



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

## Interested in learning more?

Check out the list of upcoming events offering  
"Network Monitoring and Threat Detection In-Depth (Security 503)"  
at <http://www.giac.org/registration/gcia>

## Detecting Torrents Using Snort

GIAC GCIA Gold Certification

Author: Richard Wanner, [rwanner@pobox.com](mailto:rwanner@pobox.com)  
Advisor: Leonard Ong

Accepted: November 28, 2008

### Abstract

*It is estimated that one-third of the traffic on the Internet is peer-to-peer. The fact is that peer-to-peer protocols such as BitTorrent provide a very efficient way to distribute large files such as operating system ISOs. Unfortunately that also makes peer-to-peer protocols a very efficient way to download copyright content such as music and movies. Regardless of whether corporate policy prohibits downloading of copyrighted content, or prohibits all peer-to-peer usage, it is essential to be able to detect the various aspects of peer-to-peer usage. This paper decomposes BitTorrent and the associated protocols used in conjunction with BitTorrent downloads to devise a number of different ways to detect the aspects of this traffic. This research is then used to create Snort signatures which can be implemented to detect the BitTorrent traffic in your environment.*

*Introduction*

Peer to peer applications are increasingly under scrutiny especially on corporate networks. Peer to peer applications are often cast as the villain due to their association with the downloading of copyrighted content such as music, movies, and software. However certain peer-to-peer applications have value and are permitted or at least tolerated in some corporate networks. One such example of this is BitTorrents.

Like all peer-to-peer technologies BitTorrents can be used to download copyrighted music, video, and software, through torrent tracker sites such as demonoid.com, torrentsproxy.com, mininova.org, torrentreactor.to and numerous others, (Gil, 2007). Gartner has waded into this discussion describing the impact of BitTorrents as “could be one of the most disruptive technologies in the next few years” (Prentice, McGuire, 2005). While it is true that BitTorrents can be utilized to download copyrighted content, BitTorrents also have legitimate uses. One common legitimate use is in the distribution of large file content like operating system CD and DVD ISOs. In the past organizations that wished to distribute such content had to bear the costs of servers and bandwidth required to support the download. Peer-to-peer protocols such as BitTorrent permit these organizations to substantially reduce the costs of distribution by permitting users who have even a small portion of the download to participate in distributing the content thus also distributing the CPU and network load and providing a very efficient way to distribute and download large software distributions. As one example, BitTorrent is the preferred method of downloading the Fedora Linux distribution (Fedora, 2007).

To quote Dr. Eric Cole "Prevention is ideal, but detection is a must" (Cole, 2003). With that in mind this paper looks at detecting the various aspects of BitTorrent use in the network. As a case study this paper uses the website mininova.org as a basis for analysis of network traffic created by the various aspects of BitTorrents and attempts to create snort rules to detect these phases. Mininova.org is a popular torrent tracker site for torrents containing a wide variety of content, the majority of which is copyrighted.

# 1. Anatomy of BitTorrent

There is a whole new vocabulary associated with BitTorrents. In order to improve the understanding of the rest of the paper it is best to provide a basic understanding of the terminology involved.

## 1.1. Definitions

**Downloader** – a BitTorrent client (peer) which is involved in a transfer of content, but which does not yet have the complete content available for sharing (seeding). Also referred to as a leech or leecher.

**Peer** - any BitTorrent client involved in a transfer. This includes both clients that are downloading, and those that have completed downloading and are only providing pieces to clients that are downloading (seeders). (BitTorrent, 2002)

**Seeder** – any BitTorrent client (peer) which has a complete copy of the shared content and is sharing it to the P2P network (seeding).

**Swarm** – all clients (peers) involved in a transfer, whether downloaders or seeders, are referred to as a swarm.

**Torrent File** – is a file which contains information about the content to be shared. This file contains two mandatory sections.

- An announce section which specifies the [URL](#) of the tracker.
- An info section which contains the names for the files, their lengths, the piece length used, and a [SHA-1 hash code](#) for each piece, which clients should use to verify the integrity of the data they receive.

**Tracker** – As its name suggests, the tracker is a process which keeps track of information about the download. The tracker is a web-based service which answers HTTP GET requests. The base URL is the announce URL defined in the torrent file. (TheoryOrg, 2006) The most common requests can include:

- List of peers involved in the torrent.
- Port the client is listening on.
- Statistics about the download including the total amount downloaded, the total amount

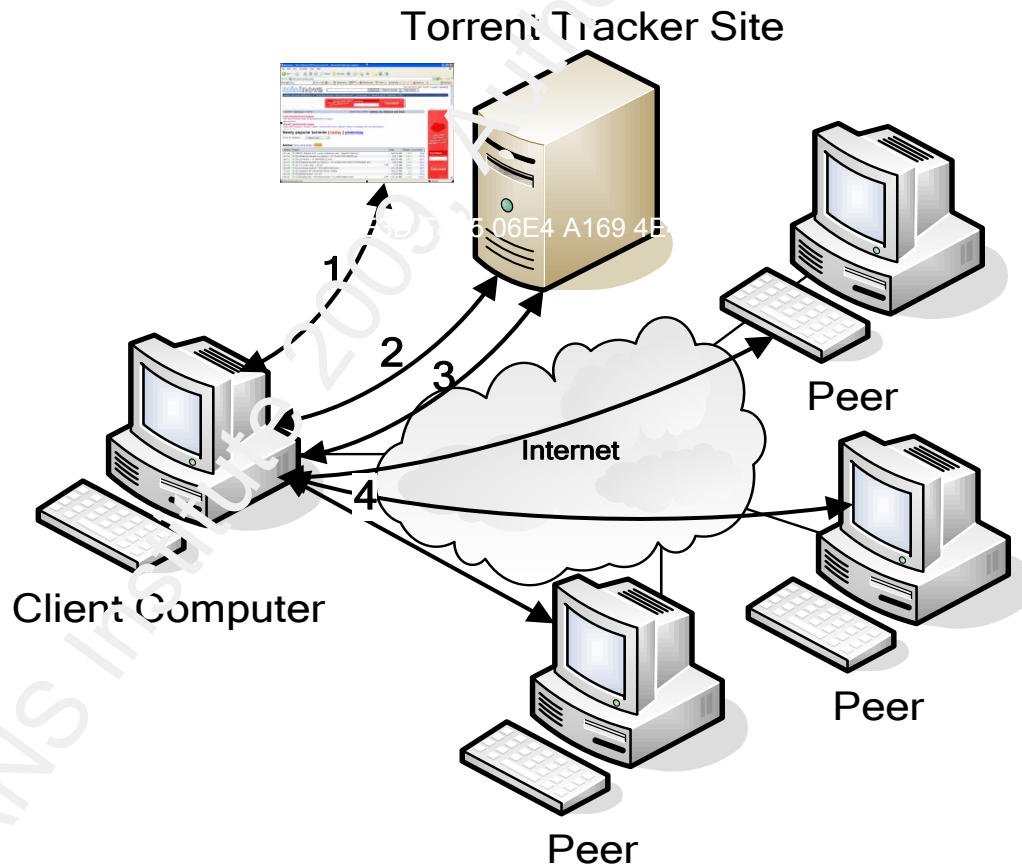
uploaded, and the amount remaining.

