University of Twente

Master Thesis

Flow-based DDoS Attack Detection on Cisco IOS

Daniël van der Steeg

Date:	August 27, 2014
Committee:	prof. dr. ir. Aiko Pras
	Rick Hofstede, M.Sc.
	dr. Anna Sperotto
Faculty:	Electrical Engineering, Mathematics and Computer Science (EWI)
Chair:	Design and Analysis of Communication Systems (DACS)

Contents

Chapter 1

Introduction

This thesis describes the process and result of the work done for my Master assignment. Usually, a Master assignment is concluded with a report which does not reach outside of the university. Instead, in agreement with my supervisors, we have decided to submit my work and results as a paper to the *14th IFIP/IEEE International Symposium on Integrated Network and Service Management* (IM'15). Although the form of the final document, which documents the work for the Master assignment, is unusual, the research does satisfy the requirements set by the faculty in [8], as I will elaborate on in this thesis. Furthermore, this document will briefly describe the subjects of my research.

1.1 Assignment

For this Master assignment, a paper has replaced the usual thesis, which must satisfy the requirements for a Master thesis. Therefore the following describes how the work done for the Master assignment fulfills the requirements.

Formulate a problem statement.
The formulation of a problem statement is usually the start of any research. In the case of this Master assignment this resulted in a set of three main research questions. The formulation of these questions was the result of the exploratory research during the Research Topics course, in preparation for the final assignment. The feasibility of the research questions was guarded by discussing them with the supervisors.

Identify relevant literature.
Both during the exploratory and final research a literature study was performed, to identify literature relevant to my research. The exploratory research was performed during Research Topics, in which an area for further research was explored through literature. During the survey

performed for the final assignment, I identified papers related to my research, as well as documentation and specifications of the relevant technologies. The literary results of the survey can be found in the final paper, and is embedded in the introduction section.

Draw up a work plan.

As the submission of the paper to IM has a concrete deadline, this consequently is the final deadline for the Master assignment. However, also including my personal preference and the availability of the supervisors, the deadline was moved slightly ahead. Based on this, a schedule was created for intermediate results. During the research, this work plan was altered multiple times, for example when our hardware to be used appeared to be malfunctioning, it took a while for the new hardware to arrive.

Adjust goals and approaches based on interim evaluations.

During the entire period of the Master assignment, the supervisor(s) and I regularly evaluated and discussed the intermediate results. Using these discussions and evaluations as a basis, the goals of the research were shifted when deemed necessary. For example, the focus of the research was shifted away from mitigation when the intermediate results showed mitigation would be impractical on the current hardware. Instead, a brief survey was done on possible solutions on newer hardware.

Analyse different possible solutions and motivate a choice between them.

Finding and analysing the different possible solutions was part of the literature study which was performed. Aside from using relevant literature, the more technical aspects also required studying technical documentations and specifications, consulting experts in the field (ICTS), and referring to technical mailing lists. Finally, also measurements were used to motivate choices between different solutions. During the entire research project, various discussions with the supervisors were held to ensure the analysis was sound and the choices made were viable.

The fulfillment of this requirement is also shown from the type of paper to be submitted to IM'15, which is an experience paper. This type of paper has a particular focus on different solutions and their practicability. More details will follow in Appendix A.

The ability to reflect on the problem, on the research/design approach, on the solution and on ones own performance.

The multiple brainstorm and discussion sessions with the supervisors, along with the evaluation of intermediate results, resulted in changes to the problem statement, the approach and the solution. This was a continuous process throughout the final project. Reflection on ones own performance requires input from others. For this I used the feedback

by the supervisors, especially regarding the more explicit tasks, such as writing and presenting. For the more technical parts of the reflection, feedback by experts was also used.

Demonstrate creativity and the ability to work independently.
The supervisors and I only defined the requirements for the prototype, such that I had the freedom to implement the algorithm as I saw fit. As there was no prior work to refer to, the implementation was build from the ground up, using the limited amount of information and tools available on the target platform. For most of the problems that arose during the final project, one or more solutions were provided by me. When more solutions were available and the problem was relatively big, they were discussed with the supervisor to ensure the most viable solution was chosen. Moreover, based on intermediate results and evaluation thereof, changes to the algorithm implementation were introduced for performance improvements. The ability to work independently has been demonstrated by progressing the work, both technically and in writing, during the time I worked remotely, or the daily supervisor was unavailable.

Communicate the research and design activities both written and in presentations.
This requirement is mostly satisfied by the final paper which is to be submitted to IM'15, for which the audience consists of people with different technical backgrounds. During the final project, a presentation was held to introduce the DACS chair to my research. The research is also communicated by the presentation at the end of this final project, and will be presented at IM'15.

1.2 Research Topics

During the final project multiple topics have been researched. The following is a list of these topics, with short descriptions and how they relate to each other and to my research.

NetFlow/IPFIX
Modern high-speed networks can feature high throughput, of many Gigabits per second [6]. Monitoring such networks requires technologies which can scale to keep up with the growing networks and increasing link rates. Technologies such as Cisco's NetFlow [1] and the recent standardization effort IPFIX [2], do not export every packet for analysis, but instead export flows. This aggregation, while there is some loss of information, allows network operators to monitor their high-speed networks in an efficient manner. Furthermore, these technologies are widely deployed and available on high-end networking devices.

