Implementing Security Control Loops in Security Autonomous Response Networks

Hristo Dimitrov
University van Amsterdam

July 11, 2014

Abstract

Security in corporate infrastructures is a big issue, because of the ever growing intrusive technologies. By implementing security as a service a lot can be improved from the situation of having administrators as security problem solvers. A software solution could share and reuse the newest techniques for detecting and fixing security threats. Having control loops actively monitoring the network and responding accordingly when an issue is detected.

This paper shows that control loops that actively monitor and adapt the network in such away that it can cope with certain types of attacks. . Furthermore we research the way such loops can be implemented in Software Defined Networks, so they can become Security Autonomous Response Networks. As a proof of concept, a Security Control Loop with two possible adjustment responses was implemented. The results show that the concept of Security Autonomous Response Networks works. Both responses have different effect on the attack and the systems functionality as a hole. Those effects can be measured and based on company policies, the responses that caused them can be rated accordingly, so that a most suitable response can be defined.

Moreover, we also explore the design space of creating control loops. One scalable and profitable way of implementing such is by using the Enterprise Service Bus architecture and having pluggable modules for the security threat detection part and for the responses.

Preface

I wrote this research paper as a Final Thesis during my Master studies in the System and Networking Engineering program at the University van Amsterdam. It contains detailed information about the research on Security Control Loops in Security Autonomous Response Networks that I conducted. The main purpose of this research paper is to create an example implementation and answer the research questions which are given in the Introduction chapter.

I would like to thank all of my teachers for giving me enough knowledge so that I would be able to conduct this thesis research. And also give special thanks to my project supervisors Marc X. Makkes and Robert J. Meijer for the great guidance and the useful feedback during my research project.

Amsterdam, July 11, 2014

Hristo Dimitrov

Contents

List of Figures

List of Tables

1 Introduction

Security is a big issue in the field of computer systems and networks, because of the ever growing intrusive technologies. Companies have administrators and security experts who are supposed to fix all security threats and problems that may occur. In order to detect attacks Intrusion Detection Systems (IDSs) are used. But in most of the cases security experts and administrators have to decide on solutions for resolving the problem and implement them manually. This has some disadvantages. First of all, there will always be someone smarter or more knowledgeable in the attacker groups and he/she will use new methods for attacking the company system. This is the case, because every company can not have the best security experts in the world available for solving issues at all time. Secondly, the administrators will always need a reasonable amount of time until they fix the issue. If they are away, they will need to first come on sight. Then they have to investigate the exact cause and consequences of the problem. Once the problem is clearly defined, they need to come up with a suitable solution and finally they need to implement it. This hole process can take from a few hours to a couple of days.

Now the innovative technology of Software Defined Networks (SDNs) [1] which are networks that are being created, configured and managed by software, allows for a complete automated network manipulation. Since those networks are virtual, a computer program can not only change the parameters and policies of such a network, but also its topology. A control loop can be used in such a network to actively monitor for security issues and trigger adequate countermeasures to those issues. If such a Security Control Loop is implemented in an SDN a Security as a Service (SecaaS) architecture can be developed. The result will be a Security Autonomous Response Network that adjusts itself in order take care of security threats. The way such a network reacts to the security problem can be different and those different countermeasure actions can be implemented in modules called responses. There can be multiple responses that can solve a given issue and the Security Control Loop is supposed to trigger the best one.

Modular implementation of a control loop will allow for different companies and organizations to share resources like security responses or intrusion detection methods with each other. This will give the companies a way to adopt the best countermeasures technologies that are available. Outsourcing the security to software will also allow for easily managing the security of very large and complex networks. A software implementation will be able to keep track of a complex network and fix security issues, without having an administrator to understand the entire complexity of the network and come up with a feasible solution. Having such complex networks will also make it harder for the attacker to understand how they work. And finally software will make it easier to organise the security of a company and implement it in a consistent state.

In section 2 some of the work which is already done on security threat detection is presented. The scope of the research is defined in section 3 "Scope". Then the methodology used to answer the research questions and the main tools used for the implementation are presented in section 4 "Approach". The details of and the reasoning for the work done are given in section 5. First the definition and implementation of the topology and the development environment are described. And after that the Security Control Loop and the responses it triggers, which are build on top of SDN implementation are explained. The results from the example implementation are presented and discussed in section 6. Finally, in section 7, the final conclusions from this research are drawn and presented compactly.

1.1 Research Questions

The research question for this project is defined as follows:

How could a security control loop be implemented as a software solution?

In order to answer that question satisfactory, two more sub-questions are defined as follows:

- What properties should the implementation of a Security Autonomous Response Network have in order to make it beneficial and effective against security threats?
- How can a Security Autonomous Response Network decide on which response will be better to execute in a given situation?

2 Related Work

There is some work already done in the field of security threat detection. One of the available methods is flow analysis. An article [2] by Michael Patterson describes the features and advantages of using flow analysis. Another approach is by using Intrusion Detection Systems (IDSs). There are a lot of tools available for this method. A report [3] by Tzeyoung Max Wu lists and describes a large variety of IDS tools. There are different kinds of IDSs, based on how they operate. A paper [4] by Damiano Bolzoni describes in detail the Anomaly-based IDSs and why they are better than signature-based IDSs.

3 Scope

This research focuses on creating an example implementation of Security Control Loops in Security Autonomous Response Networks. This is done in order to get a better understanding of how such a control loop could be implemented in software. The implementation will also have different countermeasure scenarios as responses to security threats, so that a way for evaluating those responses can be defined. Due to time restrictions the amount of the countermeasures that are going to be defined, characterized and implemented in code will be limited. Also implementing the logic that will select the most suitable countermeasure response is not in the scope of this project. And finally since there is a lot of work already done in the field of intrusion detection, there will be less focus on how does the control loop detect and classify the security problems.

4 Approach

4.1 Methodology

First an attack scenario is chosen. This scenario is going to be the base for creation of an example implementation. In order to implement an example control loop, first a suitable network and environment need to be defined and set. The properties of the development environment need to accommodate for the chosen attack scenario and also allow for different kinds of countermeasures to be taken. When the environment is set and its capabilities are defined, two simple countermeasures are chosen, which will be implemented as responses triggered from the control

loop. After the implementation of those responses and the control loop itself, they are going to be tested, so that their impact on the demonstration attack can be observed. Finally some more generic conclusions will be drawn from the resulting platform and its behaviour.

4.2 Tools

The SDN which will act as a base for the implementation of the Security Control Loop will be an OpenFlow [5] network which is set in Mininet [6]. Mininet allows the usage of Python scripts for defining and configuring the network. Therefore the control loop and the triggered responses will be programmed in Python. In addition to that some connection and bandwidth monitoring tools like Pktstat [7] and Dstat [8] will be used by the control loop to observe the state of the network.

