An Algorithm for Anomaly-based Botnet Detection

James R. Binkley
Computer Science Dept.
Portland State University
Portland, OR, USA
jrb@cs.pdx.edu

Suresh Singh
Computer Science Dept.
Portland State University
Portland, OR, USA
singh@cs.pdx.edu

Abstract

We present an anomaly-based algorithm for detecting IRC-based botnet meshes. The algorithm combines an IRC mesh detection component with a TCP scan detection heuristic called the *TCP work weight*. The IRC component produces two tuples, one for determining the IRC mesh based on IP channel names, and a sub-tuple which collects statistics (including the TCP work weight) on individual IRC hosts in channels. We sort the channels by the number of scanners producing a sorted list of potential botnets. This algorithm has been deployed in PSU's DMZ for over a year and has proven effective in reducing the number of botnet clients.

1 Introduction

Botnets [6] [5] are a current scourge of the Internet. Botnets may result in high rates of TCP syn scanning (for example, see [4]), voluminous spam, or distributed DOS attacks (see [2]).

At Portland State University, in the last few years we began to realize that many of our security incidents had a common thread which proved to be botnet related. As a result we developed an anomaly-based algorithm for detection of botnet client meshes and made it a subcomponent of our open-source ourmon [3] [9] network management and anomaly detection system. The system is currently deployed in our DMZ where we see peak traffic periods of 60k pps. In the last year, this system has proven beneficial in reducing the number of botnet clients on campus.

Our anomaly-based system combines an IRC [7] parsing component with a syn-scanner detection system aimed at individual IP hosts. The IRC parsing system collects information on TCP packets and determines an IRC channel, which we define as a set of IP hosts. We then correlate the IP host information over a large set of data sampled during the current day which tells us if an individual host in the IP channel was a scanner. We then sort the IRC channels by scanning count, with the top suspect channels labeled as possible *evil channels*. This algorithm is not signature-based in any way. It does not rely on ports or known botnet command strings. As a result, we are immune to zero-day problems. Our algorithm does assume that IRC is cleartext and that attacks are being made with the botnet mesh.

2 IRC Botnet Detection Algorithm

Our architecture relies on the observation that IRC hosts are grouped into channels by a channel name (for example, "F7", or "ubuntu" might be channel names), and that an ēvil channel is an IRC channel with a majority of hosts performing TCP SYN scanning.

The front-end data collector gathers three kinds of list tuples that are useful for benign IRC, botnet detection, and scanner detection. The tuples consist of two kinds of IRC tuples and the TCP syn tuple. The probe gathers these tuples over its thirty second sampling period. All tuples are then sent to the back-end for further processing. In the probe, the entire campus TCP syn tuple set is filtered into a smaller subset which informally consists of hosts observed sending anomalous amounts of TCP syns. This syn tuple subset is called the "worm set" and is typically orders of magnitude smaller than the entire set of IP sources found in the campus TCP syn set.

The TCP syn scanner list tuple has the following simplified form:

```
(IP source address, SYNS, SYNACKS,
FINSSENT, FINSBACK, RESETS,
PKTSSENT, PKTSBACK)
```

The logical key in this tuple is an IP source address. SYNS, FINS (all kinds), and RESETS are counts of TCP control packets. SYNS are counts of SYN packets sent from the IP source, and SYNACKS are a subset of only

those SYNS sent with the ACK flag set. FINS sent both ways are counted. RESETS are counted when sent back to the IP source. The PKTSSENT counts the total packets sent by the IP source. PKTSBACK counts the total pkts returned to the IP source. Other fields exist but are not relevant to this paper. This information is useful for determining what kind of scanning is occurring and often gives a rough network-based indication of the kind of exploit in use.

We define a metric which we call the *TCP work weight*. The *work weight* is easy to compute and is computed by the probe per IP source as follows:

$$w = (S_s + F_s + R_r)/T_{sr}$$

It is expressed as a percent. The rough idea is that we take the count of TCP control packets (SYN's plus SYNACKs sent, FIN's sent and RESETS) and divide that count by the total number of TCP packets (T_{sr}). Obviously 100% here is a bad sign and implies a true anomaly of some sort. Such a value is typically associated with a scanner or worm although some forms of P2P (and email servers) may have high work weights for shorter periods of time. The IRC module in the probe uses the TCP list as an underlying "tool", and extracts the TCP work weight from it for any IRC host.