In this research project, flow-based technologies are used to detect Distributed Denial of Service (DDoS) attacks. Specifically, data exported using Cisco's NetFlow is used on a high speed-packet forwarding device to perform this DDoS attack detection. Therefore, I have investigated how NetFlow and IPFIX works, and specifically how Cisco implements NetFlow.

Distributed Denial of Service attacks

With the growing use of the Internet, the misuse is growing as well. One attack against networks and services is the Distributed Denial of Service (DDoS) attack, which attempts to overload the target by sending a large amount of packets from different sources. Recently DDoS attacks have become more common and volumes have reached 400 Gbps, making it a serious threat [3, 7].

Many types of DDoS attacks exist. Due to the nature of DDoS attacks, most types generate a large number of flows. Consequently, by inspecting the NetFlow/IPFIX data, the attack can be detected. This is the premise of the research conducted in this final project. Using an existing algorithm from [5], flow-based DDoS attack detection has been implemented on the Cisco IOS platform.

Scripting on high-end packet forwarding devices

Many different high-end packet forwarding devices exists, most of which support some kind of command automation or scripting to assist the network administrator. To implement the detection algorithm presented in [5] on a packet forwarding device, scripting is a requirement for the target platform. I surveyed the scripting possibilities for switches from the following vendors:

- Brocade Communication Systems
- Cisco Systems
- Extreme Networks
- Dell
- Juniper Networks

From this selection, Cisco Systems and Juniper Networks appeared to offer the best scripting possibilities, as Juniper allows the use of a UNIX shell, while Cisco offers an extensive API from their TCL scripting environment. The other vendors either only provide command automation, without the possibility to react to event without the interference of an administrator, or the published API was limited in scope.

For the implementation of the DDoS attack detection, we have chosen Cisco's IOS, as one of their switches, the Catalyst 6500, is the most widely deployed switch [4]. Consequently, implementing the algorithm for this platform would cause it to be widely deployable. Therefore,

most of the practical research regarding scripting on high-end packet forwarding devices was focused on Cisco IOS.

1.3 Organization

The rest of this thesis is organized as follows. Chapter 2 discusses the deliverables for my Master assignment. In Chapter 3 the lessons I learned during the final project are discussed. The final paper to be submitted to IM'15 can be found in Appendix A.

Chapter 2

Deliverables

The deliverables for this Master assignment consist of more than this thesis, which are discussed in this chapter.

2.1 Paper

The most important deliverable of my Master assignment is the documentation of my research in the form of a paper. This paper will be submitted to the *14th IFIP/IEEE International Symposium on Integrated Network and Service Management* (IM'15). The paper itself and information regarding the conference can be found in Appendix A.

2.2 DDoS detection implementation

The research done for my Master assignment concerns the DDoS attack detection implementation for Cisco IOS. Validation has shown that the implementation can detect DDoS attacks in real-time, causing a constant load on the switch it is deployed on, which makes it interesting for deployment. By releasing the TCL scripts on GitHub[1] in an open way, we want to make sure the research is reproducible, as well as encourage further research and development. Such further research may include deployment of the scripts on the newer Supervisor Engine 2T, and implementing mitigation.

[1] https://github.com/ut-dacs/ios-ddos-detect

Chapter 3

Lessons Learned

This chapter contains some of the lessons learned during the final project, which do not directly relate to the research itself, as described in Section 1.2, but are of value nonetheless.

3.1 Cisco Catalyst 6500

During the research project, a lot of the time was spent on working with the high-end packet forwarding device used, the Cisco Catalyst 6500. Implementing the detection algorithm in TCL was a challenge in and of itself, but it surprised me how many options are available on such high-end networking devices. During this process, some interesting problems were encountered, such as reading wrong SNMP objects and some undocumented behaviour. Most of these problems could be easily overcome once the reason for the error was clear. The biggest lesson I learned from these issues, is making sure the code or command does what you expect it to do, as not all features are extensively documented. Aside from learning the scripting environment, I also learned many of the configuration and management options available on the switch I worked with, which may be useful sometime.

3.2 Writing

Documenting one's work is very important, because reproducibility is one of the main principles of science. Writing the paper for IM'15 proved to be quite a challenge, as it should be written in a very concise manner. At the start of writing, it was difficult to switch from writing report style to writing a scientific paper. During the writing process, I got lots of feedback which helped me to learn the proper style of writing. The feedback I learned most from addressed structure, both high level and low level. At the start of writing, I mostly approached writing in an ad-hoc manner, only using a

very global outline as guide. Based on the feedback, the use of bullet points to create structure, from high level to low level, became more common.

For a new paper, I would use this more structured approach from the beginning, to keep the text structured throughout the entire writing process. Furthermore, the use of language would likely be different to keep the text understandable. An example would be explaining something first and then mention it in context, instead of doing it the other way around. Finally, I would try to explain everything briefly, instead of assuming knowledge on the reader's part or relying on implicit reasoning.

3.3 OpenFlow

During the search for existing literature, one paper popped up that used OpenFlow to detect DDoS attacks. To properly understand the difference between NetFlow and IPFIX, and OpenFlow, I looked into what OpenFlow is and how is works. During the preparation of the validation setup, we had to strip the VLAN tags from the Ethernet frames and rewrite the destination MAC address. Without doing this, the Catalyst 6500 would not accept the Ethernet frames, as the frames were not addressed to it, but to the switch we received the duplicated traffic from. After looking into different solutions, only OpenFlow seemed to offer exactly what we needed. At first this was done in software, using Open vSwitch and later we switched to using a Pica8 switch. While I cannot say I have gained *a lot* of technical experience with OpenFlow, as I did with Cisco IOS, I did learn that it is a very useful new technology, and I will definitely be looking further into it.

