A Comparison of Methods to Reduce Redundancy in Internal Traffic Monitoring to Detect Botnets

Kyle D. Feuz, Neal V. Sorensen, Grigoriy Kerzehner, and Chad D. Mano

Department of Computer Science
Utah State University
Logan, UT 84322, USA
kyle.feuz@aggiemail.usu.edu {neal.sorensen, chad.mano}@usu.edu
gik@duke.edu

Abstract. As botnets become more sophisticated, the systems used to detect such botnets must also become more sophisticated. Specifically, botnets could move towards using internal communications to hide their actions from external monitors, thus an internal monitoring system is critical for protecting against these futuristic bots. Unless care is taken, internal monitoring systems can easily generate large amounts of duplicate information. In this paper we compare various methods to reduce such redundancy and suggest possible improvements to an internal network monitor.

1 Introduction

The rise of the Internet for public use also created new opportunities for public abuse. As more and more people use the Internet for shopping, paying bills, banking etc, the incentive for criminals to obtain private information on-line is greatly increased. This has led to the malicious practice known as botnets. A botnet is a network of compromised computers generaly known as bots, that can then be used to attack other computer systems [1, 4, 11]. Traditionally botnets were organized using a simple centralized command center, with a single person known as the botmaster or botherder issueing commands to the bots via an irc channel. This primitive method of control however, was easily detecable and relatively easy to eliminate. In order to continue to prosper, botnets began to evolve. One of the most prevalant forms of a botnet today uses a peer-to-peer architecture to distribute commands to the bots [5, 7]. As botnets become more and more sophisticated to evade detection and remain active, security systems must also become more sophisticated to successfully detect bot infections within a network.

Defense systems have been developed that are successfully able to detect even these more advanced peer-to-peer types of botnets. History has shown, though, that the botmasters will not sit idly by and watch their botnets being

destroyed. Once again they will evolve and seek to evade even the most advanced intrusion detection systems. In [14] a new botnet model is proposed that is capable of evading all known current detection techniques such as those used by BotHunter [6], one of the most advanced intrusion detection system currently available. It does so by coordinating activities within a subnet so that no single bot is exposed to detection by the defense system by limiting the types of external communication that each bot performs.

A defense against such a bot is also proposed but further research is needed to successfully implement the defense. The proposed model of defense against such bots involves monitoring not only the traffic at the gateway as traditional botnet detectors have done, but also monitoring traffic within the network at the level of the switch. By so doing, the bots are no longer able to hide their communications because both external as well as internal communication is monitored. [2, 14, 15]

One major problem facing the new botnet defense, is efficiency. Monitoring the network at the switch level has the potential to generate large amounts of data, of which a significant proportion may be redundant information generated by monitoring the same data at multiple locations. If this redundant information could be reduced or eliminated the efficiency of detection system would be greatly increased. The system would require less bandwidth to obtain the necessary information to successfully detect a bot, thus allowing the network as a whole more available bandwidth. The amount of processing power and memory needed to process the data would also be reduced. This would make implementing such a system all the more feasible.

2 Related Work

Bothunter is one of the more advanced intrusion detection systems(IDS). This system is a dialog-based IDS and functions by detecting dialog between computers that indicate malicious activity. These activities are given a weighted value and when a specified threshold is crossed, the computer is flagged as a bot [6]. This has proven to be a highly effective technique, but BotHunter is typically installed at the gateway and doesn't monitor internal communications. This is the weakness that the bot in [14] seeks to exploit.

As already mentioned, the solution to stop botnets from exploiting BotHunter in this manner is to move the monitoring inside the network. As indicated in [15] a full fledge BotHunter on each switch is unnecessary and impratical. However, by collaborating between BotHunter and internal switch monitors, the network can be effectively monitored. This collaboration, while effective, also introduces redundancy. Multiple switches will be seing the same traffic and reporting the same information back to bothunter. To prevent this from happening, the redundant information must be eliminated at some point in the process.

