7-2012

Covert Channel in the BitTorrent Tracker Protocol

Joseph Desimone
Rochester Institute of Technology

Daryl Johnson
Rochester Institute of Technology

Bo Yuan
Rochester Institute of Technology

Peter Lutz
Rochester Institute of Technology

Recommended Citation

Desimone J., Johnson D., Yuan B., and Lutz P. Covert Channel in the BitTorrent Tracker Protocol. In SAM'12 - The 2012 International Conference on Security and Management (Las Vegas, NV, USA, July 2012).

Covert Channel in the BitTorrent Tracker Protocol

Joseph Desimone, Daryl Johnson, Bo Yuan, Peter Lutz
B. Thomas Golisano College of Computing & Information Sciences
Rochester Institute of Technology, Rochester NY
{jwd1063, daryl.johnson, bo.yuan, peter.lutz}@rit.edu

***Abstract*— Covert channels have the unique quality of masking evidence that a communication has ever occurred between two parties. For spies and terrorist cells, this quality can be the difference between life and death. However, even the detection of communications in a botnet could be troublesome for its creators. To evade detection and prevent insights into the size and members of a botnet, covert channels can be used. A botnet should rely on covert channels built on ubiquitous protocols to blend in with legitimate traffic. In this paper, we propose a covert channel built on the BitTorrent peer-to-peer protocol. In a simple application, this covert channel can be used to discretely and covertly send messages between two parties. However, this covert channel can also be used to stealthily distribute commands or the location of a command and control server for use in a botnet.**

Keywords: Computer Security; Covert Channels; BitTorrent; Botnets; Information Hiding

I. INTRODUCTION

Cryptography is useful in providing message confidentiality, or preventing a third party from uncovering the content of a message. However, cryptography is not designed to hide evidence that the communication has occurred. For some applications, even the existence of a communication between two parties could have disastrous consequences. These applications must rely on covert channels to prevent a third party from uncovering evidence that a communication has occurred. The term "covert channels" was first coined by Lampson nearly 30 years ago [1]. These original covert channels operated on a single machine to send information from a high security level process to that of a low security level process. Today, the majority of computing devices are attached to a network. Network protocols can be used to create covert channels were messages can be sent to remote machines in a stealthy manner [2]. In most cases, covert channels make use of unintended characteristics of network protocols that can be used to store information. The number of network protocols and their complexities allows for the creation of an almost unlimited number of covert channels.

Botnets are one possible application for covert channels. Botnets are groups of compromised machines that attackers control remotely for a variety of mostly malicious purposes. This includes conducting distributed denial of service attacks, sending spam messages, stealing account information, and conducting identity theft. Traditional botnets were controlled from an IRC server where the attackers could send commands to nodes of the botnet. However, this method is rarely used today as it presents a single point of failure for the botnet and they are more easily dismantled. As a result, malware writers have tried to devise better ways to control their army of compromised machines. Command and control over HTTP or HTTPS is common nowadays due to the prevalence of these protocols on the Internet. This makes identifying botnet traffic on the network more difficult. However, locating the command and control server is still a common failure point for botnets. For example, if they have static domain addresses programmed into the malware these can be null-routed and the botnet will go offline. This has led some malware writers to rely on DNS generation algorithms where DNS names are generated in a pseudo-random fashion. While these are more difficult to take down, it is not impossible. Another even more resilient method for botnet command and control relies on peer-to-peer network protocols [3]. This design completely decentralizes the botnet and makes it very difficult for defenders to track and dismantle. Malware authors will continue to explore resilient and stealthy methods to control botnets. Utilizing covert channels are another method that botnets use to elude detection from security researchers.

BitTorrent is one example of a peer-to-peer protocol. In recent years it has exploded in popularity and is responsible for the majority of the Internet traffic in most regions of the world [4]. The high volume and common use of the BitTorrent protocol would make an ideal target for malware writers that wish to blend in with legitimate traffic. Also, most BitTorrent networks are open and require no authentication. Therefore, a covert channel in the BitTorrent protocol would be an ideal candidate for covert channels, especially in the case of botnets.

II. LITERATURE REVIEW

Much research has been done developing new types of covert channels [2]. Some of these covert channels rely on the TCP protocol to hide information. For example, Rowland proposed hiding messages in the Initial Sequence Number (ISN) field of a TCP SYN packet [5]. This sequence number is used to synchronize TCP packets in a communication and is normally randomly generated for the first packet of a TCP connection. However, a covert message can be inserted into the ISN field instead before being sent to the receiver. Other covert channels utilize the DNS protocol to send hidden information. In one such channel, information is sent over DNS lookup requests to a fake DNS server [2]. The message is encoded in the hostname field of the lookup request.