5 Example implementation of a Security Control Loop

5.1 Topology and environment

5.1.1 Scenario

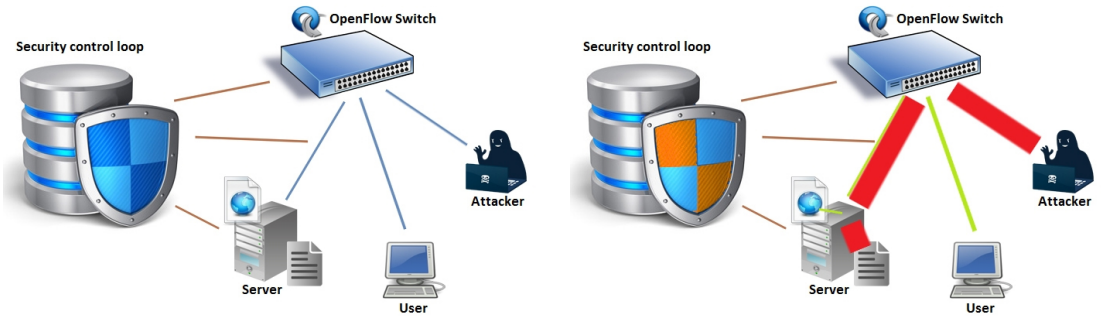
In order to implement an example of a Security Control Loop, first a security threat scenario has to be defined. There should be an infrastructure which offers some services. This infrastructure will represent a corporate network. There should also be an attack issued on the server of that infrastructure and results from this attack should be observable. No complicated topology is needed for this implementation. The Security Control Loop should operate in the same way in a really complex topology as it does in a really simple one. Of course more logic should be programmed into it when it has to to understand more complex topologies and also what it can do with them.

The security threat

A suitable attack which is both easily detectable and offers different countermeasures for its neutralization would be a Denial of Service (DoS) attack. In order to have a bigger flexibility for security response and a better way to observer the consequences of the attack, two services will be set. One of them will be under attack and the other one will be the one that the normal user tries to access.

The topology

For this implementation a single OpenFlow switch with multiple hosts is enough (See Figure 1a.). The switch will represent the company network with all the servers and hosts attached to it. One of the hosts will be set up to act as a server. This server is going to host a web page on port 8001 and some files on port 8000, both services will use the SimpleHTTPServer. This way, another host which will act as the attacker can issue a large number of requests for downloading a big file hosted on port 8000 and consume all the available bandwidth from the server, making responses from port 8001 slower for yet another host which will have the role of a user (See Figure 1b.).



(a) Topology (b) Attack scenario

Figure 1: Example implementation infrastructure

The Security Control Loop

The Security Control Loop will run in the Python script that controls the network virtualization in Mininet. This will allow the Control Loop to make changes to the virtual network as a response to a security threat. It will also allow for issuing commands on all of the defined hosts, which will be helpful both for monitoring and for reconfiguring them.

The network links

Also since the entire SDN with all its hosts will run on a single machine, the virtual network links are going to be configured to 10 Mb/s, so that there is not too much CPU overhead from all the requests and responses that need to be processed. This can be done, because in this scenario the DoS attack targets the link bandwidth and not the CPU load.

5.1.2 Implementation

Setting up the environment

For setting of the environment a Python script was created. The script first initiates a Mininet topology with the above defined characteristics and then checks the connectivity of the created network. As soon as this is done, it sets up the HTTP servers on pots 8000 and 8001 of the first host which is connected to the OpenFlow switch. Just after this is done can the Security Control Loop be started.

Simulating the attack

For simulating the defined DoS attack, a tool is needed, which can initiate a large number of downloads at the same time. The wget utility can be set to download a file multiple times or fragment it and download it from different sources, however the downloads are put on a queue and performed one by one. Even when multiple instances of the wget command are started, they still wait for each other instead of setting up multiple TCP connections for the downloads. Another option was to use a DoS tool like the Hammer [9] which is written in Python or the

Slowloris [10] which is written in Perl. Those tools are able to successfully perform a DoS attack on a specific port of a given host, but it was not possible to set a path and attack a given resource on a specific port, therefore no download of a specific file could be initiated with those tools.

```
python3 ~/DoS/hammer/hammer.py -s 10.0.0.1 -p 8000 -t155
```

```
perl ~/DoS/slowloris.pl -dns 10.0.0.1 -port 8000 -timeout 10
```

At the end the Apache Benchmark [11] tool was used for simulating the attack. For this scenario 100 000 requests were issued to a 1 MB test file and they were performed 1 000 at a time.

```
~/httpd-2.4.9/support/ab -n 100000 -c 1000 -r http://10.0.0.1:8000/test_1M.img
```

This attack was not able to fully bring down the page hosted on port 8001, but it made the response times of the page about three times slower and for the demonstration purposes of this implementation this is enough to show the attack. Also the upload bandwidth of the server was at full usage at all time.

5.2 Response definition and implementation

When dealing with security attacks, cutting off the attacker immediately may not be such a good idea. This is because when the attacker notices that he/she has been cut off completely, he/she is going to start looking for other attack vectors to attack the same system. However if the effect of the attack is neutralized without completely blocking the attacker, then this might keep him busy for a little longer, since he/she will try to find a way to make the attack work as intended, before he/she decides to give up on that attack vector. Therefore the following responses are designed to keep the attacker engaged and still neutralize the effects of the attack.

5.2.1 Attack Isolation Response

Scenario

The first simple response which will be triggered as a countermeasure to the DoS attack is going to focus on isolating the attack, so that the other services hosted on the same server become fully available. When the attacker starts flooding the server with HTTP requests to download the file hosted on port 8000, than all of the available upload bandwidth for the server gets used by the responses containing the file. Then the control loop is supposed to detect that and isolate the attack by setting up a new server on the fly, moving the attacked service to it and redirecting the incoming requests for that service to the new server. That way the newly set up server will start generating the HTTP responses containing the file and the bandwidth of the other server will become available again. The attacker on the other hand will still receive the responses to his requests, but he will be redirected (See Figure 2.).

Implementation

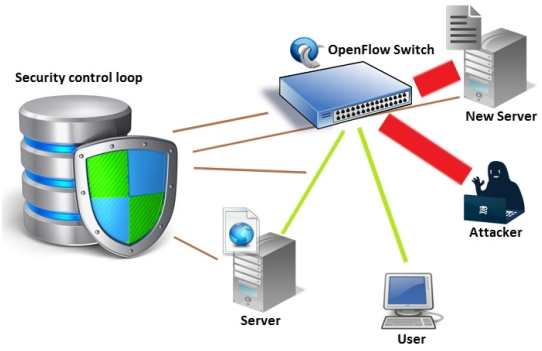


Figure 2: Attack Isolation Response of the Security Control Loop

The purpose of the Security Control Loop is to be able to detect security threats and to take countermeasure actions when it does. Since this implementation is just an example, it does not need to be able to monitor large networks and support all kinds of different responses. Therefore having a simple while loop iterating over some monitoring commands and having the countermeasure responses in conditional statements can be used for simulating the control loop. However in real Security Autonomous Response Networks where the functionality of the control loop needs to be a lot bigger, this solution may not be the best one. A better one would be to use the Enterprise Service Bus (ESB) [12] architecture, where the intrusion detection part of the Security Control Loop posts messages about detected anomalies, so that the response part can pick those up and make the needed adjustments.

