CommunityGuard: A Crowdsourced Home Cyber-Security System

Chase E. Stewart, Anne Maria Vasu, Eric Keller
{chase.stewart, anne.vasu, eric.keller}@colorado.edu

ABSTRACT

In this paper, we propose and implement CommunityGuard, a system which comprises of intelligent Guardian Nodes that learn and prevent malicious traffic from coming into and going out of a user's personal area network. In the CommunityGuard model, each Guardian Node tells others about emerging threats, blocking these threats for all users as soon as they begin. Furthermore, Guardian Nodes regularly update themselves with latest threat models to provide effective security against new and emerging threats. Our evaluation proves that CommunityGuard provides immunity against a range of incoming and outgoing attacks at all points of time with an acceptable impact on network performance. Oftentimes, the sources of DDoS attack traffic are personal devices that have been compromised without the owner's knowledge. We have modeled CommunityGuard to prevent such outgoing DDoS traffic on a wide scale which can hamstring the otherwise very frightening prospects of crippling DDoS attacks.

1. INTRODUCTION

As the Internet continues to control and define more aspects of the physical world, the consequences of Internet attacks also continue to scale. Network attacks, once an annoyance or hassle, can now translate into a loss of money, a source of physical harm, or even widespread chaos. In the past few years alone, we have seen network attacks destroy nuclear reactor development [9], hold hospitals ransom [16], and even control moving vehicles [1].

A particularly weak area increasingly targeted by attackers is the Internet of Things. It is hard to imagine the influence of computer networks and the prevalence of the Internet ever waning. Therefore, creating permanently secure and stable networks is a major technical problem that must be solved before the Internet of Things firmly takes hold. A significant challenge is that a significant segment of the IoT deployments target home deployments. Whereas enterprises have the resources and IT staff to manage security (and still get attacked), home users do not have such resources. We propose that users in the cyber world need the digital equivalent of a physical Home Security System – such as provided by ADT, which monitors physical aspects (*e.g.,* door open) of the home and automatically defend against intrusions.

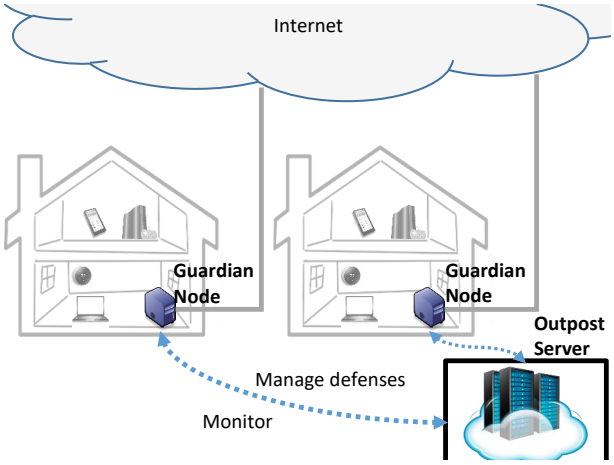

Figure 1: CommunityGuard architecture.

In this paper we present CommunityGuard (illustrated in Figure 1), a system which automatically protects home users from external attacks (much like a home security system), helps prevent home users from unwittingly being used to launch attacks on others, and provides a means for home users to look out for each other (similar to neighborhood watches where people can report suspicious activity to help their neighbors). This is all possible with the combination of network functions virtualization to launch monitoring and defenses where needed, and software defined networking to remotely manage the configuration. In essence, CommunityGuard is (in one deployment model) simply a device that resides between a home router and the cable modem, which connects to a cloud system that automatically monitors for suspicious activity and deploys the appropriate response. Through relying on the collaborative aspects of this network of devices, CommunityGuard has wide visibility that enables it to catch ever-more complicated malware and threats, and ultimately block malicious traffic from ever reaching its intended destinations.

