2009

Finding IRC-like Meshes Sans Layer 7 Payloads

Akshay Dua
Portland State University, dakshay@gmail.com

Jim Binkley
Portland State University

Suresh Singh
Portland State University

Finding IRC-like Meshes Sans Layer 7 Payloads

Akshay Dua
Computer Science Dept.
Portland State University
Portland, OR, USA
akshay@cs.pdx.edu

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 algorithm for detecting IRC-like chat networks that does not rely on Layer 7 payload information. The goal is to extract only those meshes from conventional flows where long-term periodic data is being exchanged between an external server and multiple internal clients. Flow data is passed through a series of filters that reduce the memory requirements needed for final candidate mesh sorting. Final outputs consist of two sorted lists including the fanout list, sorted by the number of client hosts in the mesh, and a secondary list called the *evil sort*. The latter consists of meshes with any host with a high TCP work weight[1] [3] indicating significant counts of scanning hosts. We are currently able to discover SSL-encoded IRC meshes as well as other chat-like meshes including MSN chat. Therefore we believe that the new algorithm will prove useful in detecting botnet meshes encrypted at Layer 7.

1 Introduction

Botnets [6] [5] remain a current scourge of the Internet, involved in computer crime, denial of service attacks, identity theft, and SPAM. According to a recent late 2006 Microsoft report [11], Microsoft's removal tool (MSRT) found 16 million instances of malware on 5.7 million hosts in a fifteen-month timeframe. Most of the malware consisted of IRC-based bots.

In a previous SRUTI paper [5], the authors called for new systems to detect botnets in more robust ways. Many techniques for command and control detection involve inside knowledge of attack commands or other string-based signatures that may simply be changed by changing the botnet's programming. In our 2006 SRUTI paper [3] we presented an algorithm for an anomaly-based system that combined knowledge of layer 7 IRC [7] protocol commands including PRIVMSG, JOIN, PING, and PONG that was aimed at IRC mesh extraction. IRC meshes were then sorted by the number of scanners detected via our TCP work weight statistic which detects scanners. The result gives us "evil meshes" which often represent an attacking IRC-based botnet.

Our current ourmon [4] [14] IRC system is based on a simple hard coded C-based lexer that simply looks for IRC commands. We can claim that it is not signature-based in the sense that it does not need to know application-level knowledge of any particular botnet family. Readers should understand that many common botnet coding families may use IRC as a delivery mechanism, but effectively function as an application on top of IRC itself. IRC is basically a communication delivery channel. For example, agobot might use the command "scan.start" delivered within an IRC PRIVMSG command. See [2] for more examples. Our IRC parsing system would absolutely not be able to find encrypted or merely obfuscated IRC meshes. An IRC system using SSL would be totally opaque to our IRC mesh extraction system. Worse even a trivial Caesar cipher encoding for the IRC protocol would cause it go blind.

As a result, we set out to develop a new extraction system that would have three goals. First it would not use any layer 7 information and would confine itself to layer 3 and layer 4. Thus it should be able to detect SSL-based IRC networks. Second even though we would use our ourmon system as the data collector, the system would be based on top of a fairly conventional flow extraction system, thus it might prove possible at a later date for the system to be implemented with conventional netflow technology. Last we would reconstruct our mesh extraction subsystem with the target of extracting IRC-like meshes defined according to a certain set of attributes. For example, topologically we define an IRC-like mesh to consist of one external server and more than one internal IP client. Such a mesh should also have long term and periodic packet exchanges between the participants

[1] This is measured as $\frac{\text{syns\_sent + fins\_sent + resets\_returned}}{\text{total TCP packets}}$.

and thus for example be unlike short term bursty web exchanges. We call our new algorithm the *small mesh detection* algorithm.