Research has also been done on categorizing types of covert channels and evaluating them based on common

criteria. These criteria include the type, throughput, robustness, and probability of detection [6]. The most common types of covert channels are storage, timing, and behavioral based. The covert channel proposed in this paper can be considered a storage channel. Throughput measures the amount of information that can be sent over a given time interval. Covert channels can range from very low throughput rates of less than 10 bits per hour to high rates of megabytes per second. The robustness measures how resilient the covert channel as it proceeds through networking devices or other layers in the networking stack. Finally, detection measures how susceptible the covert channel is to detection from active listeners along the data path. Usually these criteria conflict with each other and a designer must select the most appropriate qualities for a given application. For example, generally as the throughput increases a covert channel's probability of being detected also increases.

The use of covert channels in botnet networks is not new. Johnson, Bo, and Lutz stated that malware authors might begin utilizing covert channels as a means to evade detection [7]. Also, Butler et al. proposed using a covert channel in DNS as a method for botnet command and control [8]. However, no published paper has documented the use of the BitTorrent network protocol directly. On the other hand, Li et al. proposed using torrent files to store covert messages [9]. Torrent files contain all of the necessary information needed to download a certain file, or collection of files, using the BitTorrent protocol.

III. COVERT CHANNEL IN BITTORRENT

A. *Background*

BitTorrent is a peer-to-peer protocol used for file sharing. Each user downloading a file also simultaneous uploads pieces that they have already received to other users. BitTorrent trackers are used so peers can locate other users that are downloading the same file. Users that are actively downloading files are known as leechers, while users that have completed downloading a file but remain uploading to other users are known as seeders. The BitTorrent tracker protocol operates over HTTP. Each torrent uses a unique SHA1 hash to identify the files or group of files that can be downloaded. After a user downloads and opens a torrent file, their BitTorrent Client will perform a GET request to the tracker. This GET request contains the info hash (unique SHA1) from the torrent file, the peer id of the client, an IP and port number of the client, an event message, a numwant field, among others [10]. At the start of the download the event message is "started." The peer id field is 20 bytes of randomly generated characters unique to each client. The numwant field is the number of peers the client wishes to receive. Figure 1 shows an example announce request to torrent.ubuntu.com. As you can see, the info_hash and peer_id fields are URL encoded.

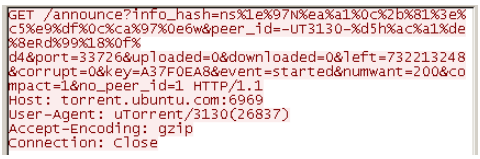

Figure 1. Example announce request

The tracker will respond to the client's request in a standard HTTP format. Figure 2 shows the packet header of the tracker response. Typically, trackers will return a maximum of 50 peers per request. Trackers will respond with a list of peers that are currently seeding or leeching the requested file. If more peers are present than the amount requested, peers are selected by the tracker randomly. This response can be in two formats. Some trackers allow the client to choose the response type, while others force a specific response. The first response format is known as a dictionary response. In a dictionary response, the server sends a list of IP, port, and peer ids of clients currently seeding or leeching the file. This response is structured in the Bencode format and contains the IP and port numbers in decimal notation. Figure 2 shows an example dictionary response from the BackTrack Linux tracker. The second response type is known as a binary or compact response. In a binary response, the server will respond with a list of IP and port numbers corresponding with other peers in network (big endian) notation. This response is typically the default as it requires less bandwidth and is also encoded with Bencode. Figure 3 shows a binary response from the BackTrack Linux tracker. As you can see from the figure, the response is not human readable. Peers are listed in the response with 4 bytes for their IP address and 2 bytes for their port number. No delimiter separates each IP/port combination in the list. The binary response omits the peer ID field entirely.