Once the control loop starts iterating, a check for ongoing security threats should be performed in every iteration. Detecting security issues is a big problem on its own and there are a lot of IDSs available which combine the latest technologies for detecting intrusion. Integrating the Security Control Loop with an existing IDS might be a better idea than creating one from scratch. Sicne Security Autonomous Response Networks are supposed to deal with security on a network level rather than on an individual host level, a Network Intrusion Detection System (NIDS) should be used instead of Host-based Intrusion Detection System (HIDS). The chosen IDS should also be flexible and configurable and it should provide a way for integration with the control loop. Bro [13] is an open-source UNIX based NIDS. It is highly configurable in terms of what it can detect and it allows for setting up custom triggers for when an intrusion was detected. Another open-source NIDS which might be suitable for integration with Security Control Loops is Snort [14]. It is cross-platform based and there are some third-party tools available to interface it for reporting, which can be used for integrating it in a SecurityAutonomous Response Network. However due to the time restrictions of this project, Bro or Snort could not be set up, configured and integrated with the control loop properly.

For this example implementation it is sufficient to just have the real time statistics of the bandwidth of the network, so that the control loop knows if there is an attack going on and also what are the effects of that attack. This was done by using Pktstat [7] to collect statistics about all connections currently set up with the server. If there are too many of them a flag that there might be a DoS attack going on is set. Then the attacker is identified by further investigation of the available statistics. If there is one source IP address, which has a lot of TCP connection

originating all from various ports, to the same port of the server, than this is considered to be the attacker. Please note that this is not a good way for detecting DoS attacks in real life, but it can be used for this implementation since we already know how the attacker is operating.

If an attacker has been identified in the control loop, then the response is triggered. A new host in the virtual network is created using the Mininet functions in Python. The service which is under attack is stopped and a new service is being set on the same port, which responds with HTTP status code 301 (Moved Permanently) and a pointer to the newly set up server. Also the stopped service is started on the new server. In Mininet all hosts share the same file-system, so there is no need to copy the hosted files from one server to the other.

Next we give an overview of the functional steps that were implemented for the Attack Isolation Response of the control loop (conditional steps are shown in brackets):

- Creating topology
- Testing the Network
- Start Services
- Start Control Loop
  - Collect TCP Connections Statistics
  - Check Number Of Connections
  - *(Determine Potential Attacks)*
  - *(Create New Server)*
  - *(Redirect Traffic To It)*

The complete code of the implementation of the Attack Isolation Response control loop can be found in appendix A.

5.2.2 Attack Limiting Response

Scenario

The second simple response will focus on limiting the attack in such a way that it becomes ineffective, instead of isolating it. Since the first response consists of a single action used to neutralize the attack, the second response will adjust the network multiple times, if this is necessary, until it stabilises. This will better demonstrate the presence of a control loop which makes continuous adjustments to the network in order to limit a security threat. This time, when the attacker starts a DoS attack on the file hosted on port 8000 and the control loop detects is, it will set up a rate-limit on that port in order to limit the attack. The logic behind this response is as follows: Whenever there is an attack on port 8000, which consumes all of the available bandwidth, then limit the attack in such a way that it only consumes half of the link bandwidth, so that the other half is still available to the other service which is hosted on port 8001.

Implementation

For this response the control loop is implemented in the same way as it was for the previous one. And so is the investigation of TCP connections and the identifying of the potential attacker.

After a DoS attack has been detected and the attackers IP address is known, some bandwidth statistics need to be checked in order to determine what part of the link bandwidth is consumed by the attack. The Pktstat tool also outputs the number of transferred Bytes per connection, however when added those numbers were different than the current bandwidth consumption, therefore another tool was needed for those statistics. The Dstat [8] tool was used. It provides bandwidth statistics and allows for saving the output to a CSV file, so by using the CSV module for Python, it was easily integrated with the Python implementation of the control loop. Based on how much is the traffic generated by the attack, a rate-limit has to be set or adjusted. Because with this set up there might be spikes in the traffic, an average bandwidth for 8 seconds time period is used instead of a single sample.

Ideally a rate-limit which limits the amount of bandwidth used by a given TCP connection to a given port should be used for this example. That way, by setting a limit per connection, one can influence the total bandwidth of all connections to that port and this will be at maximum the limit which is set multiplied by the number of current connections. However there was not found any out-of-the-box solution for placing such a rate-limit. So another approach was used. It is limiting the number of allowed TCP connections to port 8000 per IP address per second. Limiting the number of incoming connections from the attacker limits also the number of file requests that will reach the server and respectively the number of responses send back from the server. This will also result in lower upload bandwidth consumption for the server.

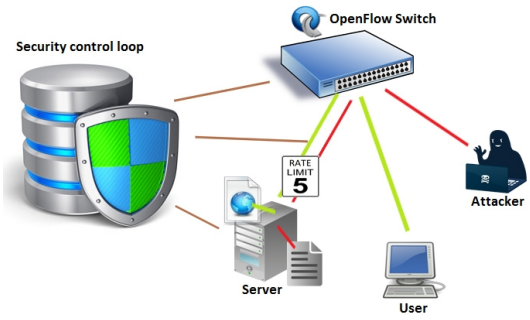
In order to have the rate-limiting adjustments visualized better in a traffic I/O graph (See Section 6.) a smaller file of size 50 KB was used for the attack. Also the amount of simultaneous connections was reduced to 15, otherwise there are too many connections that time out after the rate-limit is being set, that the Apache Benchmark tool gives up and stops issuing requests.

```
~/httpd-2.4.9/support/ab -n 10000 -c 15 -r http://10.0.0.1:8000/test_50K.img
```

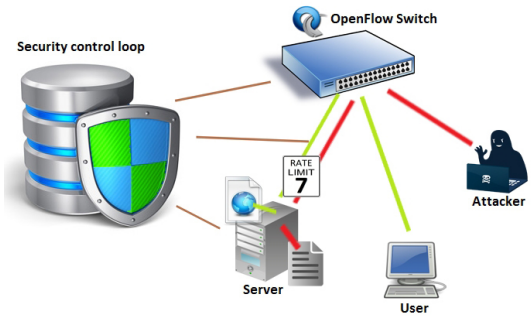
When the hole upload bandwidth of the server is consumed by the attack and there is no rate-limit set, a default rate-limit of 5 TCP connections per source IP address per second will be set for port 8000 of the server (See Figure 3a.). On every next iteration while the attack is still going on, the rate limit will be decreased if the consumed bandwidth is more than 60% of the link bandwidth or increased if it is less than 40% (See Figure 3b and 3c.). This is repeated until the consumed traffic by the attack stabilised between 40% and 60% (See Figure 3d.).