2 The Small Mesh Detection Algorithm

The goal of this algorithm is to extract IRC-like meshes. It should be understood that the algorithm starts with bi-directional flow collection and then proceeds to create candidate Layer 7 meshes out of the flows. In general, the algorithm seeks to throw out flows or meshes whenever it gets the chance simply because mesh organization is an exponential problem and thus early elimination of non-interesting data is a crucial component of the overall system. At a very high-level the algorithm has the following steps:

1. TCP bi-directional flows are collected
2. candidate flows must have a non-zero layer 7 payload and must be "small", where small is defined as less than 600 bytes
3. candidate flows must be external/internal in terms of a HOME enterprise. At a later sorting state, candidate meshes must consist of one external IP to more than one internal IP.
4. candidate meshes must exhibit long lived and active behavior over a long time period (defined as one hour at present).
5. After all data reduction filters are run, the final result is output as the *fanout list*.
6. The fanout list is also sorted according to the number of internal component hosts having a high TCP work weight. If a mesh has a non-zero number of internal hosts with a high work weight, it is also output in the *evil sort*.

The front-end ourmon probe acts as the flow collector for the small mesh system. It collects bi-directional flows with non-zero payloads and also makes sure that the flows in question are not from internal to internal "HOME" IP addresses. We call these filters *pass 1* filters and will discuss the relative statistical efficiency of our various filters in a later section. Note that the ourmon probe as outlined in our previous paper also collects per IP address TCP statistics that are output in a TCP syn tuple. This information gives us the TCP work weight. Data flows collected by the front-end consist of a list of tuples each of which have the following format:

```
(IP source address,  source port,
IP destination address, destination
port, IP protocol, packet count, total
 flow byte count)
```

Because the flows are bi-directional, they are collected as pairs with IP addresses swapped, etc. Candidate flows and TCP work weight information are passed to the back-end every thirty seconds.

The back-end script (written in perl) runs over one hour's worth of samples. It seeks to organize flow pairs into candidate meshes and represents meshes by a logical flow graph. During the construction of the flow graph, the back-end gathers statistics about the hosts and flows that allows it to eliminate meshes that are not exhibiting IRC-like behavior. This is done via three separate filter passes .

Pass two filters are run per ten minutes of data collection. Their goal is to significantly reduce the amount of data as well as to eliminate non-IRC like flows as rapidly as possible. Every host that gets filtered in this pass is added to an internal whitelist and subsequently ignored for the duration of the hourly run. It should be pointed out that we do not make any assumptions about well-known ports in terms of simplistic assumptions like IRC may be found on port 6667. This is crucial as a botnet controller might be found on any port. The pass two filter eliminates various meshes based on the following criteria:

1. internal and external web servers. A host is considered a server if it is talking to multiple clients with less than three ports. Here we use the BLINC [8] assertion - web servers may be identified by noting that during a given sampling period, the number of destination ports will be greater than the number of destination IPs.
2. multi-flow servers. If servers talk to each of their clients with many flows over the 10 minute period then we can safely eliminate them. The reason is that IRC servers usually talk to their clients using a single long lived flow rather than many simultaneous connections.
3. external clients. An external host is considered a client if the number of source ports is greater than the number of destination ports (during the sample period). This eliminates external hosts accessing internal servers.
4. internal servers - Any internal host talking from less than three ports to many external clients (greater than 10) is considered an internal server.

Pass three filters are applied to all hosts in the flow graph after an hour's worth of data is processed. Hosts eliminated at this level are not removed from the flow

graph but are ignored in any later analysis. This pass eliminates any internal hosts as possible candidates for single external servers for the simple reason that they are external. It also eliminates external hosts with only a single individual internal client.

Pass four filters represent the last filter step. At this level in general the algorithm is only concerned with meshes as a whole and eliminates meshes that do not seem to have IRC-like behavior. Meshes are eliminated if they do not have long-lived flows. IRC should survive this step because the mesh will refresh itself either with periodic PING and PONG commands at layer 7 or with client JOINS to recreate IRC connections.

