University of Twente
Department of Computer Science
Design and Analysis of Communication Systems Research Group
Cyber Security Master's Track

Consumer-Deployable Network Intrusion Detection in Public Clouds

Master Thesis

Sem J.S. Spenkelink

Supervisors:
Prof. dr. ir. Aiko Pras, Professor at the University of Twente
Dr. Jair Santanna, Assistant Professor at the University of Twente
Dr. Anna Sperotto, Assistant Professor at the University of Twente
Ir. Mark Borst, Principal Cyber Security Consultant at Northwave

Enschede, May 1st 2018

Abstract

The cloud has become a widely used platform that is ever growing. Cloud users vary from single host consumers to multinationals that are settled all over the globe. Unfortunately, where ordinary crowds move amass, so do their adversaries. Evidently, security is an important factor in any system, the cloud being no exception. While most cloud providers have a fair amount of security mechanisms available for purchase, there is no easy way for cloud consumers to monitor their own network. Cloud service providers will not provide a copy of data on a shared network, let alone allowing John Doe into their physical datacenter.

This thesis provides insight in several methods to accomplish network intrusion detection in cloud systems. The contribution is twofold. Firstly, it provides an extensive overview of the cloud landscape and the corresponding requirements to monitor the network component properly. Secondly, the thesis revises existing and novel methods to perform network intrusion detection in the public cloud environment. This research yields the most feasible option for public cloud consumers to monitor their network.

Before introducing a method to accomplish network intrusion detection in public clouds, it is fundamental to provide a well-founded set of requirements. The requirements are established based on obligatory aspects within cloud systems and requirements for traditional network intrusion detection systems. The requirements also take into account usability for the cloud consumer. Subsequently, exhaustive research regarding intrusion detection methods that ought to be applicable in public clouds was conducted. Each of these methods were thoroughly analysed and mapped onto the requirements. The most promising methods were compared in detail.

In the end, this research resulted in the most feasible method to perform network intrusion detection on a cloud environment without requiring the cloud service provider's interference. The network intrusion detection solution suggested in this thesis abides by all the requirements. The solution comes in form of an inline intrusion detection agent that taps the host's interfaces. The incident information is then aggregated and correlated in a central server. Because an agent is deployed per server, the solution scales excellently with the dynamic cloud environment. The agent can be implemented in any given (public) cloud infrastructure.

Finally, the functionality of the proposed method was tested with several heuristic experiments. The security capabilities of implementation were stress-tested and the solution was compared to an environment that lacks a cloud network intrusion detection system. No significant performance drop was observed. Based on these results, the consumer-deployable network intrusion detection system has shown to be a feasible solution for an arbitrary sized cloud environment.

Contents

Contents **ii**

1 Introduction **1**
1.1 The Cloud . . . 1
1.1.1 Service Models . . . 2
1.1.2 Deployment Models . . . 3
1.2 Intrusion Detection Systems . . . 3
1.2.1 Host Based Intrusion Detection . . . 3
1.2.2 Network Based Intrusion Detection . . . 3
1.2.3 Detection Methods . . . 4
1.2.4 HIDS and NIDS in the Public Cloud . . . 4
1.2.5 Log Monitoring and Analysis . . . 4
1.3 Problem Statement . . . 4
1.4 Research Goal . . . 5
1.5 Research Questions . . . 5
1.5.1 Subquestion 1 . . . 5
1.5.2 Subquestion 2 . . . 5
1.5.3 Subquestion 3 . . . 5
1.6 Approach . . . 5
1.7 Thesis Road Map . . . 7

2 Literature **8**
2.1 Cloud Service Providers . . . 8
2.1.1 Amazon Web Services . . . 8
2.1.2 Microsoft Azure . . . 9
2.1.3 Google Cloud . . . 9
2.1.4 Comparison . . . 9
2.2 Attack Landscape . . . 10
2.2.1 Reconnaissance, Scanning & Sniffing . . . 11
2.2.2 Network Based Attacks . . . 11
2.2.3 Denial of Service . . . 11
2.2.4 Worms . . . 11
2.2.5 Phishing . . . 11
2.2.6 Lateral Movement . . . 11
2.3 Summary . . . 12

3 Challenges **13**
3.1 Critical Areas for Cloud Computing . . . 13
3.2 Public Cloud NIDS Challenges . . . 14

4 Requirements **15**
4.1 Methodology . . . 15
4.2 Public Cloud IaaS Aspects . . . 15
4.3 NIDS Aspects . . . 16
4.3.1 Security Capabilities . . . 16
4.3.2 Performance . . . 16
4.4 Utility Aspects . . . 17
4.4.1 Management . . . 17
4.4.2 Life Cycle Costs . . . 17
4.5 Requirements . . . 18
4.5.1 Cloud Requirements . . . 18
4.5.2 NIDS Requirements . . . 18
4.5.3 Utility Requirements . . . 19
4.6 Summary . . . 19
4.7 Conclusion . . . 19

