1 Motivation

The internet is moving in the direction of increasing freedom of information, and increasingly blurred national boundary lines. At the same time as a strong sense of global community is growing, technical advances have provided greatly increased bandwidth and an enormous amount of computing power and well-connected storage. However, the increases in speed and efficiency have not brought comparable increases in privacy and anonymity on the internet – indeed, governments and especially corporations are beginning to realize that they can leverage the internet to provide detailed information about the interests and behaviors of existing or potential customers. Court cases, such as the Church of Scientology's lawsuit against Johan Helsingius[1] or the more recent OpenDVD debate[2] (and subsequent arrest of DeCSS author Jon Lech Johansen), demonstrate that the internet does not currently have an adequate infrastructure for truly anonymous publication or distribution of documents or other data.

2 Project Goals

The Free Haven Project intends to deploy a system that provides a good infrastructure for stronger anonymity. Specifically, this means that the publisher of a given document should not be known; that clients requesting the document should not have to identify themselves to anyone; and that the current location of the document should not be known. Additionally, it would be preferable to limit the number of opportunities where an outsider can show that a given document passed through a given computer.

The design is based on a community of servers (which as a whole is termed the 'servnet') where each server hosts data from the other servers in exchange for the opportunity to store data of its own in the servnet. The system is designed to store data without concern for its popularity or controversial nature. Possible uses include storing source code or binaries for software which is currently under legal debate, such as the recent DeCSS controversy or other software with patent issues; storing political speech in an anonymous fashion for people afraid that tying their speech to their public persona will damage their reputation; or even storing more normal-looking data like a set of public records from Kosovo. The system is designed more for storage than for frequent querying — it is expected that in many cases, interesting material will be retrieved from the system and published in a more available fashion (such as normal web pages) in a jurisdiction where such publishing is more reasonable. Then the document in the servnet would only need to be accessed if the other sources were shut down.

The potential adversaries are many and diverse: governments, corporations, and individuals all have reason to oppose the system. There will be social attacks from citizens and countries trying to undermine the trust in the security of the system, as well as attacking the motivation for servnet node operators to continue running nodes. There will be political attacks, using the influence of a country's leaders to discourage use of the servnet. There will be government and legal attacks, where authorities attempt to shut down servnet nodes or arrest operators. Indeed, in many cases ordinary citizens can recruit the power of the government through lawsuits or subpoenas. Multinational corporations will hold sway over several countries, influencing them to pass similar laws against anonymous networks. There will be technical attacks, both from individuals and from corporations and national intelligence agencies, targetted either at the system as a whole or at particular documents or node operators, to reduce the quality of service or gain control of part of the network. Clearly the system needs to be designed with stability, security, and longevity in mind.

More formally, requirements beyond anonymity for a stable and useful system include:

- The system must be robust: loss of perhaps up to half of the participating servers should not imply loss of any documents. In addition, the amount of damage that a compromised or otherwise 'evil' node can perform should be limited. This might be accomplished by a trust network, where each node active maintains an opinion of other nodes, and nodes inform each other when they change an opinion.

- The system must be simple: complex protocols and heuristics invite security weaknesses. It must be

[1] `http://www.penet.fi/press-english.html`
[2] `http://www2.linuxjournal.com/articles/culture/007.html`

self-contained and based on realistic technological expectations. For instance, we cannot rely on a stable international electronic cash infrastructure.

- The system must be modular enough that components can be upgraded in-place without impacting functionality.
- The system must be decentralized: to maintain efficiency, security, and reliability, no single server or small subset of the servers should be a bottleneck anywhere in the protocol.
- The system is content-neutral: popularity or popular opinion of a document should not influence its duration in the servnet.
- The system must provide flexibility on a per-server level: server operators should be able to decide how paranoid or trusting they are, how many resources to provide to the servnet, etc.
- The components upon which the system relies must be free and open source, in the sense that modification and redistribution is explicitly permitted.
- The system must provide a mechanism for anonymously inserting a document into the servnet.
- The system must provide a mechanism for anonymously retrieving a document from the servnet, including verifying that the retrieved document is identical to the original document.
- The system must provide a mechanism for expiring documents: the duration of a document should be decided by the publisher when that document is published to the servnet, and the document should be available (and immutable) until that duration expires.
- The system must provide a mechanism for smoothly adding servers to the servnet without impacting functionality.
- The system must provide a mechanism for recognizing inactive or dead servers and eventually no longer using or querying them.