After all the filter passes, we output a fanout list of candidate meshes that are either IRC or exhibit IRC-like behavior. The fanout list is sorted according to mesh size. Thus a mesh with more internal IP clients will appear higher in the list. In addition, the back-end performs the evil sort. This list is a subset of the meshes in the fanout list and may have no entries. The back-end code consults a daily TCP work weight database and looks up internal IP addresses found in fanout meshes for their maximum work weight value seen during the day. Evil meshes are then sorted by the number of clients having high work weights. A mesh with more than one scanning host may represent a botnet. We know from experience with our IRC mesh mechanism that a mesh with three or more members has a high probability of being a botnet and may represent a botnet scanning in parallel. It is important to understand that the "evil sort" part of this algorithm is basically not new and is simply a form of technology reuse from our previous IRC specific algorithm. What is new here is the mesh extraction part of the system that is not using any Layer 7 payload information.

3 Experimental Results

In this section we briefly present our experimental results. As we felt that exposure to the real-world is important, the algorithm is currently deployed in the PSU DMZ and is looking at real data. We will first review what sort of meshes our system seems to be finding. In addition we also subjected the new system to various tests primarily aimed at determining if it could both respond to a scanning IRC botmesh along the lines of the previous algorithm, and perhaps more important (given that ranking meshes by attackers is not a new idea for us) we also made sure that the new algorithm could detect IRC-like encrypted meshes. Last because this work is about data reduction, we will take a short look at a measurement study we made to both sanity check the system and find out which parts eliminated the most flows or meshes.

In terms of real-world data extraction, one very important test for our new system was to see if it would find the same external-internal IRC meshes currently found by the earlier Layer 7 system. This proved to be the case. In addition, there is the rather fundamental question of what else does the system find? During a normal school day the system finds about 20 small meshes in the morning and around 40 meshes in the afternoon. We can informally classify the meshes as falling into four camps: 1. VPNs, 2. MSN chat, 3. IRC., and 4. small but mysterious. For example out of a recent afternoon hourly run, we find roughly three quarters of the meshes are MSN chat and/or IRC, with MSN chat far outnumbering the IRC meshes (10 to 5). MSN chat runs at port 1873 and is characterized by a server run by Microsoft with a strange DNS name ending in phx.gbl. The protocol is also not IRC proper, but is similar. VPNs of various forms seem to actually dominate the statistics, not in terms of meshes, but in terms of IP hosts in the VPN mesh. The VPNs are encrypted connections to remote servers. We speculate that the connection is periodic and long-lived simply to keep NAT state fresh for broadband home systems. One example of this is gotomypc.com. Finally a number of relatively small mysterious meshes exist that are not always easy to explain. One example (recently unraveled) turned out to be a Oregon University System VOIP management system based at Oregon State University that curiously enough used TCP port 23 and was not somehow a telnet-based mesh.

We also performed a number of lab and on-line experiments to determine if our system behaved properly. For example, as the small mesh back-end program uses a per IP address database of TCP work weights created elsewhere by the ourmon system, we thus first selected an IRC mesh found in the fanout mesh by the algorithm, and tested what would happen if the local clients were found to have high work weights. High work weights were created by salting the database with false values. The result is shown in table 1. Although simplified. the output here is roughly what the new algorithm shows for any captured mesh. The external host (presumed to be a server) is given first followed by two internal clients. The L3D column gives the number of unique IP destinations for the host. The L4S and L4D columns give counts for TCP source and destination ports. The WW value gives the maximum work weight. The port list shows source ports for the external host. We also repeated this test by creating an IRC mesh with an external server, and then used nmap [12] to scan an internal darknet, thus emulating the behavior of a typical IRC-based botnet in its scanning phase. This form of test worked and results were similar to table 1. In addition, we created an SSL-based IRC mesh and made sure our algorithm would capture it. The resulting mesh may be seen in table 2.

Table 1: Small Test BOT Mesh

IP	L3D	L4S	L4D	WW	Port List
outside.166.3	2	1	2	0	6667
inside.10.206	1	1	1	90	
inside.11.30	11	7	17	90	

Table 2: Small SSL Mesh

IP	L3D	L4S	L4D	WW	Port List
outside.142.4	4	2	4	0	6697, 2523
inside.1.51	1	1	1	0	
inside.2.49	1	1	1	0	
inside.3.61	1	1	1	0	

3.1 Filter Efficiency

Finally given the multiple component nature of our algorithm which effectively decomposes into a set of smaller filters, we have attempted to determine the relative effectiveness of the various components.

