Exploring Anonymous Networking
MIT's Tor Project

Dennis Opacki
dopacki@adotout.com

August 2004

Table of Contents

Table of Figures

1.0 Introduction

When the average person thinks of security, cloak and dagger visions of "secret codes" come to mind. While encryption is an important component of security architecture, it only addresses a subset of the challenges inherent to communications security. As security and information technology professionals quibble over guidelines for appropriate encryption algorithms and key length, they sometimes miss the bigger picture. Modern commercial encryption technology tends to focus primarily on protecting the confidentiality of transmitted data, while the identities of the communicating parties, and the "when and how" of their communication remain unprotected.

To address this security shortfall, the United States Naval Research Laboratory is funding research into anonymous networking. Scientists at Massachusetts Institute of Technology have developed a free open source anonymity protocol based on a technology called "onion routing". Their project, "Tor", short for "The Onion Router", is the focus of this paper.

1.1 Goals

The guiding principle of my work was to evaluate the Tor Onion Routing software from the perspective of a private citizen seeking anonymous Internet access. Out of fairness, Onion Routing is a nascent technology, but my experience and results are representative of what an early adopter should expect.

There were three goals of my work:

- Deploy, test and evaluate the Tor network client in a production environment.
- Demonstrate that Tor alone is not sufficient to protect users against hostile web sites.
- Deploy a new Onion Router into the global Tor research network.

1.2 Methodology and Limitations

My work was conducted using the most currently available version of Tor, obtained from MIT's Free Haven website. As the software under active development, software released following my research may behave differently than the version I evaluated. Research is also ongoing to improve the performance and scalability of the public Onion Routing network.

Testing of the Tor client was performed primarily under Apple OS X on a live corporate network. While factors such as local network activity and client CPU affected the data I collected, they may provide a more realistic view of how Tor performs in production. Tor server testing was accomplished on a Debian Linux server, connected to the Internet via DS3 and Fast Ethernet connections directly into Internap’s Atlanta datacenter.

2.0 Background

2.1 Who Needs Anonymity?

When considering anonymous communications, the first things that come to mind are often abuse scenarios. Can anonymous networking be used for nefarious purposes? Absolutely. There is little to stop Internet hucksters from harnessing new technology as a means of perpetrating fraud, sending unsolicited email, and penetrating network defenses. Bear in mind that the likelihood of these individuals committing crimes is probably not tied to the availability of any one single technology. Like it or not, crime is how these folks make a living; countermeasures we take against them present little more than business continuity planning challenges on their part.

Despite an inevitable potential for abuse, anonymity technology has many legitimate purposes. Government and law enforcement are prime beneficiaries. With the rise in global terrorism, government intelligence agencies are increasingly tasked with monitoring terrorist groups on the Internet. A requirement of intelligence gathering efforts is often that the efforts go undetected. Using anonymous networking, government agents may visit the web sites of Islamist terrorist groups, without leaving a footprint. Furthermore, if there is widespread use of anonymous networking by terrorists, intelligence agencies gain an even greater advantage; site operators are unable to distinguish government intelligence browsing from that of the site’s intended audience.

Government can also benefit from anonymous networking by using it to create anonymous tip lines for whistleblowers, and anonymous means for citizens to submit leads in criminal investigations. Protecting whistleblowers becomes even more important when human rights abuses are involved. Many countries, such as China, aggressively monitor and censor Internet activity. An anonymous means of communicating dissenting information, and unfiltered first-hand reports of impropriety may prove a valuable tool for social change. It may even save lives.

Anonymous networking may benefit corporations by providing a safe means of conducting competitive research and business deals. Such technology may make it more difficult for an outside observer to detect a company performing due diligence activities, prior to completing an acquisition. Companies can also leverage anonymous networks to build truly private “virtual private networks” (VPNs). Today’s VPNs generally send routing information in-band, thus allowing an observer to build a map of which sites communicate with others. Furthermore, with some visibility into the volume of traffic transmitted between sites, an observer gains a better understanding of the significance of each site. Branch offices normally communicate with a limited number of remote sites, and do so in a predictable pattern. Main offices and data centers, on the other hand, exchange information with numerous branch offices. This technique of using in-band routing information, traffic volume and event coincidence to build a network map is collectively referred to as "traffic analysis". Resisting traffic analysis is a primary focus of anonymous networks.

Private citizens are a final beneficiary of anonymity. As Internet data repositories consolidate and coalesce, individual privacy is becoming a growing concern. One might say that anonymous networking has the potential to put the "private" back in "private citizen". A citizen need no longer disclose their identity to merely window shop in an online store. Individuals may thus be able to side step the barrage of direct marketing hurled at them by retailers. Though retailers will likely pursue circumvention measures, having invested large sums in profiling and tracking mechanisms, citizens can be on even footing for a time. Even if an "arms race" of tracking versus anonymity ensues, this is an acceptable outcome. As the cost of profiling increases, customer tracking and targeted marketing may no longer prove cost-effective, though still technologically feasible.

2.2 Anonymous Networking

2.2.1 Chaum Mixes

Generally considered the father of anonymous communications, David Chaum first proposed a system for anonymous email in 1981. [1] The system he proposed used a special mail server, called a *Mix,* to process email. Located between the sender and the receiver, the Mix used public key cryptography to act as a clearinghouse for anonymous emails. To send an email with Chaum's Mix, the sender must build a specially formatted message for transmission to the Mix per *Figure 1.*

$$\boxed{K1(\ R1,\ Ka(\ R0,\ M\ ),\ A\ )\ \mathbf{-\!-\!>}\ Ka(\ R0,\ M\ ),\ A.}$$

Figure 1. *Preparing an email for a Chaum Mix*

To accomplish this, the sender takes the Mix's public key (K1), and uses it to encrypt an envelope containing a random string (R1), a nested envelope addressed to the recipient, and the email address of the recipient (A). This nested envelope is encrypted with the recipient's public key (Ka), and contains another random string (R0), along with the body of the message being sent. Upon receipt of the encrypted top-level envelope, the Mix uses its secret key to open it. Inside, it finds the address of the recipient (A) and an encrypted message bound for (A). The random string (R1) is discarded.

