It is estimated that more than 60% 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 presentation will describe the BitTorrent protocol and the associated aspects of BitTorrent communications and provides strategies to detect BitTorrent related traffic using Snort.
Detecting BitTorrents Using Snort

Contents
Objective ...................................................................................................................................... 3
BitTorrent .................................................................................................................................. 4
BitTorrent Clients ...................................................................................................................... 5
Detecting BitTorrent .................................................................................................................. 6
Torrent Tracker Websites .......................................................................................................... 8
Anatomy of a Snort Rule ............................................................................................................ 10
Detecting BitTorrent Tracker Website Access ........................................................................ 12
Detecting Torrent Metafile Download ...................................................................................... 14
Detecting BitTorrent Metafile Content ..................................................................................... 16
Detecting the BitTorrent Handshake ......................................................................................... 18
Trackerless Torrents .................................................................................................................. 20
Traffic Shaping .......................................................................................................................... 22
BitTorrent Encryption ............................................................................................................... 23
Summary ..................................................................................................................................... 24
Appendix: Snort Signatures ....................................................................................................... 25
Objective

- Understand the mechanisms involved in BitTorrent transfers.
- Identify ways to detect BitTorrent transfers.
- Devise Snort Signatures to detect BitTorrent traffic.
- Discuss encryption which provides a way to circumvent detection.

Objective

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 BitTorrent.

Like all peer-to-peer technologies BitTorrent can be used to download copyrighted music, video, and software, through torrent tracker sites such as demonoid.com, torrentspy.com, mininova.org, torrentreactor.to and numerous others. 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. Distributing content via BitTorrent permits these organizations to substantially reduce the costs of distribution by distributing the CPU and network load across a large number of machines. As one example, BitTorrent is the preferred method of downloading the Fedora Linux distribution.

To quote Dr. Eric Cole, "Prevention is ideal, but detection is a must.” With that in mind this presentation breaks down BitTorrent transfers into their constituent parts, identifies ways of detecting those elements and devises snort rules to detect these phases.
BitTorrent

- Peer-to-peer protocol invented by Bram Cohen.
- Very efficient way to distribute large files.
- Estimated that up to 60% of all Internet traffic is BitTorrent related.

BitTorrent

BitTorrent is a peer-to-peer protocol and application invented by Bram Cohen in 2001. BitTorrent is significantly more efficient than other peer-to-peer protocols because rather than downloading a file from a node in the network that has the content, BitTorrent allows all computers which have portions of the content, regardless of how small, to participate in the distribution of the content.

The user in possession of the original version of the content makes it available by creating a tracker and making the metafile that identifies the tracker available to users that would like to download the content. This first user is called a seeder and the initial content is called a seed. Users that download the metafile and begin to download the content are called peers and the entire group involved in the transfer is called a swarm. Once any peer has successfully downloaded any piece of the content it makes that piece available for other peers in the swarm to download. Once a peer has downloaded all the pieces of the content the peer stops downloading and becomes a seeder in the swarm.

It is easy to see why BitTorrent is much more efficient than the traditional model of one provider -> one downloader model of other peer-to-peer networks and traditional file transfer protocols like FTP. BitTorrent quickly distributes the content to multiple peers and quickly makes multiple copies of the content available in the swarm. The more peers in the swarm the quicker more copies are available for downloading from the swarm.

This efficiency has made BitTorrent a very popular way to distribute content. Depending on where you are in the world, up to 60% of Internet traffic can be BitTorrent related.
BitTorrent Clients

- BitTorrent 6
- BitComet
- LimeWire
- Shareaza
- Vuze (formerly Azureus)
- µtorrent

BitTorrent Clients

In order to participate in BitTorrent transfers you must have a client installed on the computer where you wish to download the content. BitTorrent clients come and go, but there are approximately 50 BitTorrent clients in use today. However the vast majority of users most likely use one of the six on this list.

BitTorrent is the original client created by BitTorrent founder Bram Cohen and supported by his company.

It is worth noting that LimeWire and Shareza are so called universal peer-to-peer clients in that they support other peer-to-peer networks like Gnutella as well as having BitTorrent functionality. The other four clients are pure Bittorrent clients.

While each of the others have features which have created a following, they all have core functionality as defined by the BitTorrent protocol specification.


The client used for the testing detailed in this presentation is µtorrent available at [http://www.utorrent.com/](http://www.utorrent.com/).
Detecting BitTorrent

- Torrent tracker website
- Torrent metafile
  - .torrent extension
  - Metafile contents
- BitTorrent Protocol
- Distributed Hash Table (DHT)

Detecting BitTorrent

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 presentation, the figure below provides a generic high level view of the steps in participating in a typical BitTorrent swarm.