We should point out that we have over two years worth of experience with the work weight at this point. We have learned that high work weights with hosts are caused by three possible causes including 1. scanners (typically syn scanners), 2. clients lacking a server for some reason or 3. P2P hosts (usually Gnutella is the application) IP peers. In general scanners are the most common reason for a high work weight. We also know that that the work weight clusters into either high values or low values (say 0..30%). Attackers fall into the higher range. P2P clients on average fall into the lower range.

Typically the average over many samples is of interest. However in the case of IRC we decided to simply take the maximum work weight seen over all the thirty second samples for a day. This is because an otherwise normal host may be ordered remotely to do scanning for a short period of the day. One host by itself in an IRC channel with a high work weight may not be anomalous. However if a channel has ten hosts out of twelve with high work weights suspicion is justified. As a result, work weights associated with IRC channels in our summarization reports are maximum weights seen across all the samples in a daily report.

There are two IRC lists, called the *channel list* and the *node list*. The channel list has the following tuple structure:

```
(CHANNAME, HITS, JOINS, PRIVMSGS,
NOIPS, IP_LIST}
```

The channel name is the case-insensitive IRC channel name extracted from JOIN and PRIVMSG IRC messages by the IRC scanner. The probe's scanner is handcrafted C code that looks at the first 256 bytes of the L7 payload for TCP messages only and extracts IRC tokens for the four kinds of messages of interest. HITS is the total count of JOINS and PRIVMSGS, JOINS and PRIVMSGS are counts of that particular kind of message. NOIPS is the number of IP addresses in the IP_LIST, which follows the tuple. Thus a channel tuple gives a key (the channel name) with a few message count statistics and a list of IRC hosts in the channel expressed as IP addresses.

The *node list* gives per IP statistics for any IP address in any IRC channel. Informally a channel may be viewed as a directory, and a host may be viewed as a directory entry (although a host may actually be in more than one channel). The node list has the following tuple structure (not all counters shown):

```
(IPSRC, TOTALMSG, JOINS, PINGS, PONGS,
PRIVMSGS, CHANNELS, SERVERHITS, WW}
```

The key per tuple is an IP source address. Various message statistics are given including JOIN, PING, PONG, and PRIVMSG counts. The number of observed per host channels is supplied. SERVERHITS indicates the number of messages sent to/from a host. Thus this counter indicates whether a host is acting as an IRC server. The WW (work weight) as mentioned previously is derived from the TCP syn module.

One additional IRC statistic is gathered by the front-end which consists of total counts of the four kinds of IRC messages seen by the probe during the sample period. (This tuple is displayed by the back-end as an RRDTOOL-based graph - due to space limitations we cannot show such a graph in the paper). It shows that IRC is basically a slow phenomenon with only a few messages per second, even though our campus may have 5000 IP hosts active during a day. As a result our IRC evil channel analysis is based on a slower time scale, hours and days.

The IRC tuples are passed to the backend for report generation. The backend program produces an hourly text report (updated on the hour) which is called *ircreport_today.txt*. This file is available on the web for analysis. Data in this report is broken up into three major sections including global counts, channel statistics, and per host statistics. Channel statistics and per host statistical sections are further broken up into various sub-reports where data is typically sorted by some key statistic.

We can distinguish the following IRC report subsections:

1. evil channels - channels with too many hosts with a high work weight

2. channels sorted by maximum messages.

3. channels with host statistics - each channel shows the host IP in the channel with host stats.

4. servers sorted by max messages - hosts that are IRC servers are sorted by max messages.

5. hosts with join messages but no privmsgs - JOINs only but no data payloads.

6. hosts with any signs of worminess - hosts with high work weights.

For purposes of illustration in table 1 we look at one benign example which comes from the per channel host statistics section. Counts given for our example were taken from twelve hours of data (since midnight) and are typical for a small IRC chat group. [1]

In our example 1, a channel named "ubuntu" has four hosts in it. Three are local and the server (S) is remote. Total message counts (of the 4 kinds parsed) and JOIN, PING, PONG, and PRIVMSG counts are given. Maxchans is the number of channels seen during the period for that host, and maxworm is the maximum work weight seen. We do not believe this channel based on the above data is "evil".

Now let us see how this data may be correlated to plainly point out anomalous IRC-based botnet behavior.

