Darknets and hidden servers:
Identifying the true IP/network identity of I2P service hosts

Adrian Crenshaw

Abstract:

This paper will present research into services hosted internally on the I2P anonymity network, focusing on I2P hosted websites known as eepSites, and how the true identity of the Internet host providing the service may be identified via information leaks on the application layer. By knowing the identity of the Internet host providing the service, the anonymity set of the person or group that administrates the service can be greatly reduced if not completely eliminated. The core aim of this paper will be to test the anonymity provided by I2P for hosting eepSites, focusing primarily on the application layer and mistakes administrators and developers may make that could expose a service provider's identity or reduce the anonymity set[1] they are part of. We will show attacks based on the intersection of I2P users hosting eepSites on public IPs with virtual hosting, the use of common web application vulnerabilities to reveal the Internet facing IP of an eepSite, as well as general information that can be collected concerning the nodes participating in the I2P anonymity network.

Introduction:

I2P[2] is a distributed Darknet using the mixnet model, in some ways similar to Tor, but specializing in providing internal services instead of out-proxying to the general Internet. The name I2P was original short for "Invisible Internet Project", although it is rarely referred to by this long form anymore. It is meant to act as an overlay network on top of the public Internet to add anonymity and security.

The primary motivation for this project is to help secure the identity of I2P eepSite (web servers hidden in the I2P network) hosts by finding weaknesses in the implementation of these systems at higher application layers that can lead to their real IP or the identity of the administrator of a service being revealed. We also wish to find vulnerabilities that may lead to the anonymity set being greatly reduced, and compensate for them. Exposing these weaknesses will allow the administrators of I2P eepSite services to avoid these pitfalls when they implement their I2P web applications. A secondary objective would be to allow the identification of certain groups that law enforcement might be interested in locating, specifically pedophiles. These goals are somewhat at odds, since law enforcement could use the knowledge to harass groups for other reasons, and pedophiles could use the knowledge to help hide themselves, neither of which are desired goals, but with privacy matters you sometimes have to take the bad with the good. I2P was chosen as the platform since less research has gone into it verses Tor, but many of the same ideas and techniques should be applicable to both systems as they offer similar functionality when it comes to hidden services that are HTTP based. Another feature that makes this research somewhat different is that more work has been done in the past trying to detect users, not providers, of services in a Darknet.

While there are many papers on attacking anonymizing networks, most seem

[1] An anonymity set is the total number of possible candidates for the identity of an entity. Reducing the anonymity set means that you can narrow down the suspects.

[2] Full details of how I2P is implemented can be found at:
http://www.i2p2.de

to be pretty esoteric. A few previous papers that could be of use to those researching this topic are:

Locating Hidden Servers [1]

Low-resource routing attacks against anonymous systems [2]

The "Locating Hidden Servers" paper may not be directly applicable as it seems I2P goes to some effort to synchronize times and avoid clock skew problems[3]. A more directly I2P related analysis can be found on the I2P site's "I2P's Threat Model[4]" and guides to making services more anonymous can be found on "Ugha's I2P Wiki[5]". The threat model page points to many more resources and papers on possible attack vectors. More background information that will be of use during testing is listed in the approach section.

Background on I2P

Since the academic community seems to be far more aware of Tor than I2P, it may be helpful to compare the two systems and cover some of the basics concerning how I2P works. Both Tor and I2P use layered cryptography so that intermediates cannot decipher the contents of connections beyond what they need to know to forward the connection on to the next hop in the chain. Rather than focusing on anonymous access to the public Internet, I2P's core design goal is to allow the anonymous hosting of services (similar in concept to Tor Hidden Services). It does provide proxied access to the public Internet via what are referred to as "out proxies", as well as various internal services to proxy out onto the Tor and Freenet systems, but that is not its core design goal.

Every I2P node is also generally a router (and you can use the terms somewhat interchangeably when it comes to I2P) so there is not a clear distinction between a server and a mere client like there is with the Tor network. Some I2P nodes do take on more responsibility than others, such as floodfill routers that participate in NetDB to handle routing information. Unlike Tor, I2P does not use centralized directory servers to connect nodes, but instead utilizes a DHT (Distributed Hash Table), based on Kademlia[6], referred to as NetDB. This distributed system helps to eliminate a single point of failure, and stems off blocking attempts similar to what happened to Tor when China blocked access to the core directory servers on September 25th 2009[7]. I2P's reliance on a peer to peer system for distributing routing information does open up more avenues for Sybil attacks[8] and rogue peers, but steps have been taken to help mitigate this and are covered in the documentation[9].

Instead of referring to other routers and services by their IP, I2P uses cryptographical identifiers to specify both routers and end point services. For example the identifier for "www.i2p2.i2p", the project's main website internal to the I2P network, is:

---

[3] Clock skews are lightly covered here: http://www.i2p2.de/techintro.html#op.netdb

[4] I2P's Threat Model: http://www.i2p2.de/how_threatmodel

[5] Ugha's Wiki (note that you have to use an I2P proxy to access the site): http://ugha.i2p/HowTo

[6] NetDB Documentation http://www.i2p2.de/how_networkdatabase

[7] More details on China's blocking of the Tor directory servers can be found at: https://blog.torproject.org/blog/tor-partially-blocked-china

[8] I2P vs. Sybil Attacks http://www.i2p2.de/how_threatmodel#sybil

[9] More details on the inner workings of I2P, and it's mitigation techniques against Sybil attacks and rogue peers can be found in the "Technical Introduction": http://www.i2p2.de/techintro.html

```
-KR6qyfPWXoN~F3UzzYSMIsaRy4udcRkHu2Dx9syXSz
UQXQdi2Af1TV2UMH3PpPuNu-GwrqihwmLSkPFg4fv4y
QQY3E10VeQVuI67dn5vlan3NGMsjqxoXTSHHt7C3nX3
szXK90JSoO~tRMDl1xyqtKm94-RpIyNcLXofd0H6b02
683CQIjb-7JiCpDD0zharm6SU54rhdisIUVXpi1xYgg
2pKVpssL~KCp7RAGzpt2rSgz~RHFsecqGBeFwJdiko-
6CYW~tcBcigM8ea57LK7JjCFVhOoYTqgk95AG04-hfe
hnmBtuAFHWklFyFh88x6mS9sbVPvi-am4La0G0jvUJw
9a3wQ67jMr6KWQ~w~bFe~FDqoZqVXl8t88qHPIvXelv
Ww2Y8EMSF5PJhWw~AZfoWOA5VQVYvcmGzZIEKtFGE7b
gQf3rFtJ2FAtig9XXBsoLisHbJgeVb29Ew5E7bkwxvE
e9NYkIqvrKvUAt1i55we0Nkt6xlEdhBqg6xXOyIAAAA
```

This is the base64 representation of the destination. Obviously having a user type in this 516 byte chunk of data as an Identifier would be somewhat less than user-friendly, and it would not be valid in some protocols anyway (HTTP for example). I2P provides some workarounds for naming identifiers; one is called “Base 32 Names”, similar in many ways to Tor’s .onion naming convention. Essential the 516 byte Identifier is decoded (with some character replacements) into its raw value, the value is hashed with SHA256, then this hash is base 32 encoded and “.b32.i2p” is concatenated onto the end[10]. The results for the “www.i2p2.i2p” identifier shown above would be:

rjxwbsw4zjhv4zsplma6jmf5nr24e4ymvvbycd3swgiinbvg7oga.b32.i2p

This form is much easier to work with. For most eepSite users the common naming solution is to just use the local I2P address book that maps a simple name like “www.i2p2.i2p” to its much longer Base 64 identifier. There is no official DNS like service to do this lookup as that would be a single point of failure that the I2P project wishes to avoid. Each I2P node has its own series of text files that contain the name mappings in much the same way that the Internet used to use just HOSTS files to translate names to IPs before DNS was invented. There are however naming subscription services inside of I2P that can be synced to if the user wishes, though this means the user is putting some level of trust in these services not to hijack the name mappings.

A router’s ID is not the same as a service’s ID, so even if the service happens to be running on a particular router the two identifiers cannot be easily tied together. I2P also uses a few techniques to help mitigate traffic correlation attacks. While the Tor network uses a single changing path for communications, I2P uses the concept of “in” and “out” tunnels so requests and responses are not necessarily using the same paths for exchanging information. I2P also uses an Onion routing variant referred to as Garlic routing[11], where more than one message is bundled together into a “clove”. This mixing of messages using Garlic routing can lead to confusion for attackers attempting to correlate transmission sizes and timings, and if “cloves” are composed of messages from both high latency tolerant applications (e.g. email) and low latency applications (e.g. web traffic) correlation could become even harder. More comparisons between I2P, Tor and other anonymity networks can be found on I2P’s “I2P Compared to Other Anonymous Networks” page[12].