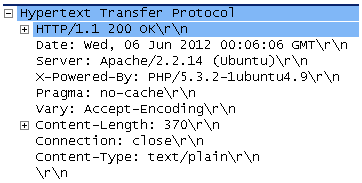

Figure 2. Tracker response header

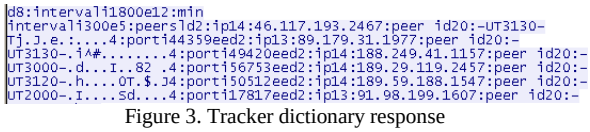

Figure 3. Tracker dictionary response

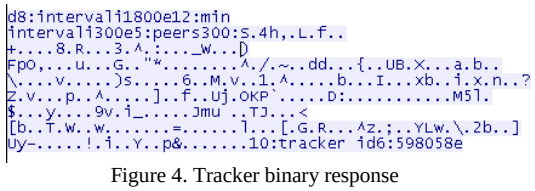

Figure 4. Tracker binary response

B. *Proposed Covert Channel*

To covertly send messages using BitTorrent trackers, a client could hide information in the peer id field during an announce request. To receive the message, one could contact the same tracker with the same info hash used as the sender and perform another announce request. The target info hash and tracker must be established prior to sending/receiving the message. The receiver's request would need to specify a dictionary type response. The server will respond with peers downloading the file which will include the peer ID of the sender containing the covert message. Figure 2 shows an example topology for this situation. The sender connects to the tracker's web interface and sends an announce request with a peer id containing a covert message. This message is then stored in the tracker's database. To retrieve this message, the receiver performs an announce request to the tracker in the same fashion as legitimate P2P clients. The server replies to the receiver with a list of clients active in the specified torrent. This reply will contain the covert message. In this covert channel, 20 bytes of information can be sent at a time. However, for a more legitimate looking covert channel, this could be reduced to 12 bytes. Most BitTorrent clients reserve a portion of the peer id field for an identifier of the client name and version number. For example, the uTorrent client begins its peer id with the string "-UT3130-" where 3130 corresponds with version 3.1.3 [11]. The remaining 12 bytes are randomly chosen characters.

Unfortunately, many trackers do not support the dictionary response format and therefore the peer id field will not be visible to the receiver. An alternative covert channel could utilize the IP field to send messages. The BitTorrent protocol allows clients to specify IPs other than that which the tracker sees in the connection. This feature allows clients to connect to the tracker through proxy servers or from the same side of a NAT device. An IPv4 address is 4 bytes which allows 4 bytes of information to be stored in each announce request. For messages longer than 4 bytes, the message can be split across multiple requests to the tracker. For each request, the peer id field must be different. Otherwise, previous bytes will be overwritten in the tracker's database. The port field could be used as a sequence number for each message so the receiver can properly reorder the message chunks upon receipt. In order for the message receiver to differentiate between valid peer IP addresses or pieces of a covert message an XOR scheme was used to encode messages. The receiver could simply reverse the encoding mechanism and verify if the resulting message was composed purely of ASCII characters. However, this requires the original message only contains ASCII characters.

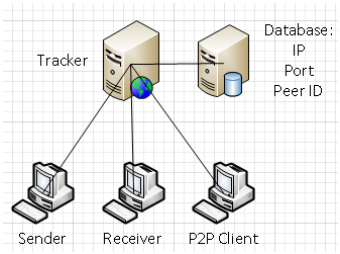

Figure 5. BitTorrent tracker topology

In an implementation of the covert channel using the IP field, we achieved a throughput rate of 20 bytes/second. A higher rate was obtained with the peer id field due to it allowing 5 times as many bytes than the IP field. Also, the peer id field covert channel has a higher degree of covertness. Hiding encrypted messages in the peer ID field would be impossible to distinguish from normal peer IDs due to their random nature. However, hiding messages in the IP field could be detected as these would resolve to IPs that are not actively participating in the download. Fortunately, most torrents will contain peers that cannot be contacted due to firewalls or peers that have closed their download client. We collected data across several popular ubuntu torrents to confirm this theory. Out of 1654 peers reported by the tracker, only 1517 could be contacted (91.7%). Additionally, some trackers are known to mix random IPs into their tracker responses to provide plausible deniability in file sharing lawsuits [12].

IV. DISCUSSION

We created proof of concept code to test the validity of the proposed covert channels. In both cases, we were successful in sending messages over the proposed covert channels. The backtrack-linux.org tracker supports the dictionary response and was used to send messages over the peer id field. To send messages over the IP and port field, the tracker from etree.org was used. An interesting issue occurred if more than 50 peers were downloading the target file. It could not be guaranteed that the receiver would see the sender's message with a single request. As a result, multiple requests were made to the server until the message was received. The selection of which torrent to rendezvous at is important for this reason. If the torrent has too many seeders and leechers then the message receiver is required to perform many requests to locate the message. However, if the torrent has too few seeders and leechers then plausible deniability is decreased. For most applications of this covert channel, I would expect a low number of total peers (<30) to be ideal. Also, torrents of copyrighted material would be less than ideal due to the higher possibility of monitoring (From the MPAA or other organizations).