Here you can see the functional steps that were implemented for the Attack Limiting Response of the control loop (conditional steps are shown in brackets):

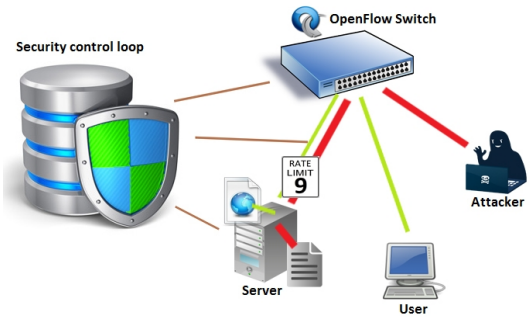
- Creating topology
- Testing the Network
- Start Services
- Start Control Loop
  - Collect TCP Connections Statistics



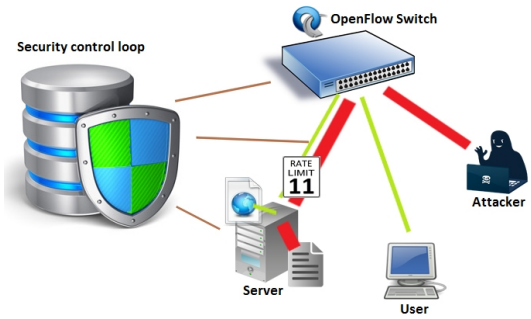
(a) A default rate-limit of 5 TCP connections to port 8000 of the server per second per source IP address is set



(b) The maximum bandwidth consumed by the attack is less than 40%, so the rate-limit is increased to 7



(c) The maximum bandwidth consumed by the attack is less than 40%, so the rate-limit is increased to 9



(d) At a rate-limit of 11 the network stabilizes and the attack consumes at most about half of the available badwidth

Figure 3: Attack Limiting Response of the Security Control Loop

- Check Number Of Connections
- *(Determine Potential Attacks)*
- *(Collect Bandwidth Statistics)*
- *(Adjust Rate Limits)*
- *(Implement New Rate Limits)*

The complete code of the implementation of the Attack Limiting Response control loop can be found in appendix B.

5.3 Readers reminder

Keep in mind that the choice for example attack and selected example countermeasures for this attack are not relevant for the conclusion of this research. They were selected, because they are easy to understand and could efficiently demonstrate how an implementation of a Security Control Loop would work. A real Security Autonomous Response Network should be able to fix all

kinds of attacks and security threats, by the means of all kinds of countermeasures implemented as responses.

6 Results

In order to show the effects of the implemented responses, a Tcpdump was started on the server which was under attack. Then both responses were tested and the generated traffic was investigated in Wireshark.

There is also a way to easily visualize live in 3D the entire topology of the Mininet virtual network by using HyperGlance [15]. In order to do that a OpenDaylight [16] or a HP VAN [17] controller has to be integrated with the OpenFlow switch. HyperGlance can than connect to one of those controllers and get all the end to end data about the virtual infrastructure that it needs. Such a visualization clearly shows in real time the network changes which are made by the response of the control loop. However the responses implemented during this research are simple and visualizing them in such a way does not make too much sense.

The I/O graph from the traffic generated by the attack in the attack isolation scenario (See Figure 4.) shows that after the attack is stated the entire outgoing bandwidth is consumed by the responses from the server. The incoming traffic is much less, since it only contains the HTTP requests. It takes about 7 seconds for the control loop to detect the attack and run the response. After the attack is redirected to the new server, the incoming and outgoing traffic are almost equal in terms of bandwidth usage. This is because for every HTTP request a HTTP redirect response which is about the same size as the request is being sent.

The I/O graph from the traffic generated by the attack in the attack limiting scenario (See Figure 5.) shows the same attack being started as in the other scenario. For this graph a moving average of 8 samples is used in order to show more clearly the variations of the traffic as a result of the rate-limit adjustments. This is also the reason why it appears to be smoother than the other graph which used a moving average of 4 samples only. In about 9 seconds the first rate-limit of 5 connections per second is set. This significantly brings down the accepted HTTP requests from the attacker, which results also in limited responses from the server. The maximum outgoing bandwidth consumption is less than 40%, so the control loop increases the rate-limit with 1 on every iteration. At a rate-limit of 11 connections per second the network stabilizes.

The effects from the two responses can now be compared on different parameters. Each response should get a grade for every parameter and those grades should be comparable between different responses. A suitable way for such grading is by using percentages, 100% being the best possible result and 0% the worst. Of course those percentages are going to be relative, but as long as they are comparable this should not matter that much.
Another aspect that needs to be taken into consideration when comparing responses is that every company can have different preferences for comparison parameters with varying importance. A company policy needs to establish which of those parameters are more important for that company, so that they can be weighted. Than based on the comparison results and the defined weights of the parameters, a rating can be calculated for every response.

By using this method, the two control loop responses that were implemented can be rated

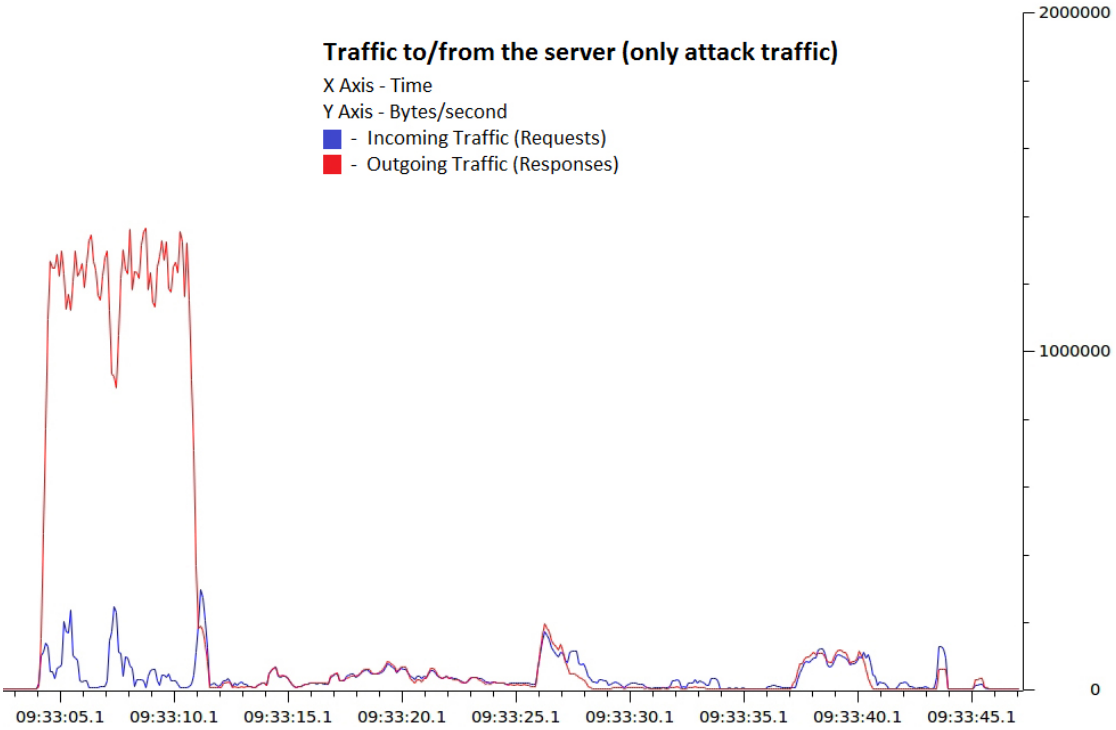


Figure 4: Generated incoming and outgoing traffic of the server in the Attack Isolation Response scenario

and compared. First the company has to define the comparison parameters and assign weights to them. Lets say that our imaginary company cares the most for the money that it spends, so a parameter "Price" is defined and its weight is set to 25. The company also wants to have high availability of its web-server which hosts the main page of the company to customers, so an "Availability of web-server" parameter is set with weight of 20. The file-server which hosts some draft reports however is not that important, so "Availability of file-server" parameter is set with weight of 5. The effect of the countermeasures on the attacker is also important, so a parameter called "Effect on the attacker" is defined with weight of 10. And finally the company would prefer if there is less traffic flowing in its network, so an "Overall bandwidth consumption" parameter is set with weight of 5.