In terms of the *front-end probe* the situation is relatively simple. In table 3 we present a statistical summarization over multiple samples. The first column gives the name of the filter stage and the second column shows the overall data reduction as a percent in flows from the total number of flows. Total flows are shown as a basis in the first row and of course make up 100% of the total flow count (on average we see 60k flows every day). The next two rows show that TCP flows outnumber UDP flows about two to one. This study focuses only on TCP and our real filters here then are the filters named *2-way TCP flows* and *2-way TCP/small flows*. In the first case we discard any TCP flow that is not bi-directional, which nets us a small data reduction of around 10% of the flows. In addition we discard flows that have no payload and only keep those with payloads less than 600 bytes total (hence small). This nets us a further reduction to around 10% of the total original flow count. The filter label *2-way small flows* indirectly shows that there are more small UDP flows than TCP flows. This may be due to DNS, but as of yet we have not explored UDP. For UDP it may be necessary to eliminate DNS servers with a whitelist or via as yet unknown attributes as elimination via whitelists could lead to false negatives.

For our *back-end* in table 4 we present a very rough estimate of data reduction based on passes and filters in passes. In general the estimates are based on total external and internal IP addresses eliminated. However in some cases flows are eliminated and in later stages of the algorithm meshes are eliminated. Given that one is looking at IP addresses, or flows, or meshes at various stages, it is hard to determine a proper strategy for comparison. For example, pass two which pre-computes all data in ten minute stages in order to reduce overall data size in general eliminates about 15% of the total flows. This number is not reflected in the table. In gross terms then it appears that the most effective component in terms of data reduction is the *no mesh* filter which simply means flows from one external source with a single internal partner. The pass 2 filter for detection of external clients is also useful at 15%.

Given that the front-end reduces the data by an order of magnitude and the back-end reduces data presented to it by 99%, we can see that the primary thrust of our algorithm and its sub-components is certainly data reduction. We believe that at this point we are successful at getting rid of everything that is not a long-lived TCP mesh. Thus we can ignore layer 7 payloads.

In terms of running time for the backend filters, on a P4 machine with 512MB of RAM, the total running time (in wallclock seconds) per one hour of data as measured by perl's Benchmark package is: Pass 1 filters - 74s, Pass 2 filters - 10.5s, Pass 3 filters - 1s, and Pass 4 (mesh) filters - 2.5s.

4 Related Work

In general the academic literature on botnet detection remains sparse. The most relevant work to our work presented here is ironically our previous work [3], which describes an anomaly-based system for detection of botnets. The algorithm parses Layer 7 payloads to extract IRC meshes and then uses the TCP work weights of the clients to determine if the mesh is a botnet. The problem is, simple encryption or encoding of the payloads can defeat the algorithm. We have thus modified it to work with Layer 3 and 4 information only. A number of valuable attribute ideas have been found in [8], but they have been reused here in a different problem space - IRC-like mesh detection - not P2P detection. Still, this reflects our own experience with attribute detection aimed at P2P mesh detection. Quite often the attributes found may be reused

Table 3: Probe Flow Reduction

filter	percent
total flows	100
TCP flows	66
UDP flows	29
2-way TCP flows	57
2-way small flows	32
2-way TCP/small flows	11

Table 4: Back-end Data Reduction

pass	filter	percent
pass 2	multi-flow servers	3.4
pass 2	external clients	15
pass 2	internal servers	.01
pass 3	no mesh	71
pass 3	internal IP only	5
pass 4	short-lived mesh	5.4

in a different problem space.