Step 1: The user connects to a web server hosting a torrent tracker site.
Step 2: The user downloads a torrent metafile file containing information on the content the user 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).

There are several points in this communication where we can identify BitTorrent transfers.

- Access to the Torrent tracker websites.
- When the BitTorrent metafile is downloaded.
- The content of the BitTorrent metafile.
- There are some unique characteristics of the BitTorrent protocol which make it possible to detect it.
- When trackerless BitTorrents are enabled (not discussed above).

More details on each of these will be covered in the next few pages.
Torrent Tracker Websites

- Tracker websites allow peers to find content.
- There are over 150 known tracker websites.
  - Mininova
  - Torrentreactor
  - Isohunt
  - ThePirateBay
  - Demonoid
  - TVtorrents

Torrent Tracker Websites

Unlike other peer-to-peer networks like Gnutella and eDonkey, BitTorrent does not have a search capability built into the network. In order to find BitTorrent content you must connect to one of the many tracker websites available on the Internet.

Tracker websites come and go all the time, but at the moment there are over 150 known tracker websites available and involved in downloading copyrighted content. You are thinking 150+ tracker sites, how does anybody find anything? The most popular sites are aggregator sites that crawl and index other sites to create a master index of available BitTorrent content.

MiniNova, Torrentreactor, Isohunt, ThePirateBay, and Demonoid are all popular aggregator sites which index music, movies, television shows, games, applications, and other kinds of content. TVtorrents.com is a little different in that it is a member only site that only tracks television show related content.

The figure below shows a typical tracker website:
Clicking on a download link, in this case the green arrows, will result in the downloading of the BitTorrent metafile for that content. When the metafile is passed to the BitTorrent client it tells the client where to find the tracker for the content. This is the first step in starting a BitTorrent download.
Anatomy of a Snort Rule

- Rule Header – describes high-level rule trigger and action
- Rule Options – detailed matching criteria and output

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

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

Snort rules are divided into two sections. The first section is the rule header, which describes under what circumstances snort triggers 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. Another 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 presentation. Let’s get on to defining some Snort rules for detecting the BitTorrent traffic.
Detecting BitTorrent Tracker Website Access

- **http GET request**
- **One signature required for each tracker website**

```
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;)
```

---

**Detecting BitTorrent Tracker Website Access**

Snort provides the capability to generate alerts to detect if certain URLs have been accessed.

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 signature looks as follows:

```
[**] [1:1100021:1] P2P mininova [**]
[Priority: 0]
04/25-10:26:08.342435 142.165.5.95:5200 -> 72.14.207.104:80
TCP TTL:127 TOS:0x0 ID:12407 IpLen:20 DgmLen:1151 DF
***AP*** Seq: 0x4A2B93C1 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 numerous 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 the popular sites.
The second drawback is that this basic 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.

Modifying the signature to:

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;)

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.
Detecting Torrent Metafile Download

- Metafile has .torrent extension
- Downloaded as HTTP GET

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;)

Detecting Torrent Metafile 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.
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.galactica.s04e10.hdtv.xvid-lmao.avi.torrent” a torrent metafile for an episode of Battlestar Galactica.

Using this information to detect the “.torrent” 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 looking for an HTTP response containing the string “.torrent”. While “.torrent” is a fairly specific string it is possible it could show up in a document or other file and create false positives.
Detecting BitTorrent Metafile Content

- Contents are a bencoded dictionary containing “keys”.
- Announce is required key which identifies the URL of the tracker.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any
(msg: "P2P torrent metafile Download";
 content:"d8:\announce";
flow:established,to_server; classtype:policy-violation;
 sid:1100000; rev:1;)
```

### Detecting BitTorrent Metafile Content

Another possible detection method is to look for a deterministic pattern in the metafile contents.

Looking into the specification for the metafile, 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<encoded string><encoded 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 the field we know that the “d” will be appended as well, making “d8:announce” a string which appears in each torrent metafile.