Chaum Mixes further improve anonymity by processing messages in batches at random intervals. By introducing a random delay, coincidence and timing attacks are made more difficult. From an outside perspective, a Mix looks like a black box with myriad inputs and outputs. As long as the integrity of the box is assured, tracking a specific message through the Mix is a difficult challenge. Mixes have proven an effective means of providing anonymous email; research has even enabled creation of cascading networks of Mixes for an additional level of assurance. Mixmaster is an open source implementation of a Mix cascade. [2]

2.2.2 Onion Routing

Though Chaum Mixes enable anonymous email communications, the latency introduced through batch processing of messages make the model poorly suited for real time communication. To extend anonymity to web browsing, new research was necessary. In the mid 1990's, three scientists at the Naval Research Laboratory, Center for High Assurance Computer Systems, proposed a means of anonymous real time communications. [3] Goldschlag, Reed and Syverson extended Chaum's work on layered cryptographic structures to build a model, whereby special data structures called *Onions* were routed to their destination via a network of *Onion Routers* (ORs). Thus, *Onion Routing* was born.

In Onion Routing, the initiator of communications has a directory of candidate ORs available on the network. The initiator selects a number of ORs at random to serve as relay stations along the path to the destination; we'll call these (X), (Y) and (Z). Public encryption keys for these nodes are available in the directory as well (PKx), (PKy) and (PKz). Using these public keys, the initiator prepares an onion for transmission per *Figure 2*.

$$\boxed{PKx(Ex, Y, Kx, PKy(Ey, Z, Ky, PKz(Ez, Kz, M)))}$$

Figure 2. *Preparing an Onion for transmission*

This seeming complex data structure is actually quite simple. Let's start our way at the onion's core, and work outwards. The core message (M) contains the communication from the initiator to the recipient. Together with an expiration time (Ez), to protect against message replay, and a set of symmetric keys (Kz) to allow the recipient to communicate with (Z), items are encrypted with the public key (PKz) belonging to the last hop (Z) before the recipient. The resulting encrypted message is combined with another expiration time (Ey), and another set of symmetric keys (Ky), and encrypted with the public key (PKy) of the middle hop (Y). Finally, the whole structure is combined with the expiration time (Ex) for the first hop (X), and a final set of symmetric keys (Kx), and encrypted with (X)'s public key (PKx). Each layer thus contains information on the next hop, a set of keys for communicating with the next hop, and a payload to forward on.

Once the initiator has built an onion, it sends it on its way. As each successive hop receives the message, it strips off its corresponding layer of encryption, revealing the layer beneath. Thus, each OR only knows about the hop before it, and the hop afterwards. Only the initiator has full knowledge of the communication.

NRL's Onion Routing proved an effective means of enabling anonymous real time communications. However, in 2001, the United States Patent Office awarded the United States Navy patent number 6,266,704, covering an "Onion routing network for securely moving data through communications networks". [4] Although the Navy expressed a willingness to discuss licensing terms, few individuals were willing to trust the government to provide them anonymity, especially if they had to pay for it.

Recognizing that it is difficult to remain anonymous when you are the only one using the protocol, the Navy elected to fund research on a second generation Onion Routing protocol, this time determined that the protocol would be free to flourish in the public domain. The open source *Tor* project was launched as part of MIT's Free Haven project. [5]

2.2.3 The Onion Router (Tor)

Tor, the second generation Onion Routing protocol, offers significant enhancements over previous implementations. Specifically, Tor offers improvements in forward secrecy, performance, and compatibility. [6] Government approved encryption algorithms were also selected, making Tor suitable for unclassified government use.

Forward secrecy is enhanced through an updated onion structure. Whereas previous implementation used a layered onion, encrypted with long-lived asymmetric keys, Tor uses "telescoping circuits", as shown in *Figure 3*. This new circuit type allows the initiator to negotiate a short-lived session key with each successive node along the path. Diffie-Hellman is used for key exchange, and the negotiated session keys are AES. This approach improves forward secrecy by eliminating the risk that nodes along the circuit could be compromised and forced to decrypt captured traffic. Once a circuit is torn down, they keys are discarded, thus also offering protection from replay attacks.

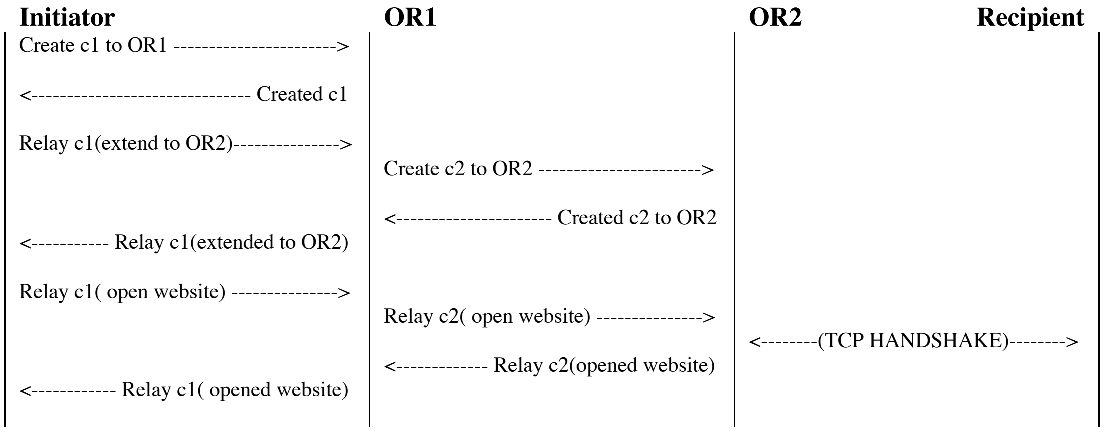

Initiator
OR1
OR2
Recipient
Create c1 to OR1 ----------------------->
<------------------------------ Created c1
Relay c1(extend to OR2)--------------->
Create c2 to OR2 ----------------------->
<---------------------- Created c2 to OR2
<----------- Relay c1(extended to OR2)
Relay c1(open website) --------------->
Relay c2(open website) --------------->
<--------(TCP HANDSHAKE)-------->
<------------- Relay c2(opened website)
<------------ Relay c1(opened website)