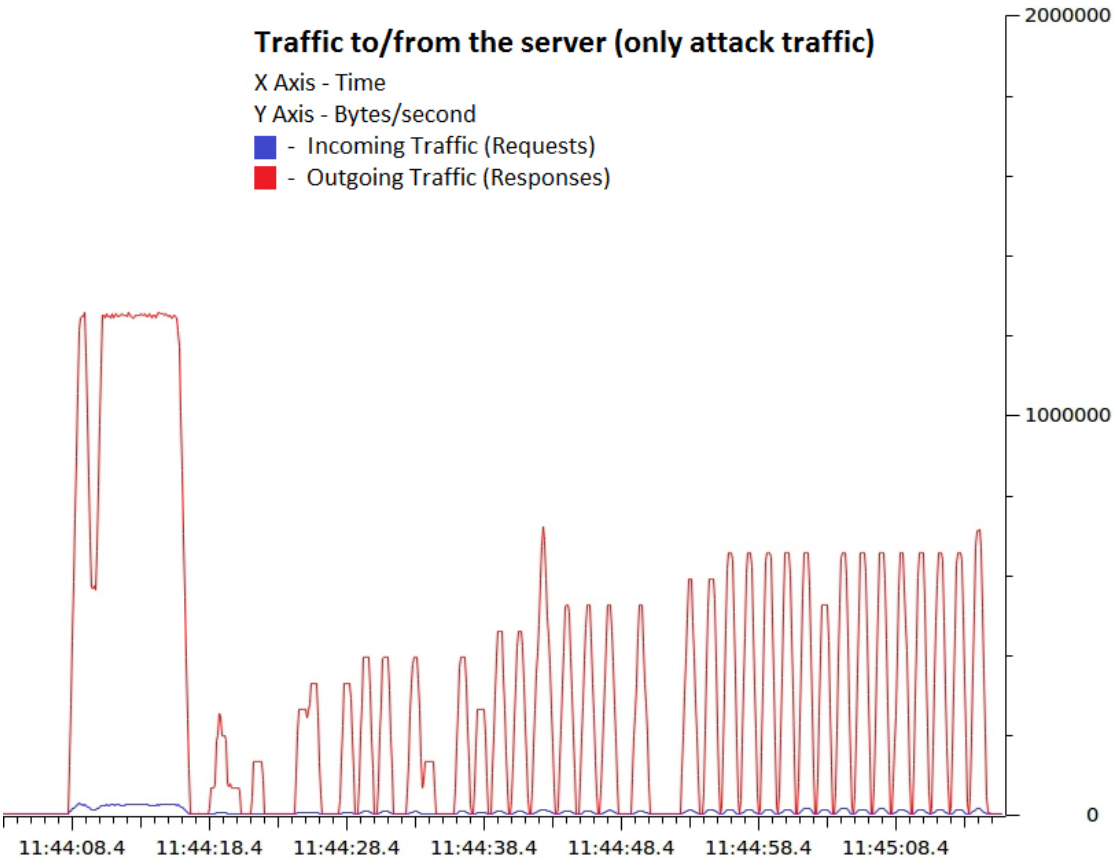


Figure 5: Generated incoming and outgoing traffic of the server in the Attack Limiting Response scenario

Then the responses can be compared and graded (See Table 1.) on every parameter:

- **Availability of file-server (5)** - The Attack Isolation Response does not lower the effect of the attack. Even if the file-server is moved, it is still under attack. The Attack Limiting Response, however, narrows the attack and so the service will also be available to other users. In this category the Attack Limiting Response is better than the Attack Isolation Response.
- **Availability of web-server (20)** - The Attack Isolation Response completely moves the effect of the attack away from the server which hosts the web page. Only the redirect messages are handled by it. On the other hand the Attack Limiting Response still dedicates half of the bandwidth of the link to the server to the attack. Because of that in this category the Attack Isolation Response is better than the Attack Limiting Response.
- **Effect on the attacker (10)** - The Attack Isolation Response will be easily visible to the attacker, because he/she will receive the redirect messages from the first server. The Attack Limiting Response may be a bit harder for the attacker to grasp, because he/she

will only see a number of his/her requests being dropped, which remains consistent over time. So in this category the Attack Limiting Response is better than the Attack Isolation Response.

- **Overall bandwidth consumption (5)** - The Attack Isolation Response will even increase the overall bandwidth consumption, because additional redirect messages will be sent. And the Attack Limiting Response will indeed lower the amount of traffic consumed by the attack. Therefore in this category the Attack Limiting Response is better than the Attack Isolation Response.
- **Price (25)** - The Attack Isolation Response sets up new virtual server, which costs money. So in the final category the Attack Limiting Response is better than the Attack Isolation Response.

Table 1: Response gradings per comparison parameter

	Attack Isolation Response	**Attack Limiting Response**
Availability of file-server (5)	0%	40%
Availability of web-server (20)	95%	50%
Effect on the attacker (10)	40%	50%
Overall bandwidth consumption (5)	10%	60%
Price (25)	70%	100%

Using the defined grades and their weights, the ratings for the two responses can now be calculated using the following formula:

$$Rating = Weight_1 \times Grade_1 + Weight_2 \times Grade_2 + ... + Weight_n \times Grade_n \tag{1}$$

The ratings are calculated as follows:

Attack Isolation Response

$$5 \times 0 + 20 \times 0.95 + 10 \times 0.4 + 5 \times 0.1 + 25 \times 0.7 = 19 + 4 + 0.5 + 17.5 = 41 \tag{2}$$

Attack Limiting Response

$$5 \times 0.4 + 20 \times 0.5 + 10 \times 0.5 + 5 \times 0.6 + 25 \times 1 = 2 + 10 + 5 + 3 + 25 = 45 \tag{3}$$

The Attack Limiting Response gets a rating of 45 which is higher than the rating of 41 for the Attack Isolation Response. Therefore according to the policies that were defined, the Attack Limiting Response will be the better choice for this company.

7 Conclusion

From the results of this research it can be concluded that the concept of Security Autonomous Response Networks work and can be implemented.