Bibliography

[1] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), 2004. http://www.ietf.org/rfc/rfc3954.txt.

[2] B. Claise, B. Trammell, and P. Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011 (Internet Standard), September 2013. http://www.ietf.org/rfc/rfc7011.txt.

[3] CloudFlare, Inc. Technical Details Behind a 400Gbps NTP Amplification DDoS Attack. http://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack, 2014. Accessed on 2 July 2014.

[4] Follett, J.H. Cisco: Catalyst 6500 The Most Successful Switch Ever, 2006. Accessed on 2 July 2014.

[5] R. Hofstede, V. Bartos, A. Sperotto, and A. Pras. Towards Real-Time Intrusion Detection for NetFlow and IPFIX. In *Proceedings of the 9th International Conference on Network and Service Management, CNSM 2013, Zürich, Switserland*, pages 227–234, 2013.

[6] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials*, 2014.

[7] Prolexic Technologies. Prolexic Quarterly Global DDoS Attack Report Q1 2014, 2014.

[8] University of Twente. Master guide Computer Science. http://www.utwente.nl/ewi/onderwijs/voorzieningen/studiegidsen/2012-2013/programme_guide_master_csc.pdf.

Appendix A

IM 2015 Paper

This appendix contains the paper which is to be submitted to *14th IFIP/IEEE International Symposium on Integrated Network and Service Management (IM'15)*. This is the final document, which contains the work done and results of my research. The track chosen for this conference is Experience Session, as this research is slightly more practical than other research.

Name:	14th IFIP/IEEE International Symposium on Integrated Network and Service Management (IM'15)
Submission:	September 15 2014
Date:	May 11–15 2015
Venue:	OCC, Ottawa, Canada
Website:	http://im2015.ieee-im.org/

Real-time DDoS Attack Detection for Cisco IOS using NetFlow

Daniël van der Steeg, Rick Hofstede, Anna Sperotto and Aiko Pras
Design and Analysis of Communication Systems (DACS)
Centre for Telematics and Information Technology (CTIT)
University of Twente, Enschede, The Netherlands
d.p.m.h.vandersteeg@student.utwente.nl, {r.j.hofstede, a.sperotto, a.pras}@utwente.nl

***Abstract*—The first papers on the detection of DDoS attacks in a flow-based fashion have appeared already several years ago. The problem with these works is that they rely on the analysis of data by analysis applications that are typically installed on or close to a flow collector. This makes existing approaches far from real-time and susceptible to DDoS attacks themselves.**

In a previous work, we have shown how to perform DDoS attack detection on a flow exporter instead of a flow collector, i.e., close to the data source and in a real-time fashion, which however required access to fully-extendible flow monitoring infrastructure. In this work, we overcome this limitation by presenting a proof-of-concept of the first intrusion detection system for Cisco IOS that detects DDoS attacks in a real-time fashion. Our analysis shows that the detection of attacks is feasible in production networks, with detection delays in the order of 20 seconds. Also, we show that our prototype generates a constant load on the underlying platform, even under attacks, underlining that it can be deployed in production networks if enough spare capacity is available.

I. Introduction

Distributed denial of service (DDoS) attacks are becoming a major technical and economical threat, flooding networks and servers with large amounts of network traffic. In early 2014 CloudFlare was hit by an amplified UDP flood attack, reaching nearly 400 Gbps in bandwidth [1]. A flow is defined in [2] as "a set of packets passing an observation point in the network during a certain time interval, such that all packets belonging to a certain flow have a set of common properties". Examples of such common properties are the source and destination addresses and ports. Large numbers of flows are created by high volume DDoS attacks, making it possible for flow-based technologies to detect such volume-based attacks [3]. Moreover, the use of flow export technologies, such as NetFlow and the recent IETF standardization effort IPFIX, are especially useful since they generate traffic aggregates. This approach reduces the amount of data to be analyzed significantly [4], as well as the necessary processing power for export, collection and analysis. Furthermore, these technologies are widely available on packet forwarding devices, making the flow data easily accessible and the technologies easy to deploy in existing networks.

Flow-based intrusion detection in general – DDoS attack detection is no exception – is traditionally performed by *analysis applications* [5]–[7], as shown in Fig. 1. These applications work based on flow data exported by *flow exporters* and collected by *flow collectors*. Since the export of flow data is heavily based on timeouts and the collection is often designed to work in time intervals of several minutes, analysis applications are subject to various delays in the detection process [8]. Especially in the case of DDoS attack detection, where overload of network infrastructure can happen very quickly, this is something that must be avoided.

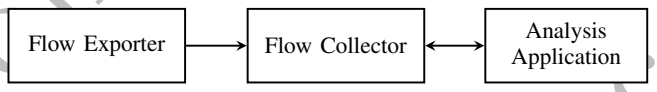

Fig. 1. Typical flow monitoring architecture.