Looking at a sniffer trace of the transfer it is possible to see the “d8:announce” in the file followed by the URLs of the trackers.
Using this information a basic snort signature to detect the torrent metafile download is:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (
  msg: "P2P torrent metafile Download";
  content:"d8:\\":announce"; flow:established,to_server; classtype:policy-violation;
  sid:1100000; rev:1;)
```

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

Breaking down the signature, it is simply looking for the string "d8:announce" in the data stream. This string should be precise enough to not create too many false positives.
Detecting the BitTorrent Handshake

Looking closely at the BitTorrent protocol specification it becomes clear that there is at least one easy way to detect the BitTorrent protocol. The BitTorrent protocol utilizes a handshake that is used between peers in the swarm to initiate a communication.

From the BitTorrent specification “The handshake is a required message and must be the first message transmitted by the client. It is \((49 + \text{len(pstr)})\) bytes long. …in version 1.0 of the BitTorrent protocol, pstrlen = 19, and pstr = "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.

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.

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:"P2P 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 reduce false positives.
Trackerless Torrents

- Decentralized method of finding peers
- Enabled peers keep distributed hash table (DHT) of available content.
- Formatted as bencoded dictionary.
- Most common command is DHT Ping
d1:ad2:id20:abcdefghij0123456789e1:q4:ping1:t2:a1:y1:qe

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;)

Trackerless Torrents

Earlier in this presentation 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. 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.

According to the DHT specification, DHT consists of a number of different queries and corresponding responses.

- 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 ad the command type of ping.
When the ping command is bencoded it will be:

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\:id2\:"; content:"ping"; classtype:policy-violation; sid:1100021; rev:1;)
```

Once DHT is enabled in the client 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\:id2\:"; content:"ping"; threshold: type limit, track by_src, count 1 , seconds 60; classtype:policy-violation; sid:1100021; rev:1;)
```
Traffic Shaping

- Some ISPs use Traffic Shaping to detect and limit peer-to-peer traffic.
- Most common methods identify and rate limit based on protocol.
- Seriously limits speed of BitTorrent transfers.

Traffic Shaping

It is estimated that more than 60% 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 peer-to-peer traffic on their networks. The most often employed technique is traffic shaping.

While a number of different methods of traffic shaping are available the most common method is via protocol detection. In fact most traffic shaping utilizes detection techniques similar to what have been discussed in this presentation.

When traffic shaping is employed it has the effect of drastically limiting the bandwidth available to peer-to-peer transfers.
BitTorrent Encryption

- Message Stream Encryption (MSE)/Protocol Encryption (PE) - BitTorrent protocol encryption specification.
- Employs obfuscation and randomized packet sizes.
- Defeats most types of traffic shaping.
- Encryption masks the contents of all aspects of the BitTorrent protocol.
- Still possible to detect tracker website requests and metafile downloads.

In order to counteract traffic shaping, the BitTorrent developers created a traffic obfuscation scheme called Message Stream Encryption (MSE)/Protocol Encryption (PE) which involves a Diffie-Helman key exchange and encryption of the header and, optionally, the body with the RC4 encryption protocol as well as randomized packet sizes.

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

There are pitfalls to enabling encryption. Besides the increased CPU overhead required to support the encryption, there is most likely a performance penalty on the network side. The reason for this is, if your client is configured to require encryption, the number of peers available to download content from is limited to those that permit encryption. In most clients encryption is not enabled by default.

The bright side is that the majority of BitTorrent users are not tech-savvy enough to be aware of the encryption capability of the clients and will not enable it.
Summary

- There are a number of different approaches that can be used to detect BitTorrent usage.
- BitTorrent encryption will defeat detection of the BitTorrent Handshake and DHT.

In summary, there are several points in BitTorrent file transfers where it is possible to identify BitTorrent transfers. The ones detailed in this presentation are:

- Monitoring the web access to the Torrent tracker websites.
- When a BitTorrent metafile is downloaded detecting the “.torrent” extension.
- Via the bencoded announce parameter in the BitTorrent metafile.
- The “BitTorrent protocol” string in the BitTorrent handshake.
- Trackerless torrent DHT ping messages.

Unfortunately BitTorrent encryption, if enabled, will break the detection of the BitTorrent handshake and the DHT ping.

The Appendix to this presentation provides all of the Snort rules which were created as a result of this research. Please feel free to implement them in your network.
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,from_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:"\x64 38 3a\announce"; flow:established; classtype:policy-violation; sid:1100011; rev:1;)

# detects BitTorrent handshake
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"P2P BitTorrent handshake";
flow:to_server,established; content:"BitTorrent protocol"; 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 bitenova"; content:"GET";
content:"bitenova"; threshold: type limit, track by_src, count 1 , seconds 60; sid:1100025; 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:1100026; 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:1100027; 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";
Detecting Bit Torrents Using Snort

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;)