We assume that there will be some generous individuals out there who believe in the goals of the system and will donate some services. Notice that efficiency isn't on the list – we can afford to have more overhead (both in time and in bandwidth) if we get stronger anonymity out of it.

3 Expected Threats

In implementing the above goals, the system is going to come under a variety of attacks. We have attemped to consider most of these in advance. There are three primary modes of attack: on the communications medium, on the Servnet, and on individual files. We have considered all three of these, and take appropriate countermeasures. In particular, we expect the following sorts of attacks:

1. Attacks on the Communications Medium
   (a) The communications medium is either vulnerable to traffic analysis or involves a high degree of latency – and thus requires a great deal of space for storing messages in transit.
   (b) Traffic Analysis Attacks are normally difficult. They can be made easier by volunteering to become part of the communications medium. By becoming part of the communications medium, it would be possible to observe the traffic more closely and get a better idea of where it's coming from or where it's going.
   (c) Attack the time-synchronization protocol to reduce the latency of the system and ease traffic analysis.
   (d) Denial of service attack on the communications medium: flooding it with messages may well increase latency beyond tolerable limits, or actually crash some of the nodes.

2. Attacks on the Servnet
   (a) Go find a physical servnet node, and prosecute the owner based on its contents.
   (b) Physically destroy a servnet node. This attacks the trust-system as well as the integrity of the data in the network.
   (c) Become part of the Servnet. Earn trust, then betray it. We need an analysis of how much trust (in megabyte-days or comparable units) you need to betray in order to do real damage.
   (d) Attack the time-synchronization protocol to make files expire earlier than expected.
   (e) Make it appear that a servnet node has violated the protocols in such a way as to reduce trust in it.
   (f) Appear to violate the protocols. When someone points this out, show him wrong and accuse him of the above attack.
   (g) Claim that the servnet or mixnet concept is patented or otherwise illegal. Sue the Free Haven Project and any known node administrators.
   (h) Trade garbage into the servnet, gain trust, then delete what you get before it expires. We need an analysis of how much damage can be done per unit of useful work.
   (i) Attack the generosity of individuals: increase the personal cost of running a servnet or mixnet node, either by adding a monetary cost to moving large quantities of data around, or by adding a bad reputation such as "harboring terrorist data and kiddie porn".
   (j) Denial of service attack on the servnet: continued flooding of queries for data or requests to join the servnet may use up all available bandwidth and processing power for a node.
3. Attacks on a File
   (a) Attack to find the publisher of a document, or anybody else who has information about the document or its location.
   (b) Attack to find people interested in a particular document. Claim to have one, and see who requests it.
   (c) Swap until you have enough control over a file you object to, then destroy it.
   (d) Conspire to make a cause "unpopular". Convince servnet node administrators that they don't want to be hosting data for these unpopular causes, and that they should manually prune their data.
   (e) Insert false shares of a file into the servnet.

4 Related Works

4.1 The Eternity Service

Ross Anderson's Eternity Service[3] is the motivation for this project. It includes a wonderful vision of how the world might work in the future, in terms of data havens. On the other hand, it relies on a stable digital cash scheme, which is certainly not available today. Further, it has a stronger correlation between ability to store data into the system and amount of real-world capital available. While our proposal does have a loose correlation between available resources and amount of influence over the servnet, it's not nearly as direct.

[3] `http://www.cl.cam.ac.uk/users/rja14/eternity/eternity.html`

4.2 The `.cz` implementation

A team of students at Charles University in Czechoslovakia decided to implement their version[4] of Anderson's idea. They use a mixnet and made overall reasonable design decisions. Tonda Bene wrote his PhD thesis[5] on this design, and provided many more concrete explanations of a lot of the details that Anderson skips over in his original paper. However, there are a number of issues that I have with their implementation:

- They implement everything they need by themselves. They're not in a vacuum – for example, the world already has 'good enough' loose time synchronization applications such as xntpd. This leads to code bloat and too many layers of abstraction, which makes verifying security very difficult.
- Almost all of their files are binary. It would be easier to examine or modify their configuration files if they were in a human-readable and human-writable format.
- BSD-limited. Their code is not ported to Linux yet. First of all, the port is probably difficult if they haven't done it yet. Secondly, a well-designed system (eg in perl) should be extremely portable already – this does not bode well.
- A centralized (single or few) trusted bank system is assumed.