Many services can be hosted inside of the I2P overlay network (IRC, Bittorent, eDonkey, Email, etc.), and the I2P team has provided an API for creating new applications that ride on top of the I2P overlay network. As the developers note on their page, many standard Internet applications are not designed with anonymity in mind, so caution should be taken when adapting an existing application to run on top of I2P. While many applications exist and could be researched

[10] Some things are better explained in source code, which you can find provided here in the Python scripting language: http://forum.i2p2.de/viewtopic.php?t=4367

[11] Garlic Routing Explanation http://www.i2p2.de/how_garlicrouting

[12] I2P Compared to Other Anonymous Networks http://www.i2p2.de/how_networkcomparisons

for application data leaks, this paper will be concentrating on eepSites which are websites internal to I2P. Some measures are taken by the default I2P install to help filter revealing information at the application level, but service providers do make mistakes that can lead to too much information being revealed.

Overview of Approach

Our main approach will be looking at the application layer and seeing what details the hosts and eepSites are giving away about themselves. This has already been done in the past against cloaked clients with much success:

Metasploit Decloaking Engine[13]

EFF project on web client identification[14] [3]

Since we are targeting the identity of servers instead of clients the exact vectors for attack will differ, but there will be some overlap. Many I2P services are hosted on nodes/routers that also act as the owner's client node so client based attacks may also be fruitful in revealing their identity. People regularly make mistakes in how they configure web servers and applications that cause too much information to be leaked out to an attacker, information that can make finding a workable vulnerability much easier. This sort of information leakage is regularly mentioned in the OWASP (Open Web App Security Project) Top 10[15] in one form or another. One of our mantras is "Specific exploits are temporary, bad configuration mistakes are forever". A few of the techniques we researched to try to reveal identifying information about the host of an eepSite include:

1. Banner grabs of both eepSites inside of I2P, and against know IPs participating in the Darknet, to reduce the anonymity set of the servers.
2. Reverse DNS and who is lookups to find out more information concerning the IPs of the I2P nodes.
3. TCP/IP stack OS finger printing.
4. Testing I2P virtual host names on the public facing IP of I2P nodes.
5. Compare the clock of the remote I2P site, and suspected IP hosts on the public Internet, to our own system's clock. We did this via the HTTP protocols "Date:" header.
6. Command injection attacks.
7. Web bugs to attempt to de-anonymize eepSite administrators or users. (This turned out more problematic than we originally thought)

There were a few challenges imposed because of the nature of the I2P Darknet. These technical challenges caused many standard security testing applications to fail completely, or give ambiguous results. Here are a few examples of the challenges:

Point 1: Communications with the eepSites is normally done via an HTTP proxy. This is somewhat more limiting connection wise than using a SOCKS proxy, and way more limiting that having a direct TCP/IP connection to the target. Also, the default HTTP proxy that comes with I2P does not support the "connect" command. While this is stated in the documentation, we first encountered this feature while trying to run an Nmap scan using proxychains,

[13] Metasploit Decloaking Engine code and details are available at: http://www.decloak.net/
[14] EFF Panopticlick https://panopticlick.eff.org/
[15] OWASP Top 10 http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

and seeing the following message when we used Wireshark to try to diagnose why our attempts were failing:

<h3>Warning: Non-HTTP Protocol</h3>

The request uses a bad protocol.

The I2P HTTP Proxy supports http:// requests ONLY. Other protocols such as https:// and ftp:// are not allowed.

While this is challenging, we got around the problem by writing some of our own scripts in Python to do the required tasks. ZZZ[16] told us that SOCKS and Connect should work if we set up the tunnels for them, but at first we were unable to get them to function. After much back and forth with ZZZ (and the sending of sections from our error logs) it seems that it's a little tricky to make a successfully connection to an eepSite via a SOCKS proxy client tunnel. We had to make sure DNS requests were being forwarded through the SOCKS tunnel; otherwise there would be an error when the DNS system tried to look up a hostname ending in .i2p, which is not a valid top level domain name on the public Internet. This setting can be made in Firefox by going into "about:config" and setting:

network.proxy.socks_remote_dns = true

However this is only a solution for one application, Firefox, so it may be of limited utility in making other applications work with the SOCKS client tunnel as a proxy.

Point 2: Perhaps because of point one, many of the tools we have experimented with so far have a tendency to give false results or hang while working on spidering an eepSite. We have created some custom scripts that compensate for these eepSite oddities, or we simply verify the results ourselves in a more manual fashion. Many of the pages hosted inside of I2P use forum, image board, or blog software that passed parameters via the file path section of the URL. This may cause a non-404 error to return, even for a non-existing file. When a spidering tool says an obsolete or vulnerable file is there, it must be verified by hand.

Point 3: Filtering of client requests makes it somewhat harder to attack the administrator of an eepSite via web bugs, or odd XSS attacks put into the logs[17]. If the administrator is hit with an XSS, it is likely they will be using I2P at the time, in which case the returned information will be coming through an outproxy and not directly from their IP. I2P automatically changes the browser agent string when an HTTP tunnel is used to "User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.6)" for outproxy, and "MYOB/6.66 (AN/ON)" for internal I2P sites. This makes putting an XSS attack in the logs of an eepSite, and hoping to get information back when the administrator checks them via an HTML based report close to nil. Many HTTP headers are filtered or normalized by I2P such as: Accept, Accept-Charset, Accept-Encoding, Accept-Language, Accept-Ranges, Referer[18], Via and From[19]. Also, add on the fact that a security conscious

[16] ZZZ is the lead developer of I2P and as the development is done pseudonymously that is the only name we have for him.

[17] XSS, Command and SQL Injection vectors: Beyond the Form http://www.irongeek.com/i.php?page=security/xss-sql-and-command-inject-vectors

[18] Though interestingly, a Referrer is still visible with JavaScript unless other precautions are taken.

[19] I2P Tunnel Information http://www.i2p2.de/i2ptunnel

administrator may be using the NoScript, TorButton (which does more anonymity functions than just switching proxies) or other privacy enhancing plugins, client side attacks may become somewhat difficult. While on the subject of client side identification and uniqueness, we did a quick test using Panopticlick. When we tried using our normal install of Firefox, Panopticlick reported that we were "unique among the 1,258,250 tested so far". However, when we used the Tor Browser Bundle[20] and set it to use our local I2P proxy, Panopticlick reported "one in 15,343 browsers have the same fingerprint as yours" which is much better. As such, it is recommended that I2P users may want to not use their default browser for I2P use, and use a dedicated browser instead.

Our experiences with testing web applications inside of I2P really highlight the need to understand how specific web apps work, rather than just running tools against them and "hoping for the best". Nathan Hamiel and Marcin Wielgoszewsk gave a great talk at Defcon 18 on the subject of writing your own tools for web application security evaluation[21], unfortunately we did not find out about their work until we had created most of our tools. For those interested, they have published their code snippets online[22].

The next major problems were legal as opposed to technical in nature. While spidering we needed to be careful not to download contraband onto my own system. There is a fair amount of child pornography out on I2P, and laws in the United States are pretty unforgiving on the issue, even if the files were obtained while doing legitimate research. As such we mostly spidered text, which is unfortunate as EXIF data in images hosted on eepSites may be of value in identifying individuals. Another issue was that some of the techniques that we were testing may not be appropriate to do against resources we do not own, so we set up our own eepSite to do many of the tests. For common web vulnerabilities that could lead to identity discloser we tested against the Mutillidae[23] training package that implements the OWASP Top 10. While not totally realistic from the stand point that Mutillidae is MEANT to be exploited, it at least acts as a proof of concept that if similar vulnerabilities are found in an I2P facing web application they could lead to identifying information.

Evaluation

Collecting data on eepSites

The first thing we had to develop was a way to check which I2P sites were currently up and responding to requests. I2P, like many peer-to-peer systems, has a fair amount of churn. This churn makes it hard to track what sites are up at any given time.

One solution to gather active eepSites would be to spider some of the popular portal eepSites like forum.i2p or ugha.i2p for URLs ending in .i2p, then continue spidering from there recursively. This recursive option can be slow however,

[20] Tor Browser Bundle http://www.torproject.org/projects/torbrowser.html.en

[21] Defcon 18: Constricting the Web - Offensive Python for Web Hackers http://vimeo.com/15554801

[22] Constricting Code Snippets http://hexsec.com/docs/ConstrictinSnippets.zip/view

[23] Mutillidae may be found at the following URL: http://www.irongeek.com/i.php?page=security/mutillidae-deliberately-vulnerable-php-owasp-top-10

many of the links are to dead sites (quite a few people seem to put up a site just for fun, then abandon it), and we may miss sites that are active but just not linked too very often.