As a demonstration of the power of CommunityGuard, we focus on Distributed Denial of Service (DDoS) attacks, which are especially poised to grow in damage and scale. DDoS is a common network attack in which a user attempts to make an extremely high number of resource requests in a

short time in order to prevent others from accessing the resource. The strength of a DDoS attack is proportional to the number of resources an attack can leverage against its opponent and the time for which they can sustain their resource requests. For this reason, the Internet of Things looming on the horizon (which promises as much as 400 zettabytes of data by 2018 [3]) has already given attackers a huge advantage in sustaining record-breaking DNS attacks against increasingly worrying targets. In events such as the attack on Dyn [8], a moderate amount of traffic passed from an unprecedented number of unique devices (mainly IoT cameras) to create an overwhelming force. In this paper, we demonstrate the use of CommunityGuard for defeating these potentially-enormous DDoS attacks by stopping the traffic at the weaker end of its path – the source subnet (home networks).

2. COMMUNITYGUARD ARCHITECTURE

Shown in Figure 1 is a high-level overview of the CommunityGuard architecture. There are two key components shown.

First, the *Guardian Node* is capable of serving as a device which all traffic between a given home's devices (computers and IoT devices) and the Internet passes through. This node is capable of having network functions deployed onto it and remotely configured. For the purposes of this paper, we use a BeagleBone Black device as a prototype, and elaborate in Sections 3.1.1 and 3.1.2.

Second, the *Community Outpost* is a central manager running on a cloud server which interfaces with each of the Guardian Nodes. The Community Outpost will monitor this traffic for any security concerns and deploy defenses as needed. For our initial prototype, we focus on the deployment of the Snort IPS system and automatic management of its configuration, elaborated in Section 3.2.

Together, these components provide crowdsourced monitoring and remotely managed security. Each Guardian Node is capable of monitoring and sending information to the Community Outpost server (the central security manager). In doing so, we get multiple vantage points and an ability to detect something in one location to help protect another. The Community Outpost will perform analysis (using all information), and deploy the defenses as needed (the snort configuration).

3. SYSTEM PROTOTYPE

Shown in Figure2 is a diagram illustrating our initial prototype of CommunityGuard, where we focus on running an intrusion prevention system (Snort) as the example network function (which can do both monitoring and protection, but our architecture is not solely tied to Snort).

The keystone of the proposed architecture is the Guardian Node which must be placed in a position where it can view the entire network traffic flowing into and out of a user's network, and block all suspicious traffic. The addition of

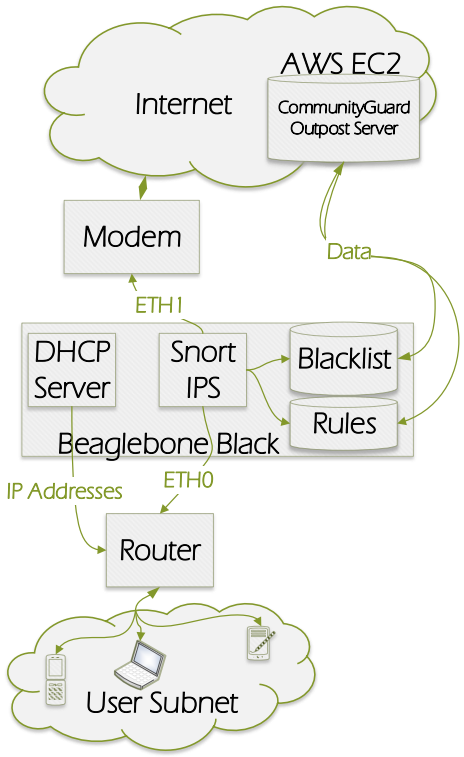

Figure 2: Prototype Overview.

the device must be as non-intrusive as possible, requiring no modification in Modem or Router configuration. Based on these factors, we chose to place the Guardian Node between a Modem and Router. For networks using a Modem and a Router on the same device, we foresee that a production scale implementation would place the Guardian Node within the same box as the Modem and the Router, therefore maintaining the same architecture as shown in Figure 1. Such Guardian Nodes placed at the entry point of home networks can create a solid net of protection that can communicate threats as soon as they emerge. It must be noted that the capability of CommunityGuard in deterring threats (and especially its ability to stop DDoS attacks) is proportional to the amount of subnets that use it – ideally there would be a Guardian Node within every household and business subnet.