5 Theoretical Analysis of Cloud NIDS Methods **20**
5.1 Methodology . . . 20
5.2 Conceptual Approaches . . . 20
5.2.1 Bump in the Wire . . . 20
5.2.2 Host based network intrusion detection . . . 21
5.2.3 Host based network traffic replication . . . 21
5.3 State of the Art Approaches . . . 21
5.3.1 IDSaaS . . . 22
5.3.2 Nested Virtualization . . . 22
5.3.3 Cloud based Intrusion Detection Service (CBIDS) . . . 23
5.3.4 Cloud Intrusion Detection System Service (CIDSS) . . . 23
5.3.5 Distributed IDS with Mobile Agents . . . 23
5.3.6 NICE . . . 24
5.3.7 H-NIDS . . . 25
5.4 Revision . . . 29
5.4.1 Cloud Revision . . . 29
5.4.2 NIDS Revision . . . 30
5.4.3 Utility Revision . . . 31
5.4.4 Summary . . . 31
5.5 Requirement Analysis . . . 32
5.5.1 Inline vs. Out of Band . . . 32
5.5.2 Comparison . . . 33
5.6 Conclusion . . . 35

6 Implementation **36**
6.1 Introduction . . . 36
6.2 Methodology . . . 36
6.3 Experiment . . . 38
6.3.1 Latency Experiment . . . 39
6.3.2 Security Experiment . . . 39
6.3.3 Impact Experiment . . . 39
6.4 Results . . . 39
6.4.1 Latency Results . . . 39
6.4.2 Security Results . . . 42
6.4.3 Impact Results . . . 43
6.5 Analysis . . . 44
6.5.1 Latency Analysis . . . 44
6.5.2 Security Analysis . . . 44

6.5.3 Impact Analysis 45
6.6 Conclusion 45

7 Conclusions **47**
7.1 Conclusion 47
7.2 Limitations & Future Work 49

Bibliography **50**

Appendix **52**

A System Specifications **53**

Chapter 1

Introduction

Over the past decades, the internet has become a vital part of our present-day infrastructure. People rely on the internet for their daily routine. The concept of being able to access internet services on-demand is extremely popular. Research from Eurostat shows that in 2014 over 25% of inhabitants of the European Union, aged between 16 and 74, actively used internet services and 21% have used the cloud to store files [42]. Moreover, in addition to individual cloud users, 21% of European enterprises used cloud computing in 2016 [17]. From those firms, 51% used cloud computing to run advanced computing services, business applications, or management software. This article also shows a steep increase of cloud usage in enterprises, of over 10%, since 2014. Interestingly, enterprises seem to rely on public clouds (e.g., hosted by Amazon or Microsoft) twice as often as its private alternative (hosted by the enterprise itself).

From the above data, we gather that cloud usage is becoming more and more popular among individuals as well as enterprises. It has to be taken into account that relatively new platforms such as the cloud allow new challenges to arise. Nevertheless, the Dutch Central Bureau for Statistics (CBS) found that cloud users are significantly less worried about their internet security than regular internet users (38% opposed to 53%) [7]. It is worrisome that a novel platform without apparent security assurances provides a safer perception.

The combination of a growing novel environment in combination with unaware cloud consumers yields an interesting attack surface for intruders. To combat this, we feel like there is a need to reassess the capabilities of cloud consumers to monitor their environment. To identify where possible cloud security issues might arise, we first dive into the concept of the cloud (section 1.1). Then we introduce the concept of intrusion detection to identify how cloud consumers can monitor the cloud (section 1.2). These introductory paragraphs sketch a broader problem, which we address in the problem statement (section 1.3). In turn, this problem yields a research goal. Section 1.5 will cover the research questions to accomplish this goal. The approach to answer these questions is addressed in section 1.6. Finally, we present the road map containing an outline of the remainder of this research (section 1.7).

1.1 The Cloud

The National Institute for Standards and Technology (NIST) states that cloud computing should adhere to the following characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity and a measured service [32]. On-demand self-service means that the consumer can provision themselves without personal interaction with the cloud service provider (CSP). This is favourable as it omits the slow bureaucratic processes. The broad network access characteristic entails that the cloud computing resources should be available via standard mechanisms used in present-day clients. Resource pooling is the dynamic allocation of resources within a pool of clients. This serves as a way for the CSP to optimise their environment as clients can share the underlying network and infrastructure (multi-tenancy). Furthermore, rapid elasticity

caters to the desirable functionality for the consumer to only pay for utilised resources. The rented cloud space can be scaled either on demand or automatically. Lastly, cloud systems should be able to measure resource usage. Together with the rapid elasticity, this defines the pay-per-use cloud model. For example, higher bandwidth usage results in a higher invoice.