When implementing a Security Autonomous Response Network there are a few important things that need to be kept in mind. The main reason for migrating to Security as a Service is because it has a bigger potential for dealing with security threats than having people responsible for that. In order to explore that potential, the implementation of a Security Autonomous Response Network has to be flexible and modular. Modularity can be achieved by having pluggable IDSs which can combine technologies for fast and efficient detection of security problems. Another important part that needs to be pluggable are the responses which are triggered as a result of detecting an issue. That way various kinds of standard or innovative countermeasures could be added to the system and waiting to be executed. And this means that knowledge can be reused and exchanged between companies or other organizations in the form of software modules. Since every countermeasure is only effective against a given set of security threats, the responses need to be classified based on what types of problems they can solve. The Security Control Loop could be implemented inside an actual loop statement which iterates until it is stopped, however this will make the implementation very inflexible and also not very scalable. A better way to implement the control loop would be to use an Enterprise Service Bus architecture. This will allow for multiple IDSs to monitor the infrastructure at the same time and also response selection and execution agents to agree on solutions and fix different problem in different parts of the network simultaneously.

When deploying a Security Autonomous Response Network in a company, there are also some important considerations. The infrastructure of the company should be designed modular whenever this is possible. That will make it more flexible and allow for a larger amount of different countermeasures to be triggered on separate parts of the whole system. When selecting and adding different responses to the countermeasure weaponry of the Security Autonomous Response Network, the company has to define policies according to which the responses should be rated. Based on the classification and the rating the most suitable response can be selected for a particular security problem that occurs.

References

[1] O. Fundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, 2012. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Software-Defined+Networking+:+The+New+Norm+for+Networks#0

[2] Michael Patterson, "Detecting and Mitigating Network Security Threats," *RTC Magazine*, 2013. [Online]. Available: http://www.rtcmagazine.com/articles/view/102912

[3] Tzeyoung Max Wu, "Intrusion Detection Systems," IATAC, Tech. Rep., 2009. [Online]. Available: http://iac.dtic.mil/csiac/download/intrusion_detection.pdf

[4] D. Bolzoni, *Revisiting anomaly-based network intrusion detection systems*, Enschede, The Netherlands, Jun. 2009. [Online]. Available: http://purl.org/utwente/doi/10.3990/1.9789036528535

[5] N. Mckeown, T. Anderson, L. Peterson, J. Rexford, S. Shenker, and S. Louis, "OpenFlow : Enabling Innovation in Campus Networks," 2008.

[6] Mininet Team, "Mininet Overview," 2014. [Online]. Available: http://mininet.org/overview/

[7] D. Leonard, "pktstat(1) - Linux man page," 2002. [Online]. Available: http://linux.die.net/man/1/pktstat

[8] D. Wieers, "Dstat: Versatile resource statistics tool," 2012. [Online]. Available: http://dag.wiee.rs/home-made/dstat/

[9] C. Yalcin, "Hammer DDos Script - Python 3," 2014. [Online]. Available: https://github.com/cyweb/hammer

[10] J. . R. Kinsella, "Slowloris HTTP DoS," 2013. [Online]. Available: http://ha.ckers.org/slowloris/

[11] Apache Software Foundation, "ab - Apache HTTP server benchmarking tool," 2014. [Online]. Available: http://httpd.apache.org/docs/current/programs/ab.html

[12] A. Manes, "Enterprise Service Bus: A Definition," *Burton Group*, pp. 1–35, 2007. [Online]. Available: http://i.i.cbsi.com/cnwk.1d/html/itp/burton_ESB.pdf

[13] The Bro Project, "Bro Manual," 2014. [Online]. Available: http://www.bro.org/sphinx/index.html

[14] The Snort Project, "SNORT Users Manual 2.9.6," 2014. [Online]. Available: http://s3.amazonaws.com/snort-org-site/production/document_files/files/000/000/001/original/snort_manual.pdf

[15] Real Status, "HyperGlance for SDN," 2014. [Online]. Available: http://real-status.com/product/sdn

[16] A Linux Foundation Collaborative Project, "OpenDaylight," 2014. [Online]. Available: http://www.opendaylight.org/

[17] Hewlett-Packard Development Company, "HP Virtual Application Networks SDN Controller," 2013. [Online]. Available: http://h17007.www1.hp.com/docs/networking/solutions/sdn/4AA4-8807ENW.PDF

A Python source code for the attack isolation response implementation of the control loop

control_loop-response_isolation.py

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import CPULimitedHost
from mininet.link import TCLink
import time
import os

# Define global variables
n = 4 # number of nodes
ncon = 0
fileserverpid = None
webserverpid = None
results = None
dos = False
counter = 0
hosts = {}
hostips = {}
hostints = {}
switchints = {}
accesslinkopts = dict(bw=10, delay='3ms', use_htb=True)
distrlinkopts = dict(bw=100, delay='1.5ms', use_htb=True)
corelinkopts = dict(bw=1000, delay='0.5ms', use_htb=True)


class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        h1 = self.addHost( 'h1', cpu=.5/n )
        h2 = self.addHost( 'h2', cpu=.5/n )
        h3 = self.addHost( 'h3', cpu=.5/n )
        h4 = self.addHost( 'h4', cpu=.5/n )
        s1 = self.addSwitch( 's1' )
```

```
        # Add links
        self.addLink( h1, s1, **accesslinkopts )
        self.addLink( h2, s1, **accesslinkopts )
        self.addLink( h3, s1, **accesslinkopts )
        self.addLink( h4, s1, **accesslinkopts )


def netSetUpAndTest():
    # Create Network
    topo = MyTopo()
    net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink)
    net.start()

    # Test Network
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    print "Testing bandwidth between h1 and h4"
    h1, h4 = net.get('h1', 'h4')
    net.iperf((h1, h4))

    # Start services on h1
    print h1.cmd( 'cd ~/fileserver/')
    print h1.cmd( 'python -m SimpleHTTPServer 8000 > /dev/null 2>&1 &')
    global fileserverpid
    fileserverpid = int(h1.cmd( 'echo $!'))
    print "Fileserver is running now on port 8000 with PID:", fileserverpid


    print h1.cmd( 'cd ~/webserver/')
    print h1.cmd( 'python -m SimpleHTTPServer 8001 > /dev/null 2>&1 &')
    global webserverpid
    webserverpid = int(h1.cmd( 'echo $!'))
    print "Webserver is running now on port 8001 with PID:", webserverpid
    time.sleep(2)

    h2 = net.get('h2')
    # Test services on h1
    print h2.cmd( 'cd ~')
    print "Services can be tested using the following commands:"
    print "h2 wget http://%s:8000/test_10K.img" % (h1.IP())
    print "h2 time curl http://%s:8001/index.html" % (h1.IP())
    CLI( net )
    return net


def tcpStats(h1):
    print "Collecting TCP connections statistics..."
    # Detecting attack
    h1.cmd( "pktstat -1 -ntT -i h1-eth0 -w 5 > /tmp/h1_bandwidth.out &" )
    time.sleep(8)
    while os.stat('/tmp/h1_bandwidth.out').st_size<=2 :
        print "Wating for connection statistics..."
        h1.cmd( "pktstat -1 -ntT -i h1-eth0 -w 5 > /tmp/h1_bandwidth.out &" )
```