Figure 3. *Establishing a telescoping circuit with Tor*

The telescoping circuit model brings two important benefits on top of forward secrecy. First, older Onion Routing implementations required a new circuit be created for each application request. Due to the overhead associated with asymmetric cryptographic operations, this made normal web browsing computationally expensive. Tor breaks this pattern by multiplexing multiple connections over a single circuit. This allows the initiator to set up a single longer-lived path across the Onion Routing network, which she can use to access multiple destinations. Tor's extensive use of less CPU-intensive symmetric key cryptography yields further performance advantages.

The second advantage of the telescoping circuit model is that it allows "leaky pipe" routing. As any node along the route is a candidate exit point, traffic analysis is particularly difficult. Individual administrators of Tor Onion Routers determine whether their ORs are used as exit nodes, and what type of traffic they will allow to exit. If administrators are concerned they may incur liability as a result of hostile traffic exiting their OR, they may configure the OR as a "middleman" mode, thus disallowing any traffic from exiting via their network.

Tor has improved compatibility over previous Onion Routing implementations, as it does not require application customization. Instead, Tor uses the industry-standard SOCKS protocol. Many modern network-aware applications come pre-installed with SOCKS capabilities. Previous Onion Routing implementations required application patching, or special network "shims", many of which were never implemented.

3.0 Project

3.1 Tor Client Evaluation

3.1.1 Client Installation

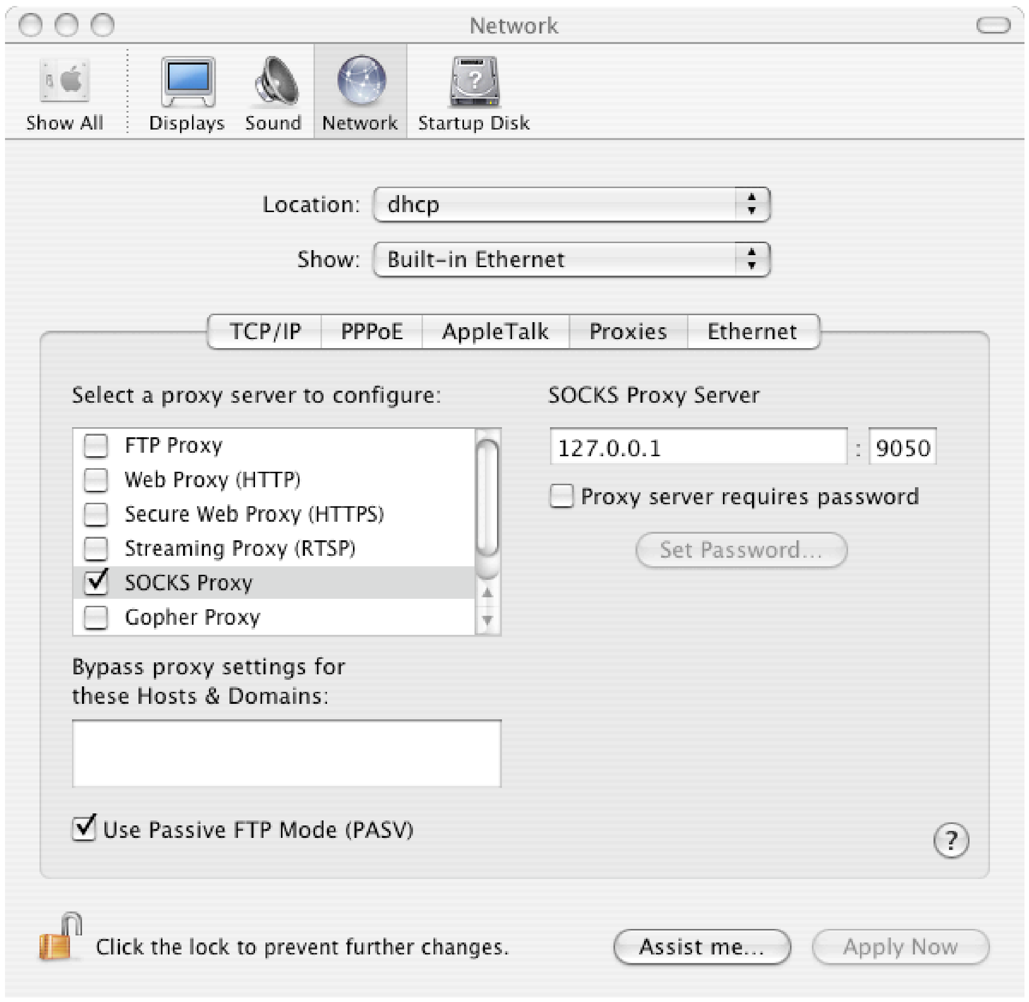
For my evaluation of the Tor client, I obtained the most current release from the Free Haven project. I built Tor from source code under both Apple OS X and Debian Linux to test cross-platform portability, though Apple OS X was my primary platform for evaluation. Installation was accomplished as follows:

```
wget -q http://freehaven.net/tor/dist/tor-0.0.8pre3.tar.gz
tar zxvf tor-0.0.8pre3.tar.gz
cd tor-0.0.8pre3
./configure
make
sudo make install
/usr/local/bin/tor
```

After starting the Tor client, I verified it properly bound to TCP port 9050, and listening was for incoming connections:

```
% telnet localhost 9050
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
▯
```

Next, I configured my core operating system to direct outbound Internet web browser connections through Tor. I accomplished this by defining a new SOCKS proxy in my network configuration menu, telling the operating system that the new proxy can be found at "127.0.0.1", port "9050".