The characteristics provide organisations with an appealing business model to migrate to the cloud. The cloud comes in different shapes and forms, which will be discussed in the upcoming section. Each of these forms has some kind of novel security challenge. To counter this, consumers should be able to apply security monitoring on their own systems. In a traditional setting this is, among other things, accomplished by deploying intrusion detection systems (IDS). However, from a consumer perspective, the cloud moves physical access of hardware to an inaccessible logical version, which yields new challenges. To stay in line with the on-demand self-service aspect of the cloud, it is desirable for consumers to deploy independently configurable and verifiable security measures. This entails that it should be possible for the CSP, consumer and possibly the managed security service provider (MSSP) to verify that certain security measures are in place.

1.1.1 Service Models

In the cloud there are different service models. Each of these models provide the customer with certain capabilities. The upcoming sections outline the three service models as described by the NISTs [32].

Software as a Service (SaaS)

SaaS is focused on providing the consumer with an application running in the cloud. Therefore, the application is accessible from various client devices through a client or program interface. The consumer has no control or management tasks over the underlying infrastructure, such as the operating systems, network, storage, servers or application capabilities. Since SaaS is all about an application, consumer security is mainly focused on identity access management. The CSP dictates the security measures of the inaccessible infrastructure listed above.

Platform as a Service (PaaS)

In PaaS, customers gain the ability to deploy consumer created or acquired applications on the platform. These applications can be created and managed with programming languages, libraries and tools that are supported by the CSP. PaaS delivers the consumer with a computing platform and solution stack in forms of a service. Other than with SaaS, the consumer now has control over deployed applications and possibly some configuration settings for the hosting environment. The underlying infrastructure remains inaccessible. PaaS is situated lower down the stack, consequently the consumer is responsible for implementing and managing more security features. On top of the SaaS responsibilities, the consumer should also take data, storage and application security into account.

Infrastructure as a Service (IaaS)

The last service model provides consumers with processing, storage, networking and other computing resources. The consumer is free to run their own software, including operating systems, on these resources. In IaaS, the CSP allows its consumers to access the underlying architecture (hardware/data-centers) by means of virtualization. CSP-staff remains in charge of the physical infrastructure, while the consumer has control over the deployed resources. Consequently, physical servers, storage, virtualization and networking are managed by the CSP, while applications, databases and operating systems are managed by the consumer. Security and the way it is integrated, however, remains a middle ground.

1.1.2 Deployment Models

The cloud has several deployment models. Firstly, the private cloud. This model is supplied for the private use of an organization, implying that it is a separate internal environment that is not susceptible to multi-tenancy problems. In this setting, only authorized users can access the resources. Still the private cloud does not necessarily have to be on the client's premises. The management and operating of the private cloud may as well be done by a third party.

On the other hand, public cloud provides services for public usage via the network. The underlying infrastructure resides completely on the CSP's premises. The CSP in turn caters its services to the client in an on-demand pay-per-use manner. Using hardware virtualization (to hide underlying physical characteristics) the CSP allows multiple users to run their own services simultaneously on the same underlying system (multi-tenancy). The allocation of client resources in a pooling manner allows, among other things, for better pricing, maintenance and elasticity. Therefore, the public cloud is particularly appealing for small and medium-sized organizations.

An alternative deployment model is the hybrid cloud, where multiple instances of the above are used simultaneously. It might be clever to split critical intellectual property and other data over public and private cloud servers. Lastly, there are community clouds where a cloud is used to serve a shared purpose. For example, when multiple businesses work on joint ventures.

Since private clouds are not vulnerable to novel security challenges such as multi-tenancy, we scope our research down to the more versatile public cloud. In the next sections we identify key features and security challenges.

1.2 Intrusion Detection Systems

Without taking into account any hybrid or distributed combinations, there are two types of intrusion detection systems; host based intrusion detection systems (HIDS) and network based intrusion detection systems (NIDS). For completeness of this research, we also take into account additional log sources that can be analysed. For example, one might use firewall logs as alternative source to verify intrusions on the network boundary. This is relevant in the cloud setting as CSPs might provision certain infrastructural, activity, diagnostic or application logs. HIDS and NIDS are usually interleaved. HIDS catches intrusions the NIDS misses out on and vice versa.

1.2.1 Host Based Intrusion Detection

As the name indicates, HIDS monitors a host. In general, it accomplishes this by monitoring a list of objects (e.g., the files from the file system). HIDS then checks logs and activity occurring on these objects for unwanted modifications, memory and data integrity, system calls and more.