**Tracker site** – a site, typically web based, which provides an index of content available for sharing and a link to download the torrent file used to initiate the download of the content.

Usually this site, or a related site, keeps track of statistics related to the torrent. Note that this site does not contain the actual content, only the tracker that points to the content.

## 1.2. Typical Swarm Architecture

Although there can be some variation in the way a typical torrent swarm is implemented and several advanced features of the BitTorrent protocol which are not considered as part of this paper, the following figure provides a generic high level view of the steps in joining a typical torrent swarm.



*Figure: Typical Swarm Architecture*

Step 1: The user connects to a web server hosting a torrent tracking site.

Step 2: The user downloads a torrent metafile file containing information on the content he wishes to download. One of the pieces of information available in the metafile is the location of a tracker (or trackers) which manage the swarm containing the content.

Step 3: Using a BitTorrent client the user connects to the tracker requesting information about this swarm. One of the pieces of information received from the tracker is the IP address information for peers involved in the swarm.

Step 4: The BitTorrent client contacts peers requesting pieces of the content. At this point the client has become a downloader peer in the swarm and will be able to download pieces of the content from other peers and will provide pieces of the content for upload to peers which do not yet have those pieces. This will continue until the client has received all of the pieces of the content. At this point the client will stop downloading and will only upload to others (seed).

## 2. Anatomy of a Torrent Transaction

Mininova.org is a popular tracker site devoted to tracking content of virtually any type imaginable, from movies, to games, software, books, and numerous other types of content. The vast majority of the content is copyrighted. This site was chosen for this case study for a number of reasons. In order to keep this paper family friendly it was important that the case study use a site that while still being representative of BitTorrent tracker sites, does not have objectionable content all over its pages. While Mininova.org does track adult content, it does not have adult ads all over its pages. In addition, while the majority of the content appears to be copyrighted, mininova.org does track some content that may not be copyright or at the very least is not being vigorously defended. Rather than risking the wrath of the Recording Industry Association of America (RIAA), or similar organizations, I will utilize content that fits into this category.

### 2.1. Web Connection

The base URL is <http://www.mininova.org/>. This page contains some background info, some advertising and an index of torrents being tracked by the site.

## Detecting Torrents Using Snort

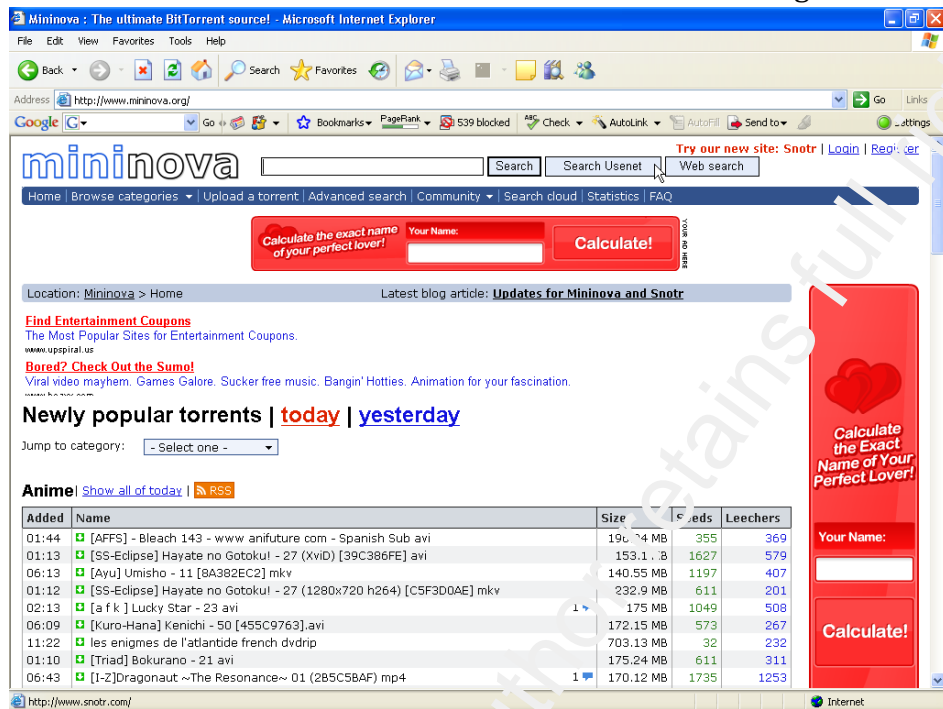


Figure: MiniNova – initial connection

The figure below shows more of the index.

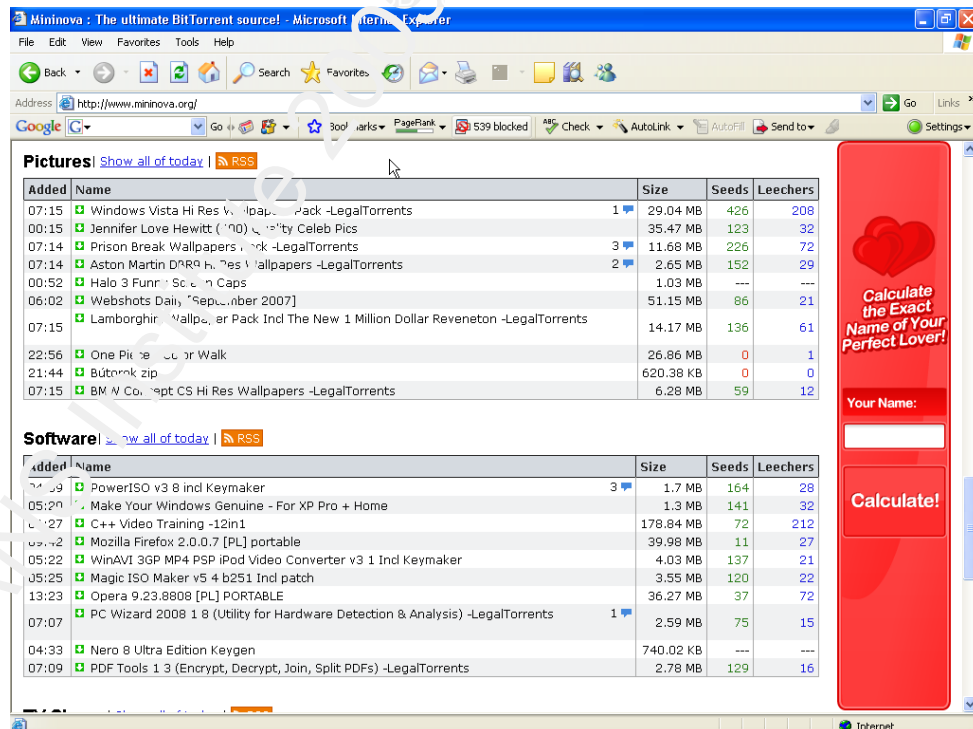


Figure: MiniNova – torrent index

Remember the goal of this exercise is to find ways to detect the torrent transactions. Utilizing snort as a sniffer to capture this session, we see that this is a typical web connection via http.