Location: dhcp

Show: Built-in Ethernet

Select a proxy server to configure:

- [ ] FTP Proxy
- [ ] Web Proxy (HTTP)
- [ ] Secure Web Proxy (HTTPS)
- [ ] Streaming Proxy (RTSP)
- [✓] SOCKS Proxy
- [ ] Gopher Proxy

SOCKS Proxy Server

127.0.0.1 : 9050

- [ ] Proxy server requires password

Bypass proxy settings for these Hosts & Domains:

- [✓] Use Passive FTP Mode (PASV)

Click the lock to prevent further changes.

Figure 4. *Configuring OS X to use a SOCKS proxy*

To verify outbound web browsing connections were traversing the Onion Routing network, I performed a brief packet capture using 'tcpdump' while requesting http://www.cnn.com.

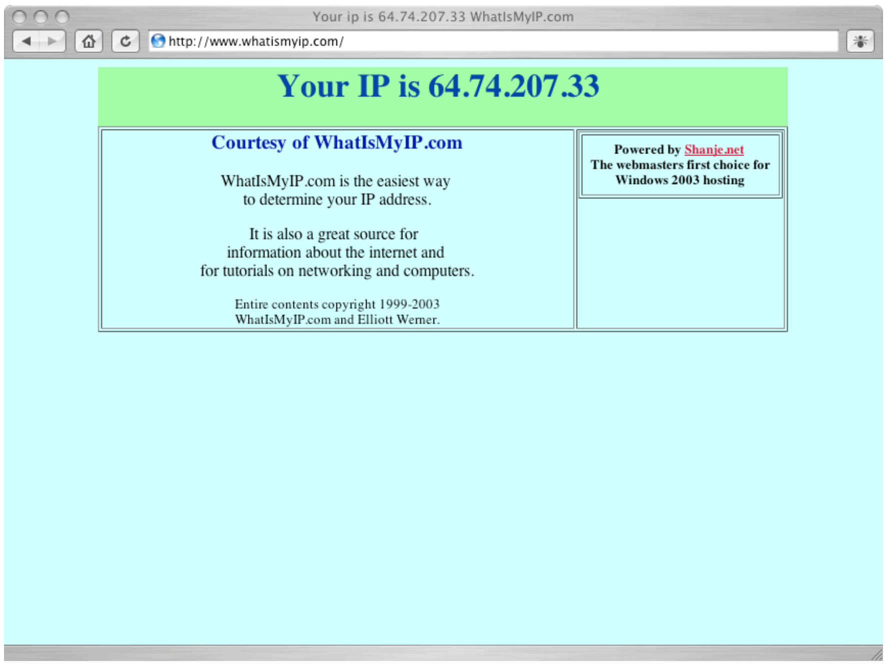
```
% sudo tcpdump -n -X -p tcp and port 9001
16:08:46.537677 IP 63.251.67.151.64951 > 128.232.110.15.9001: P
740009379:740009949(570) ack 1899141307 win 65535
0x0000   4500 0262 a079 4000 4006 2593 3ffb 4397        E..b.y@.@.%.?.C.
0x0010   80e8 6e0f fdb7 2329 2c1b a5a3 7132 98bb        ..n...#),...q2..
0x0020   5018 ffff 74de 0000 1703 0100 18f4 407a        P...t.........@z
0x0030   d64f 85fc 712b e6d7 4c59 d61a 0b94 8ccb        .O..q+..LY......
0x0040   ba62 d8d9 af17 0301 0218 6fba 3b08 8b5e        .b........o.;..^
0x0050   90ee                                           ..
16:08:46.628423 IP 128.232.110.15.9001 > 63.251.67.151.64951: . ack 570 win
17100
0x0000   4500 0028 5a37 4000 3006 7e0f 80e8 6e0f        E..(Z7@.0.~...n.
0x0010   3ffb 4397 2329 fdb7 7132 98bb 2c1b a7dd        ?.C.#)..q2..,...
0x0020   5010 42cc fbb6 0000 0000 0000 0000             P.B...........
```

The above capture shows my test machine at 63.251.67.151 (dhcp67-151-acs.acs.internap.com) exchanging data with 128.232.110.15, which resolves to "ephemer.al.cl.cam.ac.uk", on TCP port 9050. By consulting the Tor Onion Router directory at http://moria.seul.org:9031/, I confirmed that 128.232.110.15 is a valid OR.

```
router ephemer 128.232.110.15 9001 9050 0
platform Tor 0.0.7.2 on Linux ephemer i686
published 2004-08-16 14:42:01
bandwidth 800000 10000000
onion-key
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAOO0aH0wEWbFT58+OJAmTeWSa2Z3PDMY/yo82nVV/7RDRiHVLWX/0CNS
S03b/XC5/QIKRDFkpXMJ99vYUglcR/CvXfOozuj6zUJjcxyUnfbNnQVg+pe2iHvg
RMmOHjFDIvFl7dQ9OFW/4OcZTsAATEweUvtF8gY5eMeGux+cnWQdAgMBAAE=
-----END RSA PUBLIC KEY-----
signing-key
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAOgeu2O66R3fhcmc38jNs9poyDQ4j+INc3UzNihk+5XuIyETRfP27ITu
nvTnE1/PdKM20IeUB59OtS8WFgMxltK3EJlLlUyyx4JW7d1QBIq/ioXJQaqoigU1
LqaV+sgCDmK17hxmRSgGv3tM3Gcht4ogm3Gvh+U7rQiFChxDgWtVAgMBAAE=
-----END RSA PUBLIC KEY-----
reject *:*
router-signature
-----BEGIN SIGNATURE-----
I4kCwVXqV+Yn2HqjoFxm624ZjcjvPIIHEclKSIzR4c7cfvgJjMtx5jK1rLolj4MY
wrVk114TSJJcxCpgHhg+tQf5Yqe1IIjiha4GuXvov0z1F5DdBcspatIXBbAI6HQq
wcNYsryfKa8MfrvAZm5a9BVNJBWdfQt8K2fuR2qT88U=
-----END SIGNATURE-----
```

3.1.2 Anonymity Testing

To test whether Tor is properly hiding the IP address of my test machine, I consulted http://www.whatismyip.com, a free Internet diagnostic site. The site uses environment data from the web server to determine the source address of incoming web connections. This site confirmed that we were properly traversing the Onion Routing network, and exiting through 64.74.207.33, which coincidentally happens to be the Onion Router whose setup we will discuss in Section 3.2.


