called a blender. Then, when the user's jondo is started, the jondo uses the username and password to create a connection to the blender. Once the jondo is connected to the blender, it requests to join the crowd. If the blender allows the jondo to join, the blender adds the jondo to its list of crowd members. Then, the blender sends this list of members to the jondo along with a list of shared cryptographic keys to be used by the jondo to communicate with the other crowd members. Each of the appropriate keys is also sent to the

other crowd members so that they are able to communicate with the jondo. Paths in the Crowds system are static to help limit the success of statistical analysis. As a result, a new jondo cannot immediately join the crowd because the new paths it creates can easily be identified as originating from the new jondo (due to the existing static path architecture). To protect newly added jondos, Crowds does not actually allow a jondo to enter the crowd until the blender executes an event called a join commit. After a join commit is executed, all the static paths are destroyed, and the newly added jondos are allowed to participate in the crowd. Once a user's jondo has joined the crowd, the jondo begins routing the user's requests along independently random paths of other jondos. This routing process proceeds as follows. When a user submits a request, the user's jondo randomly picks a jondo from the crowd (it may select itself) and forwards the request to that jondo. When the selected jondo receives the request, it flips a biased coin (with probability p_f) to determine if it should continue forwarding the request through the crowd. If the coin-flip results in continued forwarding, the jondo forwards the request to another randomly selected jondo; otherwise, the jondo sends the request to its intended receiver. Once the request arrives at its destination, the path it took remains static. Thus, all subsequent interactions between the sender and the receiver will follow this route [20]. The following example should clarify these ideas.

In order to use the Crowds system, Cartman must initially register a username and password with the blender. Then, using this information, Cartman's jondo connects to the blender and requests to join the crowd. If the blender allows Cartman to join, Cartman waits until the blender executes a join commit. After the join commit, Cartman sends a request destined for Stan. Cartman's jondo then randomly selects a jondo from the crowd. Assuming the selected jondo is called J_1 , Cartman's jondo forwards the request to J_1 . Upon receiving the request, J_1 flips a biased coin to determine if it should forward the request or deliver the request to its final destination. If the coin-flip results in continued forwarding (the probability of this event is p_f), J_1 randomly selects a jondo from the crowd and forwards the request to that jondo. Otherwise, J_1

delivers the request to Stan. Eventually, Stan will receive the request from one of the jondos in the crowd, but he will be unable to determine the identity of the original sender because the request could have been forwarded through any number of jondos.

4 Conclusion

As online communications continue to grow in popularity, the ability to communicate in an anonymous fashion will become increasingly important. The current reliance on encryption to solve anonymity issues is insufficient, and additional steps must be taken to protect the anonymity of online participants. This paper presented a survey of the current techniques employed to provide this anonymity for online communications. In showing the strengths and weaknesses of current approaches, this paper should also facilitate the continued research of more efficient mechanisms for providing anonymity.

References

- [1] O. Berthold et al., "Web MIXes: A System for Anonymous and Unobservable Internet Access," In *Proceedings of the 2000 ICSI Workshop on Design Issues in Anonymity and Unobservability*.
- [2] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, Volume 24, Number 2, pp. 84-88.
- [3] D. Chaum, "The Dining Cryptographers Problem," *Journal of Cryptology*, Volume 1, Number 1, pp. 65-75.
- [4] I. Clarke et al., "Freenet: A Distributed Anonymous Information Storage and Retrieval System," In *Proceedings of the 2000 ICSI Workshop on Design Issues in Anonymity and Unobservability*.
- [5] W. Dai, "PipeNet 1.1," <http://www.eskimo.com/~weidai/pipenet.txt> (current 23 Oct. 2003).
- [6] M. Freedman and R. Morris, "Tarzan: A Peer-to-Peer Anonymizing Network Layer," In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*.
- [7] E. Gabber et al., "How to Make Personalized Web Browsing Simple, Secure, and Anonymous," In *Proceedings of Financial Cryptography '97 (FC '97)*.
- [8] E. Gabber et al., "Consistent yet Anonymous Web Access with LPWA," *Communications of the ACM*, Volume 42, Number 2, pp. 42-47.
- [9] S. Goel et al., "Herbivore: A Scalable and Efficient Protocol for Anonymous Communication," Cornell University CIS Technical Report TR2003-1890, <http://www.cs.cornell.edu/People/egs/papers/herbivore-tr.pdf> (current 23 Oct. 2003).
- [10] I. Goldberg and A. Shostack, "Freedom Network 1.0 Architecture," White Paper, <http://www.homeport.org/~adam/zeroknowledgewhitepapers/arch-notech.pdf> (current 23 Oct. 2003).
- [11] I. Goldberg and D. Wagner, "TAZ Servers and the Rewebber Network: Enabling Anonymous Publishing on the World Wide Web," *First Monday*, Volume 3, Number 4.
- [12] I. Goldberg et al., "Privacy-enhancing technologies for the Internet," In *Proceedings of the 42nd IEEE COMPCON (COMPCON '97)*.
- [13] Y. Guan et al., "A Quantitative Analysis of Anonymous Communications," *IEEE Transactions on Reliability*, to appear.
- [14] C. Gulcu and G. Tsudik, "Mixing Email with BABEL," In *Proceedings of the 1996 ISOC Symposium on Network and Distributed System Security*.
- [15] B. Levine and C. Shields, "Hordes: A Multicast Based Protocol for Anonymity," *Journal of Computer Security*, Volume 10, Number 3, pp. 213-240.
- [16] D. Martin, "Internet Anonymizing Techniques," *USENIX ;login: Magazine*, May 1998, pp. 34-39.
- [17] D. Martin, "Local Anonymity in the Internet," PhD Thesis, Boston University, 2000, <http://www.cs.du.edu/~dm/anon.html> (current 23 Oct. 2003).
- [18] A. Pfitzmann and M. Waidner, "Networks Without User Observability," *Computers and Security*, Volume 6, Number 2, pp. 158-166.

- [19] M. Reed et al., "Proxies for Anonymous Routing," In *Proceedings of the 12th Annual IEEE Computer Security Applications Conference*.
- [20] M. Reiter and A. Rubin, "Crowds: Anonymity for Web Transactions," *Transactions on Information and System Security*, Volume 1, Number 1, pp. 66-92.
- [21] V. Scarlata et al., "Responder Anonymity and Anonymous Peer-to-Peer File Sharing," In *Proceedings of the 2001 IEEE International Conference on Network Protocols (ICNP 2001)*.
- [22] R. Sherwood et al., "P⁵: A Protocol for Scalable Anonymous Communication," In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*.
- [23] E. Sirer et al., "CliqueNet: A Self-Organizing, Scalable, Peer-to-Peer Anonymous Communication Substrate," <http://www.cs.cornell.edu/People/egs/papers/cliqenet-iptp.pdf> (current 23 Oct. 2003).
- [24] P. Syverson et al., "Anonymous Connections and Onion Routing," In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*.
- [25] M. Waidner and B. Pfitzmann, "The Dining Cryptographers in the Disco: Unconditional Sender and Receiver Untraceability with Computationally Secure Serviceability," In *Proceedings of Eurocrypt '89*.

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.