```
        time.sleep(8)

    result = open('/tmp/h1_bandwidth.out')
    global results
    results = result.readlines()
    result.close()
    global ncon
    ncon = int(results[0].split()[0])
    print "Number of connections:", ncon


def detAttackVectors(h1):
    print "Determining potential attack vectors..."
    attsrcip = ""
    attdstipport = ""
    attsrcips = {}
    portconns = 0
    if ncon > 30 and counter == 0 :
        for i in range(1, (ncon+1)):
            if results[i].split()[2] == "tcp" :
                attdstipport = results[i].split()[3]
                attsrcip = results[i].split()[5].split(":")[0]
                if attdstipport == "10.0.0.1:8000" :
                    if attsrcips.has_key(attsrcip) :
                        attsrcips[attsrcip] += 1
                    else:
                        attsrcips[attsrcip] = 1
                    portconns += 1
        print "There are currently %s active connections to port 8000!" % portconns
        print "Sources:", attsrcips
        asi = attsrcips.keys()
        attsrcip = asi[0]
        for i in range(1, len(asi)):
            if attsrcips[asi[i]] > attsrcips[attsrcip]:
                attsrcip = asi[i]
        global dos
        dos = True
        print """The IP %s is a potential attacker! It currently has %s active
connections to port 8000.""" % (attsrcip, attsrcips[attsrcip])
    else :
        print "There are no potential attack vectors found."


def isolateResponse(net):
    h1 = net.get('h1')
    h2 = net.get('h2')
    s1 = net.get('s1')

    # Define attributes
    global counter
    global hosts
    global hostips
    global hostints
    global switchintss
    counter +=1
    print "Counter:", counter
```

```
        hosts[counter] = "nh%s" % counter
        print "Host:", hosts[counter]
        hostips[hosts[counter]] = "10.0.0.%s" % (n+counter)
        print "IP:", hostips[hosts[counter]]
        hostints[hosts[counter]] = "%s-eth0" % hosts[counter]
        print "Host interface:", hostints[hosts[counter]]
        switchints[hosts[counter]] = "s1-eth%s" % (n+counter)
        print "Switch interface", switchints[hosts[counter]]

        # Create new host and redirect the old one
        print h1.cmd( "kill -9", fileserverpid)
        h = net.addHost( hosts[counter] , cpu=1/8 )
        time.sleep(2)
        net.addLink( h, s1, **distrlinkopts )
        s1.attach(switchints[hosts[counter]])
        print h.cmd( "ifconfig", hostints[hosts[counter]],
                    hostips[hosts[counter]] )
        print "Redirecting now..."
        print h1.cmd( "~/mininet/examples/redirect.py %s &" %
                     hostips[hosts[counter]] )
        print "Redirected!"
        print h.cmd( 'cd ~/fileserver/')
        print h.cmd( 'python -m SimpleHTTPServer 8000 > /dev/null 2>&1 &')
        #Test the newly created host
        print h2.cmd( 'cd ~')
        print "h2 wget http://%s:8000/test_10K.img" % (h1.IP())
        print "h2 time curl http://%s:8001/index.html" % (h1.IP())

    def controlLoop(net):
        h1 = net.get('h1')
        global dos
        loopcounter=0
        # The control loop starts here
        while True:
            loopcounter +=1
            print "=================================================="
            print "Starting loop %s..." % loopcounter
            print "--------------------------------------------------"
            dos = False
            #check for attacks
            tcpStats(h1)
            print "--------------------------------------------------"
            detAttackVectors(h1)
            print "--------------------------------------------------"
            # Trigger response if attack is found
            if dos == True :
                isolateResponse(net)
            print "--------------------------------------------------"
            print "End of loop %s..." % loopcounter
        print h.cmd( 'kill %python')
        net.stop()

    if __name__ == '__main__':
        # Tell Mininet to print useful information
        setLogLevel('info')
        net = netSetUpAndTest()
```

```
    controlLoop(net)
```

redirect.py

```
#!/usr/bin/python

import SimpleHTTPServer
import SocketServer
import sys

PORT = 8000

class myHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def __init__(self, req, client_addr, server):
        SimpleHTTPServer.SimpleHTTPRequestHandler.__init__(self, req,
                client_addr, server)

    def do_GET(self):
        print self.path
        self.send_response(301)
        new_path = "http://%s:8000%s" % (str(sys.argv[1]), self.path)
        self.send_header('Location', new_path)
        self.end_headers()

class MyTCPServer(SocketServer.ThreadingTCPServer):
    allow_reuse_address = True