Your ip is 64.74.207.33 WhatIsMyIP.com
http://www.whatismyip.com/

Your IP is 64.74.207.33

Courtesy of WhatIsMyIP.com

WhatIsMyIP.com is the easiest way to determine your IP address.

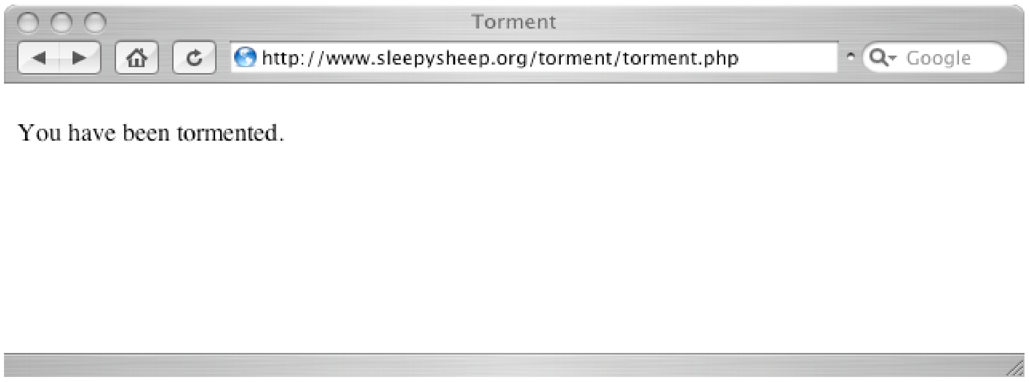
It is also a great source for information about the internet and for tutorials on networking and computers.

Entire contents copyright 1999-2003 WhatIsMyIP.com and Elliott Werner.

Powered by Shanje.net The webmasters first choice for Windows 2003 hosting

Figure 5. *Verifying anonymous Internet access*

As part of Tor's specification is that protocol sanitization is offloaded into an external application, the project maintainers strongly recommend that an anonymous proxy such as Privoxy [7] be installed in line with Tor. By sanitizing the web traffic exchanged with a hostile remote site, Privoxy may prevent your local browser from being tricked into divulging information about the local host. In this respect, active content is particularly dangerous. Under the default security policies of many web browsers, Java applets can be exploited to force leakage of sensitive or identifying information. Consider *Figure 6*.



You have been tormented.

Figure 6. *Seemingly innocuous website*

What at first glance appears to be an innocuous HTML webpage actually contains a hidden Java applet. This applet forces the web browser to reveal its IP address to the server as follows.

```
<html>
<head>
  <title> Torment </title>
</head>
<body>
<!-- No IP address, redirect. //-->
<applet code="MyAddress.class" width=0 height=0>
<param name="URL" value="torment.php?ip=" />
<param name="ACTION" value="AUTO" />
<!--<param name="TARGET" value="_STEALTH" /> //-->
</applet>
<p>You have been tormented.</p>
</body>
</html>
```

The MyAddress Java applet is freely available from http://reglos.de/. Although source code was not provided with the applet, I successfully decompiled it using Pavel Kouznetsov's JAD decompiler. The resulting source code reveals the following core logic:

```
String s1 = "unknown";
String s2 = getDocumentBase().getHost();
int i = 80;
if(getDocumentBase().getPort() != -1)
    i = getDocumentBase().getPort();
    try{
        String s = (new Socket(s2, i)).getLocalAddress().getHostAddress();
        if(!s.equals("255.255.255.255"))
           s1 = s;
    }
    catch { ..exception code.. }
}
```

By default, a browser's Java Virtual Machine executes applets in a "sandbox". The sandbox is a controlled environment with limited access to system resources; the only raw network connections a sandboxed applet may make are back to the site the applet came from. This is where our vulnerability lies. MyAddress uses the Java network classes, which are permitted access to the underlying stack. The applet obtains the IP address of the site it was downloaded from, then opens a TCP socket. It then examines the local side of the newly created TCP socket, and obtains the local address. With real IP address in hand, the applet loads a page from the original web server, thus depositing the actual client IP address in the server logs.

```
69.20.9.201 - "GET /torment/torment.php HTTP/1.1" 200 344
69.20.9.201 - "GET /torment/MyAddress.class HTTP/1.1" 200 4920
69.20.9.201 - "GET /torment/torment.php?ip=63.251.71.47 HTTP/1.1" 200 180
```

Although Privoxy provides a level of traffic filtering on top of Tor, Privoxy is not able to filter traffic that is not directed to it. Thus, even with diligent filtering of client/server traffic, Privoxy may not be able to fully protect against hostile active content. If optimum anonymity is required, active content should be disabled within the browser. An interesting area of future research would be to construct a hostile Java applet, which initiates a direct UDP connection back to the server, where a port listener directly records real IP addresses and alerts when the address is different from that in the server logs. This would serve as a rudimentary detector of anonymous browsers.