Processing and reducing redundancy is not a new subject by any means. Several algorithms have been developed for reducing redundancy in files, files systems, source code, and the world wide web [8, 9]. These algorithms and processes are great for compressing files and eliminating redundancy when all the

information is present, but if one waits until the information is sent to BotHunter to eliminate duplicate information, valuable resources have already been squandered. Therefore, we will compare several methods that reduce or eliminate redundancy from even occurring.

A common technique for reducing redundancy in network monitoring is to use probabilistic monitoring. By randomly sampling packets to be monitored, instead of monitoring all packets on the wire, the total amount of data monitored can be greatly reduced, while key statistics can still be generated. A sampling of packets works well when trying to monitor things such as bandwidth usage as this can be determined through statistical analysis [13]. In contrast, for the case of detecting specific payload signatures, a sampling of packets will not be sufficient to detect most intrusions.

[12] uses "hello" messages to allow wireless ad-hoc nodes to discover a knowledge of local topology to reduce redundancy of broadcast messages sent in an Ad Hoc Network. Similarly, one method that we compared to reduce redundancy in network monitoring, involves monitors that can discover a topology of the network and thus reduce both the number of packets monitored by the probes, as well as the number of messages sent to the collector.

The effeciency of the network is critical to many users. In fact, one of the biggest reasons for network administrators to try and eliminate bots from their network is to relieve the network of the unnecessary load. Therefore, a detection system that places a significant load on the network is highly undesirable and of little use in the real world. By discovering and comparing new ways to reduce the redundancy in internal network monitoring systems, these systems can see a substantial improvement in effeciency and utility.

3 Methods to Reduce Redundancy

We compared the following methods to detemine 1) if they are sufficient to detect the covert bots, 2) how much redundancy remains, and 3) how much overhead is accrued.

- 3.1 Monitor internal communications at a single strategic location in the network
- 3.2 Monitor traffic in one direction only with a maximum of one monitor per sub-tree
- 3.3 Use manually configured monitors
- 3.4 Use broadcasts to determine the location of the network monitors and use self-configuring monitors to behave accordingly

3.1 Single Monitor

By monitoring internal communication at single point in the network, the possibility of obtaining duplicate data is zero. With only a single monitor, no other monitor is present in the network to generate the duplicate data. The benefit

of such a setup is readily visible; no work must be done to reduce redundancy, no overhead is accrued, and the system will monitor traffic correctly no matter how the network is configured. The downside of such a system is the capability of detecting all the bots in a network is lessened. Consider Fig. 1.

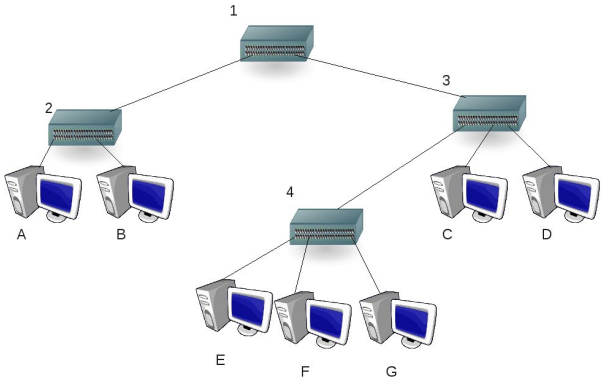

Fig. 1. Simple Network Setup. 1-4 are switches that may be monitored, A-G are computers connected to the network that could be infected.

Let's assume we decide to place a probe on 1. The probe would then be able to monitor all communications going from A or B to C, D, E, F, or G and visa-versa. Thus if either A or B was infected and communicated with another infected computer C-G, the monitor would detect it and flag the problem. If neither A nor B is infected however, the monitor will not see any of the infected bots internal traffic. While at first glance placing the probe on 1 may seem like a good place to monitor internal traffic, a better place for the monitor would be on 3. Fig 2 compares the number of different flows that can be monitored by the probe for each switch, where a flow is defined to be a path of communication, so communication from A to B and B to A would be considered one flow, and communication from A to C and C to A would be considered a separate flow. It also compares the maximum number of bots that could be infected without being detected(MNUB). In the example given above, this would be 5 because the monitor will only see the infected internal communications if A or B is infected.

