Steven Young
95-762: Privacy in the Digital Age

Proposal for a New Architecture for Anonymous Peer to Peer Networks

Introduction

Individuals and organizations recognize that privacy and security are strongly linked concepts. The desire for privacy by individuals and organizations has resulted in the creation of many privacy enhancing technologies over the last 30 years. Among these technologies are systems designed to provide anonymous communication or storage through a network such as the Internet. Starting in 1981 with David Chaum's MIX proposal and then to the introduction of the onion routing protocol in 1996, Zeroknowledge's Freedom Anonymizer service in 1999, the AN.ON/JAP Anonymizer in 2000 and the current Tor network in 2004. (Federrath 2005)

In addition, the desire for such systems is also underscored by the ongoing development and monetary support they receive. The Tor project is supported by the U.S. Department of Defense's Advanced Research Projects Agency (DARPA), the Electronic Frontier Foundation, and Human Rights Watch and by contributions from over 500 individual donations. (The Tor Project, Inc. 2008).

Motivation

Carefully combining the features of multiple anonymity technologies, it might be possible to create a more effective anonymous communication system.

A substantial amount of effort and research is currently ongoing to improve the security and ease of use of peer-to-peer anonymous communications.

Projects such as Tor which implement the Onion Routing protocol provide a way for two parties to communicate in real time with a high degree of anonymity between two communicating parties. (Dingledine, Mathewson and Syverson, Tor: The Second-Generation Onion Router 2004)

Anonymous distributed storage systems such as FreeNet are widely used and offer a way to store data in a way that provides anonymity between the publisher and the recipient of the stored data. (Clarke, et al. 2002)

Academic papers on steganographic storage and peer-to-peer steganographic storage offer insight into the possibility of storing data in a manner that makes it indistinguishable from random data and allows for plausible deniability for both the publisher and the owner of the system where the data resides. (Anderson, Needham and Shamir 2001) (Hand and Roscoe 2002)

Employing caching in a proxy between a requesting user and a content provider could make user requests independent in time from content providers. (Shubina and Smith 2003)

The introduction of time independence between the communicating parties can help mitigate the ability of an attacker to use an end-to-end timing analysis method described in (Mathewson and Dingledine 2004) and (Shmatikov and Wang 2006).

The popular anonymous communication service, AN.ON/JAP employs a caching proxy between the final mix and the Internet. (JAP Team 2006)

However, several known weaknesses exist with each of the systems described above that limit their ability to protect communicating parties and possibly hinder their adoption by privacy-seeking users.

Proposal Thesis

A system that incorporates properties of each of the above systems while remaining as similar as possible to an existing trusted platform such as Tor could provide a model for a next generation anonymous communication system that is less susceptible to common vulnerabilities of existing anonymous communication systems.

Incorporating the concept of caching requests and responses presented in (Shubina and Smith 2003) by employing caching proxies at many nodes within an onion network could significantly increase the cost to an attacker of using traffic analysis techniques to identify communicating parties.

Adding steganographic storage techniques to the caching proxies could help mitigate the potential disclosure vulnerabilities caused by storing data in caches on many nodes.

Finally, by using the above methods to expand the Hidden Service feature of the Tor network into a distributed hidden service, it might be possible to create a hidden service that is substantially less vulnerable to attacks described in (Overlier and Syverson, Locating Hidden Servers 2006).

In addition, since traffic between users to a distributed hidden service does not exit the network to access the published content, end-to-end timing analysis by a global observer as described in (Dingledine, Mathewson and Syverson, Challenges in Deploying low-latency Anonymity 2005) could become irrelevant.

The proposal is described in two parts. The first part describes suggested new system components needed for the new design. The second part describes the interaction of the components.

System Components

(Note: ***Components in italics are proposed.*** **Non-italics components already exist.**)

Distributed Hidden Service Descriptor:

Distributed Hidden Service Descriptors are similar to Tor's current Hidden Service descriptor However, unlike the current hidden service descriptors which store the node identities of hidden service introduction points; distributed hidden service descriptors do not store any node identification information. This makes it difficult to use the descriptor to associate nodes with a hidden service.

The descriptor has three parts and is generated in two phases and are stored on directory servers by distributed hidden service publishers and accessed by hidden service subscribers (users).

In the first phase, a publisher makes a request to the directory servers that a descriptor be generated. The directory servers supply a descriptor that only contains a unique URI to the publisher. The publisher needs this URI in order to properly normalize the content prior to publication as described in the "normalizing content for publication" section of this document.

In the second phase, the publisher adds the signed public key (i.e. digital certificate) that decrypts the data being published. This certificate also acts as a digital signature which serves to validate the authenticity of the data to the subscriber. (RSA Laboratories 2007)

The publisher then also adds the storage group valet descriptor which is encrypted using a separate key. In addition, the content publisher can set a flag indicating that this storage group is writable. The writable flag indicates that the content version field of the descriptor is being used and therefore must be updated as described in the "updating writable content" section of this document. At this point, the descriptor is complete and can be published to the directory.