4.3 Adam Back's Usenet Proposal

Adam Back proposed[6] a simpler implementation of the Eternity Service, using the existing Usenet infrastructure to distribute the posted files all around the world. This is an excellent idea, but on the other hand this limits the participating servers to systems which already host usenet news – administrators don't choose to specifically participate in the eternity service, and so they may well choose not to carry the 'alt' groups that comprise the service. In addition, it uses normal usenet mechanisms for retrieval or posting, so anonymity might not be as strongly preserved on either side of the transaction. Additionally, people can post data to the service, and then prove that a specific server was hosting that data (since all of them are). This is crucial for laws where mere possession of the data is illegal.

4.4 Napster

The Napster service[7] is a company based around connecting people who are offering `.mp3` files to people who want to download them. While they provide no real anonymity and disclaim all legal liability, a very important thing to note about the Napster service is that it is highly successful. Thousands of people use Napster daily to exchange mp3's; if there were greater security (and comparable ease of use), I suspect that many thousands more would participate. Napster presents a very clear argument that the Internet community wants a service like this.

4.5 India Project

While the India Project[8] has no real anonymity measures and is designed with efficiency (high bandwidth reads) in mind, it includes an implementation of Rabin's information dispersal algorithm; this might be able to be lifted entirely and reused for our project.

---

[4] `http://www.kolej.mff.cuni.cz/~eternity/`
[5] `http://www.kolej.mff.cuni.cz/ eternity/Doc/index.html`
[6] `http://www.phrack.com/search.phtml?view&article=p51-12`
[7] `http://www.napster.com/`
[8] `http://www.eecs.harvard.edu/~india/`

4.6 Rewebbers

Ian Goldberg and David Wagner wrote an excellent paper[9] proposing a way of distancing the publisher of a document from the client that retrieves that material. Specifically, it involves web proxies called 'rewebbers' that forward requests throughout the system in a similar manner to how a mixnet operates. A client requests a particular URL at a proxy site, and that site decrypts the URL into the next site and URL to fetch, which in turns decrypts it into another, and after a while it reaches the 'end' of the rewebber chain, at which point the data streams back to the client.

The paper is beautifully written, and the logic is very sound. On the other hand, it doesn't offer as much anonymity as I would like – in the rewebber case, it can be proven that data has gone through a given host. It provides a relatively fast response time in exchange for this loss of anonymity, so it's a reasonable tradeoff.

4.7 Freenet

The Freenet project[10] is the biggest related work by far. They propose a servnet where each node can process queries (presumably you query a node close to you), and then data will migrate around the servnet to areas where it is more frequently requested. This is a sound idea, and I think it will work – one of the best benefits they provide is efficiency while still separating the publisher of the data from the client. On the other hand, we have a stronger definition of anonymity than they do: where they want to be able to keep the client and publisher from identifying each other, we want to keep the client and publisher hidden from each other and from other people, and we want to make it uncertain if data has travelled through any given node of the servnet.

In addition, they base how long a file should live on its popularity: frequently requested files get duplicated around the system, whereas infrequently requested files live in only a few places or die out completely. On the other hand, I can see circumstances where a file has a 12 month lifetime but only becomes popular in the last few months of its lifetime. Examples include photos of JFK Jr saluting his father, or a (timestamped) Idaho phone book that has those ten extra names that the FBI is suddenly accused of 'erasing'. Indeed, this is already happening – `http://www.umich.edu/~newsinfo/Releases/2000/Jan00/r012000e.html` describes a case where Yugoslav phone books are being collected "to document residency for the close to one million people forced to evacuate Kosovo."

They have noble goals. But their design is not as anonymous as certain types of data might need. Simply doing a request from a strict political environment will cause a copy of the data to migrate into that environment, giving law enforcement more reason to shut those servers down. On the other hand, it may well be that such environments will close down servers without requiring 'sufficient cause', so the political and legal arguments are meaningless. It might even be that a less anonymous system is more likely to be accepted in more parts of the world.

4.8 Zero Knowledge Systems

According to the ZDNet article[11], ZKS's 'Freedom 1.0' application[12] is designed to allow users to use a nym to anonymously access web pages, use IRC, etc. The anonymity comes from two aspects: first of all, ZKS maintains what it calls the Freedom Network, which is a series of nodes which pass traffic between themselves in order to hide the origin and destination of packets. These nodes send heartbeats of data, where random garbage is added to actual encrypted data to maintain a constant level of traffic between nodes. The second aspect of anonymity comes from the fact that clients purchase anonymous 'tickets' from ZKS, and exchange these tickets for nyms – this means that even ZKS isn't able to correlate identities with their use of their nyms.