From Fig 2, we see that placing the monitor on 1 is actually the worst location in terms of obtaining the highest probability of detecting all the bots in our network. Ideally then, to use this method, one would first do an in-depth analysis of the network to determine the best possible strategic location for the probe and then deploy it. We have developed two simple formulas to facilitate this analysis.

Let x_i represent the total number of computers connected to a given port i on any particular switch. Then the maximum number of undectable bots (MNUB) on for a monitor on the particular switch can be determined by the following formula.

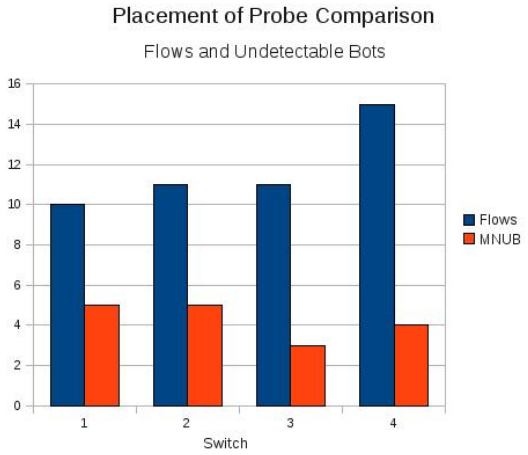

Fig. 2. Number of flows that can be monitored by a single switch and the number of bots that can evade detection.

$$MNUB = Max(x_1, x2, ..., x_n)$$

where n is the number of ports on the switch. To prove this formula to be correct, consider the following. Theorem 1: For a switch to successfully detect a bot, the bot must communicate with another bot that lies on a different port of the switch so that the communication passes through the switch. Corallary: If two bots communicate without the switch seeing the communication, both bots must lie on the same port of the switch. For these to hold true, the network cannot contain any loops. Assume the maximum number of detectable bots is greater than $Max(x_1, x2, ..., x_n)$. This implies that there is a greater number of computers that can communicate without the communication passing through the switch than $Max(x_1, x2, ..., x_n)$ which is a contradiction from the corallary. Now, assume the maximum number of detectable bots is less than $Max(x_1, x2, ..., x_n)$. This implies that there is a number of bots greater than $Max(x_1, x2, ..., x_n)$ that can communicate without the communication passing through the switch, which is a contradiction of the corallary.

The number of flows (f) monitored by any given switch can be determined using the following formula.

$$f = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (x_i * x_j)$$

This formula simply counts the number of different communcation paths that flow through a switch.

One nice thing about these two formulas, is they require only a minimal knowledge of the network. Notice that the number of switches along the path doesn't affect the outcome of the formula only the number of computers ultimately connected to the ports. Unfortunately, most real-world networks have

complex and often unknown network topologies, [3, 10] that make such an analysis practically impossible for larger networks. For this reason, we hypothesize a general rule for determining an acceptably good placement of the network monitor. In order to capture a high number of flows, while limiting the maximum number of infected bots that can go undetected, the probe should be placed on the switch with the heaviest internal workload. For smaller networks, the heaviest workload is generally found on the switch with the most computers connected to it. For larger networks, the heaviest workload is generally found on the switch closest to the center of the network where approximately half the computers are located on one side of the switch and half of the computers are located on the other side of the switch. This is based on the assumption that all computers use approximately the same amount of bandwidth and communicate equally with one another.