The descriptor completed in the second phase can be published to the director an arbitrary amount of time after both the initial request in phase 1 and after the content storage step. This allows a publisher to request a large number of descriptors at time T_1 from one location L_1 while publishing the content at a second time, T_2, from another location, L_2, and finally publishing the descriptor back to the directory at third time, T_3, and location, L_3.

By allowing these events to be separated in time, it can be substantially more difficult for an attacker monitoring the traffic of both the directory servers as well as the publisher to link the traffic these activities together. An attacker would need to be dedicated enough to monitor locations L_1, L_2 and L_3 at times T_1 T_2 and T_3 in order to employ methods to link the traffic associated with the storage of content with the descriptor.

A more complete discussion of how the introduction of random delays can make traffic analysis more difficult is presented in (Danezis, The Traffic Analysis of Continuous-Time Mixes 2005).

***Storage Group**:*

A group of **M** nodes (Storage Group Members) randomly selected by the hidden service publisher that store the published hidden service content. Storage group members are unaware of the identity of the other nodes. In addition, storage group members only respond to requests for data access from authorized valet nodes. The stored data consists of **N** unique data blocks where **M** > **N** such that the data can be reconstructed even if a certain number of storage group member nodes are unavailable.

Communication with storage group members is similar to Tor hidden service introduction points as described in (Dingledine, Mathewson and Syverson, Tor: The Second-Generation

Onion Router 2004). However, rather than relaying the communication request to a hidden server, the storage members serve their portion of content directly to the rendezvous node.

Storage Group Valet Descriptor:

A collection of all the individual storage member valet keys encrypted with the distributed hidden service access key. The storage group valet descriptor is part of the distributed hidden service descriptor. It is decrypted by the subscriber and the storage member valet keys are relayed to the rendezvous node by the content subscriber.

The individual storage member valet keys are encrypted so valet nodes and potential attackers cannot easily associate a distributed hidden service descriptor with a valet node.

The rendezvous node uses the collection of storage key hashes as a way to notify authorized storage group valets without knowing the identity of the valet nodes.

This proposal currently suggests the rendezvous node use a simple random broadcast method to notify valet nodes. A better approach could be to employ a secure routing overly method as described in (Castro, et al. 2002). Such an approach could significantly reduce the response time however it might make it easier to detect the identity of valet nodes.

Storage Member Valet Key:

A hash of a single storage member's steganographic storage key. This key serves multiple purposes. It is used by the rendezvous node to initiate communication with valet nodes without knowing their identity. In addition, it serves as a way for valet nodes to in turn notify storage group members and uniquely identify the steganographic key of the requested stored content.

The valet key also acts as a second authentication factor between the storage group member and the valet node in addition to the valet's node ID which is stored in the storage member's access list.

Although the valet keys are stored in the directory in an encrypted form, they are not considered a secret. The valet keys are known to the content publisher, the content subscriber, the storage group valet nodes and the rendezvous nodes as well as any nodes that receive that broadcast from the rendezvous node. Since the storage group members will only retrieve data when provided a proper valet key from an authorized source, it would be difficult for an attacker to use the valet keys to reveal corresponding stored content.

In order for an attacker to successfully obtain distributed hidden content, they would need to obtain all the valet keys associated with a storage group and be in control of all valet nodes in order to request the content. Then, once the attacker is in possession of the encrypted content, they would also need to know the distributed hidden service descriptor associated with the valet keys in order to obtain the plaintext of the stored content.

Finally, assuming an attacker was able to successfully obtain the plaintext content and assuming the content itself does not betray communicating parties, they would still not have

information that could directly identify the communicating anonymous parties (i.e. publisher and subscriber).

The storage member valet key and the steganographic storage key act in concert in a manner similar to the service key pair and key identifiers described in (Overlier and Syverson, Valet Services: Improving Hidden Services with a Personal Touch 2006).

***Steganographic Storage File**:*

A contiguous flat file that resides on each distributed storage node. The file is created during the installation of the node and initially filled with random data and logically divided into blocks according to a map unique to each node. All data is stored in blocks chosen according to the algorithm described in "Steganographic Storage of Content" section of this document.

Employing steganographic techniques can potentially help achieve several goals.

Storing data in pseudo-randomly chosen patterns in the storage file and not preventing 'collisions', ensures that, over time, stored data will be overwritten. This eliminates the need to explicitly notify a node to remove data and the need for a storage node to track the age of the data it stores. Eliminating the need to track the data reduces a potential vector an attacker can exploit to identify the data and associate it with a publisher or subscriber.

In addition, steganographic storage makes it more difficult to determine what data and whether a specific piece of data is stored. If content were stored without steganographic measures, an attacker with possession of the total encrypted content could more easily determine that a node was storing a portion of that encrypted content.