1.2.2 Network Based Intrusion Detection

A NIDS on the other hand, monitors packets that flow through the network, checking them for malicious content or policy violations [1]. The detection unit is usually placed as test access point (TAP) or switch port analyzer (SPAN) on a switch that mirrors the data elsewhere. Traditionally, there are two placement options for a NIDS sensor. Firstly, there is an inline option, which is a device that is placed on the network route. Consequently, the NIDS can actually stop packets from reaching their destination, possibly turning the intrusion detection system in an intrusion prevention system. However, this requires the packet analysis to happen inside this inline NIDS device, which introduces additional latency. Secondly, there is out of band NIDS, which sits outside the network. Instead out of band uses copies of the data. The copies are usually provided via a mirroring port on a switch or TAP. Out of band introduces little to no extra latency, which is desirable, especially in networks under heavy loads. On the downside, out of band looks at copies, so the data is no longer real time.

1.2.3 Detection Methods

For both HIDS and NIDS the detection comes in two main principles. An IDS either looks for behaviour that diverges from normal activity, or it matches incoming data against known patterns and signatures. These two methods are called anomaly based detection and signature (or misuse) based detection. Evidently, the latter method only works against known attacks. On the other hand, anomaly detection tends to generate more false positives than its signature based counterpart. In practice, signature and anomaly based detection are often used in conjunction.

1.2.4 HIDS and NIDS in the Public Cloud

Either two of the above detection methods have their own advantages and disadvantages. Essentially, they supplement each other. As established, HIDS is deployed on the host itself. In a cloud setting this is no different. Whether the host is a traditional physical device or a virtual machine, implementing a HIDS remains identical in the cloud. NIDS, on the other hand, proves an issue. NIDS sensors require access to the network layer, to place a tap or configure mirroring on a switch. In a public cloud setting, this is infeasible, since the underlying infrastructure of CPSs like Microsoft or Amazon are inaccessible. Even if data mirroring was possible, it would still be unfavourable due to the multi-tenancy characteristic of the cloud.

1.2.5 Log Monitoring and Analysis

Event and log correlation from different detection units is often performed in a centralised machine. This so called SIEM (security information and event monitoring) is essentially an extra layer on top of all security controls. Besides IDSs and other security devices, a SIEM can also collect logs from most sources within your environment. In the cloud there are a lot of SaaS, PaaS and even IaaS instances that generate logs. These logs range from system and application logs to active directory, user activity, virtual machine (VM) or cloud security appliance related logs. Essentially, cloud providers provide security of their own, so in light of this research it is definitely something to take into account on top of IDS.

1.3 Problem Statement

In this introductory chapter, we defined and elaborated on the different cloud spectrums and intrusion detection. The initial section of this thesis shows the lack of security awareness among public cloud consumers, making this an interesting field of study. Furthermore, section 1.2.4 shows that network intrusion detection is complicated in a cloud setting. The combination of three factors - the troublesome public cloud security setting, unaware end users, and NIDS possibilities - has motivated us to look into a consumer deployable NIDS for public clouds.

While there are plenty of challenges within the scope of this research, the main problem is the inaccessibility of network intrusion detection from a consumer perspective. In traditional systems consumers can perform network intrusion detection by plugging NIDS sensors into the network or infrastructure. However, in a public cloud environment this poses a problem. The network layer is entirely hosted on the premises of the cloud service provider and therefore not accessible. The service providers might provide some logs, allowing clients insight in their underlying infrastructure (for example information regarding creation of new virtual machines), but there is no universal logging for all network based attacks. Even if some CSPs would provide sufficient logging for consumers to cover themselves from network based attacks, there is no guarantee that this applies to all CSPs out there. The situation sketched above is the existential issue that actuates this research. There are additional challenges that arise within the scope of this research. These will be addressed in chapter 3, where we touch upon all the aspects of cloud, NIDS and its surrounding environment.

1.4 Research Goal

The goal of this research is to find a network intrusion detection method that is deployable in public cloud settings. The method should be a general concept that is applicable in any public cloud. The goal is to achieve a method that can be implemented from a consumer perspective, so it is not prone to cloud vendor restrictions. Lastly, the devised method should comply with cloud and traditional NIDS requirements.

1.5 Research Questions

As established, the research goal is to find a NIDS method for public clouds that the consumer can deploy themselves. This leads to the following research question:

What is the most effective way of accomplishing consumer-deployable network intrusion detection in public clouds?

However, this question is difficult to interpret on its own. Therefore, we divide it into three subquestions.

1.5.1 Subquestion 1

Firstly, we need to identify what characteristics belong to the cloud and to a NIDS. Based on these findings we will determine requirements that should hold for a public cloud NIDS. These requirements will also include usability requirements that are relevant for the cloud consumer. This leads to the following subquestion:

What are the requirements for deploying a consumer-deployable NIDS for public clouds?

1.5.2 Subquestion 2

Once we have identified requirements, we will dive into literature to find different NIDS methods and revise them, taking into account subquestion 1.5.1. This leads to the following subquestion:

Based on these requirements, what are the theoretically most promising consumer-deployable NIDS methods for public clouds?