On the one hand, the Freedom Network looks like it does a good job of actually demonstrating an anonymous mixnet that functions in real-time rather than the several-hour delays of more conventional

[9] `http://www.cs.berkeley.edu/~daw/classes/cs268/taz-www/rewebber.html`
[10] `http://freenet.sourceforge.net/`
[11] `http://www.zdnet.com/pcmag/stories/firstlooks/0,6763,2413285,00.html`
[12] `http://www.zks.net/`

mixmasters. On the other hand, it's solving the problem of the mixnet (anonymous communication) rather than the problem of data storage (anonymous storing and retrieving). In addition, the clients cost $50 (it uses only Freedom Network servers) and there is still a centralized system for obtaining nyms (which makes sense in the context of a company, but makes less sense in the context of a community-based service).

5 Some Proposed Mechanisms

There are a number of ways to satisfy the above design requirements. Some thoughts on the design and implementation of various components follow.

5.1 Mixnets

The primary way we intend to achieve anonymity is by using what are known as class II remailers (specifically, Mixmaster remailers), which are more commonly known simply as 'mixnets'. A mixnet is designed to forward a message from the sender to the recipient in a manner that makes it difficult to correlate sender to recipient. Lance Cottrell does an excellent job[13] of describing the anticipated threats, and how Mixmaster addresses each of these threats.

5.2 System Summary

Each server has a public key and one (or perhaps more) remailer reply blocks, which together can be used to provide secure, authenticated, anonymous communication with a given servnet node. Every machine in the servnet remembers information for some of the machines in the servnet (the public key, a remailer block or address, and some characterization of trust for each machine). 'Clients' are users (inside or outside the servnet) that want to retrieve a file. Only machines in the servnet are allowed to insert files into the network. The amount of storage space a given machine can use is limited by the amount of space it's willing to host – this is loosely enforced by other servers losing trust in a server that 'drops' data.

The servnet is dynamic: data moves from one server to another every so often, based partly on chance and partly on each server's trust of the others. Servers transfer data by trading. This means that the only way to introduce a new file into the system is for a given server to use (and thus provide) more space on its local system.

5.3 Storing

When a server on the servnet wants to introduce a new file into the servnet, it breaks it into shares using Rabin's information dispersal algorithm[14], and then for each share, it finds a machine on the servnet which it trusts and which is willing to make a trade, and then trades the share away. More specifically, it does the following.

1. Encrypt the file as desired (since any client can retrieve any file if it knows its lookup key (as below), we leave privacy at this level up to the user), and maybe divide the file into pieces if it's too long or pad it if it's too short (how to make this decision is described below under Trading).
2. Break the file F into shares $f_1, \ldots, f_n$ where any k shares are sufficient to recreate the file (say, $k = \frac{n}{2}$).
3. Generate a public/private key pair PK/SK to be used for signing each of the shares.
4. For each share, build a data segment "(Index, Data, Expiration, hash(Data+Expiration), PK)" and sign this segment with SK. In this case, index is some deterministic value which indicates which share this is.
5. Enter the shares into the local server's space.

[13] `http://www.obscura.com/~loki/remailer-essay.html`
[14] `http://www.eecs.harvard.edu/~india/ida/`

5.4 Retrieving

In order to retrieve a file, a client must first know the PK which was used to sign the shares — he learns this from a post to usenet or some similar external means (or because he was the original author of the document). From here, he must locate a servnet server that is willing to do the query for him. The server can be contacted over its remailer address, but presumably there will be public cgi's available which automate the process of doing the query. The client comes up with a key pair PK/SK for this transaction, as well as a remailer reply block. The servnet server does a broadcast of "('request', PK_{file}, PK_{client}, reply block)" (signed by this servnet node – is this signature necessary?) to all servnet nodes that it knows about. These broadcasts can be queued and then sent out in bulk to conserve bandwidth (since the mixnets are going to introduce some delay, adding a bit more by sending requests perhaps once an hour won't affect things much).

The broadcast request can optionally specify which share index is desired. This allows clients who only want one share to be able to query them specifically. For instance, a node wanting to confirm that a given share is still 'alive' might query only for that share.