Another option is to parse though the host.txt file I2P uses for name to cryptographic identifier mappings, and check each i2p service for availability. I2P's SusiDNS allows the user to subscribe to host mapping services. The address book services we subscribed to were:

http://www.i2p2.i2p/hosts.txt
http://i2host.i2p/cgi-bin/i2hostetag
http://stats.i2p/cgi-bin/newhosts.txt
http://tino.i2p/hosts.txt

This gave us 1538 host names in our address book on 10/27/2010 at approximately 1pm EST.

The final solution was to use a combination of both methods. A Python script was created that simply checked the status code returned by an eepSite when it's root document was requested, as well as doing a banner grab for the server type the eepSite's web daemon reported. While the reported server can be modified by the system administrator to not contain extra platform information, or even to return false information, not all administrators bother. This Python script could be used directly with the local I2P access proxy, or could be chained to another intercepting proxy for extra functionality. In general, intercepting proxies are meant to be run locally and allow the user to modify requests before they are sent out to the server, and many offer extra functionality such as spidering and scanning for common misconfigurations. We chose ZAP (Zed Attack Proxy[24]) as the intercepting proxy to chain to, and used it to do the needed spidering and site scraping. ZAP is a fork of the Paros Proxy project, and seemed to work well for the task at hand.

The Python script we created uses multiple threads to iterate though the hosts.txt file located at:

C:\Windows\SysWOW64\config\systemprofile\AppData\Roaming\I2P\hosts.txt

The choice of thread count is somewhat arbitrary. We did not want to overwhelm the local proxy servers with too many I2P requests, however doing the status checks and banner grabs one at a time would have been prohibitively slow. We obtained reasonable results with a thread count anywhere between 10 and 25. While testing, a scan with 100 concurrent threads found 104 active eepSites in 798.585 seconds and another scan using 10 threads found 112 in 5934.425 seconds. Keep in mind that these results are not completely predictable as outside events may have caused differences in speed and the number of eepSites reported, but it seems the local I2P proxy can handle multiple threads without dropping too many connection attempts. As such, we opted for faster scans by using more threads.

For the sake of space we will not insert the source code of our probing scripts into this paper, but our sample Python scripts are available from the author's

[24] Zed Attack Proxy http://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

website or on request[25]. The following is a quick synopsis concerning the function of each script:

I2PMassGrabber-headers.py
Checks the status of each I2P host listed in an I2P host.txt file to see if it's up, and then generates CSV and HTML formatted output with the hostname, status, and server banner. Input file and proxies will have to be changed based on user settings. This script also collects page scrapes that can be reviewed.

real-IP-banner.py
Grabs HTTP banners from an Internet facing IP so we can compare, sort and filter later.

dump-and-sort-i2p-router-ips.py
NetDB scraping code used to obtain a list of IPs from our local NetDB cache. The RegEX needs some work as some invalid IPs work their way into the resulting output text. Generates or adds to a file named all-sorted-uniq.txt, so this script can be ran by a scheduler to collect the IPs of I2P nodes over time.

time-stamp-server.py
Compares times stamps found in the HTTP headers of both Internet IPs and I2P sites to the local clock, along with retrieval times, generating a CSV file and a synopsis in HTML.

virtual-server-test.py
I2P Virtual Host checking script. This script uses a large CSV file to try specific I2P host names on a given public IP to see if a different page is returned. It saves scrapes of these pages to a time stamped directory.

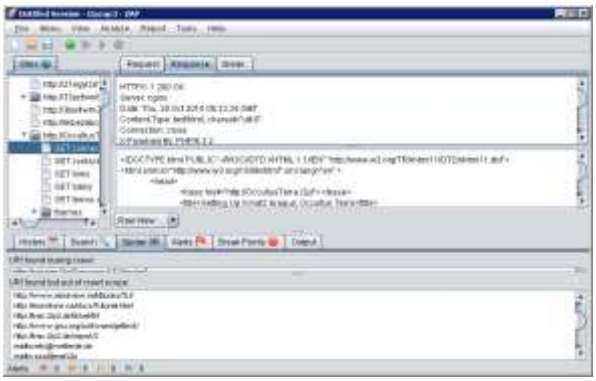
All of the scripts above will need to be tweaked by their users as the options are set by variables in the code, as oppose to command line flags. Also, the author is a Python novice so it's likely that the code could be cleaner and better optimized.

By setting the I2P banner grabbing Python script to use ZAP as its proxy, and then chaining ZAP to the local I2P HTTP proxy, we were able to do both banner grabs with the script and load the URL information into ZAP so that it could be used to do more spidering and scanning later. The output of the Python script went to two time-stamp named files, one HTML formatted for direct use in a browser, and one CSV file used to feed other applications. Here is an example of the CSV files format:

```
"bitcoin4cash.i2p","200","Apache"
"shpargalko.i2p","200","Apache/2.2.15 (Win32)
PHP/5.3.2"
"darrob.i2p","200",""
"ufm.i2p","200","Apache/2.2.8 (Ubuntu) PHP/5.2.4-
2ubuntu5.12 with Suhosin-Patch"
```

CSV is a convenient format to work with as it can be easily imported into other tools, especial Microsoft Excel and Access. The findings from the spidering and scans done by ZAP will be covered lightly in future sections. The intercepting proxy's biggest benefit to an attacker is in finding possible web applications to exploit via ZAP's spidering, file/directory brute-forcing and scanning features. I2P eepSite administrators should be aware that just because a file or folder on their site is not advertised does not mean it can't be found by an attacker.

[25] Current versions of the I2P probing scripts can be found at the following URL http://www.irongeek.com/host/i2p-probe-scripts.zip



Concurrently with the scanning of sites with ZAP and banner grabbing of eepSites with the Python script we attempted to run Wireshark[26] and captured the network traffic to disk. While the data being sent on the network is encrypted, just knowing who is communicating with us over I2P may be revealing. We can filter the traffic for nodes we know are peering with us in the I2P network based on the known port numbers we are using. These ports are not fixed, but we can find the ones we are using by going into the local console at:

http://127.0.0.1:7657/config.jsp

and taking note of the ports that are currently set. Since our I2P host was using UDP and TCP ports 12668 at the time, we set the capture filter to be "port 12668" to help eliminate extraneous data. While testing with the sniffer we ran into a bug that caused the Wireshark application to crash. To alleviate this problem, we used a simpler tool that comes with the Wireshark package called dumpcap to only write the packets to a file without displaying or parsing them. The command we issued was:

```
dumpcap -i
\Device\NPF_{E97777A0-5863-4741-
AA42-FD3E02B2BD4C} -s 0 -f "port
12668" -w g:\dumpcap.pcap -a
duration:3600
```

The command above uses the following parameters:

-i to tell dumpcap which network interface to use (if you are not sure which of your local interfaces to use, see the local interfaces options by using the –D flag)

-s to set the snap length so that we capture the whole packet

-f specifies the capture filter to use, thus emanating packets we may not care about

-w locates the pcap file to output

–a tells dumpcap to stop capturing under certain circumstances (in this case after one hour)

We could then look at the created pcap file later in Wireshark without fear of our packet capture being interrupted because of a problem in the GUI or protocol parsing sections of Wireshark's code base.

Upon looking at the I2P client closer, we realized a more efficient way to find know I2P nodes would be to scrape the contents of our NetDB directory using a regular expression to find IPs, then filter it for unique entries and remove invalid IP matches. The "dump-and-sort-i2p-router-ips.py" script was created for this purpose. On November 9th 2010 this netted us 1099 nodes, of which 172 seemed to be running a webserver that returned status code 200.

We took the end points we found in I2P via our network capture and NetDB scraping and scanned them with a slightly

[26]Wireshark
http://www.wireshark.org/

modified version of our Python banner grabbing script. The main things we had to change were how the script partitioned the data (comma instead of equal sign) and removed the use of the local I2P proxy. We originally wished to scan though the I2P proxy so that we would not have to worry about our ISP asking us why we were attempting a scan for port 80 across multiple IPs, but the outproxy seemed to strip the server type header information so we had to query the IPs directly over the public internet. We logged the server header strings for web services so we could later compared those to the headers returned by the eepSites we scanned.

Another source of useful information was doing a reverse DNS of the IP addresses. At first we did this by loading our pcap file using the "Network name resolution" option, sorting by hostname, and looking at the available endpoints under the statistics menu option. For example, one of the hosts was named awxcnx.de, but there is also an awxcnx.i2p. Both seem to belong to the public German Privacy Foundation so that example is not a big deal as it was likely deliberate (telecomix.org/telecomix.i2p and privacybox.de/privacybox.i2p are similar examples), but internal to external naming conventions is something to keep an eye out for. For example, if we see a name like "thor.schmelz.com" we might want to scour I2P for people interested in Norse mythology or Marvel Comics.