Since it is more difficult for an attacker to determine what data is stored, it is easier for a node operator to assert plausible deniability. (Hand and Roscoe 2002)

In order to reduce the problem of hash collisions, the steganographic storage file must be sufficiently large and the block size must be sufficiently small to allow a large number of blocks. In addition, independently encrypted replicas of stored blocks are simultaneously written to the file. When data is retrieved, these replicas can help detect collisions and corrupted data. An overview of the storage algorithm is in the "Steganographic Storage of Content" section of this document.

A more detailed description of storage algorithm is in the "A local Steganographic File System" section of (Hand and Roscoe 2002). In addition, alternative but similar steganographic storage algorithms are described in detail in (ZHOU 2005).

Steganographic Storage Key:

A combination of encryption key, initial block location hash value as well additional information that can be used to perform a validity check on the data. The storage key is generated when data is stored and serves to locate blocks where data and replicas are stored and to reverse the encryption done when the data was stored.

Storage keys are unique to each storage group member node and known only to the storage group node where the data is stored. Authorized valet nodes posses a hash of this key called the storage member valet key.

Storage Group Valet Nodes:

Nodes are randomly selected by individual storage group members and are provided with storage member valet keys at the time of content storage by a storage member.

Valet nodes are the only nodes that can initiate a request to a storage group member retrieve or update stored data. This is enforced by an access control list maintained by each storage group member.

Storage Group Valet Nodes extend the hidden service valet node proposal presented in (Overlier and Syverson, Valet Services: Improving Hidden Services with a Personal Touch 2006) by the additional use of the valet storage keys as a location mechanism.

In this proposal, the valet storage keys are used as the method rendezvous nodes discover valet nodes in lieu of publishing the identity of the valet nodes in the directory.

Rendezvous Node:

A node randomly chosen by the subscriber to a hidden service that serves as a communication intermediary between the subscriber and storage group nodes.

Rendezvous Nodes are already part of the current Tor hidden service specification and they are used by this proposal in the same manner as described in current Tor design document. (Dingledine, Mathewson and Syverson, Tor: The Second-Generation Onion Router 2004)

Local Web Server:

An application such as Apache Server or Microsoft IIS server that runs on a subscriber's computer and serves HTML content to the subscriber's web browser via standard HTTP requests.

In order to protect privacy, the web server must be properly configured. At the minimum, it should be configured to only serve content from the hidden service domain (e.g. *.onion) to and it should only communicate through the privacy proxy software on a subscribers machine. In addition, served content must only be resident in memory and never stored on disk.

Since the web server will operate through the privacy proxy, accessing content from other distributed hidden service descriptors is transparent to the web server as the proxy will automatically initiate the process of obtaining the descriptor and retrieving the content.

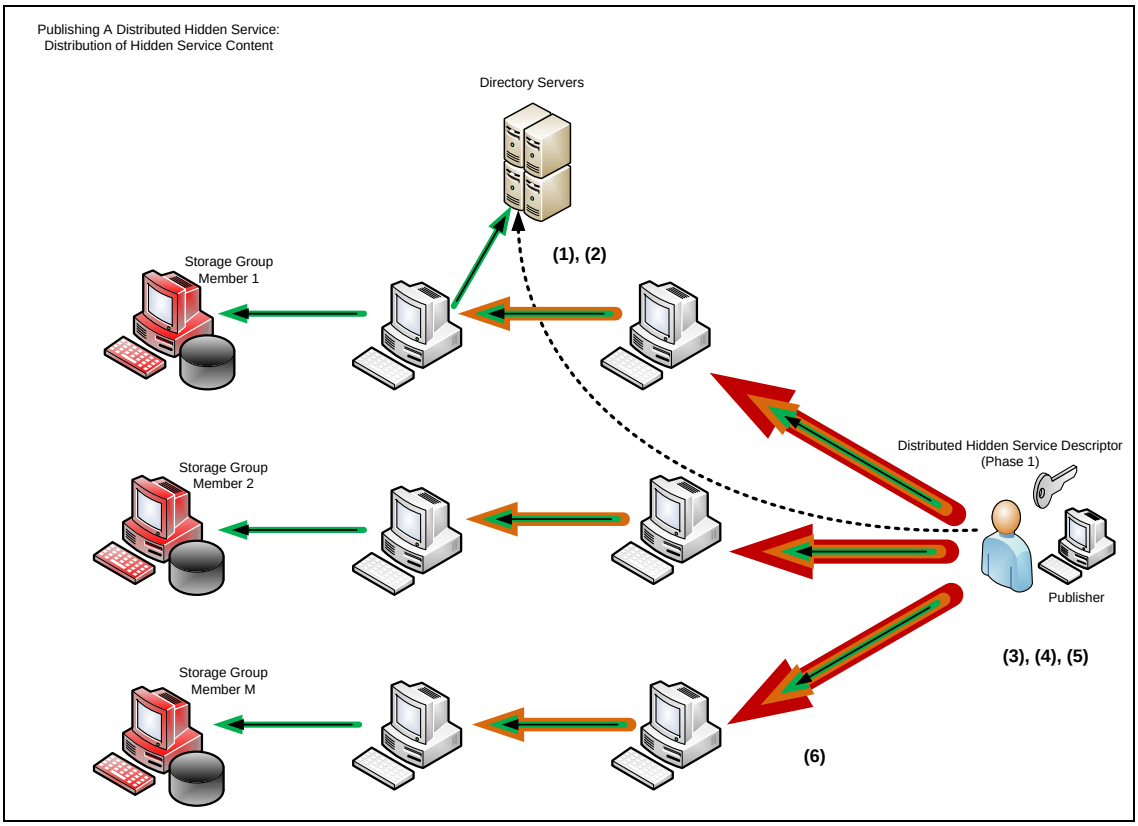
Steven Young
95-762: Privacy in the Digital Age