Recent work has shown that moving detection closer to the data source decreases detection delays significantly, from at least 165 seconds to 10 seconds [9]. The presented DDoS attack detection algorithm runs on a platform targeted at passive data export based on flows, namely INVEA-TECHs FlowMon platform. The goal of this paper is to investigate whether the detection algorithm presented in [9] can be deployed on a widely available platform. In this context, we target Cisco's IOS platform and in particular the Cisco Catalyst 6500, which is one of the most widely deployed packet forwarding devices [10]. We focus in particular on the operational experience of performing intrusion detection on a packet forwarding device in a production network.

The remainder of this paper is structured as follows. Section II introduces NetFlow/IPFIX terminology which is used in this paper. An overview of the original detection algorithm from [9] is given in Section III. In Section IV, we present an approximation of the metric for the detection algorithm. The implemented prototype is discussed Section V, which will be used for the validation presented in Section VI. In Section VII, we elaborate on further possibilities for DDoS attack detection and mitigation in Cisco IOS. Finally, we close this paper in Section VIII where we draw our conclusions.

II. Flow Metering & Export

In this section, we briefly introduce the terminology related to flow metering and export, which will be used throughout this paper.

Flow metering and export are the two tasks performed by a *flow exporter* [4], as shown in Fig. 1. Packets in the network

confidential

are aggregated into flows by the Metering Process. When a new flow is observed, an entry for this flow is created in the *flow cache*. This cache is a table in which information is stored regarding all active flows in the network [4]. Aside from the key of the flow, i.e., the fields that identify a flow, some extra information is accounted, such as the number of packets and bytes in the flow. When a flow cache entry expires, for example when the flow has been active or inactive for too long or because of resource constraints, a flow record is *exported*, i.e., it is inserted in a NetFlow or IPFIX Message and sent to a collector for storage and analysis. If the cache is full and a flow cache entry cannot be created, it is considered a *flow learn failure* [11]. This can happen during periods of very high traffic if the flow cache is under-dimensioned, for example.

III. Detection Algorithm

We use an existing algorithm that has proven to satisfy the requirements of being lightweight, accurate and real-time in the context of DDoS attack detection, described in [9]. The algorithm[1] runs on a fixed time interval and measures the number of flow cache entry creations, as this metric was shown to be most usable of the four metrics presented in [9]. Based on this measurement a forecast is made for the measurement value of the next interval. In case the number of flow cache entry creations is too high in comparison with the past measurement values, the measurement sample is considered to be anomalous. However, because Internet traffic shows strong diurnal patterns, such as strong increases and decreases in the number of flow cache entry creations during the start and end of a working day respectively, the algorithm also learns the normal behaviour of the network over a 24 hour period. The forecasted value is therefore defined as:

$$\hat{x}_{t+1} \quad = \quad b_t + s_t \ , \tag{1}$$

where $\hat{x}_{t+1}$ is the forecasted value for the next interval, b_t is the base component, sometimes also referred to as permanent component, which represents the trend of the Internet traffic, and s_t is the seasonal component which represents the diurnal patterns.

Several enhancements to this algorithm are discussed in [9]. First, in order to decrease memory usage, the values used for retaining seasonal patterns, s_t, are stored per hour and are interpolated to estimate the value for a given time. Second, to prevent the algorithm from learning malicious traffic patterns, values such as s_t and b_t are discarded during an attack. Last, since the traffic patterns during the weekend usually differ from the patterns during weekdays, distinction is made between weekend and weekdays for season memory. This results in two training periods, one for weekdays and one for weekends.

IV. Monitoring Information Available in IOS

The detection algorithm considered in this paper, which has been summarized in Section III, is heavily based on a single metric, namely the number of flow cache entry creations per time interval. This metric is easily accessible on the flow monitoring platform used in the original work of the prototype ([9]), INVEA-TECH's FlowMon. Since that platform has been designed with extendibility in mind, this information is directly available from the platform's API. However, the amount of information available in IOS strongly depends on the path the packet or flow has taken through the router or switch. More precisely, packets are switched either in hardware or in software, although most packets are hardware-switched. On the campus network of the University of Twente (UT), for example, 99.6% of the traffic is hardware-switched [11]. Situations that trigger a packet to be switched in software are fragmented packets and packets that require ARP resolution [12], for example. For flows handled in hardware, information on the number of flow cache entry creations is not directly available. To approximate this metric, we use the following information available from the flow metering and exporting process:

- Number of *flow cache entries* (F_c).
- Number of *exported software flow records* (F_e).
- Number of *flow learn failures* (F_f). Accounts for the number of packets that could not be mapped to a flow cache entry, due to an under-dimensioned flow cache [11].

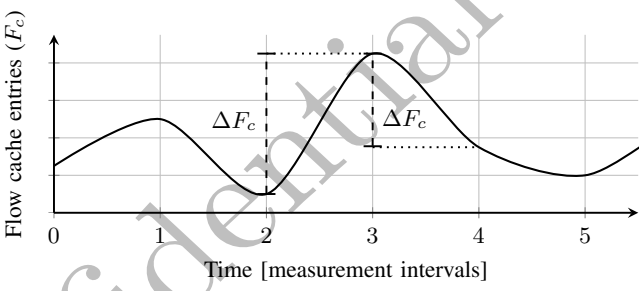

Fig. 2. Measuring flow cache entry creations in Cisco IOS.