3.2 Multiple Monitors, One per Sub-Tree

As an extension of the previous model, we also looked at monitoring traffic in one direction with monitors on multiple switches that are not not descendants of each other. The key concept here is that networks can be arranged in a tree-like structure. If loops exist, they can be ignored using the Spanning Tree Protocol (STP) or one of the other similar protocols. The switch can then be configured to monitor only incoming traffic and ignore all traffic on the root port. By setting up the monitor in this way and never placing any monitor in the same sub-tree as another monitor, redundancy can still be completely eliminated and the probability of detecting all bots in your network is greatly increased. As with monitoring at a single point, this method can potentially have zero network bandwidth overhead if the switches are configured manually. For more dynamic networks that use the STP, the monitors could be configured based on the overhead data that is already present due to the STP so again no additional network overhead would be necessary. The downside of this setup is the possibility of missing a bot still exists. Also, as more probes would be used to monitor the network, this method could be slightly more costly than the previous one. As with 3.1 a knowledge of the network topology can help better place the monitors but by having monitors on multiple subtrees, this knowledge is much less critical to detecting a majority of the bots on a network.

3.3 Manually Configured Monitors

The next method we compared is to use manually configured monitors. Each monitor must be manually configured to monitor certain switch ports, and/or IPs/MAC addresses. In order to configure the switches, one must first know the topology of the network on which the monitors will be placed. As discussed in 3.1 this can be very difficult for some networks even with the available third party tools. Assuming however, that a topology can be determined, it is then a matter of deciding which switches will monitor which flows. This allows for virtually any configuration of the network monitors, with zero network overhead and the

possibility for any amount of coverage desired. This is one solution to the limit of one monitor per sub-tree. Using manually configured monitors presents a few drawbacks as well. The intial setup could be quite time consuming especially for a large network. Also, the system would be very static and inflexible, possibly failing to monitor the network correctly when the network changes. If MAC or IP addresses were used to configure the monitors, the flexibilty and adaptability would be further diminished. Consider, for example, a laptop, which may move around from switch to switch. Obviously, if the mac address of the laptop were used to control which switch monitored it, the laptop, could very easily go undetected a large percentage of the time.

3.4 Self-Configured Monitors

Another solution to allow multiple monitors per sub-tree, is for the monitors to be self-configuring. By building on what has been discussed in the previous methods, our self-configuring method is able to handle multiple switches in the same subtree with only a slight increase in overhead traffic, while still completely eliminating all redundancy once it is in place. [12] used a learned knowledge of the local topology to reduce the amount of broadcast traffic that needs to be sent over a wireless ad-hoc network. For our third attempt to reduce redundancy, we will use a similar concept to implement self-configuring switches which discover a small portion of the local topology to determine which ports to actively monitor.

Our method works as follows: 1) At least one switch at the top of a subtree is configured as the broadcaster (multiple broadcasters are possible for the purpose of introducing protection against the failure of a broadcaster, but this will also increase the amount of overhead traffic that is generated to configure the monitors), 2) The broadcaster will broadcast a request asking for a response from other monitors, 3) The monitors mark the port on which they received the request as the root port, 4) Each monitor replies to the request acknowledging that they are indeed monitoring the network, 5) Each monitor watches for responses to the request and marks the port on which the response was received as a monitored port. 6) Monitors turn of monitoring on the ports marked as monitored or root, 7) Steps 2-5 are repeated periodically, and the monitors are re-configured as necessary. See Fig.3

This method currently one major limitation. When descending any given subtree, once a switch with a monitor is seen, there must not be any non-monitoring switches between that switch and any subsequent switches. This is due to the monitors relying on the fact that if a response to a broadcast request is seen on a port, all computers on that port are assumed to be monitored. If a non-monitoring switch was in the middle of two monitoring switches, this would no longer be the case. When monitoring all switches in a network, this is a non-issue, and even most other scenarios should be able to handle this stipulation.