The code that we wrote in implementing and evaluating CommunityGuard is available on Bitbucket at https://bitbucket.org/ChaseEStewart/advnetsysfinal/ [4].

3.1 Guardian Node

3.1.1 Hardware Design

The BeagleBone Black, a reasonably powerful embedded device, was chosen to prototype Guardian Node since it satisfied the minimum requirements to implement the envisioned end product. The on-board 10/100 Mbps Ethernet port provided one network interface, while another interface

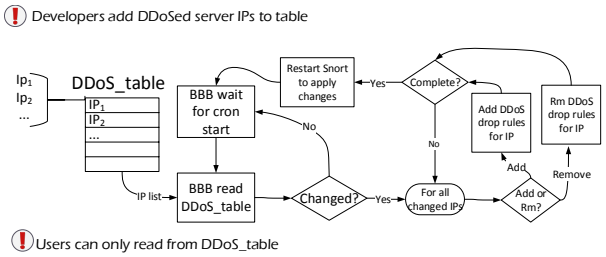

Figure 3: Outbound DDoS prevention Algorithm

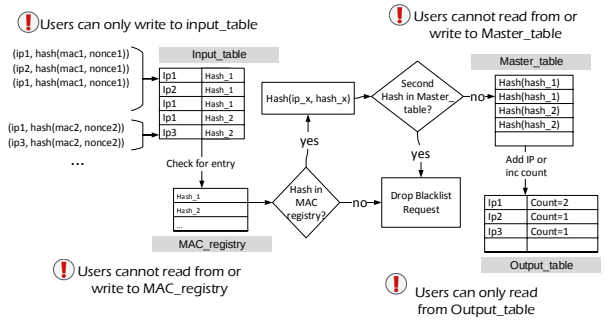

Figure 4: Community Outpost push and pull.

was added using an USB to 10/100 Mbps Ethernet adapter. This prototype is deemed sufficient to display the core mechanics of the Guardian Node.

3.1.2 Software Design

The software design of a Guardian Node is depicted in Figure 2. The device has a DHCP server configured so that a Router can be added behind it. It passes all traffic from one network interface to the other. The Guardian Node runs Snort [15], a powerful Intrusion Detection/Prevention tool, to monitor traffic at both interfaces. A set of Snort rules are configured to block any suspicious traffic that satisfies any of the rules. Snort can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflow, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and many more. It also provides a black-list and white-list functionality that can be used to conditionally block and unblock malicious IP addresses. Snort was chosen as the IPS due to its large amount of documentation, open source software and up-to-date rule-lists, and its high regard in security communities.

Communication of malware alerts is a key aspect of CommunityGuard. This communication is done on a periodic basis by a set of three cron jobs running on the Guardian Node. These cron jobs do the following:

- Keep Snort's general rules up-to-date by pulling rules and IPs from rule repositories.
- Push this particular guardian node's suspicious traffic to the Community Outpost and also to pull new rules and blocked IPs from the Community Outpost.
- Check for DDoS server beacons and generate new anti-DDoS rules if necessary.

The first cron job runs periodically and reads a log created by Snort that contains IP address of sources that have sent bad traffic. It then parses the source IP addresses present in the log, and updates the same to the database on the Community Outpost (described in the following section).

A second cron job runs periodically to fetch information from a DDoS watch list available on the Community Outpost described below as part of the Outgoing DDoS prevention mechanism (server-side is described in Section 3.2.3). If the database provides new IP addresses (for which rules do not exist on the Guardian Node yet) that are currently being DDoSed, the cron job creates and adds Snort rules to drop all such outgoing DDoS traffic, thus clipping off the attack at the compromised source itself. This algorithm is depicted in Figure 3; Such a mechanism, when repeated over all Guardian Nodes, can prevent a DDoS attack from the source.