The number of flow cache entry creations since the last measurement can be approximated using the following definition:

$$F = \Delta F_c + \Delta F_e + \frac{\Delta F_f}{c_f} \tag{2}$$

When flow cache entries are exported, F_c will decrease which will cause the approximation to be less accurate if the measurement intervals are too long. For example, in Fig. 2, if the measurement were to cover two intervals, from $t = 2$ to $t = 4$, the ΔF_c will not consider the peak at $t = 3$. By polling F_c more frequently, we can observe the changes more accurately, such that we observe the positive ΔF_c at $t = 3$ and the negative ΔF_c at $t = 4$, which is caused by exports. Then, if ΔF_c is negative, instead we use an estimation of previous ΔF_c values. Since the number of entries in the flow cache (F_c) only regards hardware-switched flows, we also add the number of exported software-switched flows (F_e), which can be easily obtained from IOS. Finally, by adding F_f it is possible to also regard flows that should have been created but were not, which

[1] In [9] two algorithms are described. We use Algorithm 2, which showed the best results.

confidential

is especially the case during a high intensity DDoS attacks. However, F_f counts every packet for which it failed to learn the flow as one *flow learn failure*. To compensate for the fact that F_f is expressed in packets while the other metrics are expressed in flows, we divide F_f by the average of the number of packets per flow, represented by c_f in Equation 2.

V. Implementation

The Embedded Event Manager (EEM) – part of Cisco's IOS that handles real-time network event detection – allows for the definition of *policies*, which can be used to execute an applet or script when certain events are triggered. For example, emails can be sent to network administrators when round-trip times reach a certain limit, or when network route changes occur. Another event type is based on time. This event can, among others, be scheduled at fixed time intervals. In this work, we use two time-based policies, implemented as TCL scripts:[2]

- **Measurement policy** – Determines the first component for our approximation of the flow-based metric: the number of flow cache entries (F_c), as described in Section IV.
- **Detection policy** – Retrieves the remaining components: the number of exported software flows (F_e) and the number of flow learn failures (F_f). Also, it implements the actual DDoS attack detection algorithm.

To obtain all three components, which are all made available using the SNMP protocol, we use a feature of the EEM environment that provides access to local SNMP objects. The reason for splitting the measurement policy from the detection policy is that we require a higher resolution for the former to detect changes more accurately, as described in Section IV.

Policy invocations are memoryless, and since we want to share data – both between policy runs and between policies – a method for sharing data needs to be implemented. Due to the fact that the filesystem is flash-based, we generally want to avoid excessive write actions that will shorten the memory's lifespan. The EEM environment therefore offers a *Context library* for this purpose; it allows for saving TCL variables to memory instead of writing them to disk. Besides for keeping track of our data between policy runs, we also use this feature to exchange information between the two policies, as the result of the measurement policy is needed by the detection policy.

The two policies discussed before are executed by the EEM at their respective intervals, which have been selected based on the runtime of the respective policies. When the switch is however under heavy load, its higher CPU utilization will cause the policies to take longer to execute. To avoid the policies from skipping an execution when the runtime of the policy exceeds the length of the interval, the prototype utilizes a feature from the EEM that can set a maximum policy runtime. If this runtime is exceeded, the policy terminates forcibly and data is lost. In the case of the detection policy, the algorithm has to start again from the learning phase as all state data is lost. If the measurement policy terminates prematurely, the measured number of created flow cache entries will be lower, as it missed a measurement, which will slightly impact the accuracy of the algorithm. To prevent the detection policy from being killed, a margin has been added to the interval which allows it to run longer if necessary, but never longer than the interval at which it is executed. The average runtime of the detection policy is 2–3 seconds under normal conditions, and has shown to reach 7–8 seconds under stress. Therefore, the final interval chosen for the detection policy is 10 seconds. For the measurement policy, measurements have shown that 2 seconds is provides an optimal balance between detailed measurements and loss of data due to termination.

[2]The open-source TCL scripts can be retrieved from https://github.com/ut-dacs/ios-ddos-detect/

VI. Validation

In this section, we describe the validation of this work, starting by identifying the requirements in Section VI-A. Next, we give a description of the validation setup in Section VI-B as well as specifics regarding the deployment, after which we discuss the results in Section VI-C.

A. Requirements

Three requirements were defined for the original algorithm: 1) it should be lightweight in terms of CPU and memory utilization, 2) the accuracy should be high enough to ascertain a low number of false positives/negatives, and the detection delay should be reduced to roughly 10% of conventional intrusion detection approaches [9]. However, since the the Cisco Catalyst 6500 is a high-speed packet forwarding device that has not designed for performing intrusion detection tasks, special care must be taken to not overload the device and possibly interrupt forwarding activities. We therefore relax the real-time requirement to detection within 30 seconds, while the CPU and memory utilization must be 10% or lower. Since the accuracy of the algorithm has already been validated in [9] and because it is invariant to the underlying implementation platform, we discuss the accuracy requirement only briefly.

B. Setup & Deployment

The implementation described in Section V has been developed on a *Cisco Catalyst 6500* with the *Supervisor Engine 720*. We have used this in combination with the *WS-X6708-10G-3C* line card for 10 Gbps Ethernet connectivity. The traffic used for validation comes is mirrored from the uplink of the UT campus network to the Dutch National Research and Education Network SURFnet. This traffic consists both of educational traffic, i.e. traffic generated by faculties and students, and traffic of campus residences. The link has a wire-speed of 10 Gbps with an average throughput of 1.8 Gbps during working hours. Furthermore, the flow data is exported to a flow collector, such that attacks detected by the prototype can be validated manually.