3 Botnet Examples

Let us look at three examples which illustrate the operation of our algorithm. Table 2 gives us three items from our evil channel report. The purpose of this selection is to illustrate, on the one hand, the effectiveness of our algorithm in detecting evil channels while on the other hand showing some borderline cases that require additional analysis (e.g., examining port reports) First, we present a botnet client mesh. By definition, the server is off-campus and a few hosts have been captured on-campus to become part of the botnet. We look at two subsections of the hourly IRC report to find our evil channel which is named "F7". We look at our evil channel sort, and discover that F7 shown in table 2 is named as a channel in that list and occupies a high rank in the list.

Channel F7 is high in the evil channel list simply because it has 4 out of 6 hosts with high work weights. The "evil" flag at the end of the column is set to E if a potential evil channel has more than 1 anomalous host. Next we look at the report sub-section which breaks host statistics out for the channel F7.

[1] All IP addresses have been changed and are represented as symbolic addresses. In addition the reader should note that our current output format is simply an ASCII report. However we represent it here in tabular format.

In table 3 we see the part of the report that shows hosts in a channel. In channel F7, we have one remote server and five infected local hosts. Four of those hosts have very high maximum work weights. We know from experience with the work weight (and also by looking at logs from both Ourmon and other systems) that the hosts are performing SYN scanning. Ourmon logs for the syn tuple will typically show that the hosts in question have been performing scanning aimed at Microsoft exploits on port 445 (typically lsass-based exploits, for example, see [4]).

We have used ngrep in the past to prove beyond a shadow of a doubt that examples like our F7 botnet client are indeed malign. At this point in time, we no longer feel the need to use a tool like ngrep to prove that ourmon has detected an evil mesh. However the reader might desire to see such proof and in addition ngrep can still be very useful as an aid in host forensics. For example, one may be able to gather valuable clues about the exploit used. An ngrep sent from a local client to the server in question (net2.1) showed messages like the following:

```
# ngrep -q host net2.1
T net1.1:1053 -> net2.1:30591 [AP]^
  PRIVMSG #F7 :[Lsass]: Fuxed IP: net1.2
```

Here we see a report from a bot client back to the server that host net1.2 has been exploited. The exploit used is also mentioned.

The other two examples in table 2 are not evil channels. s3reporter is a IRC game (which is why all participating IP addresses are marked as servers) for which we sometimes get a high work weight. However, since the high work weight is associated with a remote host (see table 3), we do not consider it further. The third example has a local IP host with a high work weight which implies evil channel. However, this is a borderline case (with only one client) where the high work weight may be because of software glitches (e.g., meetingmaker loss of server causes this type of bot-like behavior) or a p2p outage of some sort. These types of channels require additional analysis where we need to examine port reports in more detail. Some of the specifics that we look for include, for example,

- L3/L4 dst counts of unique L3 and L4 destinations. This can suggest whether a host is traversing IP destinations or ports or both.

- EWORM - a flag system to indicate if traffic exists that is 2-way or if network errors exist. E.g., E and R indicate ICMP errors or RESETS returned to the host. O indicates a lack of FINS. M indicates no non-control packets returned.

Table 1: Benign IRC Channel - Channel/Host Report

channel/ip	tmsg	tjoin	tping	tpong	tprivmsg	maxchans	maxworm	server
ubuntu/net1.host1	11598	1282	1912	1910	6494	4	43	H
ubuntu/net1.host2	7265	938	619	622	5086	3	0	H
ubuntu/net1.host3	17218	1926	4123	4100	7069	5	37	H
ubuntu/net2.host1	28152	3222	3913	3904	17113	8	0	S

Table 2: Malign and normal IRC Client Botnet - Evil Channel Report

channel	msgs	joins	privmsgs	ipcount	wormyhosts	evil
F7	118	19	99	6	4	E
s3reporter	2259	25	2234	3	1	E
thespicebox	23	8	15	2	1	E

Table 3: Malign and Benign Channels - Channel/Host Report

channel/ip	tmsg	tjoin	tping	tpong	tprivmsg	maxchans	maxworm	server
F7/net1.1	1205	24	377	376	428	2	42	H
F7/net1.2	113	6	39	43	25	1	96	H
F7/net1.3	144	2	60	61	21	1	94	H
F7/net1.4	46	3	12	14	17	1	90	H
F7/net1.5	701	2	343	345	11	1	90	H
F7/net2.1	1300	19	587	593	101	1	16	S
s3reporter/net1.1	3949	25	844	846	2234	1	5	S
s3reporter/net2.1	6899	36	794	794	5275	2	90	S
s3reporter/net3.1	4525	21	704	702	3098	2	19	S
thespicebox/net1.1	3106	101	433	661	1911	2	83	H
thespicebox/net2.1	10943	373	1828	2037	6705	4	43	S

- Sampled destination ports sometimes help to characterize the nature of the attack. For example, given that in our network we know that Microsoft file share ports are blocked, scanning of port 139 and 445 is deeply suspicious. In addition nearly all of our bot clients caught in the last year were scanning those ports.