Finally, the third cron job runs once every day at some random time and fetches the latest bad IP list from the Community Outpost and adds it to the Snort IP Blacklist. It also fetches Snort rules from Emerging Threats [7] to defend against new and emerging attack patterns. These three cron jobs, combined with Snort running on all Guardian Nodes, together establish a system of protection that learns, communicates, and has the capacity to prevent most malware attacks.

3.2 Community Outpost

The Community Outpost is the source of intelligence for the Guardian Nodes that it services. This server runs a database and a couple of cron job timed events to process the information within its database. This server uses its database tables to aggregate threats presented by all collected guardian nodes, process them to determine valid threats, and then present them for individual guardian nodes to read and process. As this server is an obvious target for one who would wish to halt the CommunityGuard service, it is foreseen that the server will be elastically-scaling and secure in a production level implementation.

3.2.1 Attack Models

We assume that a user will not attempt to attack or compromise the availability or function own Guardian Node since our system aims to protect users who are unaware that their devices have been compromised. Furthermore, the Guardian Node itself can be fortified from external attacks through a hardened OS image and security best practices. This leaves a malicious user two possible vectors of attack for causing harm to a particular user- either attempting to get harmful traffic through the Guardian Node, or else reverse-engineering

the Guardian Node to attack the Community Outpost itself. We designed the functionality of the Community Outpost while keeping the following attack models in mind:

A malicious user may attempt to:

1. add useful IPs to the blacklist to block them for users
2. remove blacklisted IPs from the blacklist
3. read user data from the list

3.2.2 Server Blacklist architecture

When any guardian node is created, a SHA256 hash of the node's MAC address and a random nonce stored on the Guardian Node's memory is entered into the mac_addr_registry table, as shown in Figure 4. This hash value will serve as a key for writes, to ensure that writes can only be initiated from a valid Guardian Node.

The Guardian Node is only allowed write permission to the IPv4_input table and read permission from the IPv4_output table and DDoS_output table. This policy prevents attack models 1, 2, and 3 and ensures that even if a malicious user were to reverse-engineer the device to discover a means to contact the Community Outpost directly, the amount of access they could gain is minimal. When a Guardian Node is scheduled to push suspicious traffic to the Community Outpost, it is allowed to write only into the IPv4_input table. It will write the following pair to the input table:

$$\{ip_{suspicious}, SHA256(int(MACaddr), nonce)\}$$

The Community Outpost will process the input table to recalculate new threats at a certain frequency, and then drop the current input table. For each entry in the input table, the server first checks the SHA256 passed by the Guardian Node to ensure that this key is already within the mac_addr_registry. The entries that do not have a valid SHA256 hash are dropped, leaving only the valid requests. For all valid requests, a second SHA256 hash is computed, this time a hash of both the previous SHA256 result and now also the IP address declared as suspicious

$$SHA256(\{ip_{suspicious}, SHA256(int(MACaddr), nonce)\})$$

. The Community Outpost checks whether this hash is already in the IPv4_master_list server, which is a list of the SHA256, IP combinations that have previously been entered-if this is a new suspicious IP, or it is a known suspicious IP being reported by a new and valid Guardian Node, it will be pushed to the IPv4_output table. This second hash function ensures that a given Community Outpost instance receives only one vote towards a given IP address, as a means to prevent against attack model 1.

If this IP is a new entry, it will be entered into the table with a count of 1; if it already exists within the table, its count will be incremented. Once the count tallied against the IP reaches a sufficiently high number proportional to the number of reported threats, the IP will be pushed as output to requesting Guardian Nodes, and blocked automatically by all Guardian Nodes. Relying on a count before pushing a suspicious IP also helps prevent attack model 1 from occurring.