The extract from the dump below shows a three-way handshake to 87.253.147.140.

```
05/25-02:56:54.588235 172.20.10.70:2498 -> 87.233.147.140:80
TCP TTL:128 TOS:0x0 ID:54717 IpLen:20 DgmLen:48 DF
*****S* Seq: 0xB88EC74D Ack: 0x0 Win: 0x4000 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

```
05/25-02:56:54.597841 87.233.147.140:80 -> 172.20.10.70:2498
TCP TTL:50 TOS:0x0 ID:0 IpLen:20 DgmLen:48 DF
***A***S* Seq: 0x448D2F5 Ack: 0xB88EC74E Win: 0x16D0 TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

```
05/25-02:56:54.59/847 172.20.10.70:2498 -> 87.233.147.140:80
TCP TTL:128 TOS:0x0 ID:54719 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x288EC74E Ack: 0x448D2F6 Win: 0x470 IpLen: 20
```

A quick nslookup confirms that 87.233.147.140 is [www.mininova.org](http://www.mininova.org)

```
# nslookup 87.233.147.140
```

Non-authoritative answer:

140.147.233.87.in-addr.arpa    name = [www.mininova.org](http://www.mininova.org).

More interestingly a little more research utilizing domain tools.com's (domain tools, 2007) reverse IP lookup tool shows us that the only thing hosted on this site is Mininova in a few different flavours.

87.233.147.140      Look Up

There are 3 domains hosted on this IP address.

1. [Mininova.com](http://Mininova.com)
2. [Mininova.net](http://Mininova.net)
3. [Mininova.org](http://Mininova.org)



Figure: MiniNova – domaintools.com reverse lookup

By clicking on one of the items in the index, we can get to the torrent detail screen. This screen provides more information about the content including size, availability of the content, and the download link.



Figure: MiniNova – torrent detail

Again, this is just a standard web transaction utilizing http. It does not provide us any new way of detecting this traffic.

Clicking on the download link will result in the download of the torrent metainfo file.

From the sniffer trace we can see the HTTP GET for the torrent metainfo file. Notice in this trace, and the subsequent screenshot, the metainfo file has an extension of .torrent.

```
05/25-02:57:04.916664 172.20.10.70:2544 -> 87.233.147.130:80
TCP TTL:128 TOS:0x0 ID:55150 IpLen:20 DgmLen:894 DF
***AP*** Seq: 0x7F0AD9D Ack: 0x10BEB7D3 Win: 0x4470 TcpLen: 20
47 45 54 20 2F 74 6F 72 72 65 6E 74 73 2F 39 32 GET /torrents/92
33 39 33 39 2E 74 6F 72 72 65 6E 74 20 48 54 54 3939.torrent HTTP
50 2F 31 2E 31 0D 0A 41 63 63 65 70 74 3A 20 69 P/1.1..Accept: i
6D 61 67 65 2F 67 69 66 2C 20 69 6D 61 67 65 2F image/gif, image/
78 2D 78 62 69 74 6D 61 70 2C 20 69 6D 61 67 65 x-xbitmap, image
2F 6A 70 65 67 2C 20 69 6D 61 67 65 2F 70 6A 70 /jpeg, image/pjp
```

Note that we are now communicating with 87.233.147.130 to download the torrent metainfo file.

```
# nslookup 87.233.147.130
```

Non-authoritative answer:

```
130.147.233.87.in-addr.arpa name = loadbalancer.mininova.org.
```

Clicking on Open will result in the torrent metainfo file being downloaded and transferred to the µtorrent client.

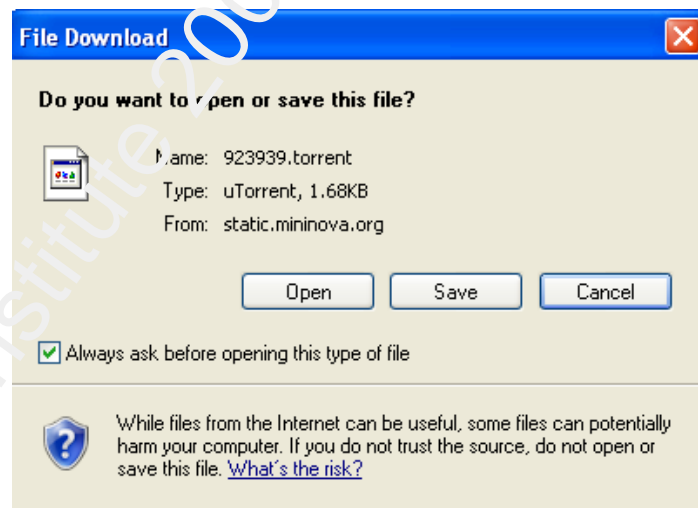


Figure: MiniNova – torrent download

In this case the BitTorrent client used is µtorrent. µtorrent when it is passed the torrent metainfo file it opens a dialog box which permits you to define what content to download, where to store it, and a few other parameters. Once “OK” is clicked control is passed to the µtorrent client to start downloading the content.

Rick Wanner

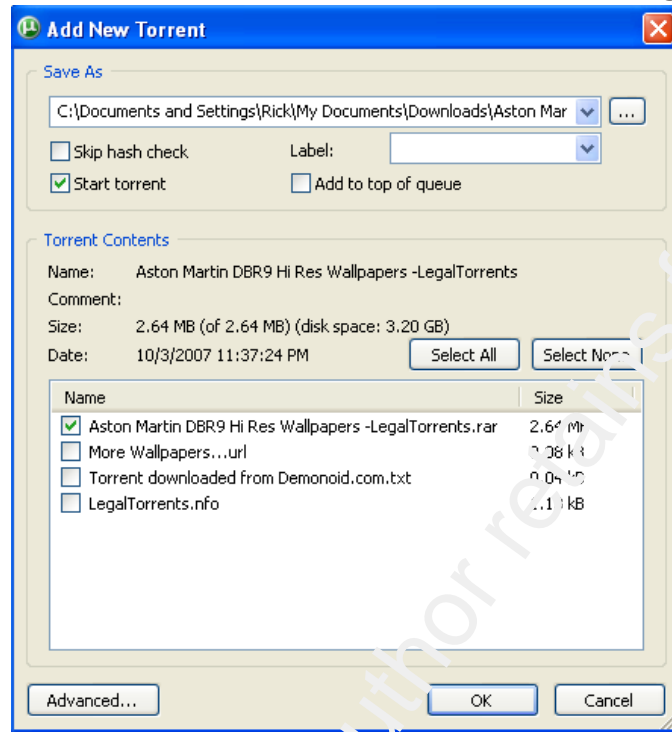


Figure: Mini 'ova – µtorrent download

Once “OK” is clicked control is passed to the µtorrent client to start downloading the content pointed to by the metafile.