One thing we stumbled upon while looking at names was an organization that seemed to have quite a few I2P nodes. Nimbios.org had 25 I2P members according to our pcap file. Upon doing a reverse lookup on the IPs we scraped from out local NetDB, we were able to find forty-four unique IPs belonging to NIMBIOS. We were rather curious what the "National Institute for Mathematical and Biological Synthesis" was using I2P for, so we emailed them. Seems I2P is part of the standard build for that organization. Proxad.net, Wanadoo.fr and Goaland.net also seem to have a fair share of nodes. This sort of analysis might be useful for those wanting to spot potential Sybil attacks.

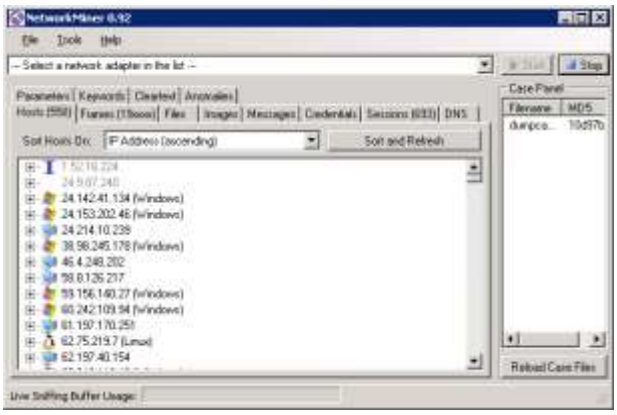
Overview analysis of the data

In this section we will cover interesting statistics based on some of the data we collected. While not all of it will be directly germane to anonymity, it does reveal things that we find interesting about the users of I2P and the IP networks they connect from.

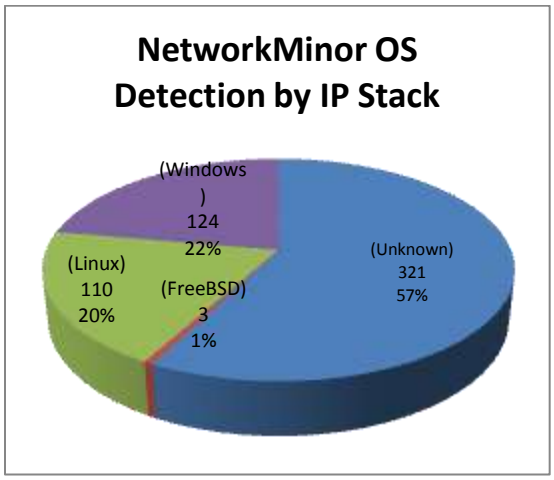
One advantage of using the Wireshark suite to dump packet to file is that it supports the libpcap file format, which is also supported by pretty much all tools that use the libpcap libraries. Once the pcap was created we were able to load it into NetworkMiner[27] for further analysis. When it comes to the TCP/IP protocol, some of the RFCs are ambiguous, and some vendors implements their TCP/IP stacks in peculiar ways. Items like initial TTL, Windows size, "don't fragment" settings and other options vary depending on who wrote the stack. These minor differences can be used to help finger print the type of host we are communicating with. NetworkMiner does passive OS fingerprinting, giving us a great deal of information about the IP stacks of the hosts we are in contact with, and based on the IP stack fingerprint we can make likely guesses as to what OS is running on

[27] NetworkMiner http://networkminer.sourceforge.net/

the remote hosts. NetworkMiner uses the fingerprint databases from previous tools such as p0f, Ettercap, FingerBank and Satori. Below is a screenshot of NetworkMiner's output.



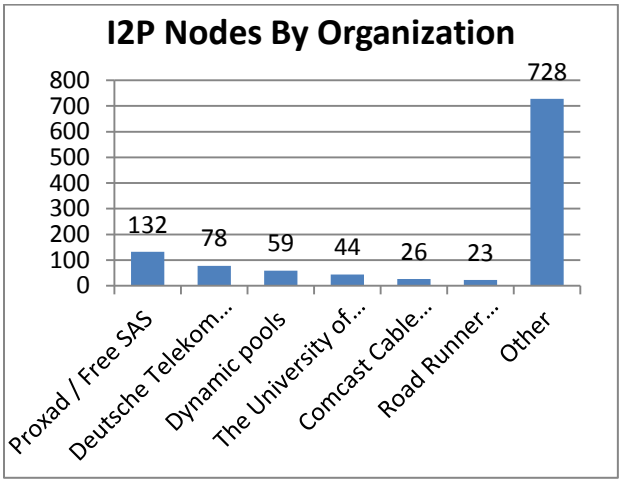
Since the current version of NetworkMiner does not allow us to dump the parsed data to a text file, we used Nirsoft's SysExporter[28] to extract the text from the treeview control, and a simple text editor to format it as we wished for loading into other applications. During our hour long capture we found 558 unique IPs communicating with us in the I2P network. The following pie graph gives a breakdown of the detected Operating Systems.

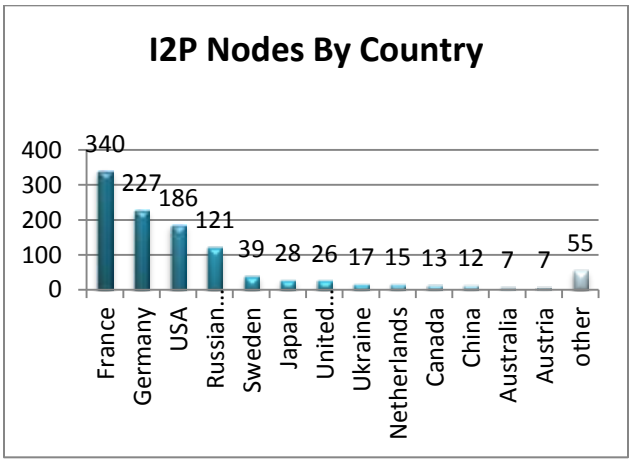


[28]SysExporter
http://www.nirsoft.net/utils/sysexp.html

While the IP fingerprint might somewhat lessen the anonymity set, it's not as clear as a banner grab of the reported server type.

Other information of interest is the location and responsible organization of the I2P node based on its IP and Whois record. There are many ways to obtain this information, but IPNetInfo[29] seemed the easiest to use because of the bulk of IPs we had to look up. The dataset collected on 11/09/2010 by scraping our local NetDB gave the following results.





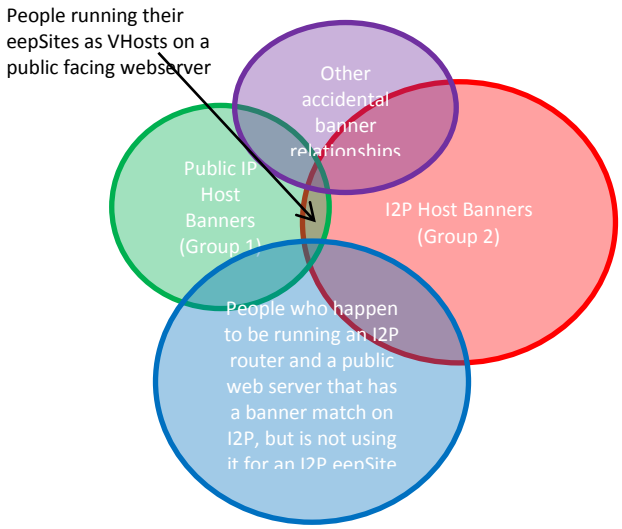
[29]IPNetInfo
http://www.nirsoft.net/utils/ipnetinfo.html

Now that we have various methods we can use to obtain data about the nodes in I2P, what information can we ascertain about their identity?

Correlating server banners grabbed from inside of I2P and off of the public Internet

One of our reasons for banner grabbing eepSites inside of the I2P network and known nodes from the public Internet is to see if we can correlate header information. Not all of the server banners were particularly unique, such as "Server: Apache". Also, not all servers returned a server banner at all. Because of churn in the network it's best to speak of results based on data collected at a given time. We will use the data collected on 11/09/2010 to illustrate some of our points. Of those banners returned facing internally to the I2P network we obtained the information that can be found in Table 1 of the appendix. Table 2 of the appendix contains the banner counts for I2P nodes that had a public Internet facing HTTP server and returned a banner with code 200 as the HTTP status.

As can be seen from the collected data from 11/09/2010, some of the banners give detailed information about their hosts regarding the platform and modules in use. When we used better techniques to harvest the IPs of participating I2P nodes we obtained a larger data set, but the data from 11/09/2010 illustrates the point. The end goal of the banner grabbing was to correlate external IPs to internal eepSites. There are of course false positives that are hard to estimate. Also, most of the banners are not in a one to one relationship, but even if they are not it helps to cut down on suspects and may help in obtaining a subpoena for search in freer countries, or cause the "Gestapo/Jack-booted-Thugs" to say "hey, we only have to kick down 10 doors instead of 500!" in more repressive regimes.