Other botnet detection techniques include machine learning, honeynets and signature based detection. Machine learning classification algorithms are used in [10] and [9] to identify botnet meshes. Although the algorithms use Layer 3 and 4 information only, their false positive and negative rates can vary with the training set used. This can be a real problem because finding accurate training data, especially one that includes malicious (botnet) traffic can be challenging. Further, since the classifiers work with individual flows, they fail to identify the mesh, i.e. the flows that belong to the same botnet. Honeynets [6] or darknets [1] are very effective in collecting information about botnets. However, they may not be easy to deploy internally for those with limited time for analysis or limited IP address space. It is also not always clear that information found in darknets is disseminated to those who need to know it. (Of course email about abuse may be ignored as well.) Signature based detection can be implemented using an IDS like snort [13], and can be effective in finding known bots like the ones discussed in [2]. The drawback, of course, is that modified versions of old bots, completely new bots, or encrypted command and control communication with bots can defeat this system. Layer 3 and 4 based anomaly detection on the other hand, can detect such a system. Drawbacks of anomaly based detection include the discovery of IRC-like networks that may be botnets but have not been used for attacks yet, thus there are no anomalies. Our technology to some extent depends on bot controllers actually launching attacks; of course, there is no guarantee that we can detect every infected system. On the other hand, an analyst can use our technology to learn the current set of small meshes and may thus note new meshes at a layer point that constitute passive botnets. Also, anomaly detection can sometimes be too late. It is certainly more useful to detect an initial attack with a signature than to wait for hosts to get infected and show signs infection. Many of these techniques can go hand in hand and need not be considered competitive. In fact they are often complimentary. Most discussed techniques like honeypots, signature detection, and anomaly detection can be useful in different ways for studying or detecting botnets.

5 Conclusion

In this paper we have presented a flow-based system for extraction of small meshes that may also be coupled with our previous TCP work-weight to suggest attacking meshes. This work may prove useful in the detection of stealth botnets. The result does not depend on Layer 7 data or attributes. It only uses Layer 3 IP addresses and Layer 4 TCP ports. The overall algorithm may be viewed as a set of filter passes that aim to reduce the overall flow data to a small set of meshes with one external system and more than one internal IP host. Only meshes with long-term periodic data are considered. This system is currently deployed in our network and is both stable and has been proven to detect the same IRC meshes as our previous Layer 7 based system. In addition it has been shown experimentally to detect SSL-based IRC meshes and in actual reality has detected some previously unknown external VPN systems.

The white paper [5] calls for systems to detect botnets via more robust detection means. We believe that our system is more robust than previous systems. In terms of

future work, we intend to investigate UDP small meshes. Also given the nature of the Ourmon system itself, which tends to extreme data aggregation in the probe, we would like to reexamine our various component filters to see if any of the current back-end filters might be pushed into the probe. This would hopefully improve the overall scalability of the system.

Acknowledgment

This work was supported by the NSF under award NeTS-NR: 0435328.

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, S. Singh, An Algorithm for Anomaly-based Botnet Detection. *In Proceedings of Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI '06)*, San Jose, CA, July 2006.

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

[5] E. Cooke, F. Jahanian, and D. McPherson, The zombie roundup: Understanding, detecting and disrupting botnets. *In Proceedings of Usenix Workshop on Steps 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] T. Karagiannis, K. Papagiannaki, M. Faloutsos, BLINC: Multilevel Traffic Classification in the Dark. *Proceedings of the 2005 conference on Applications,, technologies, architectures, and computer communications*, Philadelphia, 2005.

[9] WT Strayer, R. Walsh, C. Livadas and D. Lapsley Detecting Botnets with Tight Command and Control *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 2006

[10] C. Livadas, R. Walsh, D. Lapsley and WT Strayer Usilng Machine Learning Technliques to Identify Botnet Traffic *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 2006

[11] Microsoft. Windows Malicious Software Removal Tool: Progress Made, Trends Observed. `http://www.microsoft.com/downloads/details.aspx?FamilyId=47DDCFA9-645D-4495-9EDA-92CDE33E99A9\&displaylang=en`, November 2006.

[12] Nmap security scanner `http://insecure.org`, April 2007.

[13] Snort IDS web page. `http://www.snort.org`, April 2007.

[14] Sourceforge Ourmon web page. `http://ourmon.sourceforge.net`, April 2007.