1.5.3 Subquestion 3

Lastly, when we have identified what method works best based on our revision, we will implement this method. Consequently, the implementation is tested and its performance will be verified based on an identical server without our implemented solution. This leads to the last subquestion:

How does the theoretically most promising method perform in a heuristic experiment, taking all requirements into account?

1.6 Approach

The above research questions need to be answered chronologically, as the answer of subquestion 1 provides the foundation for answering subquestions 2 and 3. Similarly, in subquestion 3 we will implement a method based on our findings in subquestion 2. For clarity, the next three sections will outline our approach for answering all research questions.

Approach Plan Subquestion 1

The first subquestion can broadly be divided into three components: consumer utilities, network intrusion detection system and public clouds. Based on these aspects we need to determine requirements that should be covered in the implementation of a consumer deployable NIDS for public clouds. We tackle this problem by delving into literature based on public clouds and intrusion detection systems. The requirements will be derived from academic and well-regarded sources.

Approach Plan Subquestion 2

Taking into account the problem statement and the research questions, we have to identify some core matter regarding the subject. Therefore, it will be necessary to acquire information about the following topics:

(a) What attack scenarios are relevant for network or cloud based adversaries?

(b) What academic research is available to implement NIDS in public clouds?

(c) In what way can traditional NIDS be tweaked for it to work in public clouds?

(d) What alternative techniques are available to cover network based attacks, other than NIDS?

(e) What are present-day cloud service providers doing to procure their customers with logging?

(f) What are present-day MSSPs and security providers doing to protect public cloud customers?

(g) What are the pros and cons for each of the identified techniques?

For a large part, these questions will be covered in the literature study. In section 2.2 attack landscape will be introduced to get an overview of what consumers should be able to protect themselves against. For a final verdict about best NIDS deployment and techniques in the cloud, we will have to take into account network attacks that should be detected. It also allows us to find security gaps in each of the detection techniques discussed in section 2.2.

Furthermore, for topics (b), (c) and (d), a broad overview of relevant intrusion detection methods will be investigated. The techniques that will be evaluated, will be described in sections 5.3 and 5.2. What CSPs and MSSPs put into practice (topics (e) and (f)) is also shortly touched on in the literature chapter (section 2.1).

The key aspect of this subquestion lies in identifying possible ways to achieve proper network intrusion detection possibilities in the public cloud. Once we have accumulated an exhaustive list of methodologies, we will compile all positive and negative aspects for all of these methods. In this process we will emphasise on the requirements defined in subquestion 1. Once we have accumulated all necessary information, an extensive theoretical review will be conducted for each of the methods gathered from literature. The review will include as many pros and cons as we can find from literature and logical evaluation. This includes aspects like detection performance (false-positives and false-negatives), overhead, response time and ease of use (deployment and editing configurations). To verify the necessity of a good network intrusion detection method for public clouds, we also cross-reference this with what present-day CSPs, Cloud MSSPs or Cloud IDS providers do to protect their customers.

The process described above will lead to a critical revision of the theoretically most promising methods that are gathered from literature. In this revision the methods are classified in terms of pros and cons based on the predefined requirements.

Approach Plan Subquestion 3

The critical revision acquired in subquestion 2 allows us to choose the theoretically most feasible approach. To verify this, we intend to perform a heuristic experiment with the most promising concept(s). This might also be a combination of several approaches that defeat the cons and

enhance the positive aspects of each technique. For the experiment, we will set up a cloud networking test environment. In this environment we deploy the NIDS or NIDS alternative. The implemented method will be tested with ingress, egress and local traffic that will include, but not be limited to, relevant attack scenarios. In the end it should be possible to verify in what way the new method contributes to intrusion detection in comparison with the capabilities that CPSs deploy natively.

1.7 Thesis Road Map

The remainder of this thesis will be structured as follows:

- **Chapter 2 - Literature.** This chapter contains background information regarding cloud service providers. To develop a public cloud NIDS, we first need to familiarise ourselves with the public cloud infrastructure and what services it offers. Furthermore, this section includes information about the attack landscape surrounding the cloud. The presence of vulnerabilities to network based attacks provides insight regarding what a public cloud NIDS should be capable of.
- **Chapter 3 - Challenges.** As with all research, there are some challenges to overcome. In chapter 3 we define what challenges a consumer-deployable NIDS in the public cloud faces. This chapter defines the problems to overcome, which will aid us when defining requirements.
- **Chapter 4 - RQ1: Requirements.** Now that we have determined the challenges and landscape of our research, we start off by defining requirements the NIDS should comply with. Defining requirements is fundamental in the process of answering research questions two and three.
- **Chapter 5 - RQ2: Public Cloud NIDS Revision.** This chapter contains an extensive revision of different cloud NIDS methodologies. We formulate pros and cons based on the requirements that were defined in chapter 4.
- **Chapter 6 - RQ3: Experiment.** From our findings in research question 2, we devise the theoretically best NIDS in the cloud and implement it in a major public cloud platform. The final research question provides us with a measure in quantitative performance of the NIDS, based on the requirements defined earlier.
- **Chapter 7 - Conclusion.** This chapter closes the thesis by drawing our final conclusions and identifying any limitations and future work.