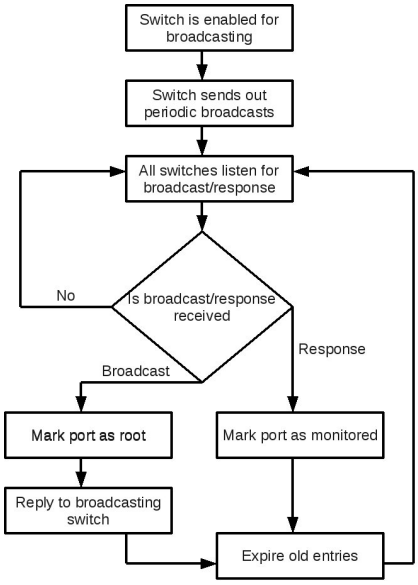

Fig. 3. flow chart for self-configuring monitors.

4 Experiments

In order to compare bandwidth usage overhead, as well as to ensure that each method correctly eliminates redundancy and detects the bots, we conducted experiments for each method. Each method was setup and run on the network shown in Fig. 1. We installed the bots discussed in [15] and they provide almost exclusively the only traffic on the network. Switch 1 was connected to an external network which included traffic beyond our control. This most likely had an effect on our results which we discuss at the end of this section. Computers A, C, and E were infected and actively participated in the botnet. Computers B, D, and F were not active for these experiments.

We calculated overhead as follows. Any communication using the ports dedicated for the use of BotHunter were flagged as monitoring overhead. Additionally, any communications using ports dedicated to the passing of IPFIX messages were also flagged as overhead. Finally, communications using the specified discovery port were marked as overhead as well. The size in bytes of the packets were totaled and overhead was calculated to be $\frac{BytesOverhead}{TotalBytesTraffic}$.

To provide for the maximum amount of consitancy in our experiments, each method was testing using the same procedure. BotHunter was activated, as were the bots. We then began to monitor the traffic passing through each switch. The collecor process was started on a computer connected to the external network on switch 1. Finally, we started up the exporter(s).

The only difference in each experiment was the configuration of the monitors. For method 3.1, the monitor was placed on 3 to provide the best coverage of the

network. Method 3.2 was also setup to provide the best coverage possible. One monitor was place on 2, and the other monitor was placed on 4. The root port on each switch was then designated to correspond with the hierarchy depicted in Fig 1. When manually configuring the probes for method 3.3, probes were placed on 2, 3, and 4. The ports that were being monitored by another probe were disabled. Lastly, method 3.4 placed probes on all switches 1-4. The monitor on 1 was designated as the broadcaster, and all monitors were enabled to listen and respond to the broadcasts.

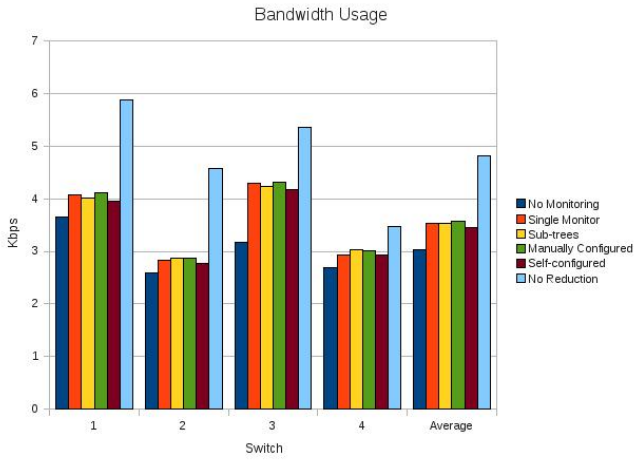

Fig. 4. Graph depicting the total bandwidth used on each switch in the network

Before testing any particular method, we ran the bots and collected the traffic that passed through each switch to give us a baseline of the amount of data we could expect to be generated by the bots alone. Each method was then run, and the total bandwidth used by each was measured as well. Fig. 4 shows the results. As is clearly seen, each method increases the amount of bandwidth used. This is the overhead cost of monitoring internal to the network. However, as can also be seen, the different methods don't appear to differ by much in terms of the amount of total bandwidth they use.