Thus, in the end, while out algorithm clearly shows the presence of evail botnets, for many boderline cases, we need to resort to additional analysis.

4 Related Work

In general the academic literature on botnet detection is sparse. Furthermore we are not aware of any other anomaly-based system for detection of botnets. Known techniques include honeynets and IDS systems with signature detection. Honeynets [6] or darknets might be distributed [1] or local and can certainly prove beneficial in terms of providing information about botnet technology. However they may not be easily deployed in a commercial environment and do not necessarily help with the question of whether host X has worm Y. Knowledge of useful signatures and behavior of existing botnet systems is another venue for detection. The paper [2] presents a good introduction to botnets and analyzes botnet architecture. An open-source system like snort [8] can be used for detection of known botnets.

The problem with signatures is of course one may lack the required signature for a bot known elsewhere, or a bot may be new to the world, locally unknown, or changed, thus defeating previously known signatures. Anomaly detection on the other hand may detect such a system. Problems with anomaly detection can include detection of an IRC network that may be a botnet but has not been used yet for attacks, hence there are no anomalies. As our technology depends on hackers actually launching attacks, there is no guarantee that we can detect every infected system. One can also argue that anomaly detection is ”too late”. It is certainly better to detect an initial attack with a signature when it first occurs and get an exploited system fixed before it is used for spam or denial of service attacks. We believe signatures and anomaly detection are often complimentary and should not be viewed as somehow competitive. All of these techniques (honeypot, IDS, and anomaly detection) can be useful and provide slightly different set of information.

5 Conclusion

In this paper we have presented our current system for bot anomaly detection. We discussed how we combined TCP-based anomaly detection with IRC tokenization and IRC message statistics to create a system that can clearly detect client botnets and how also gross statistical measures can easily reveal bot servers. This system is currently deployed in our network and works well.

The white paper [5] calls for systems to detect botnets via more robust detection means. We believe our current anomaly-based detection system is an advance in the art, but it could be easily defeated by simply using a trivial cipher to encode the IRC commands. As a result we would lose information about mesh connectivity. On the other hand we believe that detection and correlation of attacking meshes of hosts is a valuable contribution. Our current system should be made more general in terms of attacks and include email and DOS attack indicators. For future work, we intend to pursue an anamoly-based algorithm that will work only with layer 3 and layer 4 statistics.

References

[1] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, The Internet Motion Sensor: A Distributed Blackhole Monitoring System. *In Proceedings of the Network and Distributed Security Symposium*, San Diego, CA, January 2005.

[2] P. Barford, V. Yegneswaran, An Inside Look at Botnets, *Special Workshop on Malware Detection, Advances in Information Security*, Springer Verlag, 2006

[3] J. Binkley, B. Massey, Ourmon and Network Monitoring Performance. *Proceedings of the Spring 2005 USENIX Conference, Freenix track*, Anaheim, April 2005.

[4] CERT Advisory CIAD-2004-10 Multiple Vulnerabilities in Microsoft Products `http://www.cert.org/advisories/ciad-2004-10.htm`, April 2004.

[5] E. Cooke, F. Jahanian, and D. McPherson, The zombie roundup: Understanding, detecting and disrupting botnets. *In Proceedings of Usenix Workshop on Stepts to Reducing Unwanted Traffic on the Internet (SRUTI '05)*, Cambridge, MA, July 2005.

[6] The Honeynet Project and Research Alliance. Know Your Enemy, Tracking Botnets. `http://honeynet.org/papers/bots`, March 2005.

[7] J. Oikarinen, D. Reed. Internet Relay Chat Protocol. IETF RFC 1459, May 1993.

[8] Snort IDS web page. `http://www.snort.org`, March 2006.

[9] Sourceforge Ourmon web page. `http://ourmon.sourceforge.net`, December 2005.

[10] Wikipedia web page. `http://en.wikipedia.org/wiki/Internet_Relay_Chat`, December 2005.