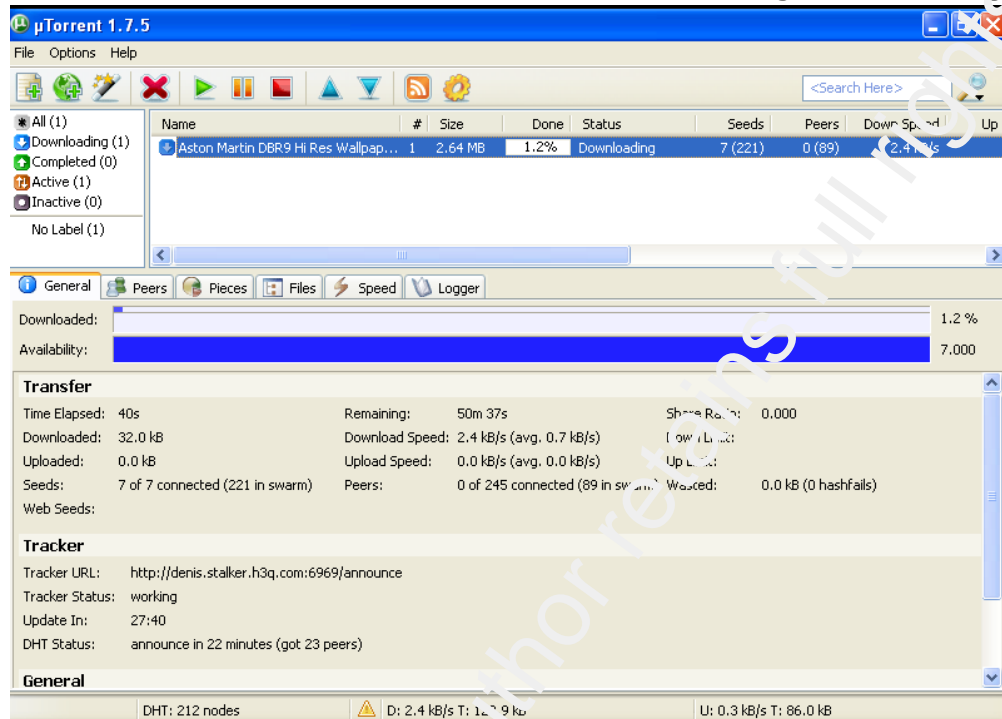


Figure: µTorrent client download

Once the download has completed the torrent will switch to seeding mode, where it is no longer downloading content, but is only seeding for others to download.

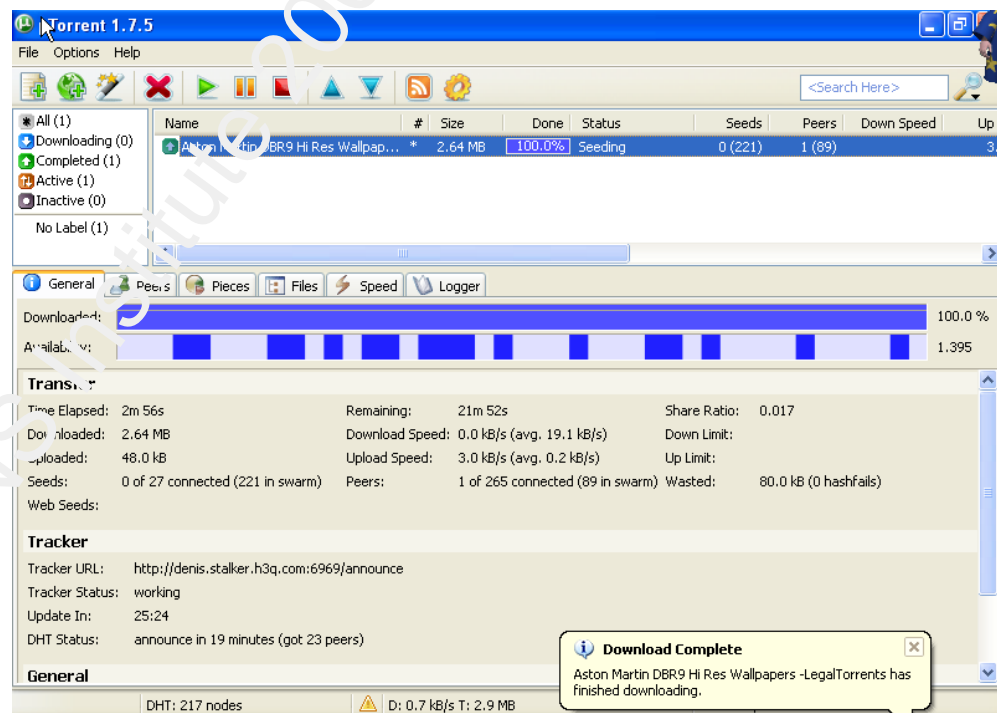


Figure: µtorrent seeding

### 3. Building the Snort Signatures

#### 3.1. Anatomy of a Snort Signature

While it is beyond the scope of this paper to go into details on how to build snort signatures, a basic tutorial will improve the clarity of the remainder of the paper.

Snort rules are divided into two sections. The first section is the rule header, which describes under what circumstances snort trigger the rule based on high level characteristics of the network traffic flow such as direction of flow, protocols, IP addresses, ports, etc. The header also contains the action that should be taken if the rule is triggered. The most common action is “alert”, which is the one that we will use for all of the rules we are building in this paper, although others are available.

The second section is the rule options section which may contain additional, more detailed, matching criteria and describes what snort should output if the signature is triggered.

In the signature below:

```
alert tcp any any -> 10.10.10.0/24 80 (content:"GET"; msg:"WWW GET detected"; sid:1000001; rev:1;)
```

The portion of the rule up to the open round bracket is the rule header and within the brackets are the rule options.

In this example the action is “alert”. The source information is on the left side of the “->”. In this case the rule is set to trigger on any TCP traffic with any source address and source port. The right side of the header indicates to match on a destination IP in the 10.10.10.0/24 subnet and a destination port of 80.

The options section checks the content of the matched packet to see if it contains the string “GET”, a common string in web transactions. If the content matches it will put out an alert with the messages “WWW GET detected”. There are a couple of other options which bear some explaining. In order to more easily identify a particular snort rule, each rule should be assigned a sid, or snort identifier. Snort reserves all sids below one million for itself, so user generated rules should have a sid of one million or greater. The option used in this rule is the rev: which tells the version number of the particular rule.