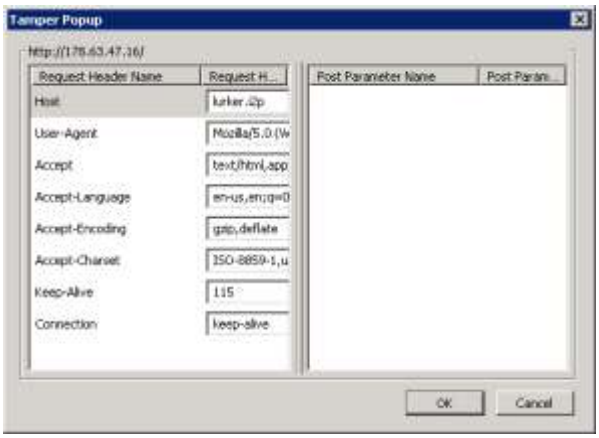


For our test of using banner grabs to correlate external IPs to internal eepSites we first focused on the relations that were one to one. We used a combination of Access and Excel to find these correlations and statistics by importing the CSV files we created earlier and doing a few simple SQL queries. Here is a table of the one to one relationships from an earlier dataset we created:

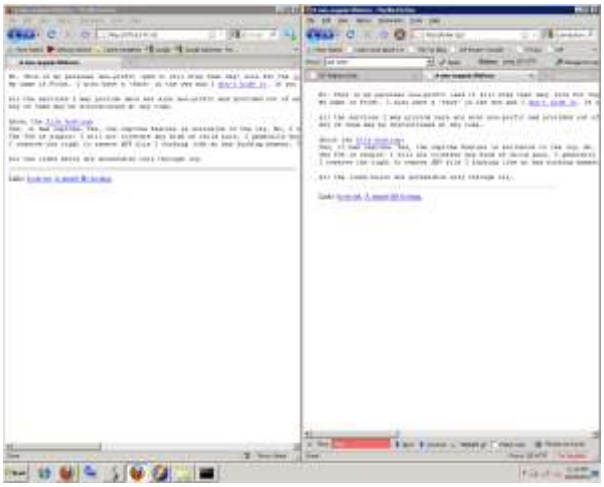
1 to 1 IP to I2P Banners		
I2P hostname	**IP**	**Banner**
medosbor.i2p	89.31.112.91 (host-89-31-112-91.academ.org)	Apache/2.2.13 (Linux/SUSE)
ipredia.i2p	97.74.196.206 (ip-97-74-196-206.ip.secureserver.net)	Apache/2.2.3 (CentOS)
xorbot.i2p	178.77.75.23 (www.gernot-schulz.com)	Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch
trac.i2p2.i2p	46.4.248.202 (bilbo.srv.welterde.de)	nginx/0.6.32
lurker.i2p	178.63.47.16 (fleshless.org)	nginx/0.7.65

While this is not conclusive, it does reduce the anonymity set and allows us to take further steps to verify the suspicion that they are the same host.

From hitting the IP 178.63.47.16 and receiving back a page that only said “It works!” (a default page on some web server installs) we suspected the server was using virtual hosting to host more than one site on the same IP. Using the Firefox plugin TamperData[30] we modified our request header to have the suspected eepSite’s hostname (lurker.i2p):



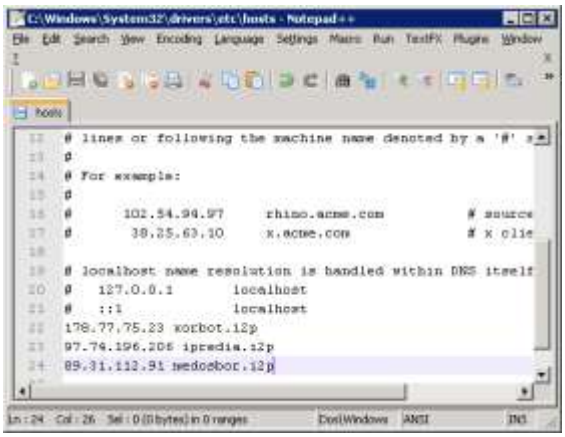
This gave us the results we were looking for.



Since the pages are the same it seems at least in this case we found the Internet facing IP of an eepSite. Based on the text of the page, lurker.i2p’s owner Frost is not really trying to hide his connection to the site, but still this is a promising proof of concept for correlating eepSites to IP hosts via server banners.

As for the other pages we tested by hand, 46.4.248.202 (bilbo.srv.welterde.de) already returns the I2P project page without any manipulation of the host header, so it’s pretty clear it is connected to trac.i2p2.i2p. 178.77.75.23 (www.gernot-schulz.com) was a little harder. Tamper Data was used to insert “xorbot.i2p“ as the host to request, but something was going wrong, possibly the use of cache control headers from the server caused issues. We switched to using ZAP’s intercepting proxy features to have more control, and then set the requested host header, but without success. The next idea was to just do it the old fashion way and add entries to map the IPs to host names in our Windows host.txt file.

[30] TamperData https://addons.mozilla.org/en-US/firefox/addon/966/



Alas, this also failed. As the public page on the site makes it obvious that Mr. Schulz is into crypto, and we know he is using I2P, he may still a likely suspect. The I2P facing side of ipredia.i2p was having communications issues at the time we were performing our checks, so we could not test it. 97.74.196.206 would only show the "LXA Server Administration Tool" as the root document no matter the host name used (although we later found ipredia.i2p on a different IP once we had collected more Internet facing hosts to test against). Medosbor.i2p and 89.31.112.91 (host-89-31-112-91.academ.org) host the same site, so that is a fairly obvious connection. 89.31.112.91 returned a blank page by default, so we used the Windows host.txt file to set the name mapping, allowing us to easily pass the medosbor.i2p host name in the HTTP request that went over the public Internet. Medosbor, like some other sites, does not really seem to want to hide as they own "Medosbor.com" as well.

For one off checks, using the cURL[31] tool is a good option. For example, we could use the following two command lines:

```
curl 178.63.47.16
curl -H "Host: lurker.i2p" 178.63.47.16
```

and then observe returned results to see if they match.

All of this is fine for one to one checks, but if multiple I2P host banners match multiple Internet host banners, something more automated is required. We wrote several iterations of a script to try the entire set of Internet host to I2P host correlations, and test each IP for each suspected I2P hostname. To be more through, we could check every IP for every virtual host, but this greatly increases the number of checks that would have to be done and does not seem to be likely to net better results (and when we tested, this was the case). Using our data set from 11/09/2010 it would take 583 checks if we matched our tests by banner, but 19092 if we checked all possible IPs for all possible I2P hostnames regardless of the banner.

At first we looked at all of the returned pages manually instead of just having the script say if the returned page was different than the default root document, this however was a chore. Earlier version of the virtual host matching script (virtual-server-test.py) used a simple string compare to see if the sites were different when using host headers, but this led to a lot of false positives. If the page returned a date stamp, or the name of the host requested, the page would look different to a simple matching if statement, but really the site was the same functionally. Luckily we were able to use Python's difflib to compare two sites, and only flag them as different if they varied by 25%.

[31] cURL
http://curl.haxx.se/

Using these methods we believe we have de-anonymised the following sites using I2P/Internet facing web servers:

I2P Hostname	Likely Real IP
lurker.i2p	178.63.47.16
bzr.welterde.i2p	188.40.181.33
docs.i2p2.i2p	188.40.181.33
openmusic.i2p	188.40.181.33
paste.i2p2.i2p	188.40.181.33
syndie.welterde.i2p	188.40.181.33
www.i2p2.i2p	188.40.181.33
matterhorn.i2p	188.165.45.229
awxcnx.i2p	62.75.219.7
directedition.i2p	68.33.184.167
forum.i2p	82.103.134.192
ugha.i2p	82.103.134.192
bolobomb.i2p	83.222.124.19
ipredia.i2p	84.55.73.228
teknogods.i2p	84.234.26.123
jonatan.walck.i2p	85.229.85.244
medosbor.i2p	89.31.112.91
colombo-bt.i2p	93.174.93.93
www.i2p2.i2p (mirror?)	94.23.12.210 94.23.46.106 46.4.248.202
mathiasdm.i2p	94.23.52.151
privacybox.i2p	94.75.228.29

Granted, this is not a huge percentage of the 111 I2P hosts we were working with, but it does show that this is a legitimate attack vector worthy of consideration. Improvements could be made by sampling for longer times, and more frequently to help compensate for churn in the network.