Publication of Distributed Hidden Service

Distribution of Hidden Service Content [figure 1]

1. Publisher establishes onion circuit to directory server and requests distributed hidden service descriptor(s) be created.
2. Publisher queries directory and obtains a list of nodes that are currently active and willing to store data.
3. Publisher randomly selects **M** nodes to be used as storage nodes from directory list.
4. Publisher normalizes content for publication by performing steps described in the "normalizing content" section of this document.
5. Publisher prepares content by signing, encrypting and then splitting content into ***N*** chunks. Where ***M*** **>** ***N***.
6. Publisher distributes ***N*** chunks of content to ***M*** storage nodes with ***M*** – ***N*** duplicate chunks through established onion circuits.(Note: An arbitrarily long delay can exist between steps 1 and 2.)

Figure 1



Steven Young
95-762: Privacy in the Digital Age

Steganographic Storage of Content

[Steps 2–4 described below are paraphrased from the "Writing and Reading a Single Block" section of (Hand and Roscoe 2002)]
[Step 5 is based on the "Distributing the Block Store" section of (Hand and Roscoe 2002)]

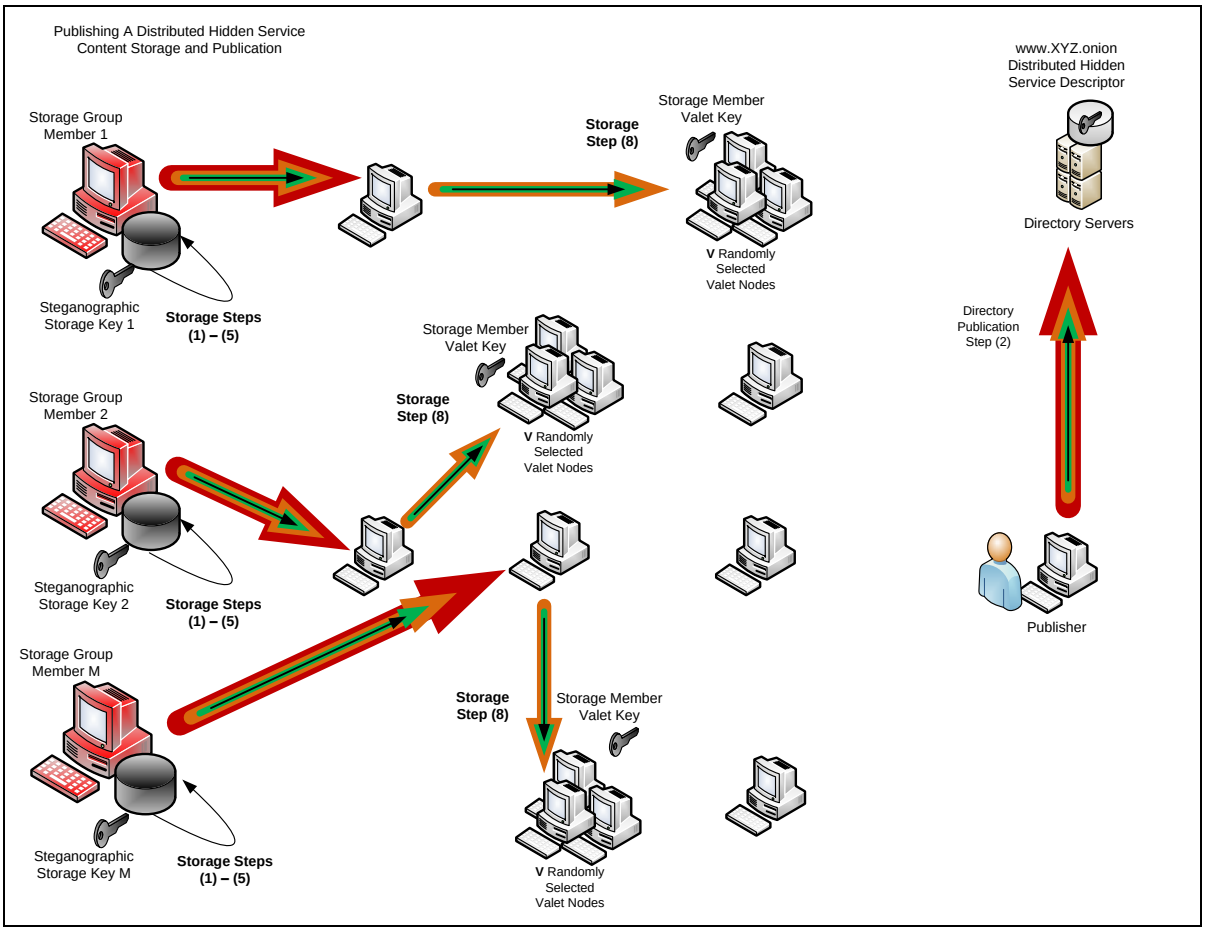
1. Each Storage Group Member breaks apart the content chunk, N_i delivered to it by the publisher into ***n*** blocks where the block size is equal to the block size employed in the steganographic file store.
2. ***i*** blocks from n are chosen to be replicas where $0 \leq i \leq n$.
3. The ***n*** blocks are encrypted and stored in pseudo-random locations within the steganographic file store according to a hash function based on a key, ***K***.
4. The ***i*** replica blocks are each encrypted using an individual encryption key per block. The per-block replica key is: $k_i = E_k(h_i)$ and each replica block is stored in a location within the storage file that corresponds to its partner: $b_i = h_i \bmod X$**.** Where ***X*** is the total number of blocks within the steganographic storage file.
5. Each storage group member computes a hash of the key, **H(*K*)**, it generated in step 3 above to encrypt and store the contents.
6. Once all blocks are stored, each node returns **H(*K*)** to the publisher via the onion circuit and the circuit is then torn down. (Note: For clarity diagram does not show these existing circuits)
7. Each storage group member randomly selects ***V*** valet nodes from the directory of available nodes.
8. Each storage group member establishes a onion circuit with its chosen valet members and publishes the steganographic storage key hash, **H(*K*).** In addition, the storage group member stores the valet node's onion unique identifier in an access list.