3.2.3 Server Outgoing DDoS prevention

As mentioned above in Section 3.1.2, the Community Outpost also provides a method to prevent outgoing DDoS traffic from a Guardian Node's subnet out to a DDoS target. This system is much more simple than the blacklist method described above: in this case, the DDoS victim list will be edited by CommunityGuard administrators, and only reading will be allowed for all Guardian Nodes. CommunityGuard administrators will confirm the existence of a DDoS attack through an alternate channel, possibly a third party DDoS [6] working with CommunityGuard administrators, and then add the relevant IPs or CIDR ranges to the IPv4_ddos table. Guardian Nodes will use the cron described in Figure 3 to add and remove rules to prevent outgoing DDoS attacks.

4. EVALUATION AND PERFORMANCE

We evaluated CommunityGuard by both the effectiveness of its intended operation 4.1 as well as by network performance while the system was running in-line 4.2.

4.1 Operation

To test the effectiveness of the system, we evaluated two core capabilities of CommunityGuard which are described below. Our *Test Setup* includes two Guardian Nodes present on different networks, which were used to test the system to check if blacklisted IP addresses updated by one Guardian Node was communicated to the other. Both Guardian Nodes connected to the Community Outpost server with the same periodicity.

4.1.1 Prevent and log incoming malicious traffic

Snort rules were configured to block IP traffic that contained malicious content. There are thousands of open-source snort rules available that can alert and drop malicious traffic like worms, illegal attempts at accessing FTP or telnet, and a range of attacks [15]. However, we did not wish to receive actual malicious traffic in testing due to infrastructure and resource restrictions. Instead, the team wrote a few snort rules to treat some arbitrary safe traffic as malicious and configured rules to drop such traffic. The cron job (discussed in 3.1.2) parsed the log and updated the DB about the source from where possible malicious traffic was originating. Depending on a majority of votes (this majority was artificially elevated in our test case since only two Guardian Nodes were present), the Community Outpost proceeded to conditionally add the malicious source IP to the blocked list. Newly confirmed bad IP addresses are fetched when the cron job responsible for adding blacklisted IP addresses next runs on the Guardian Nodes.

4.1.2 Prevent outbound DDoS attacks

This was tested by adding ”fake” DDoSed IP addresses to the DDoS watch list on the Community Outpost. The DDoSed IP addresses were friend machines in another network. In a real world implementation, we assume that a Third Party DDoS protection service [6] that has been specifically employed for the job would update the DDoSed IP addresses to the CommunityGuard administrators. The team added the DDoSed IP addresses manually in order to test this functionality. A TCP SYN DoS attack was simulated with hping3 [12] [10]. The cron job that was checking for new updates from the database fetched these IP addresses every minute and checked for a match in the existing list of DDoSed IP addresses maintained at each Guardian Node. New Snort drop rules were created for these IP addresses if a match was not found. The new rules prevent all outbound DDoS traffic to the IP addresses fetched from the DDoS watch list.

An important point to note here is that the network was still allowed to send legitimate traffic to these attacked IP addresses because Snort rules were configured to monitor the pattern of a DDoS attack before dropping that traffic. One such Snort rule is shown in Figure 5 which detects an outgoing SYN flood attack from the home network to an external network. Figure 5 also shows the log created by Snort when it detects and drops an outgoing DDoS attack as soon as the cron job fetches the DDoS watch list from the Community Outpost. The Snort rule detects TCP-SYN DDoS attack patterns and drops all such outgoing traffic. However, benign TCP traffic is still allowed to pass through. This was tested using the netcat tool [14] to communicate between the attacking system and the attacked system via TCP while Snort was actively blocking all outgoing TCP-SYN DDoS attacks to the target IP.