Chapter 2

Literature

In the previous chapter, we defined the two main aspects of this thesis; the cloud model and intrusion detection systems. Based on these two topics, background information was gathered. For cloud services, we look at the leading three cloud service providers according to Gartner [30]. Section 2.1 is dedicated to identifying what security mechanisms are already in place for the different providers. This allows us to identify whether NIDS is already covered within the scope of currently deployed services within cloud platforms. In section 2.2 we define relevant attacks in public cloud environments. This background information provides reasoning why NIDS is useful in the cloud.

2.1 Cloud Service Providers

To protect their own property as well as their customers', cloud service providers apply security of their own. The level of security provided in cloud systems is an indication of how important it is to monitor network traffic for consumers. Therefore, it is interesting to see what features major CSPs have deployed. In this section we scope down to the IaaS service model, as this requires and facilitates the most consumer based security measures.

2.1.1 Amazon Web Services

Present day, Amazon is probably the leading CSP on the market. The Amazon Web Services (AWS) platform [3] offers a large variety of cloud services of which Elastic Compute Cloud (EC2) is their main IaaS product. Out of the box, AWS utilizes a variety of cloud-network monitoring systems. These should provide protection against several of the traditional network based attacks. AWS claim to protect against: 1) Distributed denial of service attacks (DDoS), due to 'world class infrastructure, proprietary DDOS mitigation techniques and homing across a number of providers', 2) Man-in-the-middle attacks (MITM), due to SSL-encrypted endpoints, 3) IP Spoofing, due to the infrastructural design of hosts not being able to send traffic with source other than its own, 4) Port Scanning, due to all ports being closed by default and 5) Packet Sniffing, due to the hypervisor not allowing VMs running in promiscuous mode to sniff traffic that is intended for another VM [4]. Furthermore, AWS provides monitoring and logging via their Cloudtrail[1] and Cloudwatch[2] services. Cloudtrail logs any API activity including SDKs and command line tools. With Cloudwatch it is possible to import any type of logging to process. These can be logs from applications, systems, networks or even services like Cloudtrail. Log analysis from systems can be used to search, among other things, for malicious logins or DDOS attacks. Additionally, AWS uses security groups that can be assigned to any instance or group of instances to administer what traffic can flow to which places. AWS also provides a web application firewall (WAF) service which

[1] https://aws.amazon.com/cloudtrail/
[2] https://aws.amazon.com/cloudwatch/

has similar functionality as a security group, with the addition of conventional rules that block regular attack patterns such as SQL injections and cross-site scripting (XSS). Lastly, AWS offers a virtual private cloud (VPC) which provides almost all benefits of an on premises infrastructure, with the benefits of the cloud. The VPC can be monitored with VPC flow logs, keeping track of all packet movement across the VPC. These logs can, once more, be integrated with Cloudwatch.

2.1.2 Microsoft Azure

Azure is Microsoft's version of an IaaS full of cloud services that consumers can use to build, deploy and manage applications [34]. Within Azure the services are similar to AWS. There are options for compute, networking, storage and more. Azure's Security Center enables the user to prevent, detect and respond to threats [11]. On top of that, Azure comes with standard capabilities such as encryption for data in transit and at rest. Furthermore, Microsoft offers logging with Azure Monitor[3], which offers infrastructure, system, and application level data about the throughput of a service and the surrounding environment. Security specific event logs are customizable in the platform and can be managed in the Azure Security Center[4]. For securing the network, Azure defined network access controls [12]. These controls can be enforced via the Network Security Groups (NSGs), which are cloud based stateful packet filtering firewalls to evaluate any traffic flowing in or out a VM, a group of VMs or even entire subnets. Note that the packet filtering is stateful and cannot be compared with the full packet inspection capabilities traditional NIDS encapsulates. Similar to AWS's VPC, Azure also includes the ability to deploy virtual networks (VNets).

2.1.3 Google Cloud

Google's cloud computing environment has been around since 2008, but their current IaaS service, Google Compute Engine, has only been here for a couple of years. Even though Google is relatively new in the market compared to the above two, they cannot be left out. Google Cloud [18] is supported by the same infrastructure Google uses for their core services like YouTube or their search engine. In terms of security, Google have implemented several essential defaults across their entire infrastructure. Examples of these security services include, but are not limited to, DoS mitigation, encryption in transit and at rest, secure authentication and inter service access management [19]. Within their own infrastructure Google applies monitoring (and network intrusion detection) that is focused on information gathered from the internal network, employee actions and knowledge of vulnerabilities. These sources do not surface to consumer level [20]. However, all platform API requests are logged for the user to monitor with Google's stackdriver platform tools for logging[5] and monitoring[6]. Interestingly, both these services are also available for AWS[5,6]. For private networking, Google also deployed a VPC consisting of a comprehensive set of networking capabilities, including granular IP address range selection, routes, firewalls, VPN, and Cloud Router.