The network traffic used in [9] differs from the network traffic used in this work, both from its nature (backbone traffic vs. campus traffic) and volume. It is therefore clear that we have to adjust the parameters of the detection algorithm to achieve a similar accuracy as in [9]. As such, we have selected

confidential

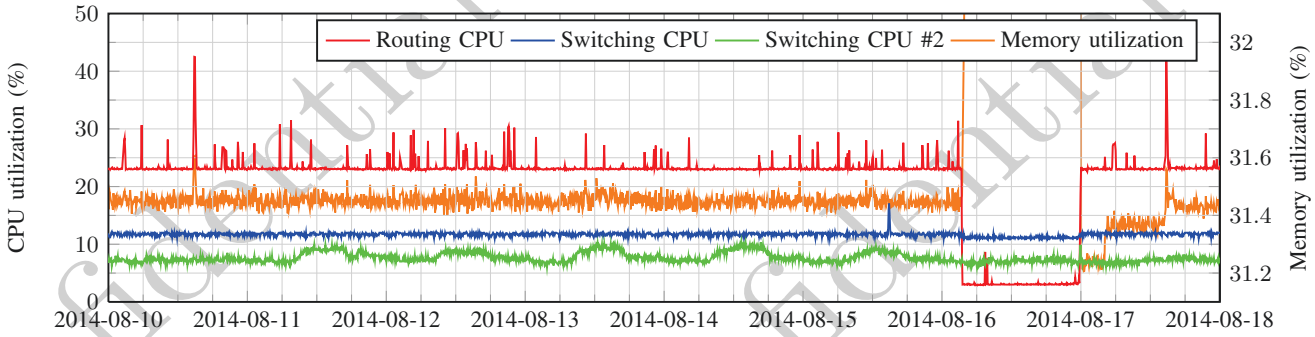

Fig. 3. Load of the Cisco Catalyst 6500 over time.

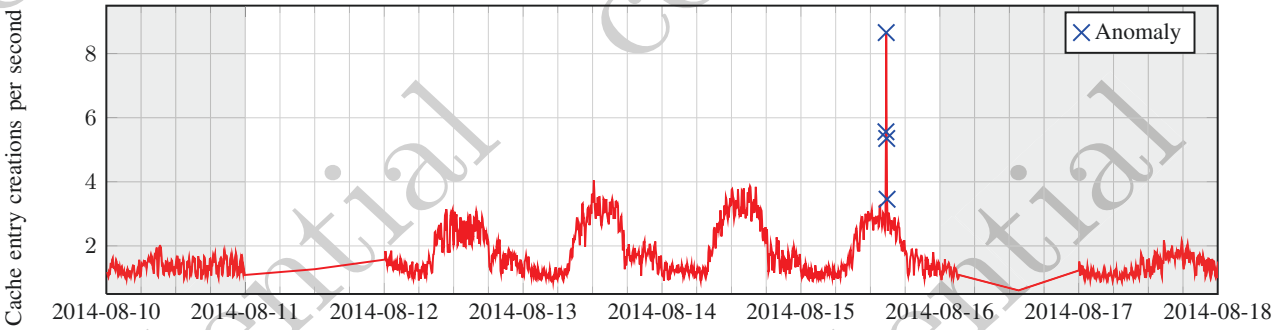

Fig. 4. Flow cache entry creations per second (averaged per 5 minutes), as processed by the detection algorithm over time.

the optimal parameters[3] for our observation point. For the parameter c_f used in our approximation of the number of flow cache entry creations, as described in Section IV, we obtain $c_f = 59.8133$ packets per flow on average.

C. Results

The most important requirement to be validated in this work is that the implementation must be light-weight, such that the implementation does not interfere with the primary activities of the packet forwarding device, namely routing and switching. We measure the resource consumption both in terms of CPU and memory utilization. In Fig. 3, the CPU load of the device is shown together with the memory utilization, averaged over 150 seconds. Using SNMP, the load of the CPU is measured for three components, namely the *routing CPU*, which handles L3 traffic, and two *switching CPUs*, which process traffic at L2. Once a routing or switching decision has been made by the CPU, hardware handles subsequent packets if possible. Furthermore, the *routing CPU* also handles the network management – including the EEM –, as most of this is done on L3. Consequently, our EEM policies also run on the *routing CPU*, and as such any load caused by our policies should account to the load of the *routing CPU*. During the period from August 16 3:00 to August 17, the measurement policies were not running due to a malfunctioning in the EEM. This caused the CPU utilization to drop, and the memory utilization to peak due to a memory leak.

In Fig. 3, the policies are active during the entire measurement period. Because the CPU utilization of most individual processes is reported as 0–1% and only peaks are reported as more than 1%, we only consider the overal CPU usage. Consequently, the overhead of managing and executing only the policies cannot be observed. This overhead is caused by processes such as the *Chunk Manager*, which handles memory allocation, *EEM Server*, which manages all EEM policies and applets, and *SNMP ENGINE*, which handles all SNMP requests. Because the overhead of operating our policies is caused by multiple processes, which also run when our implemented policies are disabled, we have measured the difference in CPU and memory utilization between operation with and without our policies. To measure this, the switch has been rebooted to clear all memory and CPU utilization. During this measurement we have observed a load on the *routing CPU* of 4%, combined with a memory utilization of 31.3%. Next, we have enabled our policies and measured the CPU and memory utilization again, observing an increase of 19% in CPU utilization, and an increase of 0.13% in memory utilization. This accounts for the average constant load added by our implementation.