One other aspect of Snort worth pointing out is that by tradition user created snort rules are placed in the local.rules file in the Snort rules directory.

This is a very basic snort rules primer, but it should be enough to permit understanding of the examples provided in this paper. Let's get on to defining some Snort rules for detecting the BitTorrent traffic.

### 3.2. Web Session

This provides one possible way of detecting access to this site. Snort provides the capability to create signatures to detect if certain URLs have been accessed. A check for the various Mininova domains in the URL would provide a way of detecting access. We could also potentially use the IP address, but this will probably be less effective. The IP address is more likely to change than the domain name.

A simple snort signature to detect access to the Mininova site would be:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P mininova"; content:"GET";
content:"mininova"; sid:1100021; rev:1)
```

Breaking the signature down, it looks for outbound HTTP GET traffic with a content of "mininova". When implemented the alert generated by this alert looks as follows:

```
[**] [1:1100021:1] P2P mininova [^*]
[Priority: 0]
04/25-10:26:08.342435 142.160.5.95:5200 -> 72.14.207.104:80
TCP TTL:127 TOS:0x0 ID:12407 IpLen:20 DgmLen:1151 DF
***AP*** Seq: 0x4A2993C1 Ack: 0x38D70FBD Win: 0x403D TcpLen: 20
```

This approach does have some drawbacks. The first is that this type of signature will only work for identified sites for which signatures exist. While there are a relatively small set of high runner torrent tracker sites on the Internet which count for the majority of traffic, there are thousands of others that exist. It could be onerous to try to create and keep a list up to date of all sites. However this is a reasonable method if you want to detect access to obviously bad sites.

The second drawback is that this signature will be noisy. A web page is composed of several elements that are all loaded independently. So a typical web page will cause this signature to be triggered many times all related to the same access. The main page of Mininova generated 27 instances of the above alert.

Snort does provide a mechanism for thresholding alerts. The alert will still trigger and thus consume resources on the snort probe, but it will only be displayed based on the threshold values.

For example modifying the signature to:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "FTP mininova"; content:"GET";
content:"mininova"; threshold: type limit, track by_src, count 1, seconds 60; sid:1100021; rev:1;)
```

This defines a threshold that will only display one alert per 60-second interval for each source IP. This is an effective way to threshold these alerts to a reasonable level.

Technically, there is no problem with browsing the torrent tracker websites, the real potential issue begins when the download begins. Let's see if we can utilize aspects of the torrent transaction to detect this traffic.

### 3.3. Torrent metainfo File Download

As discussed earlier the key to initiating a download utilizing the BitTorrent protocol is the download of a metafile which contains the tracker information for the torrent. There are a couple of ways we could detect this action. The first is to watch for the .torrent file extension.

Looking at the sniffer trace for this transaction you can clearly see the response for the request to download the torrent metafile.

0000	00 15 b7 77 d9 ee 00 10 db e4 e3 00 08 00 45 00	...w.... ..E
0010	01 21 15 bb 40 00 35 06 76 7b 54 13 ae 2d ac 14	!..@.5. v{T -..
0020	0a 4c 00 50 18 cb 18 c3 88 6c a2 00 60 50 50 18	.L.P.... .l. .P.
0030	19 e9 20 62 00 00 48 54 54 50 2f 31 2e 31 20 32	.. b..HT T/9 1 2
0040	30 30 20 4f 4b 0d 0a 53 65 72 76 65 72 3a 20 41	00 OK..S eue: A
0050	70 61 63 68 65 2d 43 6f 79 6f 74 65 2f 31 2e 31	pache-Cr yate/1.1
0060	0d 0a 43 6f 6e 74 65 6e 74 2d 64 69 73 70 6f 73	..Conten s-dispos
0070	69 74 69 6f 6e 3a 20 61 74 74 61 63 68 6d 65 6e	ition: a ttachmen
0080	74 3b 20 66 69 6c 65 6e 61 6d 65 3d 62 61 74 74	t; filen ame=batt
0090	6c 65 73 74 61 72 2e 67 61 6c 61 63 74 69 63 61	lesstar g alactica
00a0	2e 73 30 34 65 31 30 2e 68 64 74 76 2e 78 76 69	.s04e10. hdtv.xvi
00b0	64 2d 6c 6d 61 6f 2e 61 76 69 2e 74 6f 72 72 65	d-lmao.a vi.torre
00c0	6e 74 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65	r..Cont ent-Type
00d0	3a 20 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 2d	:applic ation/x-
00e0	62 69 74 74 6f 72 72 65 6e 74 0d 0a 54 72 61 6e	Bittorre nt..Tran
00f0	73 66 65 72 2d 45 6e 63 6f 64 69 6e 67 3a 20 63	sfer-Enc oding: c
0100	68 75 6e 6b 65 64 0d 0a 44 61 74 65 3a 20 54	hunked.. Date: Tu
0110	65 2c 20 32 39 20 4a 75 6c 20 32 30 30 38 20 30	e, 29 Ju l 2008 0
0120	34 3a 34 39 3a 30 30 20 47 4d 54 0d 0a 0d 6e 6e	4:49:00 GMT....

Figure: torrent metafile download

Looking closely you can see that this is an HTTP response and that it contains the name of the torrent metafile in this case “battlestar.galacticass04e10.hdtv.xvid-lmao.avi.torrent” a torrent metafile for an episode of Battlestar Galactica.