2.1.4 Comparison

In terms of security, we identified that in broad lines all CSPs have similar functionality. However, there are a few key differences worth noticing.

Access Control

Security groups (AWS and Azure) have a management advantage over firewalls (Google Cloud), as they reduce the number of distinct configurations that have to be maintained. Therefore, the chance of (human) error is belittled. Other than that, they function similarly, so applying

[3] https://docs.microsoft.com/azure/monitoring-and-diagnostics/
[4] https://docs.microsoft.com/azure/security-center/
[5] https://cloud.google.com/logging/
[6] https://cloud.google.com/monitoring/

both would provide marginal benefits. However, AWS has an additional web application firewall, which adds some IDS capabilities from regular attack patterns. This is not natively possible in Azure or GC. Furthermore, AWS's security groups have the advantage that they can employ a different security group as a (partial) rule source for another. Azure's NSGs, on the other hand, are completely static.

Networking

All three CSPs allow for virtual networking. Google Cloud has one extra addition, that networks can exceed the resources, which allows deployment across multiple regions. This reduces demand for complex VPN configurations. Similarly, a single subnet is not bound to the address space of the parent network, allowing different subnets to be deployed on different IP ranges within the same network.

Monitoring

In terms of monitoring, it is hard to say what differences exist. In all three environments, there are over 1000 metrics that can be monitored.

Disaster Recovery

While all parties have some documentation on disaster recovery. Azure is the only provider that has a service to automatically perform disaster recovery.

Penetration Testing

In Google Cloud, consumers are allowed to perform penetration tests on their cloud to test its security. The other vendors require some kind of authorization process.

Marketplace

While this is not in the scope of what CSPs provide themselves, it is worth mentioning that in terms of marketplaces, there is a major difference between the three providers. AWS has by far the most third party appliances, followed by Azure. The Google Cloud marketplace is still somewhat limited.

2.2 Attack Landscape

There is a vast amount of attack scenarios that are relevant for the cloud. In essence, all these attacks should be caught by an intrusion detection system, but not all of them are network related. For this thesis it is relevant to verify that network intrusion detection systems are not obsolete in a cloud setting. If all attack scenarios can be mitigated or detected with a HIDS or alternative log source, the purpose of this research would be defeated. In order to provide foundation for a cloud NIDS, we studied previous research in the field of network attack taxonomies [24][23]. Based on this research, we divide network based attacks in some basic categories. These categories are not exhaustive, but merely a basic classification of attack vectors relevant on the network. Furthermore, research from [28] provides an extensive taxonomy on attacks in cloud computing environments. It covers the attack surfaces within the different cloud layers and classifies the scenarios. Adversaries can launch attacks on different surfaces within the cloud. There are attacks for the network, the hosts and the hypervisor. Security measures in the hypervisor are the task of the CSP. Even though the network layer is completely on premises of the CSP, the data also flows through the users network. Therefore, network-based attacks are interesting within the scope of our research.

2.2.1 Reconnaissance, Scanning & Sniffing

Traffic-heavy reconnaissance attacks like most scanning attacks are trivial regarding their detection. NIDS should easily be able to detect the vast amount of data flow that differ from regular data flows. In a similar manner, HIDS can also detect scans across the entire environment. However, a single HIDS can only detect a scan to the host itself. When correlating attacks across the entire network, it is also possible to detect distributed scans. On the other hand, (passive) sniffing - the concept of copying or reading packets that flow across the network - is near impossible to detect with intrusion detection tools.

2.2.2 Network Based Attacks

In the taxonomy of Hansman et. al. a wide variety of network attacks is defined [23]. In their taxonomy we find: spoofing, session hijacking, wireless attacks, web application attacks, parameter tampering, cookie poisoning, database attacks and hidden field manipulation. While some of these attack vectors can be covered with hosts or firewall logs, the majority of these attacks are usually detected on the network. Especially web application attacks like XSS and CSRF are detected while they traverse through the network [25] [26]. The same goes for most spoofing attempts [36].

2.2.3 Denial of Service

Denial of Service (DoS) attacks come in many forms; most noticeably flooding attacks with different protocols such as TCP, UDP or ICMP floods. The distributed version (DDoS) is particularly popular. Cloud services are mainly vulnerable to network based DoS attacks, making this an interesting attack scenario. Evidently, the network itself it the place to detect these network based DoS attacks. Detecting denial of service on host level will often not suffice, as the goal is to flood the host.