Each server that receives the query will check to see if it any shares with the requested PK_{file}, and if it does it will encrypt each share in the enclosed public key PK_{client}, and then send the encrypted share through the remailer to the enclosed address. These shares will magically arrive out of the ether at their destination; once enough shares arrive, the client recreates the file and is done. If not enough shares arrive, then the client has failed to obtain the file (most likely, it is lost). Hopefully this case will never happen, since we'll take active steps to maintain availability for each share in the face of compromised servers.

5.5 Trading

Trades are considered 'fair' based on the two-dimensional currency of 'time*duration'. For instance, a 1 megabyte file that expires in 2 months is roughly equivalent to a 2 megabyte file that expires in 1 month. However, the economics don't have to work this way: it could well be that a given server requires a trade in its advantage before it's willing to complete a transaction, because it doesn't have much trust in the server offering the trade. Alternatively, it could be that a server would be willing to make trades where it ends up with more megabyte-days; such servers might build up a reputation of being easy to trade with.

When a server wants to make a trade (frequency of trade should be a parameter set by the server operator), it chooses another server from its list of known servers (based on trust and history), and offers a share along with some set of hints describing what it's willing to trade for. If the other server is interested, it responds with a share of its own. The negotiation is finalized (assuming the transport medium is reliable) when the first server sends an acknowledgement of acceptance back to the second.

If the offered share is not acceptable, a server can simply not respond, or it can respond with some description of why it refused the trade. This gives some indication of which trades might be accepted in the future.

It is suggested that when a server A trades a share to a server B, server A should keep a copy of the share around for a while, just in case B proves untrustworthy. This will increase the amount of overhead in the system by a factor of two or so, and provide greatly increased robustness. In this case, when a query is done for a share, the system responding should include a flag for whether it thinks it's the 'primary provider' of the data, or just happens to have a copy still lying around. This flag will help with accountability, as described below.

5.6 Expiration

Each share has an expiration date, after which it's ok to delete it. This means that if an operator wants to quit being a server, it's polite to wait until all the shares he has expires. (One way to hasten this would be to trade away for a really big share that expires soon). Expiration dates should be chosen based on how long the introducing server wants the data to last, compromising based on size of file and likelihood of finding a server willing to make the trade. Servers should be wary of accepting shares that they're not going to be able to trade away easily, because then they might be committed to keeping that file on their system until it expires.

5.7 Accountability and Redundancy

With the protocol described so far, shares could get swallowed by malicious nodes and nobody would ever notice. This has two ramifications: first of all, over time shares will disappear, eventually causing files to be unrecoverable. Secondly, malicious nodes can continue to be malicious — there are no mechanisms in place for identifying and excising bad nodes.

One solution to this would be for the publisher of each share to maintain a copy of his own, and periodically query the servnet for the share. If the share has disappeared, he will simply drop some of the data currently on his system and insert his backup copy of the share. (At this point, he might stop trusting the server to which he traded that share.) However, this makes the entire system very fragile, and slow to react to malicious nodes. A better solution would be to somehow keep track of the current location of each share. This would allow a given server to know exactly when a share disappeared, and which node was responsible for it at the time. Broadcasting knowledge of bad nodes would quickly update the servnet on suspected malicious nodes, limiting the amount of damage that can be done.

However, having a static location for keeping track of location information for each share defeats the purpose of having them scattered throughout the servnet. Specifically, this provides a centralized target for anyone wishing to learn more about a given share.

One possible solution to this is to make two identical copies of each share, and use the 'buddy system'. In this case, each share would be responsible for maintaining information about the location of the other share. When a share moves, it should send notification to the other indicating this move. (Specifically, the server sending the share should send notification, and then after the transfer the server receiving the share should send notification.) Periodically, it should query for itself, to make sure its buddy is still alive. If its buddy should stop responding, then the remaining share is responsible for announcing which node has responsibility for it when it disappeared.

Optionally, an abandoned share can spawn a new copy. This will make the service much more robust, but it's also possible that it could cause a population explosion of shares: if a share is out of touch for a little while but isn't dead, then both shares will end up spawning new copies of themselves. This is a strong argument for not letting shares replicate.

Another more subtle problem with having both shares on the move is that the mixnet has nontrivial latency. This means that a share might have already been traded away by the time notification arrives from the other share. Should nodes keep forwarders for packets they've recently traded away? How long should they keep the forwarders? Clearly this proposed solution needs some more work.