Using this information to detect the .torrent extension and extrapolating from the signatures we have looked at above, a basic snort rule could be:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P .torrent metafile"; content:"HTTP/";
content:".torrent"; flow:established, to_server; classtype:policy-violation; sid:1100010; rev:1;)
```

Breaking down the signature, it is simply looking for an HTTP response containing the string “.torrent”. While “.torrent” is a fairly specific string it could show up in a document or other file and create false positives.

Another possible method would be to look for a deterministic pattern in the metafile contents.

Looking into the specification for the metafile (theory.org 2008), we see that the metafile contains an announce section that is composed of the tag “announce” followed by the URL of the tracker information. The catch is that all contents of the metafile are bencoded. Without getting into too much detail on bencoding, the metafile is composed of fields of the form “d<bencoded string><bencoded element>e”. A bencoded string is of the form <length of string>:<string>. In this case the “announce” tag when bencoded will be “8:announce”. Since it is at the beginning of



0000	00	15	b7	77	d9	ee	00	10	db	e4	e3	00	08	00	45	00	.....E.
0010	05	dc	15	bf	40	00	35	06	71	bc	54	13	ae	2d	ac	14	....@.5. q.T.-..
0020	0a	4c	00	50	18	cb	18	c3	89	6b	a2	00	60	50	50	10	L.P.....k.. PP.
0030	19	e9	1e	03	00	00	64	38	3a	61	6e	6e	6f	75	6e	63	.....d8 :announc
0040	65	31	30	33	3a	68	74	74	70	3a	2f	2f	32	2e	74	74	e103:htt p://2.tr
0050	61	63	6b	65	72	2e	74	76	74	6f	72	72	65	6e	74	73	acker.tv torrents
0060	2e	63	6f	6d	2f	54	72	61	63	6b	65	72	53	65	77	76	.com/Tra ckerserv
0070	6c	65	74	2f	64	66	31	32	64	32	31	33	32	35	64	76	let/df12 d21325d6
0080	37	61	37	33	35	39	64	63	35	62	38	33	61	37	33	33	7a7359dc 5b83a753
0090	33	62	36	39	31	61	64	35	66	35	32	63	2f	33	39	36	3b691ad5 f52c/596
00a0	37	34	35	2f	61	6e	6e	6f	75	6e	63	65	31	33	3a	61	745/anno uncel3:a
00b0	6e	6e	6f	75	6e	63	65	2d	6c	69	73	74	6c	6c	31	30	nnounce- 1ist1110
00c0	33	3a	68	74	74	70	3a	2f	2f	32	2e	74	74	61	63	6b	3:http:/ /2.track
00d0	65	72	2e	74	76	74	6f	72	72	65	6e	74	74	2e	63	6f	er.tvtor rents.co
00e0	6d	2f	54	72	61	63	6b	65	72	53	65	72	74	6c	65	74	m/Tracke rservlet
00f0	2f	64	66	31	32	64	32	31	33	32	35	64	36	37	61	37	/df12d21 325d67a7
0100	33	35	39	64	63	35	62	38	33	61	33	33	33	33	62	36	359dc5b8 3a7533b6
0110	39	31	61	64	35	66	35	32	63	2f	33	39	36	37	34	35	91ad5f52 c/596745
0120	2f	61	6e	6e	6f	75	6e	63	65	31	33	33	3a	68	74	74	/announc e103:htt
0130	70	3a	2f	2f	31	2e	74	72	61	33	6b	65	72	2e	74	76	p://1.tr acker.tv

Using this information a basic snort signature to detect the torrent metafile download is:

In the content: section the match string contains a “\” which is used to escape the “:”, since a “:” is a special character in snort.

### 3.4. BitTorrent Protocol

Rick Wanner

### 3.4.1. BitTorrent Handshake

If we look at the BitTorrent protocol a little closer there are a couple of ways that could detect the behavior of BitTorrent using snort. For example the BitTorrent protocol (Cohen, 2008) utilizes a handshake that is used between peers in the swarm to initiate a connection.

From the BitTorrent specification (theory.org, 2008) “The handshake is a required message and must be the first message transmitted by the client. It is  $(40 + \text{len}(\text{pstr}))$  bytes long. ...in version 1.0 of the BitTorrent protocol,  $\text{pstrlen} = 19$ , and  $\text{pstr} = \text{"BitTorrent protocol"}$ ”.

The current version of the BitTorrent protocol is 1.0. With a little translation from protocol specification to English, this means that the protocol length will be 19 decimal bytes and the string “BitTorrent protocol” will be present in the output.

To lay it out in a more familiar format, here is a representation of the first 20 bytes of the BitTorrent handshake packet.

0	15	16	32
Pstrlen = 19			B
i			t
T			o
r			r
e			n
t			
p			r
o			t
o			c
o			l

Figure: BitTorrent protocol – first 22 bytes

This is only the first 20 bytes, the part we are interested in, this is followed by 8 reserved bytes, a 20-byte hash, and a 20-byte peer identifier string.

Looking at a capture of the handshake the 13 hex that corresponds to the 19 decimal pstrlen, and the “BitTorrent protocol” string is clearly visible.

0000	00	0f	66	36	92	ec	00	1a	73	71	b6	c3	08	00	45	00	..f6....sq....E.
0010	00	6c	66	2d	40	00	80	06	62	f0	c0	a8	01	65	cf	2f	.lf-@...b....e./
0020	a0	31	e3	0d	f7	43	51	b6	6b	8e	c4	29	21	a9	50	18	.1...CQ. k..)!..P.
0030	00	44	d9	23	00	00	13	42	69	74	54	6f	72	72	65	6e	.D.#...B itTorren
0040	74	20	70	72	6f	74	6f	63	6f	6c	00	00	00	00	00	10	t protoc ol.....
0050	00	01	f4	4b	51	c3	b0	2f	6e	9a	bf	dd	40	1c	79	14	..KQ../ n...@.y.
0060	fd	19	80	c0	9f	18	2d	55	54	31	37	37	30	2d	f3	9f	.....-U T1770-..
0070	c9	dd	f1	a8	3b	99	30	f0	4b	e4							....;.0. K.

Figure: BitTorrent handshake capture

This is a very unique pattern which can be used to identify a BitTorrent protocol in progress.

Using this information a basic snort signature to detect the BitTorrent handshake is:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP BitTorrent handshake";
flow:to_server,established; content:"BitTorrent protocol"; classtype:policy-violation; sid:1100012; rev:1;)
```

Breaking down the signature, it is simply looking for the string “BitTorrent protocol” in the data stream. This signature has the potential to cause a significant number of false positives, since any stream containing “BitTorrent protocol” will trigger this signature regardless of whether it is a BitTorrent handshake, or a harmless text file. We may need to find a way to make this signature more specific to eliminate those false positives.

## 3.4.2. Reserved Bits

As one way of making the above signature more specific I looked deeper into the protocol specification, it appears at first glance that there are 8 reserved bytes that are not used today and are supposed to be initialized to null. Looking at the capture it is easy to see the reserved bytes, unfortunately, they do not appear to be all nulled.

0000	00	0f	66	36	92	ec	00	1a	73	71	b6	c3	08	00	45	00	..f6....sq....E.
0010	00	6c	66	2d	40	00	80	06	62	f0	c0	a8	01	65	cf	2f	.lf-@...b....e./
0020	a0	31	e3	0d	f7	43	51	b6	6b	8e	c4	29	21	a9	50	18	.1...CQ. k..)!..P.
0030	00	44	d9	23	00	00	13	42	69	74	54	6f	72	72	65	6e	.D.#...B itTorren
0040	74	20	70	72	6f	74	6f	63	6f	6c	00	00	00	00	00	10	t protoc ol.....
0050	00	01	f4	4b	51	c3	b0	2f	6e	9a	bf	dd	40	1c	79	14	..KQ../ n...@.y.
0060	fd	19	80	c0	9f	18	2d	55	54	31	37	37	30	2d	f3	9f	.....-U T1770-..
0070	c9	dd	f1	a8	3b	99	30	f0	4b	e4							....;.0. K.

Figure: BitTorrent handshake - reserved bytes

This is a lesson in not including data in your signatures without validating that it is correct. Despite substantial investigation I have not been able to determine what the bytes set in the reserved area are used for.