3.1.3 Installing Privoxy

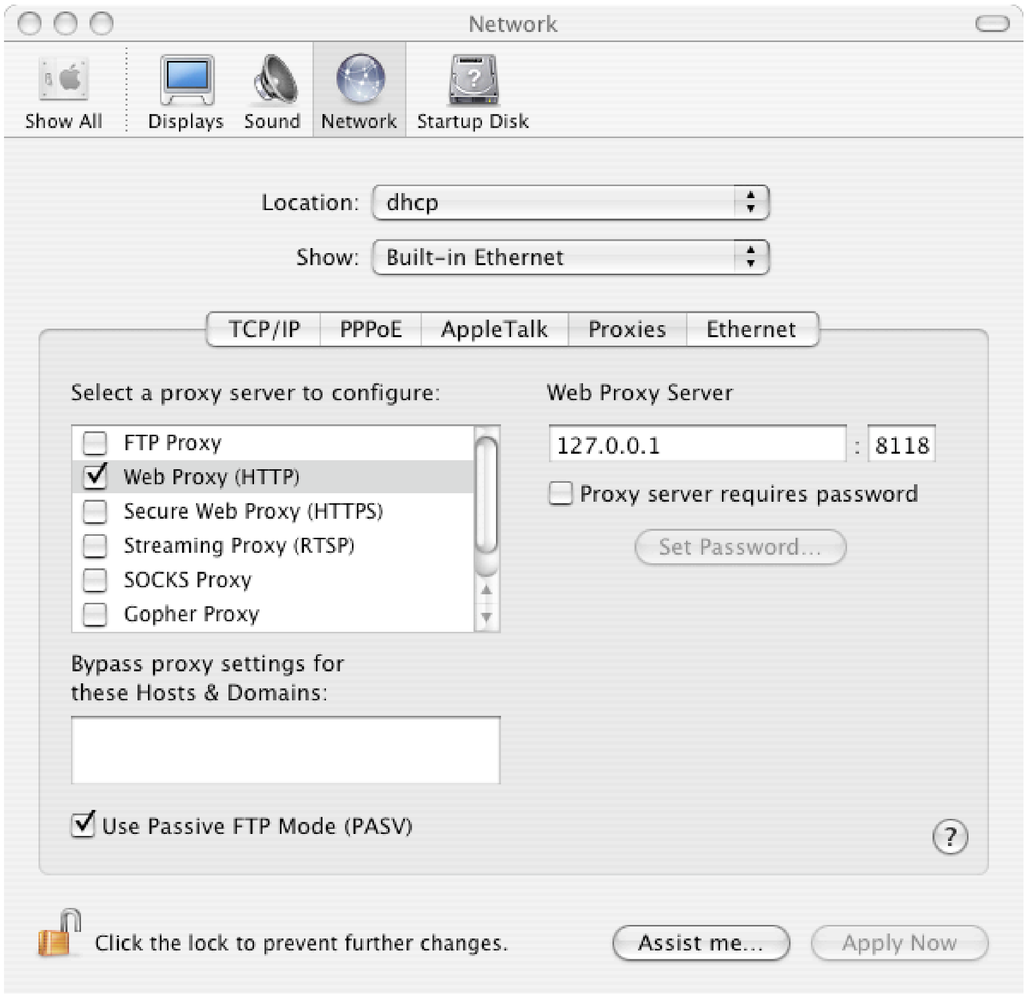
In the interest of gaining the most complete understanding of the Tor user experience, I obtained and installed the recommended Privoxy packages. I was able to obtain Privoxy as an OS X binary package, which made for an expedient installation. To configure Privoxy for Tor, I added a single line to Privoxy's configuration file, in "/Library/Privoxy/config".

```
forward-socks4a / localhost:9050 .
```

The above configuration directive causes all incoming traffic to Privoxy to be forwarded through Tor, on its way to the Internet. After starting Privoxy, I it was listening at TCP port 8118.

```
% telnet localhost 8118
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
▯
```

Lastly, I updated the operating system configuration to replace the SOCKS proxy definition I created when installing Tor, with a http proxy definition pointing to "127.0.0.1", port "8118".



Location: dhcp

Show: Built-in Ethernet

Select a proxy server to configure:

- [ ] FTP Proxy
- [✓] Web Proxy (HTTP)
- [ ] Secure Web Proxy (HTTPS)
- [ ] Streaming Proxy (RTSP)
- [ ] SOCKS Proxy
- [ ] Gopher Proxy

Web Proxy Server: 127.0.0.1 : 8118

- [ ] Proxy server requires password

Bypass proxy settings for these Hosts & Domains:

- [✓] Use Passive FTP Mode (PASV)

Figure 7. *Configuring OS X to use Privoxy*

The resulting path to remote web sites was as follows:

```
Browser → Privoxy → Tor → ORx → ORy → ORz → Destination
```

3.1.4 Performance Testing

With the recommended Tor configuration in place, I conducted performance testing. Focused on determining whether latency introduced through complex proxy chaining and Onion Routing were sufficient to discourage widespread adoption of Tor, I collected data in three network configurations:

- Baseline – No proxy, direct connection to remote site
- Tor – Connections made using Tor as a SOCKS proxy
- Tor+Privoxy – Connections are passed through Privoxy and Tor

To gather my data, I wrote a Perl wrapper around the Unix “curl” command. The wrapper allowed me to specify a website to test and the number of samples to gather.

```
#!/usr/bin/perl

use strict;
use Getopt::Long;

sub CURL {'/usr/local/bin/curl -s -o /dev/null '}
sub FMT  {'-w \'%{time_total}\n\' '}

my $proxy = "";
my $socks = "";
my $url = "";
my $samples = 1;
my $command = CURL.FMT;

GetOptions("proxy=s"=>\$proxy, "socks=s"=>\$socks, "url=s"=>\$url,
"samples=s"=>\$samples);

$proxy && ($command.="--proxy ".$proxy);
$socks && ($command.="--socks ".$socks);
$url && ($command.=' '.$url);

for (my $i=0; $i < $samples; $i++) {
        system($command);
}
```

I selected CNN’s home page (http://www.cnn.com) as my testing target, as it is a well-known site with sufficient infrastructure to withstand my testing. I selected a sample size of one thousand samples. New requests were sent immediately following the return of the previous test. Though this is not representative of the browsing behavior a human might exhibit, all three tests were performed using the same method. Thus, I was still able to draw useful conclusions from the data. All performance testing was performed on a full-duplex Fast Ethernet link, connected directly to Internap’s Atlanta facility.