5.8 Trust Networks

This system is built off some amount of trust in the other nodes of the servnet. In particular, the protocol supports some enforcement of good practice by the other nodes, but there is still plenty of opportunity for nodes to obey the rules until they are sufficiently trusted, and then start breaking the rules in subtle ways. For instance, a node might sometimes fail to provide a receipt to a share's buddy during a trade, introducing confusion as to whether the other side is trying to trick the buddy into believing that the share had been traded away. More destructive would be to never query the buddy shares for any of the shares on your server, or even wrongly accuse a different node of losing that share. The list goes on and on. However, with careful trust management, each node ought to be able to keep track of which nodes it trusts; with the cushioning provided by Rabin's information dispersal algorithm, only a significant fraction of the nodes turning evil at once will result in actual loss of files.

5.8.1 Protocols for informing neighbors

One possibility is for each node to keep a status for the other nodes it knows about – 'known but not trusted', 'known but uncertain', 'known and trusted'. Each of these could carry a weight associated with it, and when nodes perform adequately or fail to perform adequately, the weight shifts. When a node crosses a threshold, then a message is broadcast to other nodes indicating this change. How the message is interpreted is left entirely up to each recipient, based on how much it trusts the broadcasting node's opinion.

5.8.2 Introducing new nodes

New nodes are introduced by contacting an established servnet node and asking to join the servnet. At this point, that node will provide some data for the new node to store (but not delete that data from its own system), and regularly check on the data. It will notify the rest of the servnet of a 'new but untrusted' node along with the reply block and public key for the new node. Gradually, it will begin to increase its trust in the node, and it will incrementally let other nodes know that it believes the new node to be trustworthy. The new node won't be able to trade away any shares until it finds a servnet node that trusts it enough to make the trade. This probation period should cut down on the number of unreliable nodes in the servnet.

5.8.3 Purging old nodes

Old nodes need to be removed from use after a while, since continued mail bombardment from the mixnet to a closed account will eventually slow down the service (as well as angering a lot of systems administrators). On the other hand, any operation which can remove nodes from the servnet must be handled very carefully, because it's easy to introduce security vulnerabilities that let an adversary disable a node.

One solution is to allow the trust network to gradually lose trust in a given node to the point that it doesn't ever send trade requests to it. On the other hand, broadcast requests will still go to it. This means that we might consider a sort of 'ping' query for a node, to make sure that it's still responding to its mail. On the other hand, this introduces new denial of service attacks where a node might answer pings but not other queries. I think the benefits of being able to notice dead nodes outweigh the denial of service concerns and the extra protocol.

5.9 Garlic Routing – how to broadcast over a mixnet

An optimization for making servnet broadcasts more reasonable is something we came up with called 'garlic routing'. Just as garlics look a lot like onions until you unwrap them, a garlic packet looks a lot like an onion packet until it gets to a certain hop in the mixnet. At this point, that hop decrypts the outer layer, and finds that there are now a group of onions inside, rather than just another onion. It then forwards these new onions (which might well be garlics, but it doesn't matter at that point) as usual.

The benefit that this provides is in shifting the responsibility of sending hundreds or perhaps thousands of messages from each server node onto the mixnet itself. This means that a given server node simply sends one message into the mixnet, and it can expect that all servers will receive this message.

This might not provide as much benefit as we would hope – it reduces the amount of outgoing mail by a factor of n, but it does not affect the amount of incoming mail (one per hour per server in the servnet, potentially). This idea also requires a change in the mixnets, to support this sort of routing. It remains to be seen if this would be a worthwhile approach to take.

6 Defenses against Expected Threats

In order to counter the anticipated attacks, we have evolved a number of defenses:

1. Physical Security: This is the responsibility of individual servnet node owners. In the event of failure here, we fall back on the robustness of the data storage system. *2a, 2b*

2. The Trust System: This serves two functions. First, it ensures that anybody entering the servnet provides useful work in proportion to the amount of damage they can do. Second, it provides a measure of accountability. Those who do good things are trusted to do good things again, and those who do bad things are trusted to do bad things again. *2c, 2e, 2f, 2h*

3. The Mixnet: This preserves the essential anonymity and untracability of the service. Its latency also protects against traffic analysis. *1a, 1b, 2b, 3a, 3b*