Mitigating this attack

The first mitigation for eepSite owners would be either to configure their server not to return a server banner or to just return a very non-distinctive banner such as the aforementioned "Server: Apache" (this is likely the result of using the ServerTokens directive set to ProductOnly). Documentation on how to do this should be available from the makers of the webserver software. This is not a complete solution to attackers checking for virtual hosts, an attacker can still choose to do the slower check from a larger pool of candidates. Keep in mind, even if the server does not return the requested virtual host to someone that requested it, an error prone banner match may still be enough depending on the laws of the country for someone to physically visit and search the server. If an attacker wished to reduce the anonymity set further, they could launch a Denial of Service attack against the IP of a suspected I2P host, as pointed out by a poster on ZZZ's forums[32]. However, if no identifying information was returned that helped to reduce the anonymity set in the first place, an attacker would have to try to DoS many more hosts, and test many more for response times. This could lead to more ambiguous information for the attacker and more anonymity for the eepSite host. As such, we recommended that future versions of I2P may want to look into filtering identifying server headers by default when an "HTTP Server" type tunnel is created. Much the same was already done for identifying browsers user agent strings on the client side. After reading an early draft of this paper Mathiasdm submitted a modification to the HTTP server tunnel code to automatically replace the HTTP server header with "Server: I2PServer". When version 0.8.2 was released on 12/22/2010 it implemented a change to automatically remove the HTTP Server header entirely, making mitigating verbose HTTP Server

[32] I2P vs. DoS of IP address http://zzz.i2p/topics/761

headers yourself somewhat moot. While this means that the HTTP Server header can no longer be used to reduce the number of IPs that need to be checked for Virtual Hosts, information about the server type may still be gleaned from X-Powered-By headers and verbose error messages. Also, with the currently small size of the I2P network, probing every I2P node without filtering by Server header is still feasible. As the I2P network grows, this may no longer be the case.

The Server string may not be the only item in the headers that allows for fingerprinting the system. Some HTTP daemon extensions may append other headers that can be revealing. For example, ASP.NET and PHP may add an "X-Powered-By" header that will reveal information about the server that will reduce its anonymity set. A case in point is forum.i2p:

```
Date: Wed, 01 Dec 2010 21:02:21 GMT
Server: Apache
X-Powered-By: PHP/5.2.13-pl0-gentoo
```

Notice that while the server string is fairly generic, the X-Powered-By is pretty specific. This can be used to help eliminate other candidates that have the string "Server: Apache" in their headers. Fortunately these headers can be disabled in PHP [33] and ASP.NET[34]. The ordering of headers may also be useful in some cases, though server types (Apache, IIS, etc.) generally seem to keep a standard order.

If a site does not currently return useful headers it may be revealing to check out historical records of its previous headers from before mitigations were put in place. If an attacker goes to:

http://i2p.to/frame.php?page=info&host=somesite.i2p

and replaces somesite.i2p with the site they are interested in they may find useful information in the past headers the site returned. For those interested in more information about how HTTP headers may be used by attackers it is recommended that they visit the Shodan project's[35] website.

The second and stronger mitigation is to either not run the eepSite on a web server with a public facing IP, or to make sure that the virtual host for the I2P site is only set to respond to requests from the localhost (where the I2P router is running) or trusted IPs. An example section in an Apache httpd.conf file might look something like the following:

```
#Don't show Apache version in errors
ServerSignature Off
# Say only "apache" in server banner
ServerTokens Prod

# Make a default virtual host
NameVirtualHost 0.0.0.0
<VirtualHost *>
  DocumentRoot "/somepath/htdocs"
</VirtualHost>

# Host two eepSites that only listen
# on the loopback address

NameVirtualHost 127.0.0.1
<VirtualHost 127.0.0.1>
   ServerName myeepsite1.i2p
   DocumentRoot "/somepath/eep1"
</VirtualHost>
```

[33] Disable PHP X-Powered-By header: http://phpsec.org/projects/phpsecinfo/tests/expose_php.html

[34] Disable ASP.NET X-Powered-By header: http://www.asp101.com/articles/wayne/pryingeyes/default.asp

[35] Shodan HQ http://www.shodanhq.com/

```
NameVirtualHost 127.0.0.1
<VirtualHost 127.0.0.1>
   ServerName myeepsite2.i2p
   DocumentRoot "/somepath/eep2"
</VirtualHost>
```

If a web server does not respond to probes from the Internet confirmation of it hosting an I2P service becomes much harder. Also note that the httpd.conf example above uses the "serversignature off" and "servertokens prod" directives to help reduce the amount of information returned by error messages and HTTP headers.

Clock Differences

While clock skew has been covered in the literature before [4], it seems rather difficult to implement its use for de-anonymizing hidden services. Previous efforts have had to implement their own test networks because real world/deployed anonymizing networks (Tor in this case) were so variable in their response times that the clock skew measurement methods could not obtain dependable results. I2P eepSites seem more dependable than Tor hidden services when it comes to response times, so perhaps these techniques should be revisited.

Rather than look at clock skew, and have to apply complicated statistical analysis to compensate for the latency caused by I2P, we looked at total clock differences as measure by reading the time stamps of the HTTP headers returned by eepSites. If the time difference is significantly beyond the total time it takes to retrieve the page this may be useful for spotting likely suspect IPs hosting I2P sites. It should be noted that I2P does do some synchronization of clocks and timing, but this is for the I2P package itself and not the host's clock nor other services running on the host.

To test the idea we took sites like ipredia.i2p (84.55.73.228) which we had already de-anonymized using the virtual host method and checked their clocks as reported by their HTTP headers against our own system's clock. When we checked the HTTP timestamp of 84.55.73.228 the time difference was -4325.582 seconds with a retrieval time of 0.353 seconds. When we checked ipredia.i2p the time difference was -4321.663 seconds with a retrieval time of 8.946 seconds. Since the clock difference was significantly greater than the retrieval time, this would be a pretty clear example of a badly set clock giving away an IP to I2P relationship. After the initial tests, we tried to correlate the clocks of other IP and I2P hosts. One standout worth mentioning is the pair error.i2p and 130.241.45.216. Both shared the same server header "Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny8 with Suhosin-Patch", but doing the virtual host check against 130.241.45.216 for error.i2p did not return definitive results. The clocks tell a different story however. When we checked the HTTP timestamp of 130.241.45.216 the time difference was 4488.434 seconds with a retrieval time of 0.702 seconds. When we checked the HTTP timestamp of error.i2p the time difference was 4490.365 seconds with a retrieval time of 4.894 seconds. This makes a connection between these two hosts seem very likely. With clock differences on the order of an hour it's pretty easy to spot suspected hosts, but with proper analysis the needed time difference could be greatly reduced.

Mitigating this attack

As mentioned before, not running an eepSite on a public IP would be a good first step. Also, making sure that the time is properly synchronized with a reliable and widely used NTP server and the time zone is set correctly would help. The reason we specify a widely used and reliable NTP server is that synchronizing against an NTP system that is significantly off may also reduce the anonymity set.

Command Injection attack

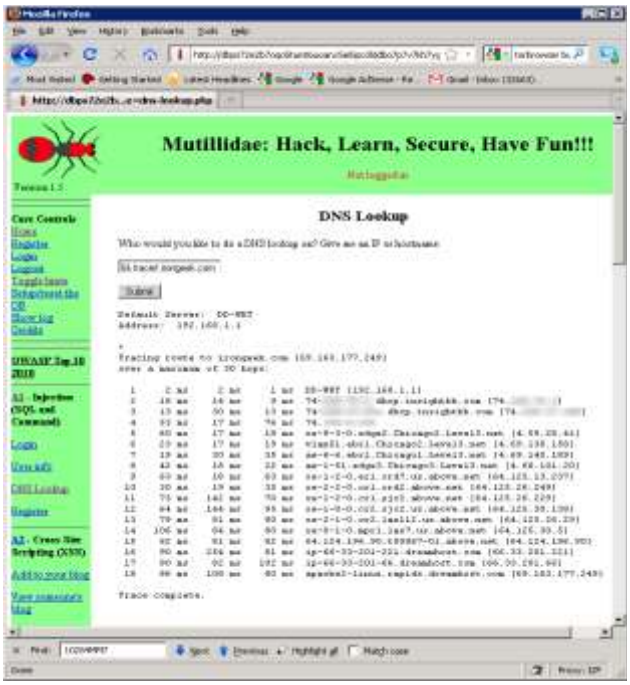
A Command Injection Vulnerability occurs when improperly sanitized input, be it from a web form, get request, cookie or header, is fed into an application that then uses the input as part of a command that is to be issued at a shell. A similar flavor of vulnerability is the Code Injection attack, where the attacker attempts to get their code inserted as part of the application. A slightly less related attack is the SQL Injection attack, where the attacker uses input to try to change the nature of an SQL query. All of these attack vectors are of interest because it is possible to use them to force an [5] eepSite's host to make a connection to an attacker controlled host from outside of the I2P network, thus revealing their identity.

Since mounting this particular attack on someone else's system might be ethically or legal questionable we put up our own eepSite to test against. For common web vulnerabilities that could lead to identity discloser we installed the Mutillidae training package that implements the OWASP Top 10[36] as a test bed. While this is not a realistic test in the sense that the Mutillidae web application is deliberately designed to be compromised, it still works as a proof of concept for how these common web vulnerabilities could be used to identify a system.