To validate the behaviour during an attack and because of

[3]The parameters used in this work are: $c_{threshold} = 3.0$, $M_{min} = 7000$, $c_{cusum} = 4.0$, $\alpha = \frac{2}{N+1}$, where $N = 540$, and $\gamma = 0.4$.

confidential

the absence of DDoS attacks, we have injected a SYN flood attack on August 15. This was done by running Masscan[4] on a server, which is directly attached to the switch to prevent the attack from leaking into the network. The source IP addresses and ports are randomly generated in order to obtain a large number of flows, like an actual DDoS attack would. The attack lasted for around 13 minutes. It started and ended with approximately 4k packets per second (PPS), and peaked at 20k PPS for a minute. During this attack, we only observe a minor increase of the load of the *switching CPU*, caused by the increased number of packets to be switched, and no increase in load for the *routing CPU*. As such, we conclude that the CPU load caused by our implementation during attacks does not peak and instead only consists of the constant load. The peaks in the load of the *routing CPU*, visible in Fig. 3, are likely the effect of other routing or management processes on the Catalyst 6500, as such processes are handled by the *routing CPU*. In terms of memory utilization, we clearly observe a stable pattern in Fig. 3, aside from the memory spike during the malfunctioning of the EEM. We do not observe any increase in memory utilization during the attacks, therefore we conclude the memory utilization does not create big peaks.

Considering the above measurements, we conclude that the memory utilization does satisfy the requirement of using 10% of memory or less. However, the 19% CPU load caused by our implementation does not satisfy the requirement of 10% CPU utilization or less. As the Catalyst 6500 is a packet switching device and not meant to perform network attack detection, such other activities should not interfere with its main purpose of operation. As a load of 19% is probable to cause interference with the routing and switching tasks, we conclude that our implementation does not satisfy the requirement to be light-weight. The difference between the measured constant load and the lack of peaks in Fig. 3 can be explained by the fact that the amount of traffic does not change the number of computations performed by the policies, as only the calculated values are different. Furthermore, the short and frequent execution of the policies will be averaged out to a constant added CPU load. Especially the short, 2 second interval of the measurement policy increases the load. However, increasing this interval would decrease the measurement resolution, as described in Section IV.

The second requirement is the detection delay. This requirement, like the accuracy, has already been validated for the prototype in [9]. Our implementation uses an interval of 10 seconds between invocations of the algorithm, instead of 5 seconds as described in [9] because of the runtime of the algorithm, as described in Section V. This results in detection delays of multiples 10 seconds, with a minimum of 10 seconds. The attack visible in Fig. 4 was detected within the second interval, resulting in a detection delay of 20 seconds.

The final requirement considered in this work is the accuracy of the DDoS attack detection. In Fig. 4, the number of flow cache entry creations per measurement interval is shown, averaged over 5 minute intervals. Weekends are shaded in light-gray. Diurnal patterns are clearly distinguishable and due to the nature of the traffic, we can also observe the difference between weekdays and weekends. Two anomalous periods are clearly visible. The first, starting on August 11, is a learning phase of the algorithm, as the algorithm was started in the weekend. The second, starting on August 16 around 3:00, is due to a malfunctioning of the EEM, which caused the policies to stop running. The generated attack on August 15 is cleary distinguishable in Fig. 4. It resulted in 2–3 times more flow records than predicted by the algorithm, and lasted for roughly 13 minutes. Multiple detection marks are shown, as the attack overlapped multiple 5 minute intervals.

[4]https://github.com/robertdavidgraham/masscan

VII. Discussion

In the previous sections we have investigated whether it is possible to deploy DDoS attack detection on a Cisco Catalyst 6500. Although the detection of (DDoS) attacks is a crucial first step, it merely serves the ultimate goal: attack mitigation.

In [9], not only a detection algorithm is presented, but mitigation is discussed as well. When the detection algorithm is run and a measurement sample is considered malicious, mitigation is started. By counting the number of exported flow records per source IP address, attackers can be identified. As soon as more than 200 flow records with three packets or less have been exported per second for a particular source IP address, the source IP address is blacklisted. These blacklisted IP addresses are added to a firewall to block traffic from the attacker. Furthermore, to prevent the flow collector from overloading, the flow records with these IP addresses are not sent to the collector. When the algorithm detects the end of the attack, the created rules are removed from the firewall.

The information used to identify attackers in [9] is not available in IOS; Only the total number of exported cache entries is available. An alternative approach for identifying attackers is to analyze the contents of the flow cache. However, the IP addresses of attackers will be overrepresented in the cache during a DDoS attack, since attackers generate large amounts of traffic, which result in a large number of flow cache entries. However, the time to retrieve and process the entire flow cache under load – which consists of at least 128k entries, depending on the used hardware – can take up to tens of seconds, making timely mitigation using this approach hardly possible.