Figure 8 represents the normal page load time for simple http requests sent directly to the target site. Access time is generally near 0.3 seconds, with peak access time of 1.4 seconds. Starting at sample 400, I began seeing some irregularity, most likely due operating system “housekeeping” activity on the test machine, which I did not adequately control for. Baseline performance was still excellent, and it is unlikely a human user would notice such a small variation in latency.

In *Figure 9* we examine the latency introduced by directing web requests through Tor. Though we continue to see a number of transactions being completed within one second, our general latency has increased dramatically. The "wave" patterns present in the data likely represent Tor creating new circuits to vary the transmission path, roughly every minute. Most responses occur in less than five seconds, though some take almost twenty seconds. Recall that I am only requesting a single html page, and that human browsing activity spawns a request for each object in the page. Latency of five seconds per object will surely be noticeable by a human, and twenty-second object loads will surely be aggravating. Still, given some assurance of anonymity, this level of performance still proves usable.

When we introduce Privoxy into the equation the browsing experience is further degraded. Data collected with Tor and Privoxy, presented in *Figure 10*, shows latency under five seconds for the majority of requests, though our peaks are now significantly higher. A number of requests took more than twenty seconds to complete, and one request took close to thirty. Only the most determined user would to tolerate a connection of this nature. Others may interpret such delays as a network interruption or software failure.