The tracker will provide a min update interval to the client as part of an announce request. This is the minimum amount of time a client should wait before performing another announce request. Typically, trackers will remove clients that have not announced for twice the min update interval. As a result, the messages need to be reposted within

this time frame to ensure they remain on the tracker. Typical min update intervals are 1 hour.

Both of these covert channels could be used in botnets. The BitTorrent trackers could be used as a rendezvous point for botnet clients and their controller. As an alternative to a domain name used to locate the IP of a command and control server, the IP address could send in the covert channel. This method would be more resilient to takedown than a domain name. One possible weakness that could be exploited to enumerate all nodes of the botnet would be to send the IP address of a monitoring server to the target tracker. This would be possible to anyone who knows the rendezvous locations and algorithm. To prevent insights into the size of the botnet and to prevent a possible hijack, public key cryptography could be used to sign the IP address. Unfortunately, utilizing 1024 bit RSA key would require almost as many bits for message signing and would be much too large for the covert channel. Elliptical curve cryptography (ECC) has the advantage of having an equivalent level of security with much smaller key sizes. For example, ECC with a 160 bit key offers the same level of security as RSA with 1024 bits [13]. Starnberger, Kruegel, and Kirda proposed using ECC in their botnet protocol with a 112 bit key [13]. This allows for 40 bit messages to be securely sent if a maximum cipher text length of 20 bytes is desired. Thus, a botnet controller could encode the command and control server using a single message in the peer id covert channel.

V. Future Work

The BitTorrent specification should be analyzed for other potential covert channels. Also, mitigation or detection procedures should be developed for the proposed covert channels in this paper. For example, trackers should disallow the dictionary model response. Also, trackers should not allow peers to announce from arbitrary IPs. The info hash field should also be investigated as the potential for a covert channel. Many trackers will begin tracking any info hash that is announced to it. Therefore, arbitrary information could be inserted in this field to store up to 20 bytes of a message.

VI. Conclusion

BitTorrent is one of the most commonly used protocols on the internet. Covert channels exist in the BitTorrent protocol that can be used to send messages in a stealthy and resilient manner. These covert channels could be used by a botnet to distribute commands or send the location of a command and control server.

References

[1] B. W. Lampson, "A note on the confinement problem," *Communications of the A.C.M.*,vol. 16, no. 10, pp. 613–615, Oct. 1973.

[2] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44-57, Oct. 2007.

[3] P. Wang, S. Sparks, and C. Zou, "An advanced hybrid peer-to-peer botnet," in *First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.

[4] H. Schulze and K. Mochalski, "Internet Study 2008/2009," 2009. [Online]. Available: http://www.ipoque.com/resources/internet-studies/

[5] C. H. Rowland, "Covert Channels in the TCP/IP Protocol Suite," Technical Report, *First Monday: Peer Reviewed Journal on the Internet*, Jul. 1997.

[6] D. Johnson, B. Yuan, P. Lutz, and E. Brown, "Covert channels in the HTTP network protocol: Channel characterization and detecting man-in-the-middle attacks," 2010. [Online]. Available: https://ritdml.rit.edu/ha ndle/1850/14797.

[7] D. Johnson, B. Yuan, and P. Lutz, "Behavior-based covert channel in cyberspace," 2009. [Online]. Available: https://ritdml.rit.edu/handle/18 50/14795.

[8] P. Butler, K. Xu, and D. Yao, "Quantitatively analyzing stealthy communication channels," in *Applied Cryptography and Network Security*, vol. 6715, J. Lopez and G. Tsudik, Eds. Springer Berlin / Heidelberg, 2011, pp. 238–254.

[9] Z. Li, X. Sun, B. Wang, and X. Wang, "A steganography scheme in P2P network," in *IIHMSP '08 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Harbin, China, 2008, pp. 20 –24.

[10] B. Cohen, "The BitTorrent Protocol Specification," 2008. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html.

[11] "Bittorrent Protocol Specification v1.0," 2011. [Online]. Available: http://wiki.theory.org/BitTorrentSpecification.

[12] "Perfect Deniability," 2007. [Online]. Available: http://opentracker.blog .h3q.com/2007/02/12/perfect-deniability/.

[13] G. Starnberger, C. Kruegel, and E. Kirda, "Overbot: a botnet protocol based on Kademlia," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks*, New York, NY, USA, 2008, pp. 13:1–13:9.