Now consider Fig. 6. This graph shows the amount of overhead that each method incurred from the perspective of each switch. As is plainly visible, using only one monitor presents a significant reduction in overhead traffic. Using one monitor per subtree, still provides a reduction in overhead traffic although not as much. For unknown reasons, the self-configured monitors performed better than the manually configured monitors. This could be at least partially due to the fact that the network traffic varied as the test were run.

The last thing we compared was the maximum number of undetectable bots of each method. See Fig. 5. Because of the nature of our test, each method was able to successfully detect all the bots on the network. However, we looked at the

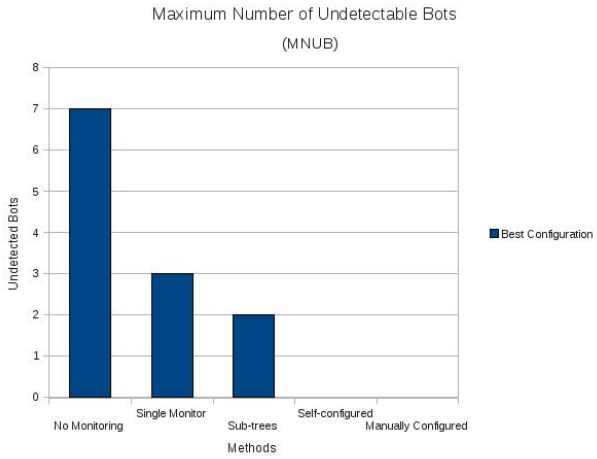

Fig. 5. Graph depicting the maximum number of undetectable bots assuming the best possible configuration using the network shown in 1

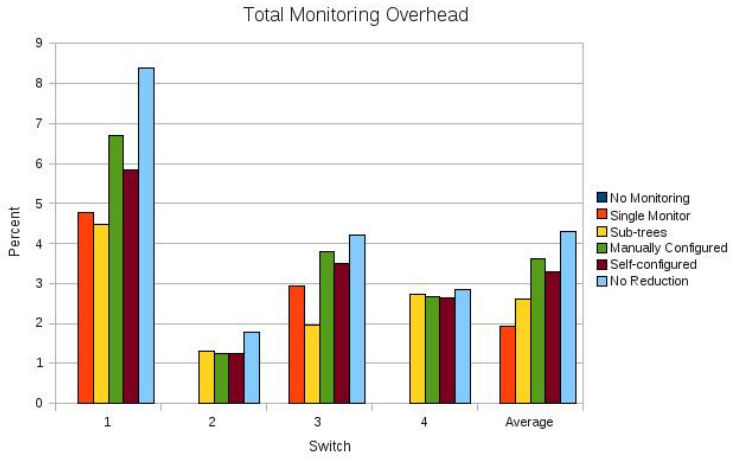

Fig. 6. Graph depecting the percent of overhead traffic that occurred due to the monitoring process

number of bots a system could overlook if different computers were infected. We assume the network is configured as in Fig. 1. As expected a single monitor has the potential of missing the highest number of bots, while both self-configured monitors, and manually configured monitors, when placed on every switch, do not miss a single bot.

We can see that in our limited testing, no method showed considerable deviation from another in terms of bandwidth usage. This is good news for method 3.4 as it shows that in comparison to the bandwidth necessary to perform the monitoring, the additional bandwidth to perform discovery is insignificant, but monitoring on multiple switches does produce a significant increase in overhead. The self-configuring monitors as well as the manually configured monitors clearly excel at detecting all possible bots, while the simpler methods, are not as complete when it comes to detecting bots. Because detecting all bots requires a monitor at nearly every switch, cost could become an issue as well.

5 Future Work

We did not do any testing of our hypothesis on the best location for a single network monitor. An in-depth study of network topologies and the determining the best location to place the network monitor to maximize the probability of detecting all bots could prove to be invaluable in promoting cheap yet efficient internal network intrusion detection systems.