Rick Wanner

### 3.5. Distributed Hash Table (DHT)

Earlier in the paper it was described that BitTorrent networks are supported by tracker sites which are used to find peers that have pieces of the content. But what if the tracker is unavailable?

For just this purpose the Distributed Hash Table (DHT) feature was added to create the concept of trackerless torrents (BitTorrent, 2009). If the client has enabled DHT then each client keeps a table of all known peers involved in the swarm, and how to contact them. In effect each peer becomes a tracker. (Gibbocool, 2009)

To enable DHT in µtorrent click “Enable DHT Network” on the BitTorrent screen of the preferences.

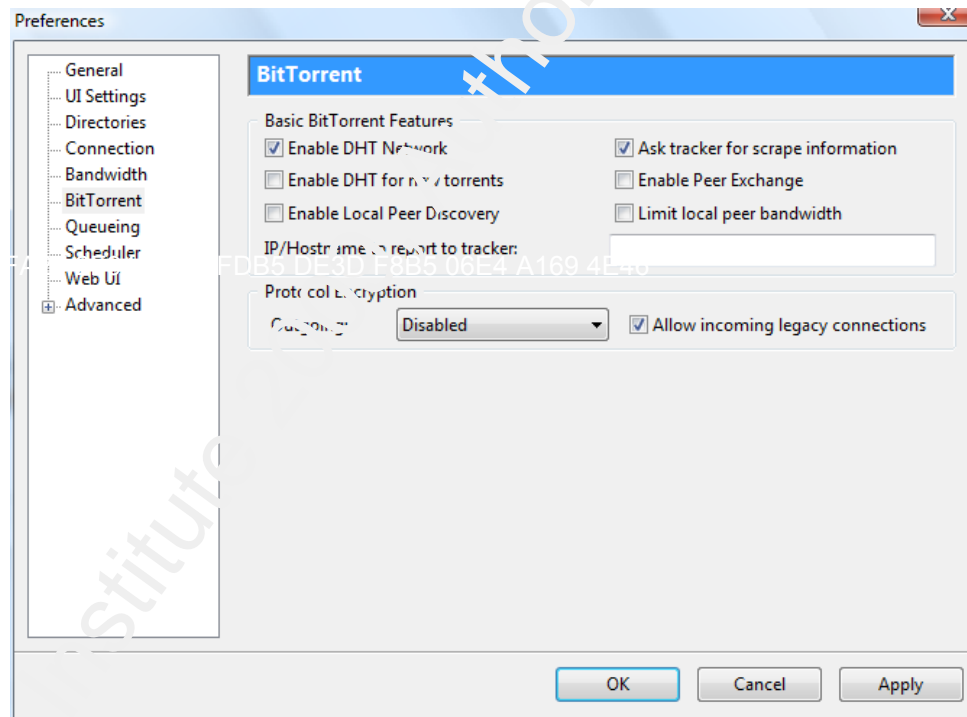


Figure: Enabling Distributed Hash Table (DHT) in µtorrent

According to the DHT specification, DHT consists of a number of different queries and corresponding responses. (theory.org, 2009)

- Ping – used to check if a peer is available.
- Find\_node – used to find the contact information for a peer.
- Get\_peers – requests a list of peers which have pieces of the content.
- Announce\_peer – announces the contact information for the peer to the network.

The most basic and most frequently used query is ping, so that is a reasonable place to start in detecting DHT usage.

According to the specification a DHT ping is a bencoded string consisting of a single argument which is a 20 byte node ID and the command type of ping.

When the ping command is bencoded it will be (theory.org 2008):

```
d1:ad2:id20:abcdefghij0123456789e1:q4:ping1:t2:aa1:y1:qe
```

where “abcdefghij0123456789” is a placeholder for the id.

One possible signature to detect DHT ping would be:

```
alert udp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrent DHT ping";
content:"d1\ad2\id20\."; content:"ping"; classtype:policy-violation; sid:1100021; rev:1;)
```

This signature generates a significant number of alerts, so a threshold is probably appropriate to keep this alert from filling the database.

```
alert udp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrent DHT ping";
content:"d1\ad2\id20\."; content:"ping"; threshold: type limit, track by_src, count 1 , seconds 60;
classtype:policy-violation; sid:1100021; rev:1;)
```

## 4. Encryption

It is estimated that more than 1/3 of the traffic on the Internet is peer-to-peer. This has lead to certain ISP's trying to find ways to reduce the impact of BitTorrent traffic on their networks. The most often employed technique is traffic-shaping. Traffic shaping allows ISPs to detect and limit the bandwidth utilization of BitTorrent traffic via protocol detection.

In order to counteract this approach the BitTorrent developers created a traffic obfuscation scheme called Message Stream Encryption (Azureus, 2007) which involves a Diffie-Helman key exchange and encryption of the header and optionally the body with the RC4 encryption protocol.

BitTorrent encryption can be enabled on the BitTorrent menu in the preferences in the Protocol Encryption section.

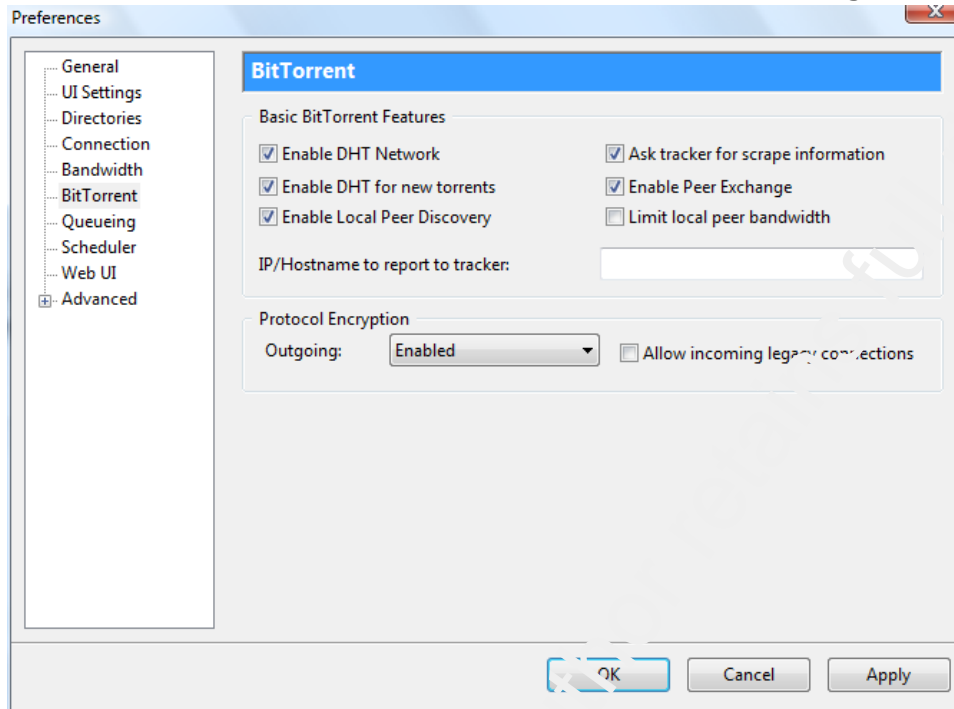


Figure: BitTorrent encryption

Once enabled, all BitTorrent traffic after the download of the .torrent metafile will be encrypted. This means that detection of torrent traffic can still be done by URL and by triggering on the metafile, but that detection of any of the BitTorrent protocol components will fail due to the traffic being encrypted.