4.2 Performance

4.2.1 Load test

The Guardian Node is designed to be inserted in-line somewhere before the Router, which implies that the Guardian Node could be introducing a possible bottleneck and stalling traffic coming into and going out of the personal network. Therefore, it was necessary to verify the performance of the Guardian Node while it was fully functional. Speed tests were taken at intervals of 5 minutes under light, average and heavy load conditions to conform performance compliance. Light load was simulated by opening 1-3 video Live streams, a few browsers with medium flash content, one active upload and few Social media pages over 6 different devices ranging from desktops to tablets and mobile phones. Average load was simulated by opening 3-5 live streams, 4-6 pages with high flash content and 2-3 active uploads distributed over the same set of devices. Heavy load was simulated by increasing to 6-8 live streams, 8-10 pages with high flash content and 4-5 active uploads distributed over the same set of devices.

Snort rule

```
drop tcp $HOME_NET any -> $EXTERNAL_NET any (flags: S; msg:"Possible TCP DoS"; f
low: stateless; threshold: type both, track by_dst, count 70, seconds 10; sid:10
0001;rev:1;)
```

Snort log

```
"Possible TCP DoS",TCP,12/11-08:22:48.025236 ,192.168.3.4,41224,10.0.0.2,80
"Possible TCP DoS",TCP,12/11-08:22:58.021326 ,192.168.3.4,51812,10.0.0.2,80
"Possible TCP DoS",TCP,12/11-08:23:08.033561 ,192.168.3.4,16528,10.0.0.2,80
"Possible TCP DoS",TCP,12/11-08:23:18.019386 ,192.168.3.4,44599,10.0.0.2,80
```

Figure 5: Sample Snort DDoS prevention rule and log

Figure 6 shows the performance represented as a linear plot. As can be seen, the performance data is very similar between the baseline and the test case. On an average the test case appears to be slightly lower than the baseline case which can be significantly attributed to the following hardware deficiencies.

- USB to 10/100 Mbps Ethernet adapter – The Guardian Node hardware, which in this case was a BeagleBone Black, has only one on-board 10/100 Mbps Ethernet port. An USB to 10/100 Mbps Ethernet adapter was added since the Guardian Node required two network interfaces. Speed of transfer was therefore severely limited by the speed of conversion.
- Slow SD card writes – The Guardian Node was running a Linux Operating System that was mounted on a SD card. Transfer speeds were also affected by comparatively slow SD card write times.

4.2.2 Performance vs. number of Snort rules

Snort provides the freedom to configure as few rules or as many rules as required. The Load Test mentioned in Section 4.2.1 was done with approximately 7000 Snort rules configured and functioning. Speed tests were also conducted for a number of rules ranging between 100 to 10000. It was seen that the number of rules did not affect speed test results as much as one would expect. Speed measurements indicated similar values to those shown in Figure 6. This was attributed to the fact that Snort processes utilized around 150 to 200 MB of RAM in all cases ranging between 100 to 10000 rules. Figure 7 shows the RAM utilization without Snort running, Snort running with 156 rules and Snort running with 9863 rules.

The CommunityGuard team feels that the set of running Snort rules can be further optimized for better processor and RAM utilization. Optimization of Snort rules will be pursued as future work. It is to be noted that increasing traffic from 100 Mbps to 400 Mbps will increase the RAM requirement to almost 1GB for 100-10000 rules. The implementation of Snort on the Guardian Node was using much less RAM since network traffic was limited to 100 Mbps by the Ethernet card. Dealing with higher traffic speeds would require specialized hardware. In an ideal implementation, the Guardian Node would have a network card with performance specifications equivalent to it’s router’s network card.