if __name__ == '__main__':
    handler = MyTCPServer(("", PORT), myHandler)
    handler.allow_reuse_address = True
    print "serving at port", PORT
    handler.serve_forever()
```

B Python source code for the attack limiting response implementation of the control loop

control_loop-response_limiting.py

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.cli import CLI
from mininet.node import CPULimitedHost
from mininet.link import TCLink
import time
import os
import csv

# Define global parameters
n = 4 # number of nodes
ncon = 0
deliprule1 = None
deliprule2 = None
deliprule3 = None
attacked = "h1"
port = 8000
protocol = "tcp"
seconds = 1
linkspeed = 1000000
hitcount = 5
results = None
dos = None
max_bandwidth = None
accesslinkopts = dict(bw=10, delay='3ms', use_htb=True)
distrlinkopts = dict(bw=100, delay='1.5ms', use_htb=True)
corelinkopts = dict(bw=1000, delay='0.5ms', use_htb=True)


class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        h1 = self.addHost( 'h1', cpu=.5/n )
```

```
        h2 = self.addHost( 'h2', cpu=.5/n )
        h3 = self.addHost( 'h3', cpu=.5/n )
        h4 = self.addHost( 'h4', cpu=.5/n )
        s1 = self.addSwitch( 's1' )

        # Add links
        self.addLink( h1, s1, **accesslinkopts )
        self.addLink( h2, s1, **accesslinkopts )
        self.addLink( h3, s1, **accesslinkopts )
        self.addLink( h4, s1, **accesslinkopts )


def netSetUpAndTest():
    # Create Network
    topo = MyTopo()
    net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink)
    net.start()

    # Test Network
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    print "Testing bandwidth between h1 and h4"
    h1, h4 = net.get('h1', 'h4')
    net.iperf((h1, h4))

    # Start services on h1
    print h1.cmd( 'cd ~/fileserver/')
    print h1.cmd( 'python -m SimpleHTTPServer 8000 > /dev/null 2>&1 &')
    time.sleep(2)
    h2 = net.get('h2')
    # Test services on h1
    print h2.cmd( 'cd ~')
    print "Fileserver can be tested using the following command:"
    print "h2 wget http://%s:8000/test_10M.img" % (h1.IP())
    CLI( net )
    return net


def tcpStats(h1):
    print "Collecting TCP connections statistics..."
    # Detecting attack
    h1.cmd( "pktstat -1 -ntT -i h1-eth0 -w 5 > /tmp/h1_bandwidth.out &" )
    time.sleep(8)
    while os.stat('/tmp/h1_bandwidth.out').st_size<=2 :
        print "Waiting for connection statistics..."
        h1.cmd( "pktstat -1 -ntT -i h1-eth0 -w 5 > /tmp/h1_bandwidth.out &" )
        time.sleep(8)

    result = open('/tmp/h1_bandwidth.out')
    global results
    results = result.readlines()
    result.close()
    global ncon
    ncon = int(results[0].split()[0])
```

```
    print "Number of connections:", ncon


def detAttackVectors(h1):
    print "Determining potential attack vectors..."
    attsrcip = ""
    attdstipport = ""
    attsrcips = {}
    attdstipports = {}
    if ncon > 10 :
        for i in range(1, (ncon+1)):
            if results[i].split()[2] == "tcp" :
                attdstipport = results[i].split()[3]
                attsrcip = results[i].split()[5].split(":")[0]
                if attsrcips.has_key(attsrcip):
                    attsrcips[attsrcip] += 1
                else:
                    attsrcips[attsrcip] = 1
                if attdstipports.has_key(attdstipport):
                    attdstipports[attdstipport] += 1
                else:
                    attdstipports[attdstipport] = 1
        print "Destinations:", attdstipports
        print "Sources:", attsrcips
        asi = attsrcips.keys()
        attsrcip = asi[0]
        for i in range(1, len(asi)):
            if attsrcips[asi[i]] > attsrcips[attsrcip]:
                attsrcip = asi[i]
        adip = attdstipports.keys()
        attdstipport = adip[0]
        for i in range(1, len(adip)):
            if attdstipports[adip[i]] > attdstipports[attdstipport]:
                attdstipport = adip[i]
        print "Most used attacker IP is %s, used %s times." % (attsrcip,
                    attsrcips[attsrcip])
        print "Most used attacked IP and port are %s, used %s times." % (attdstipport,
                     attdstipports[attdstipport])
        print "-----------------------------------------------"
        banStats(h1)
    else :
        print "There are no potential attack vectors found."


def banStats(h1):
    print "Collecting bandwidth statistics..."
    h1.cmd( "rm /tmp/band_h1.csv" )
    h1.cmd( "dstat -nt --nocolor --output /tmp/band_h1.csv &" )
    time.sleep(8)
    h1.cmd( "kill %dstat" )
    h1.cmd( "sed -i '1,6d' /tmp/band_h1.csv" )
    global max_bandwidth
    max_bandwidth = None
    band = csv.DictReader(open("/tmp/band_h1.csv"))
    for row in band:
        bandwidth = int(row["send"].split(".")[0])
```

```
            if max_bandwidth == None or max_bandwidth < bandwidth:
               max_bandwidth = bandwidth

        if max_bandwidth != None:
            print "Max bandwidth consumed by the attack:", max_bandwidth
        else:
            print "Error (No bandwidth statistics saved.)"
        global dos
        dos = True


def limitResponse(net):
        print "Adjusting rate limits..."
        h = net.get( attacked)
        global deliprule1
        global deliprule2
        global deliprule3
        global hitcount
        change = True
        if deliprule1 == None and deliprule2 == None and deliprule3 == None:
            # Add the IP to the list:
            print h.cmd ( """iptables -A INPUT -p %s --destination-port %s --syn
-m state --state NEW -m recent --set --name RATELIMITED""" % (protocol, port))
            deliprule1 = """iptables -D INPUT -p %s --destination-port %s --syn
-m state --state NEW -m recent --set --name RATELIMITED""" % (protocol, port)
            # Drop if exceeded limit:
            print h.cmd ( """iptables -A INPUT -p %s --destination-port %s --syn
-m state --state NEW -m recent --update --seconds %s --hitcount %s --rttl
--name RATELIMITED -j DROP""" % (protocol, port, seconds, hitcount))
            deliprule2 = """iptables -D INPUT -p %s --destination-port %s --syn
-m state --state NEW -m recent --update --seconds %s --hitcount %s --rttl
--name RATELIMITED -j DROP""" % (protocol, port, seconds, hitcount)
            # Accept if inside limit:
            print h.cmd ( """iptables -A INPUT -p %s --destination-port %s --syn
-m state --state NEW -j ACCEPT""" % (protocol, port))
            deliprule3 = """iptables -D INPUT -p %s --destination-port %s --syn
-m state --state NEW -j ACCEPT""" % (protocol, port)
            print "Rate limit is now set to %s connections per %s second!" % (hitcount,
                        seconds)
        else :
            # Calculating new hitcount value
            if max_bandwidth > (linkspeed/10)*6 :
                hitcount -= 1
            elif max_bandwidth < (linkspeed/10)*4 :
                hitcount += 1
            else :
                change = False
            if change :
                print h.cmd ( deliprule1 )
                print h.cmd ( deliprule2 )
                print h.cmd ( deliprule3 )
                 # Add the IP to the list:
                print h.cmd ( """iptables -A INPUT -p %s --destination-port %s --syn
-m state --state NEW -m recent --set --name RATELIMITED""" % (protocol, port))
                deliprule1 = """iptables -D INPUT -p %s --destination-port %s --syn
-m state --state NEW -m recent --set --name RATELIMITED""" % (protocol, port)
```

```
            # Drop if exceeded limit:
            print h.cmd ( """iptables -A INPUT -p %s --destination-port %s --syn
-m state --state NEW -m recent --update --seconds %s --hitcount %s --rttl
--name RATELIMITED -j DROP""" % (protocol, port, seconds, hitcount))
            deliprule2 = """iptables -D INPUT -p %s --destination-port %s --syn
-m state --state NEW -m recent --update --seconds %s --hitcount %s --rttl
--name RATELIMITED -j DROP""" % (protocol, port, seconds, hitcount)
            # Accept if inside limit:
            print h.cmd ( """iptables -A INPUT -p %s --destination-port %s --syn
-m state --state NEW -j ACCEPT""" % (protocol, port))
            deliprule3 = """iptables -D INPUT -p %s --destination-port %s --syn
-m state --state NEW -j ACCEPT""" % (protocol, port)
            print "Rate limit is now set to %s connections per %s second!" % (hitcount,
                        seconds)


def controlLoop(net):
    h1 = net.get('h1')
    global dos
    loopcounter=0
    # The control loop starts here
    while True:
        loopcounter +=1
        print "=================================================="
        print "Starting loop %s..." % loopcounter
        print "--------------------------------------------------"
        time.sleep(2)
        dos = False
        #check for attacks
        tcpStats(h1)
        print "--------------------------------------------------"
        detAttackVectors(h1)
        print "--------------------------------------------------"
        # Trigger response if attack is found
        if dos == True :
            limitResponse(net)
        print "--------------------------------------------------"
        print "End of loop %s..." % loopcounter
    print h.cmd ( deliprule1 )
    print h.cmd ( deliprule2 )
    print h.cmd ( deliprule3 )
    net.stop()

if __name__ == '__main__':
    # Tell Mininet to print useful information
    setLogLevel('info')
    net = netSetUpAndTest()
    controlLoop(net)
```