## 5. Summary

This paper broke down BitTorrent and its related protocols into its constituent parts and showed how this knowledge can be used to create viable IDS signatures. Since the time the original research was performed signatures to detect some similar aspects of the BitTorrent application and protocol have been released by projects such as Bleeding Threats and more recently Emerging Threats (Jonkman, 2009). Although BitTorrent was used as the case study for this paper, the goal of this paper was to show how most applications can be broken down into its constituent parts and the related protocols with the aim of being able to decompose the components and create IDS signatures that will allow you to detect that application. With a little research and testing the concepts in this paper can easily be applied to most applications and IDS combinations to detect violations of acceptable use or other policies.

## 6. References

About Reverse IP. Retrieved October 12, 2007, from domaintools.com Web site:

<http://www.domaintools.com/reverse-ip>

Azureus Project, (2007, Dec 1). Message Stream Encryption - AzureusWiki . Retrieved May 14, 2008, from Azureus Wiki Web site: [http://www.azureuswiki.com/index.php/Message\\_Stream\\_Encryption](http://www.azureuswiki.com/index.php/Message_Stream_Encryption)

Cohen, B (2008, Jan 8). BitTorrent Protocol Specification. Retrieved March 10, 2008, from

[http://BitTorrent.org/beps/bep\\_0003.html](http://BitTorrent.org/beps/bep_0003.html) Web site: [http://BitTorrent.org/beps/bep\\_0003.html](http://BitTorrent.org/beps/bep_0003.html)

Cole, E (2003, December). SANS GSEC Lectures. SANSFire 2003, Washington, DC.

Fedora Project, (2007). FedorMain - Fedora Project Wiki. Retrieved April 28, 2007, from FedoraMain - Fedora Project Wiki Web site: <http://fedoraproject.org/wiki/>

Gibbocool, (2009, Dec 12). A guide to DHT (Trackerless Torrents). Retrieved May 16, 2009, from Mystery Axiom Forums Web site: <http://forums.mystery-axiom.com/showthread.php?t=24717>

Gil, P (2009, January 28). The Top 35 Torrent Sites of 2009. Retrieved January 28, 2009, from about.com Web site: [http://netforbeginners.about.com/od/peers/aring/a/torrent\\_search.htm](http://netforbeginners.about.com/od/peers/aring/a/torrent_search.htm)

Jonkman, Matt (2009). Emerging Threats. Retrieved May 25, 2009, from Emerging Threats Web site: <http://www.emergingthreats.net/>

Prentice, S, Macguire M (2005, June 27). Don't Overlook Legitimate Uses of File-Sharing Technology. Retrieved April 28, 2007, from Gartner Inc. Web site:

[http://www.gartner.com/DisplayDocument?doc\\_cd=129361](http://www.gartner.com/DisplayDocument?doc_cd=129361)

Roesch, M Green, C (2006, Apr 7). Snort Users Guide. Retrieved June 3, 2007, from Snort, the defacto standard for intrusion prevention/detection Web site:

[http://www.snort.org/docs/snort\\_manual/2.6.1/snort\\_manual.pdf](http://www.snort.org/docs/snort_manual/2.6.1/snort_manual.pdf)

Theory.org, (2008, Feb 12). Theory.org. Retrieved March 16, 2008, from BitTorrentSpecification - TheoryOrg Web site: <http://wiki.theory.org/BitTorrentSpecification>

Wikipedia, (2007, May 14). BitTorrent - Wikipedia, The Free Encyclopedia. Retrieved May 15, 2007, from Wikipedia.org. Wikipedia, The Free Encyclopedia Web site: <http://en.wikipedia.org/wiki/BitTorrent>

## Appendix: Snort Signatures

This section provides a list of the final versions of the signatures derived from the research for this paper. Assuming your snort configuration is properly configured and properly defines \$HOME\_NET, you should be able to cut and paste these signatures into the local.rules file.

# to detect .torrent file extension in HTTP GET

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P .torrent metafile request";
content:"HTTP/"; content:".torrent"; flow:established,to_server; classtype:policy-violation; sid:1100010;
rev:1;)
```

# to detect torrent metafile download

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrent metafile download";
content:"|64 38 3a|announce"; flow:established; classtype:policy-violation; sid:1100011; rev:1;)
```

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"P2P BitTorrent handshake";
flow:to_server,established; content:"BitTorrent protocol|0000 0000|"; classtype:policy-violation;
sid:1100012; rev:1;)
```

#detects various torrent tracker sites

# this is a list of high runners, but is far from complete

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P TVTorrents"; content:"tvtorrents";
threshold: type limit, track by_src, count 1 , seconds 60; sid:1100020; rev:1;)
```

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P mininova"; content:"GET";
content:"mininova"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100021; rev:1;)
```

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P thepiratebay.org"; content:"GET";
content:"thepiratebay"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100022; rev:1;)
```

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrentreactor"; content:"GET";
content:"torrentreactor"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100023; rev:1;)
```

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P demonoid"; content:"GET";
content:"demonoid"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100024; rev:1;)
```

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P bitsoup"; content:"GET";
content:"bitsoup"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100025; rev:1;)
```

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P bitnova"; content:"GET";
```



```

content:"bitenova"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100026; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrentportal"; content:"GET" ,
content:"torrentportal"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100027; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P youtorrent"; content:"GET";
content:"youtorrent"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100028; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P isohunt"; content:"GET";
content:"isohunt"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100029; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrentz"; content:"GET";
content:"torrentz"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100030; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrentscan"; content:"GET";
content:"torrentscan"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100031; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrentmatrix"; content:"GET";
content:"torrentmatrix"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100032; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrents.to"; content:"GET";
content:"torrents.to"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100033; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P filemp3.org"; content:"GET";
content:"filemp3"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100034; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P filemp3.org"; content:"GET";
content:"filemp3"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100035; rev:1;)
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrentspy"; content:"GET";
content:"torrentspy"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100036; rev:1;)

# detects DHT ping traffic
alert udp $HOME_NET any -> $EXTERNAL_NET any (msg: "P2P torrent DHT ping";
content:"d1\ad2\id20\"; content:"ping"; threshold: type limit, track by_src, count 1 , seconds 60;
classtype:policy-violation; sid:1100021; rev:1;)

```