Directory Publication [figure 2]

1. Publisher generates the storage group valet descriptor by encrypting the steganographic storage key hashes received from each storage node. With the addition of the storage group valet descriptor, the Distributed Hidden Service Descriptor is complete and ready to be published.
2. The Distributed Hidden Service Descriptor is published to the directory.

(Note: An arbitrarily long delay can exist between steps 1 and 2.)

Figure 2



Accessing a Distributed Hidden Service

Establishing Rendezvous and Service Location [figure 3]

1. Subscriber obtains Hidden Service URI from public channel (e.g. accessing a public website via onion network and obtaining hidden service URI.)
2. Subscriber obtains hidden service key from another channel (e.g. e-mail or website)
3. Subscriber obtains distributed hidden service descriptor and a list of active nodes from directory and randomly chooses a node from the list to act as a rendezvous node.
4. Subscriber uses hidden service key to decrypt storage group valet descriptor.
5. Subscriber establishes an onion tunnel to rendezvous node and delivers storage group valet descriptor to it.
6. Rendezvous node forwards storage group valet descriptor as well as its own node ID to random nodes and waits for connection requests from storage group members.

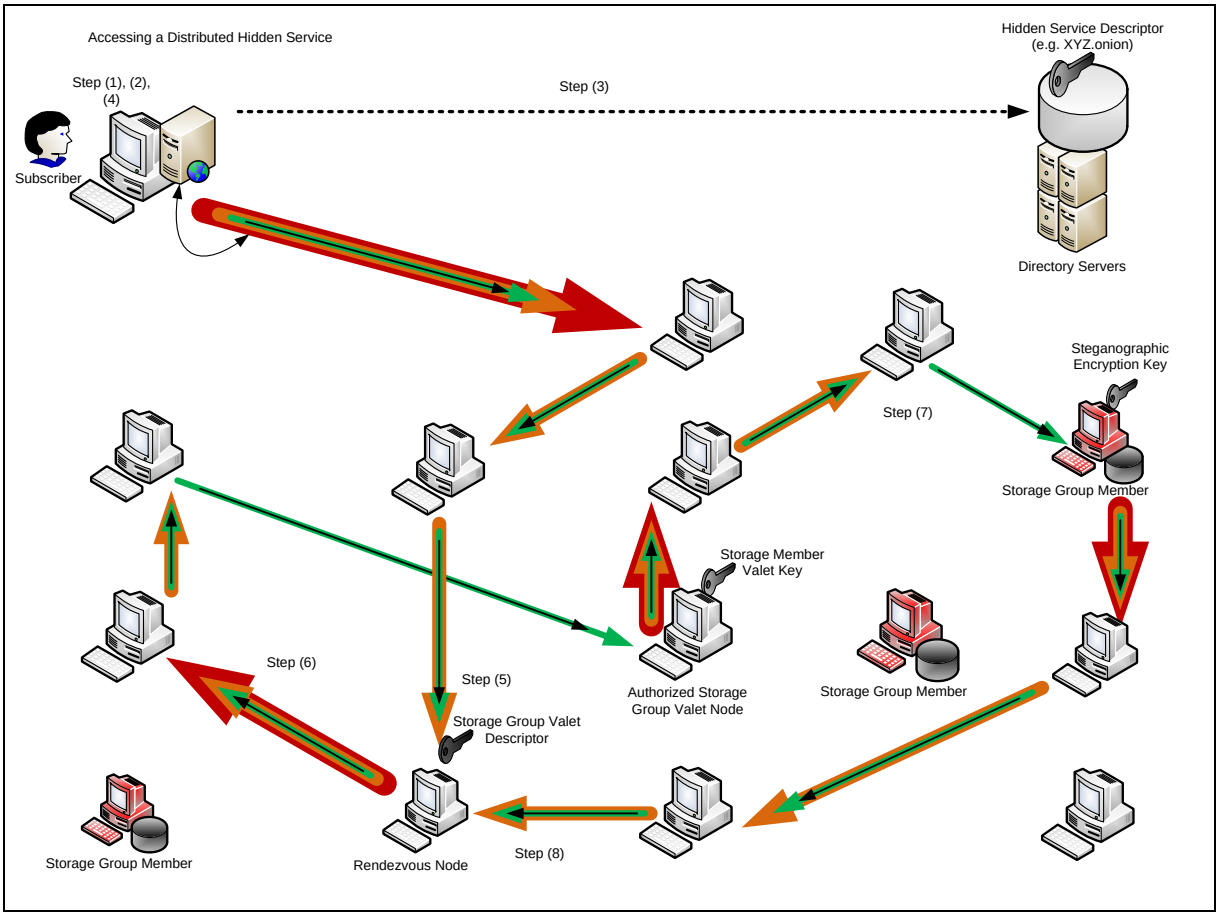
7. Storage Group Valet Nodes see matching storage group valet descriptor and notify storage group member of rendezvous node. (Duplicate requests are ignored by storage group members)
8. Storage group members establish onion circuits to rendezvous node and rendezvous node notifies subscriber connection is complete.

[The speed of step 6 could be improved using an overlay routing network protocol such as Tapestry as described in (Hildrum, et al. 2002)]

Accessing Content [figure 3]

1. The privacy proxy on subscriber's computer makes standard HTTP requests to web server service also running on client machine. The web server serves web pages via standard HTTP responses to the subscriber's browser through the local privacy proxy.
2. The web server accesses content through the onion circuit to the rendezvous point as described in the "serving content" section of this document.

Figure 3



Normalizing Content for Publication

1. Any references to content outside of the onion network found in content are removed.
2. All relative references are prefixed with the generated hidden service descriptor URL. (i.e. /index.html -> xyz.onion/index.html)