A different approach to implement mitigation is the use of an IOS feature that keeps track of the top $x \in (0, 200)$ flows featuring the highest volume, either in terms of packets or bytes, referred to as *NetFlow Top Talkers*. This feature cannot show the top talkers by the number of flows a source address produces, which will be very high for DDoS attackers. Furthermore, it is very likely that legitimate users will be in the top talkers list, as they can generate just as much packets and bytes. We therefore conclude that it is very hard to identify the attackers and set aside mitigation in this work, while focussing on detection.

confidential

VIII. Conclusions

This paper has presented a prototype for detecting DDoS attacks on Cisco's IOS platform, based on the algorithm presented in [9]. Our results show that detection of flooding attacks is possible within tens of seconds, making real-time detection on a widely available switching platform possible. However, our prototype has also shown to cause a high CPU load of 19%, which may cause interference with the routing and switching processes. The cause of this is the frequent polling for reliable monitoring information used in our implementation. According to various network operators we have stayed in touch with during this work, if the capacity of the packet forwarding device is available, it should be possible to run our DDoS attack detection in a production environment. While it is possible to deploy our implementation with only 20–30% CPU available, for example, it would require to be run with a lower priority, to not interfere with the routing and switching processes. As this may cause instability to our prototype, it is advised to at least have 40% CPU available.

The first requirement is the small footprint of the implemented detection algorithm. The validation results show that during attacks there is no visible increase in CPU and memory usage. However, when monitoring the overall increase in CPU and memory usage compared between not running and running the detection prototype, an increase of 19% CPU load, and 0.13% memory usage can be observed. While the memory usage satisfies the requirement of using 10% or less of the available resources, the CPU load does not satisfy this requirement. Validation of our prototype in the network of the University of Twente has shown that detection delays of 20 seconds are feasible for high intensity attacks, satisfying the real-time requirement of detection within 30 seconds. This corresponds to twice our measurement interval of 10 seconds. Smaller measurement intervals may decrease detection delays, but will make it more likely that our detection runs overtime and is killed by a management process. The last requirement for our implementation is detection accuracy. Our validation results show that the number of false positives is low, while the detection rate is high, because of which we conclude that our prototype is accurate.

Mitigation is the next step, after detection. Our investigation has shown that while it is possible to obtain enough information to identify possible attackers, the command used to obtain this information can take tens of seconds when the switch is under heavy load, which occurs during flooding attacks. We therefore conclude that real-time mitigation is impossible on the hardware used in this work.

Possible future work includes investigating alternative implementations on different hardware. The successor of the *Supervisor Engine 720*, the *Supervisor Engine 2T*, contains more powerful hardware and provides additional functionality. This more powerful hardware is likely to influence the load caused by our implementation in a positive way, and potentially even allow for real-time mitigation. Furthermore, a brief investigation has shown that the *Supervisor Engine 2T* has the option of using events upon flow cache entry creations. This could replace our approximation of the number of flow cache entry creations, as described in Section IV, and make it more accurate and possibly faster.

Acknowledgments

Special thanks go to Vosko Networking B.V. and Cisco Systems International B.V. Amsterdam for providing the network devices used in this work, and Jeroen van Ingen Schenau, Roel Hoek, Niels Mejan and Jan Markslag (University of Twente) for their help in setting up the measurement infrastructure. This work was partly funded by FLAMINGO, a Network of Excellence project (ICT-318488), and SALUS, a STREP project (ICT-313296), both supported by the European Commission under its Seventh Framework Programme.

References

[1] CloudFlare, Inc., "Technical Details Behind a 400Gbps NTP Amplification DDoS Attack," 2014, accessed on 25 August 2014. [Online]. Available: http://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack

[2] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011 (Internet Standard), Internet Engineering Task Force, September 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7011.txt

[3] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[4] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, 2014.

[5] A. A. Galtsev and A. M. Sukhov, "Network attack detection at flow level," in *Proceedings of the 11th international conference and 4th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking*, 2011, pp. 326–334.

[6] H. A. Nguyen, T. Tam Van Nguyen, D. I. Kim, and D. Choi, "Network Traffic Anomalies Detection and Identification with Flow Monitoring," in *5th IFIP International Conference on Wireless and Optical Communications Networks, WOCN'08*, 2008, pp. 1–5.

[7] N. Muraleedharan, A. Parmar, and M. Kumar, "A Flow based Anomaly Detection System using Chi-square Technique," in *Proceedings of IEEE 2nd International Advance Computing Conference IACC'10*, 2010, pp. 285–289.

[8] R. Hofstede and A. Pras, "Real-Time and Resilient Intrusion Detection: A Flow-Based Approach," in *Proceedings of the 6th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS'12, Ph.D. Workshop*, ser. Lecture Notes in Computer Science, vol. 7279. Springer Berlin Heidelberg, 2012, pp. 109–112.

[9] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras, "Towards Real-Time Intrusion Detection for NetFlow/IPFIX," in *Proceedings of the 9th International Conference on Network and Service Management, CNSM'13*, 2013, pp. 227–234.

[10] Follett, J.H., "Cisco: Catalyst 6500 The Most Successful Switch Ever," 2006, accessed on 25 August 2014. [Online]. Available: http://www.crn.com/news/networking/189500982/cisco-catalyst-6500-the-most-successful-switch-ever.htm

[11] R. Hofstede, I. Drago, A. Sperotto, R. Sadre, and A. Pras, "Measurement Artifacts in NetFlow Data," in *Proceedings of the 14th International Conference on Passive and Active Measurement, PAM'13*, ser. Lecture Notes in Computer Science, vol. 7799. Springer Berlin Heidelberg, 2013, pp. 1–10.

[12] Cisco Systems, Inc., "Catalyst 6500/6000 Switch High CPU Utilization," 2012, accessed on 25 August 2014. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/63992-6k-high-cpu.html

confidential