The self-configuring monitors were designed to be dynamic, so that if a switch failed or was removed, the others would continue to correctly monitor the network. However, the current implementation will only work correctly if a leaf monitor fails, failure of a monitor higher up in the tree hierarchy will lead to computers that appear to be monitored, but in reality are not. This could be corrected by passing information such as IPs monitored with the broadcast response. Also failure of the broadcasting switch will lead to complete transmittance of all redundant information. For this reason it is recommended, that multiple top-level sub-tree monitors be enabled with broadcasting.

More extensive testing could be conducted on each of these methods, using a bigger more realistic network setup. Key things to look for include the percent of bots detected as well as the number of false positive generated. Also, the amount of redundancy present in the collection is critical to determine the efficiency of a given solution.

We also considered two other methods for processing and eliminating redundancy, 1)process duplicate data at the central collector and then notify monitors to stop monitoring the specified communication channel, and 2) probes which monitor the communication of other probes and learn which flows are already being monitored. The first method shows some promise in being dynamic and low maintenance, however we did not have sufficient time to implement this method. The second method also has promise of being dynamic and low maintenance, but when using the IPFIX protocol, messages are not easily interpreted,

and a full fledge collector would be necessary with each probe. If an alternative was found this could prove to be an acceptable solution to reducing redundancy.

6 Conclusion

In Conclusion, to detect potential threats from malicious software traffic needs to be monitored within the network. This generates a significant amount of redundant data which if reduced will lead to a more efficient detection system. Using the methods we have compared here to reduce redundancy, there is no clear answer as to which configuration is best. Each method has merit for certain situations, but each falls short of being the perfect solution. In our testing, each method was able to successfully eliminate all redundancy from the system. To do so, each had to sacrifice perfomance at some other location. The methods presented in 3.1 and 3.2 sacrificed the ability to detect all bots. The method presented in 3.3 requires more time to setup, and does not adapt to changes in the network. The final method presented in 3.4 sacrificed a small amount of network bandwidth to discover a partial topology of the network, and could lead to an increase in cost due to the greater number of monitors present.

References

1. Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 41–52, New York, NY, USA, 2006. ACM.
2. Lokesh Babu Ramesh Babu. Master's thesis.
3. Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz. Topology discovery in heterogeneous ip networks: the netinventory system. *Networking, IEEE/ACM Transactions on*, 12(3):401–414, June 2004.
4. Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: understanding, detecting, and disrupting botnets. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 6–6, Berkeley, CA, USA, 2005. USENIX Association.
5. Julian B. Grizzard and The Johns. Peer-to-peer botnets: Overview and case study. In *In USENIX Workshop on Hot Topics in Understanding Botnets (HotBots07*, 2007.
6. Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)*, August 2007.
7. Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association.
8. J. Howard Johnson. Identifying redundancy in source code using fingerprints. In *CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research*, pages 171–183. IBM Press, 1993.

9. Purushottam Kulkarni, Fred Douglis, Jason LaVoie, and John M. Tracey. Redundancy elimination within large collections of files. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 5–5, Berkeley, CA, USA, 2004. USENIX Association.
10. Bruce Lowekamp, David O'Hallaron, and Thomas Gross. Topology discovery for large ethernet networks. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 237–248, New York, NY, USA, 2001. ACM.
11. Bill McCarty. Botnets: Big and bigger. *IEEE Security and Privacy*, 1(4):87–90, July 2003.
12. Wei Peng and Xi-Cheng Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. pages 129–130, 2000.
13. Joseph Reves and Sonia Panchen.
14. Brandon Shirley and Chad D. Mano. A model for covert botnet communication in a private subnet. In *Networking*, pages 624–632, 2008.
15. Brandon Shirley and Chad D. Mano. Sub-botnet coordination using tokens in a switched network. pages 1–5, 30 2008-Dec. 4 2008.