4. Cryptographic Signatures of Fragments: Strong cryptography protects us against spoofing of shares, as well as providing a convenient tagging mechanism. *3e*

5. Data Duplication: By dividing a file into many shares, not all of which are needed to reconstruct it, we gain protection against the destruction or duplicity of Servnet nodes. *2c, 2b, 2h, 3c*

6. Legal Protections: Information illegal in one place is frequently legal in others. Global oppression of a piece of information is relatively rare. The content-neutral policies mean that there is no reason to expect that the server operator has looked at the data he holds, which might make it more difficult to prosecute. *2a, 2g*

7. The Time-Synchronization Protocol: We rely on its security as an abstracted object. *1c, 2d*

8. Volunteerism and the Hacker Ethic: Owning a node of this service is still going to put an administrator in a potentially tricky situation. We rely on the Hacker ethic and a commitment to free information flow to provide volunteers *2i, 3d*

7 Modules

There are a number of aspects of this project that need to be done in order for the project to succeed. A number of them are relatively modular, meaning they would be good projects for a Urop or perhaps AUP.

7.1 User Interface

The server operator should have some convenient interface by which he can specify parameters for the server, as well as status and statistics. Similarly, the CGI's for doing retrieves and stores need to be convenient, and we will want to distribute a client-side application which automates the process of getting a reply block for the mixnet, building a key, and also reassembling the shares after they've arrived.

7.2 Mixnet

The theory for how a mixnet should work seems relatively mature. We need to research which code bases are available, and whether there are mixnets actually up and available that we can use. If there are, how difficult would it be to add the garlic routing idea? Do these mixnets need help with finding more nodes, or do they prefer a small number of very reliable nodes? Are we going to have to start our own?

Most mixnets are done via sendmail these days, I believe. This is very useful because some of the nodes can be behind firewalls. On the other hand, this means that we can't do optimizations that involve UDP or other more untraceable packet types.

7.3 Trust Network

Nodes will transmit messages to each other based on change in trust of any other node. Presumably such 'web of trust' networks have been studied before (eg PGP), and the algorithms for increasing and decreasing trust based on another node's opinion can be at least based on this prior research.

This is an extremely large project, based partly on researching other solutions and partly on actually designing and implementing a solution that fits for this scenario. Some very simple heuristics can be used at the beginning, but as the system gets used more heavily, more intelligent rules need to be determined and put in place.

One approach that we're considering is for each servnet node to simply remember how much useful service it has received from each other node, and only trusting nodes with at most the amount of useful work they've already provided. This means that we can limit the amount of damage a given node can do, and make sure that we'll always get at least 50% useful return out of each node.

7.4 Publicity

Without servers, the servnet is nothing. There are many people out there who would be willing to commit some space to the servnet (or perhaps to the mixnet as well, if we do not find a suitable mixnet in place already), providing the software is secure and cleanly written. Having contacts among the right communities will be crucial to making this idea actually work.

Other considerations include working with the other projects (like Freenet) that are implementing similar ideas, to make sure we don't end up overlapping work or alienating each other.

7.5 Documentation

And without advocacy and publicity documents describing the motivation and details of the servnet, again we will go nowhere. In addition to that form of documentation, we will also need to more fully justify the design goals and decisions that we made along the way. Part of this will be so new programmers can understand why we did things a certain way; part of this is so teams attempting a similar idea in the future can learn from our experiences; part of this is simply so we can keep everything straight once the project grows beyond a few people.

7.6 Main Skeleton

In addition to having a good mixnet and a good set of heuristics for building and maintaining trust, we need the actual code that will store files, move files, retrieve files, encrypt files, decrypt files, etc. This can probably be written as a couple of Perl glue scripts, but it needs to be very clean and extensible. We should focus on implementing as little as possible ourselves, and reusing as many external packages and applications as possible. We should assume that the systems we're running on will have modern applications (eg perl, sendmail, various crypto applications, etc) that we can rely on.

8 Acknowledgements

Quite a few people discussed the ideas and designs contained herein. They include:

- Brian Sniffen wrote the draft for the above sections on attacks and defenses.
- David Molnar and Seph Sokol-Margolis (as well as Brian) provided valuable comments and suggestions.
- Professor Rivest in his role as my thesis advisor has helped to chip away many of the extra layers of confusion I had added into my ideas. Also, he's very useful at providing perspective for why I choose certain design decisions, and which decisions are most important.