3. If references to other hidden service descriptors are found, the proper storage group valet descriptors are embedded in the content. This allows content to reference other hidden services and enabled the web server to automatically resolve and obtain that content.

Serving Content

The web server is configured to only communicate through the privacy proxy agent on the local machine. This allows link URLs to be properly resolved through the onion network and ensures that any external links that could betray user's activities are not followed.

All the content associated with a hidden service descriptor is obtained at once and unpacked by the web server into RAM. No plaintext content is stored on disk.

References to other hidden service descriptor URLs are resolved and the content is retrieved automatically.

Since the storage group nodes are unaware of each other's identity, the web server must act as a coordinator to ensure that content flagged as writable, such as a blog page, is properly updated by the storage group. The web server does this by sending update requests through the circuit established with the rendezvous node.

Updating Writeable Content

1. Web server sends storage group member an update request.
2. When each storage group member receives a write request, it generates a new steganographic storage key and encrypts and stores the new data as described in the "Steganographic storage of data" section of this document. The original key is kept but marked as referencing the new key.
3. Storage group member notifies its valet nodes that the storage member valet key they have is 1 version behind. The storage member valet key is not changed.
4. Once a sufficient number of valets acknowledge the update, the storage member notifies the web server that the data is updated.

After a sufficient number of storage members acknowledge the update, the web server establishes an onion circuit to the directory server and updates the service descriptor with the current version number.

Weaknesses and Potential Remedies

Economics of Participation:

In order to allow for the storage of content, participating nodes must be willing to yield disk space. And in order to ensure the proper operation of steganographic storage each node must be willing to pre-allocate a set amount of disk space regardless of whether or how much data they intend to store in the anonymous system. In addition, the creation of the storage space may take a considerable amount of time (several minutes to hours) further increasing the economic cost to participate in the network.