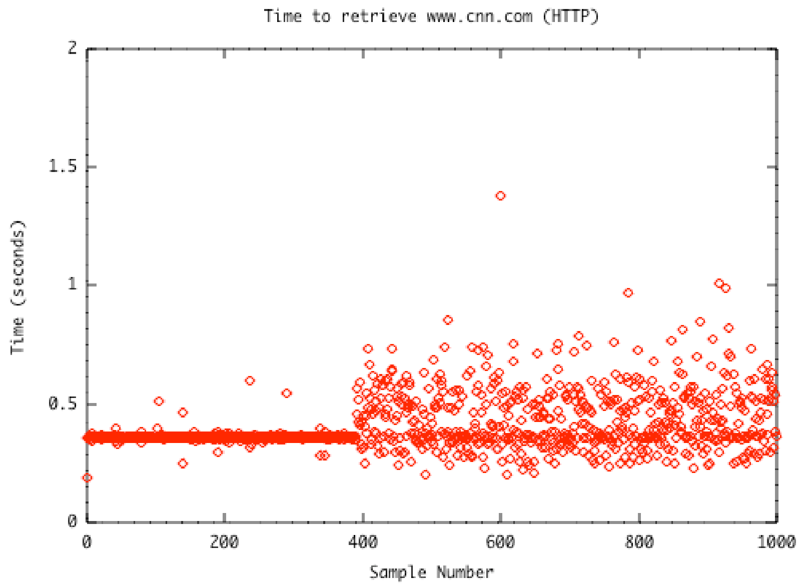


Figure 8. *Accessing the CNN website without Tor*

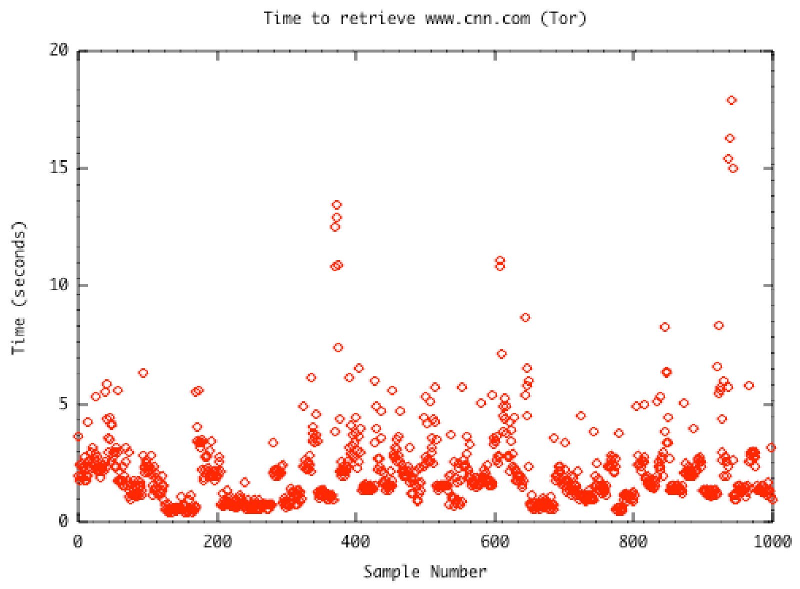


Figure 9. *Accessing the CNN website with Tor*

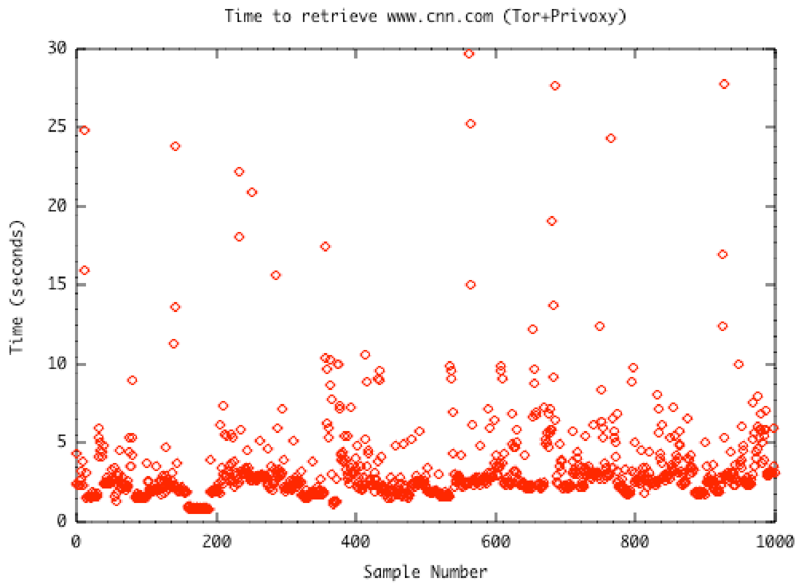


Figure 10. *Accessing the CNN website with Tor and Privoxy*

3.2 Tor Server Installation

Building a Tor Onion Router and adding it to network did not prove difficult. The Tor server software was included in the same distribution I used in Section 3.1.1. I built the software under Debian Linux as follows:

```
wget -q http://freehaven.net/tor/dist/tor-0.0.8pre3.tar.gz
tar zxvf tor-0.0.8pre3.tar.gz >/dev/null
cd tor-0.0.8pre3
./configure
make
sudo bash
make install
cd /usr/local/etc/tor
cp torrc.sample torrc
mkdir /usr/local/var/lib/tor
cd /usr/local/var/lib/tor
sudo chmod nobody:nogroup .
sudo -u nobody /usr/local/bin/tor
```

Although similar to building and installing the Tor client, adding a node to the Tor network required several additional steps. When I started the Tor client for the first time, it automatically generated a set of Onion Router keys, which were placed in "/usr/local/var/lib/tor". In order for the new node to be accepted into the network, I had to send the node's "fingerprint" to the MIT network administrator. I accomplished this by mailing the contents of the "fingerprint" file to tor-ops@freehaven.net.

```
root@mirror1:/usr/local/var/lib/tor# cat fingerprint
inap1 49BD 25ED C7D3 21C7 6A0B CC89 AEE7 1EF6 E6D4 C58D
```

As participating in an anonymous network necessarily implies a certain degree of integrity and ethics, I avoided direct observation of Onion Router traffic. To verify operation of the new node, I checked the process table and verified Tor was listening on the proper port (TCP/9050). *Figure 5* further confirms operation.

```
root@mirror1:/usr/local/var/lib/tor# ps auxwww| grep tor | grep -v grep
nobody   25324 11.9  0.1  4536 2960 ?        S    16:02   0:03
/usr/local/bin/tor
nobody   25325  0.0  0.0  2700  916 ?        S    16:02   0:00
/usr/local/bin/tor
nobody   25326  0.0  0.0  2708  980 ?        S    16:02   0:00
/usr/local/bin/tor
nobody   25327  0.0  0.0  2716  988 ?        S    16:02   0:00
/usr/local/bin/tor
nobody   25329  0.0  0.0  2788 1540 ?        S    16:02   0:00
/usr/local/bin/tor

root@mirror1:/usr/local/var/lib/tor# telnet localhost 9050
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
▯
```

4.0 Summary

4.1 Findings

Though I was successful in deploying and using the Tor client for anonymous browsing, I was disappointed with the extent to which network performance was degraded. When browsing with Tor alone, network response time hovered around five seconds per request. Though sluggish, this level of performance is still acceptable if resistance to traffic analysis is an important requirement.

However, Tor is not designed to protect the initiator from hostile content providers, as I demonstrated by analyzing the MyAddress Java applet. If this additional level of protection and anonymity is required, the user must deploy a third-party anonymous proxy such as Privoxy. Performance testing, with Privoxy, yielded disappointing results. Spikes in page load time reached the mid-twenty second range multiple times, peaking at thirty seconds. Such latency would likely be interpreted by the initiator as an inability to connect, and would frustrate most users. Perhaps through a dial-up or other low bandwidth connection, response time would be less of an issue.

The high latency I experienced in my tests was likely the result of Tor network administrators allowing particularly busy or under-provisioned nodes into the Tor routing fabric. Though site autonomy is advantageous from an anonymity perspective, this practice may be self-defeating. If network performance is unacceptable, fewer people are likely to use the network. As we saw with Chaum's Mixes, anonymity is derived from the ability to hide in a crowd. No crowd, no anonymity. Thus, acceptable network performance should be a consideration.

Adding new Onion Routers, homed off of a high-bandwidth network is a way to help improve performance. Creating a new node proved an extremely straightforward and well-documented process. Exit policy of Tor ORs is flexible, allowing nodes to easily conform to local security policy. As the number of available ORs in the Tor network likely has a direct impact on network performance, I recommend that network administrators consider bringing Tor ORs on line. The process is painless, and the benefits to society are plentiful.

Onion Routing is a fascinating technology, and one that offers sufficient promise to bear continued observation. However, in its current state, the global Tor network does not provide sufficiently good performance to enable seamless use. More research is likely necessary to improve Tor's performance and scalability before widespread adoption is practical.

4.2 Future Work

There two future areas of research I would like to pursue related to Tor:

- Further research into hostile active content as a means of defeating anonymous networking technology
- Investigation of route optimization technology as a potential means of improving both network response and Onion Router autonomy

Though some research has already been performed on optimizing OR selection for location diversity [8], perhaps building on their paper's conclusions would make an interesting project.

5.0 References

[1] **Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms**
David Chaum, *Communications of the ACM*, Volume 24, Number 2, February 1981

[2] **Mixmaster Home Page**
Internet Site, August 2004
http://mixmaster.sourceforge.com

[3] **Hiding Routing Information**
Goldschlag, Reed and Syverson, Naval Research Laboratory
Workshop on Information Hiding, Cambridge, UK, May 1996

[4] **United States Patent #6,266,704**
Reed, et al., July 24, 2001
http://tinyurl.com/4bp2u

[5] **Free Haven Project**
Internet Site, August 2004
http://www.freehaven.net

[6] **Tor: The Second-Generation Onion Router**
Dingledine, Mathewson, and Syverson
Proceedings of the 13th USENIX Security Symposium, August 2004

[7] **Privoxy Home Page**
Internet Site, August 2004
http://www.privoxy.org/

[8] **Location Diversity in Anonymity Networks**
Feamster and Dingledine, 2004
http://www.freehaven.net/doc/routing-zones/routing-zones.ps