Figure 6: Results of Snort throughput tests: test=CommunityGuard, baseline=no CommunityGuard

RAM utilization at Guardian node without Snort instance

```
KiB Mem:    503912 total,   280868 used,   223044 free,    20912 buffers
KiB Swap:        0 total,        0 used,        0 free.    58264 cached Mem
```

RAM utilization at Guardian node with 156 Snort rules configured

```
KiB Mem:    503912 total,   479568 used,    24344 free,    20856 buffers
KiB Swap:        0 total,        0 used,        0 free.    58244 cached Mem
```

RAM utilization at Guardian node with 9863 Snort rules configured

```
KiB Mem:    503912 total,   481016 used,    22896 free,      568 buffers
KiB Swap:        0 total,        0 used,        0 free.    28896 cached Mem
```

Figure 7: RAM Utilization on the Guardian Node

5. RELATED WORK

Collaborative Network Security is not a new topic, and some previous work has been done in this domain. Chen, Dong et al. [2] describe a collaborative security for multi-tenant data centers using a security center. Mu, Chen, et al. [13] propose a collaborative security management system for Metropolitan Area Networks using a P2P network. Both papers do not address security for a common user's local network or large scale vulnerabilities introduced by personal devices that have weak security such as IoT devices. Also, both papers do not address situations where a malicious user might try to block valid traffic for all peers by sending bad data to the central security system, nor do they deal with preventing an outbound DDoS attack. These are aspects we have dealt with in our paper. Mirkovic, Prier et al. [11] have talked about preventing a DDoS attack at the source by monitoring 2-way traffic periodically and at all points of time. They do not talk about cloud-sourcing this information to prevent similar outbound DDoS attacks on all networks and mostly focus on detecting DDoS attacks based on the pattern. As far as DDoS attacks are concerned, we mostly focus on running up-to-date EmergingThreats rules, as well as the cloud sourcing aspect of preventing a large scale DDoS attack.

6. CONCLUSIONS AND FUTURE WORK

As network security will only become more crucial over time, we aspire to a future where users' personal area networks and devices defend themselves and each other from emerging threats. To this end, we introduced the design, prototype, and evaluation of CommunityGuard, an in-line home cyber-security system where a Guardian Node in each home shares new threats with nodes in other homes. This provides a sort of herd immunity against new attacks. As soon as an attack claims a victim, all of the victim's peers will be informed, and will repel the same attack against their own subnets.

As this is only a first step toward our overall vision, we envision much future work. This includes large scale testing (deploying in many homes), exploration of alternate hardware platforms, integration into routers, expanding on the correlation across anomalies seen in various Guardian Nodes, and exploring monitoring and defense mechanisms beyond snort, such as deploying specific DDoS scrubbing network functions (as was done in Bohatei [5] for ISP networks).

7. REFERENCES

[1] M. Anderson. Black hat 2014: Hacking the smart car, 2014.

[2] Z. Chen, W. Dong, H. Li, P. Zhang, X. Chen, and J. Cao. Collaborative network security in multi-tenant data center for cloud computing. *Tsinghua Science and Technology*, 19(1):82–94, Feb 2014.

[3] I. Cisco Systems. Cisco global cloud index: Forecast and methodology, 2015-2020, 2016.

[4] C.Stewart and A. Vasu. Avant-guard source code, 2016.

[5] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. Bohatei: Flexible and Elastic DDoS Defense. In *24th USENIX Security Symposium (USENIX Security)*, Aug. 2015.

[6] J. Grady. Worldwide ddos prevention products and services, 2013.

[7] T. Green. Emerging threats faq, 2016.

[8] S. Hilton. Dyn analysis summary of friday october 21 attack.

[9] M. Holloway. Stuxnet worm attack on iranian nuclear facilities, 2015.

[10] Hping. Hping active network security tool.

[11] J. Mirkovic, G. Prier, and P. Reiher. Attacking ddos at the source. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 312–321, Nov 2002.

[12] R. Moyers, P. Dunning, C. Marchany, and G. Tront. Effects of wi-fi and bluetooth battery exhaustion attacks on mobile devices, 2010.

[13] B. Mu, X. Chen, and Z. Chen. A collaborative network security management system in metropolitan area network. In *2011 Third International Conference on Communications and Mobile Computing*, pages 45–50, April 2011.

[14] netcat. Netcat 1.10.

[15] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[16] Symantec. Ransomware and businesses 2016, 2016.