Mutillidae has multiple vulnerabilities we could choose from, but for our testing we chose to use the Command Injection vulnerability located in the DNS Lookup application. The way this application is designed to work is as follows: The user enters a host name or IP to lookup, then the application uses the system's nslookup command to find the requested information and return it to the user. However, since the DNS Lookup application is issuing this nslookup command with a simple PHP "shell_exec" function, extra commands can be tacked onto the end of the input (using a ; in Linux or a && in Windows) which will also be executed. Since in this case the output of the command is reflected in the resulting HTML of the returned page, all the attacker has to do is read the results directly.

For this test we used the simple string "&& tracert irongeek.com" as our injection. As can be seen in the output, this trace route command totally bypasses the I2P proxy, and the results show the true IP of the host running the eepSite (which we blurred in the screenshot).

[36] OWASP Top 10 http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project



This particular Command Injection vulnerability reflects the output back to the web browser, but this sort of verboseness is not necessary. An attacker could also use a blind Command Injection that utilizes a network related command (like ping on netcat for example) to make a connection back to a host the attacker controls, then sniff for incoming connections to find the true IP for that eepSite's host.

Similar attacks could also be carried out via a Code Injection that inserts networking functionality into the application to communicates back with the attacker, or conceivably via an SQL Injection that uses a stored procedure (xp_cmdShell in MSSQL comes to mind).

Mitigating this attack

Short of a full out code review, watching for security news related to the web applications used and keeping the applications on the eepSite patched and up-to-date is the best course of action. For home grown web applications it would be a good idea to review OWASP's material on the subject of avoiding various types of injection attacks[37]. Another solution may be to massively lock down the eepSite's firewall rules not to allow any sort of egress to the outside Internet. While some may disregard this section of the paper since we only tested against our own deliberately vulnerable application, these sorts of flaws do exist in real web applications and pop up fairly regularly (though usually not as obvious or simple to exploit as in the sample DNS Lookup application). A simple search for "injection" under the web section of Exploit-DB[38] should be quite revealing as to how common these sorts of problems are.

Summary of results

Exact statistics on the reliability of attacks are not easy given the amount of churn in the I2P network. This churn can be somewhat compensated for by collecting data over a longer period of time, but the figures are not exact and there is not complete visibility into the network. An eepSite may be found, and then disappear, before an associated IP can be probed (and vice versa). Of the 119 I2P hostnames we have in our set we found 21 IPs via either querying for the I2P host name in the host header, or because the IP returned the same page as the I2P eepSite. One of these was an outdated version of jonatan.walck.i2p that had been moved to a new location, which we found out about by emailing the administrator. We have four candidates for www.i2p.i2p because of mirroring. Clock difference attacks only gave us one new "likely" de-anonymized

[37] OWASP Command Injection http://www.owasp.org/index.php/Command_Injection

[38] Exploit Database http://www.exploit-db.com

eepSite, though some of the eepSites found via other methods could also be found with this attack. This clock difference method shows promise for further testing and refinement. The command injection attack was only carried out against a test system, real world results would of course vary based on the site that was being attacked and what web applications they had installed.

Conclusions and Discussion

As can be seen from the sections above, even if an anonymity network is well designed on its lower levels, applications that are run on top of it can still compromise the identity of the users if certain data is not properly sanitized. This may especially be the case when applications designed on and for the public Internet are shoehorned into working on an anonymity network without certain mitigations being put in place. It should also be noted that the attacks above may prove more useful if the collected data is accumulated over a longer period of time to help compensate for the natural churn of the network, and the lack of a central location to query to find all nodes in the network.

Besides the techniques we have outlined above, there are many more avenues that could be explored in future research. We concentrated our work on eepSites inside of I2P, but IRC, eMule and BitTorrent usage could also be interesting to research for identity leaks. We have already done some work in revealing information about IRC users in I2P based on the "Request: USER" information their IRC client provides (see the /whois command).

This paper concentrated on looking for the Internet hosts of services directly, but targeting the administrators via whatever contact information they provide and enticing them to visit a site the attacker controls could also be fruitful. This may not reveal the IP of the eepSite host if the administrator is not using it as an I2P client as well, but in many cases the IP of one of the administrator's workstations is good enough. There are numerous ways to find the IP a client is coming from that could bypass the browser's proxy settings. For example, when we visited the aforementioned Decloak.net while using I2P it was able to discern our true IP via the Flash plugin we had installed. For this reason, it is recommended that people who really wish to stay anonymous may want to forgo the use of plugins like Flash. We wished to look into various JavaScript XSS vectors as well, but certain technical and time limitations held us back. Also of interest might be metadata in documents posted on eepSites or in Deepsites[39]. Quite a few people have been doing research into the metadata located inside of JPEGs, MS Office docs, PDFs and other data files [6]. Using tools like FOCA[40] this data can be extracted to reveal real names, user names, IPs and other related data [7].

While these application level attacks do not break the I2P anonymity system directly, they can lead to compromising identities. Certain architecture changes could be made to make these attacks more difficult, but there is no way to completely protect users and administrators from

[39] Deepsites are akin to FreeNETs distributed storage system. More details are available at: http://duck.i2p/

[40] FOCA may be downloaded from: http://www.informatica64.com/DownloadFOCA/

making mistakes without also limiting their freedom to choose what to do with the anonymity platform. Administrators should be cautious when providing services inside of I2P, make sure there are not unintended leaks, and understand the nature of the application or service they are trying to deploy. From an attacker's perspective, why bother to pick a lock when you can crawl through an open window?

Thanks to the following individuals who reviewed this paper: ZZZ, Mathiasdm, Echelon, Tuna, Bart Hopper, Gene Bransfield, David Comings, Rick Hayes, Keith Pachulski and Dr. Apu Kapadia.

Works Cited/Related Work [8]

[1] L., and Syverson, P. Øverlier, "Locating hidden servers," in *Symposium on Security and Privacy*, May 2006.

[2] Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, Douglas Sicker Kevin Bauer, "Low-resource routing attacks against tor," in *WPES '07 Proceedings of the 2007 ACM workshop on Privacy in electronic society*, 2007.

[3] Peter Eckersley, "How Unique Is Your Web Browser?," Electronic Frontier Foundation, 2010.

[4] Steven J. Murdoch Sebastian Zander, "An Improved Clock-skew Measurement Technique," in *In 17th USENIX Security Symposium*, San Jose, 2008.

[5] Brian Neil Levine, Clay Shields Marc Liberatore, "Strengthening Forensic Investigationsof Child Pornography on P2P Networks," in *CoNEXT* , Philadelphia, 2010.

[6] Larry Pesce, "Document Metadata, the Silent Killer…," SANS Institute, 2008.

[7] Enrique Rando, Francisco Oca and Antonio Guzmán Chema Alonso, "Disclosing Private Information from Metadata, hidden info and lost data," in *BlackHat Europe*, 2009.

[8] Aaron Johnson, Paul Syverson Joan Feigenbaum1, "Preventing Active Timing Attacks in Low-Latency Anonymous Communication," in *Privacy Enhancing Technologies Symposium (PETS 2010)* , Berlin, 2010.

[9] Eugene Y. Vasserman, Eric Chan-Tin Nicholas Hopper, "How Much Anonymity does Network Latency Leak?," University of Minnesota, Minneapolis, MN 55455 USA, 2007.

[10] Steven J. Murdoch and George Danezis, "Low-Cost Traffic Analysis of Tor," University of Cambridge, Computer Laboratory, Cambridge, 2005.

[11] Steven J. Murdoch, "Hot or Not: Revealing Hidden Services by their Clock Skew," University of Cambridge, Cambridge, 2006.

[12] Werner Sandmann, Christian Wilms, Guido Wirtz Karsten Loesing, "Performance Measurements and Statistics of Tor Hidden Services," in *SAINT '08 Proceedings of the 2008 International Symposium on Applications and the Internet* , 2008.

[13] Saumil Shah, "An Introduction to HTTP fingerprinting," in *Black Hat Asia*, 2003.

[14] "OWASP Top 10," OWASP, 2010.

[15] Steven J. Murdoch, "Covert channel vulnerabilities in anonymity systems," Univesity of Cambridge, 2007.

[16] Michael K. Reiter, Chenxi Wang, and Matthew K. Wright Brian N. Levine, "Timing attacks in low-latency mix-based systems," in *Financial Cryptography, 8th International Conference*, 2004.

[17] Nikita Borisov Prateek Mittal, "Information leaks in structured peer-to-peer anonymous communication systems," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2008, pp. 267-278.

[18] Jean-François Raymond, "Traffic analysis: Protocols, attacks, design issues, and," in *Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 2000, pp. 10-29.