According to (Danezis and Anderson, The Economics of Censorship Resistance n.d.), requiring participants to allocate resources to serve random data that the participants themselves are not privy to nor interested in will likely decrease the number of people willing to participate in such a system.

The economic costs of storage associated with participation also raise a related issue, fairness. The overarching goal providing anonymous communication makes it difficult if not impossible to enforce a storage quota for publishers without potentially compromising their identity.

This proposal does not attempt to address these economic issues.

Potential Reduction in Speed:

Since the desired content is distributed across multiple nodes, the amount of time it takes for a request to be fulfilled is dependent on more nodes.

In addition, the proposed 'random broadcast' method of locating valet nodes could take an arbitrarily long time and it would be difficult to determine a proper time out.

Two potential solutions could alleviate this problem.

- Data can be distributed across more nodes (i.e. increase **M** while keeping **N** constant)
- A secure routing algorithm such as the one described in (Castro, et al. 2002) can be used instead of the random broadcast method described in this proposal

Compromised Storage Group Member or Valet Node:

If an attacker is able to compromise a storage group member or a valet node, they could potentially associate stored data blocks with a distributed hidden service descriptor.

If an attacker can successfully associate the stored content with a hidden service descriptor, they can potentially determine the identity of subscribers by monitoring the network traffic of a suspected subscriber and correlating the subscriber's network activity with disk activity to the stored content.

This attack is similar to the attack described in (Overlier and Syverson, Locating Hidden Servers 2006).

However, a few factors in this proposal make the attack described in (Overlier and Syverson, Locating Hidden Servers 2006) more difficult.

- Not all storage group members are aware of requests. Since there are **M** nodes storing **N** chunks of data, **M – N** nodes store redundant data. Therefore, the design of this proposal can ensure that the least number of storage group members are aware of requests. This reduces the odds that the attacker will see all requests and therefore make it more difficult to correlate subscriber network activity with storage group access.
- Since storage blocks are regularly overwritten due to hashing collisions, it is possible that while an attacker is monitoring access, the data stored by the compromised storage group member becomes corrupt. When this occurs, the member notifies its valet nodes and requests are no longer sent to that member.

If all members of a storage group were compromised, then it becomes substantially easier for an attacker to correlate subscriber traffic with a hidden service descriptor.

Key Distribution:

This proposal requires a subscriber obtain an additional encryption key, the hidden service key, outside of the network. A compromise of this key could allow an attacker to discover the corresponding hidden service descriptor.

With the key, an attacker would obtain suspected hidden service descriptors and employ a brute-force method to determine which descriptor the key unlocked.

A potential remedy for this vulnerability would be for the directory servers to restrict the number of hidden service descriptors that can be obtained within a given time. While this can't prevent the attack, it makes it more costly in terms of time and effort.

TCP Connection Establishment:

Since the onion and similar networks employ TCP/IP to establish connections between nodes, a global observer can see when connections are created and broken. Although the anonymous circuits that run over the underlying TCP network connections are somewhat hidden, an observer can use the TCP/IP connection patterns to infer a more detailed understanding of activity.

For example, if an attacker disables an operating onion routing node, they can wait and observe the resulting new TCP/IP connection requests between other nodes to reestablish anonymous circuits. If an attacker is also privy to a targeted end user's communication, they can repeat the process to isolate which nodes are relaying the user's traffic.

The remedy is to use UDP/IP rather than TCP/IP as the underlying network communication protocol between nodes. Since the UDP protocol is connectionless and does not maintain any state information and since the packet payload will always be encrypted, it will be more difficult for an observer to tell when new connections are being established.

Lack of Mechanism to Ensure Adherence to Protocol Rules:

When an attacker compromises a node, they can modify the software on their node to ignore protocol features that help ensure privacy. For example, an attacker can reduce the minimum number of hops required for an anonymous circuit.

This is already a known issue with the current generation of peer to peer systems. However, it could be more damaging in a system that actually stores data for users.

A potential resolution would likely require a means for nodes that are directly connected to verify the code running on their partner node. If the neighbor fails the code check, a Shoot-the-Other-Node-In-The-Head (STONITH) style mechanism can be invoked to alert other nodes of the potential corrupt node.

Conclusion

The model presented was intended to describe a potential way multiple privacy enhancing technologies might be combined to produce a stronger platform for anonymous communication.

Many potential weaknesses with proposed model were highlighted along with potential strategies to overcome those weaknesses.

Given only this overview, it is impossible to determine whether this model can actually improve the ability of an anonymous network to protect the identity of its users. However, it can be used as a basis for a more detailed model.

If provided with a model with sufficient detail, it could be possible to generate an IO automata model as described in (Lynch 1996). This model could potentially be compared with the existing IO automata that models the onion routing protocol as presented in (Feigenbaum, Johnson and Syverson, A Model of Onion Routing with Provable Anonymity 2007) to as a metric to determine whether it increases or decreases the anonymity of communication.

Works Cited

Anderson, Ross, Roger Needham, and Adi Shamir. "The Steganographic File System." 2001.

Castro, Miguel, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. "Secure routing for structured peer-to-peer overlay networks." *Proceedings of 5th Usenix Symposium on Operating Systems Design and Implementation.* Boston, MA, 2002.

Clarke, Ian, Scott Miller, Theodore Hong, Oskar Sandberg, and Brandon Wiley. *Protecting Free Expression Online with Freenet - Internet Computing, IEEE.* IEEE , 2002.

Cleversafe. "Information Dispersal Algorithms." *CleverSafe.* 2007. http://www.cleversafe.org/dispersed-storage/idas (accessed September 2008).

Danezis, George. "The Traffic Analysis of Continuous-Time Mixes." (Springer Berlin / Heidelberg) 3424/2005 (2005).

Danezis, George, and Ross Anderson. *The Economics of Censorship Resistance.* Cambridge, United Kingdom: University of Cambridge, Computer Laboratory.

Dingledine, Roger, Nick Mathewson, and Paul Syverson. *Challenges in Deploying low-latency Anonymity.* The Free Haven Project, 2005.

—. "Tor: The Second-Generation Onion Router." Proceedings of the 13th USENIX Security Symposium, 2004.

Federrath, Hannes. "Privacy Enhanced Technologies: Methods - Markets - Misuse (LNCS 3592)." *Proceedings of 2nd International Coonference on Trust, Privacy, and Security in Digital Business.* Heidelberg: Springer-Verlag, 2005. 1 - 9.

Feigenbaum, Joan, Aaron Johnson, and Paul Syverson. "A Model of Onion Routing with Provable Anonymity." 2007.

Feigenbaum, Joan, Aaron Johnson, and Paul Syverson. *Probabilistic Analysis of Onion Routing in a Black-box Model.* ACM Press, 2007.

Goldschlag, David M., Michael G. Reed, and Paul F. Syverson. "Hiding Routing Information." Washington, D.C.: Naval Research Laboratory, Center For High Assurance Computer Systems, 1996.

Hand, Steven, and Timothy Roscoe. *Mnemosyne: Peer-to-Peer Steganographic Storage.* Burlingame, CA: Sprint Advanced Technology Lab, 2002.

Hildrum, Kirsten, John Kubiatowicz, Satish Rao, and Ben Zhao. *Distributed Object Location in a Dynamic Network.* Berkeley, CA: Computer Science Division, University of California at Berkeley, 2002.

JAP Team. "Architecture of the Anonymization Service." *JAP - Project Anonymity in the Internet.* German Research Foundation. 2006. http://anon.inf.tu-dresden.de/desc/desc_anon_en.html (accessed October 2008).

Lynch, Nancy. *Distributed Algorithms.* Morgan Kaufmann, 1996.

Mathewson, Nick, and Roger Dingledine. *Practical Traffic Analysis: Extending and Resisting Statistical Disclosure.* The Free Heaven Project, 2004.

"Onion Routing Network Requirements Document." *Onion-Router.* May 24, 2002. http://www.onion-router.net/Archives/requirements.pdf (accessed October 2008).

Overlier, Lasse, and Paul Syverson. "Locating Hidden Servers." Oakland, CA: IEEE CS Press, 2006.

—. "Valet Services: Improving Hidden Services with a Personal Touch." Proceedings of the 2006 Privacy Enhancing Technologies Workshop, 2006.

RSA Laboratories. "What is a digital Signature and What is Authentication?" *RSA Labortories.* 2007. http://www.rsa.com/rsalabs/node.asp?id=2182 (accessed October 17, 2008).

Serjantov, Andrei, and Steven Murdoch. "Message Splitting Against the Partial Adversary." In *Lecture Notes In Computer Science Volume 3856/2006*. Springer Berlin / Heidelberg, 2006.

Shmatikov, Vitaly, and Ming-Hsiu Wang. *Timing analysis in low-latency mix networks: attacks and defenses.* Austin, TX: The University of Texas at Austin, 2006.

Shubina, Anna M., and Sean W. Smith. "Using caching for browsing anonymity." *ACM SIGecom Exchanges*, 2003: 11-20.

The Tor Project, Inc. "Tor: Sponsors." *The Tor Project, Inc.* October 13, 2008. http://www.torproject.org/sponsors.html.en (accessed October 17, 2008).

United Nations. "United Nations Universal Declaration of Human Rights." *United Nations.* 1948. (accessed 2008).

ZHOU, XUAN. *STEGANOGRAPHIC FILE SYSTEM.* Singapore: National University of Singapore, 2005.