Appendix

Table 1
I2P Facing Banner Counts (11/09/2010 dataset)

Server Banner	Count
blank	39
Server: Apache	14
Server: lighttpd/1.4.22	6
Server: Apache/2.2.15 (Win32) PHP/5.3.2	4
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny8 with Suhosin-Patch	4
Server: Apache/2.2.14 (Unix) mod_ssl/2.2.14 OpenSSL/0.9.8l DAV/2 PHP/5.2.12	3
Server: Apache/2.2.15 (Debian)	3
Server: WSGIServer/0.1 Python/2.5.2	3
Server: Microsoft-IIS/6.0	3
Server: nginx/0.8.53	2
Server: Apache/1.3.27 (Linux/SuSE) mod_ssl/2.8.12 OpenSSL/0.9.6i PHP/4.3.1 mod_perl/1.27	2

Table 1
I2P Facing Banner Counts (11/09/2010 dataset)

Server Banner	Count
Server: Apache/2.2.11	2
Server: Apache/2.2.11 (Win32) PHP/5.2.8	2
Server: Apache/2.2.14 (Ubuntu)	2
Server: lighttpd/1.4.23	2
Server: nginx	2
Server: Apache/2.2.13 (Linux/SUSE)	1
Server: AnomicHTTPD (www.anomic.de)	1
Server: thttpd/2.25b 29dec2003	1
Server: lighttpd/1.4.19	1
Server: Apache/1.3.34 (Debian) mod_python/2.7.11 Python/2.4.4c0 PHP/5.2.0-8+etch16	1
Server: Apache/2.0.55 (Linux/SUSE)	1
Server: Fred 0.5 (build 5107) HTTP Servlets	1
Server: Apache/2.2.11 (Win32) DAV/2 mod_ssl/2.2.11 OpenSSL/0.9.8i PHP/5.2.9	1
Server: Apache/2.2.14	1
Server: Apache/2.2.12 (Ubuntu)	1
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.12 with Suhosin-Patch	1
Server: Apache/2.2.16 (Ubuntu)	1
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch	1
Server: Apache/2.2.14 (Win32) DAV/2 mod_autoindex_color PHP/5.3.1 mod_apreq2-20090110/2.7.1 mod_perl/2.0.4 Perl/v5.10.1	1
Server: nginx/0.7.67	1
Server: nginx/0.7.65	1
Server: nginx/0.6.32	1
Server: CherryPy/3.1.2	1

Table 2
Public IP Banner Counts (11/09/2010 dataset)

Server Banner	Count
Server: Apache	21
Server: Apache/2.2.3 (CentOS)	18
Server: Apache/2.2.14 (Ubuntu)	11
Server: Apache/2.2.12 (Ubuntu)	8
Server: Apache/2.2.16 (Debian)	7
Server: lighttpd/1.4.19	6
Server: Microsoft-IIS/6.0	6
blank	5
Server: Apache/2.2.16 (Ubuntu)	4
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch	4
Server: Apache/2.2.9 (Debian)	3
Server: Microsoft-IIS/5.1	2

Table 2
Public IP Banner Counts
(11/09/2010 dataset)

Server Banner	Count
Server: lighttpd/1.4.28	2
Server: Apache/2.2.9 (Debian) mod_ssl/2.2.9 OpenSSL/0.9.8g	2
Server: lighttpd/1.4.26	2
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch mod_ssl/2.2.9 OpenSSL/0.9.8g	2
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny8 with Suhosin-Patch	2
Server: httpd	2
Server: nginx/0.7.62	2
Server: Apache/2.0.52 (CentOS)	2
Server: nginx	2
Server: Apache/2.0.52 (Red Hat)	2
Server: nginx/0.7.65	2
Server: WSGIServer/0.1 Python/2.5.2	2
Server: Apache/2.2.11 (Ubuntu) PHP/5.2.6-3ubuntu4 with Suhosin-Patch mod_ssl/2.2.11 OpenSSL/0.9.8g	2
Server: nginx/0.6.35	2
Server: Apache/2.2.6 (FreeBSD) mod_ssl/2.2.6 OpenSSL/0.9.8e DAV/2	1
Server: Apache/2.2.15 (Mandriva Linux/PREFORK-3.1mdv2010.1)	1
Server: Apache/1.13.9 (Red Hat)	1
Server: Apache/2.2.16 (Unix) PHP/5.3.2	1
Server: Abyss/2.5.0.0-X2-Win32 AbyssLib/2.5.0.0	1
Server: Apache/2.0.52 (BlueQuartz)	1
Server: Apache/2.2.8 (ASPLinux)	1
Server: Apache/2.2.16 (Win32)	1
Server: Apache/2.2.10 (Linux/SUSE)	1
Server: Apache/2.2.13 (Unix) mod_ssl/2.2.13 OpenSSL/0.9.8k PHP/5.2.12	1
Server: Apache/2.2.11 (Debian) mod_gnutls/0.5.1 PHP/5.2.9-2 with Suhosin-Patch mod_ssl/2.2.11 OpenSSL/0.9.8g	1
Server: Apache/2.2.14 (Win32) SVN/1.6.3 mod_ssl/2.2.14 OpenSSL/0.9.8k PHP/5.3.0 mod_ftp/0.9.6 DAV/2	1
Server: Apache/2.2.14 (Win32) PHP/5.3.1	1
Server: Apache/2.2.14 (Unix) mod_ssl/2.2.14 OpenSSL/0.9.8l DAV/2	1
Server: Apache/2.2.14 (FreeBSD) mod_ssl/2.2.14 OpenSSL/1.0.0 DAV/2 SVN/1.6.9	1
Server: Apache/2.2.8 (Ubuntu) DAV/2 SVN/1.5.1 PHP/5.2.4-2ubuntu5.12 with Suhosin-Patch mod_ssl/2.2.8 OpenSSL/0.9.8g mod_wsgi/2.0 Python/2.5.2 mod_perl/2.0.3 Perl/v5.8.8	1
Server: Apache/2.2.11 (Ubuntu) PHP/5.2.6-3ubuntu4.6 with Suhosin-Patch	1
Server: Apache/2.2.13 (Linux/SUSE)	1
Server: Apache/2.2.16 (EL)	1
Server: Ilonia/1.0.28 (Unix) mod_bash/1.10 FBI/0.0.1 oae/KG10.01	1
Server: Zope/(Zope 2.10.6-final, python 2.4.4, darwin) ZServer/1.1 Plone/3.1.1	1

Table 2
Public IP Banner Counts
(11/09/2010 dataset)

Server Banner	Count
Server: thttpd/2.25b 29dec2003	1
Server: Some random file server	1
Server: Roxen/5.0.403-release2	1
Server: RomPager/4.51 UPnP/1.0	1
Server: OmniSecure/3.0a5	1
Server: nginx/0.8.53	1
Server: nginx/0.7.67	1
Server: nginx/0.6.39	1
Server: nginx/0.6.32	1
Server: Microsoft-IIS/7.5	1
Server: lighttpd/1.5.0	1
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny3 with Suhosin-Patch	1
Server: Jetty(6.1.x)	1
Server: Apache/2.2.8 (Ubuntu) mod_python/3.3.1 Python/2.5.2 PHP/5.2.4-2ubuntu5.10 with Suhosin-Patch mod_ssl/2.2.8 OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8	1
Server: gateway	1
SERVER: EPSON_Linux UPnP/1.0 Epson UPnP SDK/1.0	1
Server: dhttpd/1.02a	1
Server: Cherokee/1.0.8 (Ubuntu)	1
Server: Apache/2.2.9 (Fedora)	1
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny9 with Suhosin-Patch mod_ssl/2.2.9 OpenSSL/0.9.8g mod_perl/2.0.4 Perl/v5.10.0	1
Server: Zope/(Zope 2.9.7-final, python 2.4.6, linux2) ZServer/1.1	1
Server: Apache/2.2.9 (Debian) PHP/5.2.6-1+lenny4 with Suhosin-Patch	1
Server: Apache/2.2.9 (Debian) mod_fastcgi/2.4.6 mod_gnutls/0.5.1	1
Server: Apache/2.2.9 (Debian) DAV/2 SVN/1.5.1 PHP/5.2.6-1+lenny9 with Suhosin-Patch mod_ssl/2.2.9 OpenSSL/0.9.8g	1
Server: Apache/2.2.9 (Debian) DAV/2 mod_fastcgi/2.4.6 Phusion_Passenger/2.2.15 PHP/5.2.6-1+lenny9 with Suhosin-Patch mod_python/3.3.1 Python/2.5.2 mod_ssl/2.2.9 OpenSSL/0.9.8g mod_perl/2.0.4 Perl/v5.10.0	1
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.12 with Suhosin-Patch mod_ssl/2.2.8 OpenSSL/0.9.8g	1
Server: Apache/2.2.8 (Ubuntu) mod_python/3.3.1 Python/2.5.2 PHP/5.2.4-2ubuntu5.12 with Suhosin-Patch mod_ssl/2.2.8 OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8	1
Server: lighttpd/1.4.22	1