2.2.4 Worms

Worms are standalone viruses that can replicate themselves between machines, rather than staying within a single infected machine. In order to spread, a worm often uses a computer network. Therefore, it is very much a network attack that is relevant for our research.

2.2.5 Phishing

Phishing is an attack that tries to mislead the target by disguising itself as a trustworthy entity. Typically phishing attacks use mail spoofing or instant messaging. There are many degrees of phishing. Some adversaries send mass phishing attacks under the assumption that some people will fall for it anyhow. Others specifically target certain instances or people with more sophisticated phishing attacks (spear phishing). There are also phishing attacks that forge entire websites, make use of phone calls, or social engineering. While there are advanced techniques that detect phishing attacks going over the network [40], a NIDS usually detects phishing attempts fairly simple. Firstly, phishing campaigns are often tracked with network based signatures. Additionally, when phishing is successful, credentials often traverse over the network in plaintext, which is easily tracked by NIDS.

2.2.6 Lateral Movement

Lateral movement is not so much an attack on itself. Rather it is used to create a larger attack-space for the intruder. Lateral movement is already relevant in traditional systems. An attacker can escalate his privileges or compromise additional machines via the initial attack. In cloud networks lateral movement expands its form as there might be VM escapes. This is when a malicious tenant breaks down the guest operating system to gain access to the hypervisor or other

guests, underlying operating systems and physical infrastructure [43]. Detecting lateral movement attacks with a network analysis tool has already been researched and shown feasible [47].

2.3 Summary

In this chapter, we have investigated cloud service providers from a security perspective. For each of the major CSPs, we have documented what security capabilities can be deployed. Given that fees apply for most of the advanced security mechanism, these features may or may not be a feasible option. We also identified the attack landscape of the cloud. The combination of network attacks and lack of network based detection mechanisms in cloud systems make this research purposeful.

Chapter 3

Challenges

The literature review has shown us that there are plenty network based attacks that are not covered by CSP countermeasures, HIDS or other log sources. There is no universal method that allows consumers to monitor their systems for network intrusions in an efficient, non reliant and complete manner. Now that we have sketched the defender and attacker landscape, we observe the challenges attached to the setting. The goal of this chapter is twofold. Firstly, the challenges brought up in this section will yield a proper motivation to conduct this research. Secondly, it helps identifying hazardous challenges that may arise during the research, so we can adapt our solution and tailor it to omit these challenges. The concept of a consumer deployable network intrusion detection system for public clouds surfaces challenges. Some of these emerge from the cloud platform, others from network intrusion detection, and most of them from the combination we intend to investigate. Therefore, this chapter is divided into two sections. In section 3.1, the critical aspects of cloud computing are identified. These aspects will be taken into account when they are deemed to be relevant in the network intrusion detection scope.

3.1 Critical Areas for Cloud Computing

Before looking into network intrusion detection, we need to examine the critical areas of cloud computing that require monitoring. The trusted computing group (TCG) identifies six areas for security concern in cloud computing that cloud consumers should take into account [22].

1. **Data at rest.** Any data stored on a cloud server
2. **Data in transit.** The data traversing through the network as well as ingress and egress traffic.
3. **Authentication.** Access control and corresponding policies
4. **Separation of customers.** One of the more novel security concerns is the separation of customers. In a cloud setting separation is logical (e.g., through VM(M)s). How is security accomplished and verified in this setting?
5. **Cloud legal and regulatory issues.** Policies and practices applied by the CSP. Part of the IaaS model is the outsourcing of your infrastructure. It is important to know what the CSP does on the infrastructure's end. This goes hand in hand with compliance, audit, legal actions and security policies.
6. **Incident response.** What happens when your cloud environment is breached?

Not all of these six factors are relevant when looking at network intrusion detection. However, there is often a relation between them. For example, data in transit is relevant as it traverses directly over the network. Data at rest is just placed somewhere on a server, not traversing the

network. Seemingly, this is less relevant in the network. However, network intrusion detection systems are able to monitor connection requests on the network. An example would be the detection of SQL injection signatures. The injection could easily tamper data at rest.

While all critical areas listed above have some relevant scenario for NIDS, there is a single one that is essentially different to traditional environments; separation between customers. While in on-premises there is a physical separation between customers or even within different domains of one customer, the public cloud uses the same physical hardware with a logical separation on top. This introduces some novel vulnerabilities as addressed in section 2.2.

3.2 Public Cloud NIDS Challenges

Now that we know the critical areas cloud computing produces, the hostile landscape, and the issues with NIDS in cloud settings, it is possible to define the challenges a consumer deployable NIDS will yield within this cloud platform. The challenges defined below yield motivation for conducting research and will be overcome by our final solution of performing NIDS in public cloud systems, while CSP interference is unnecessary.

- **NIDS deployment.** As described in the problem statement, deployment of NIDS is the main challenge addressed in this thesis. Since network data is not accessible, we need to deploy our NIDS on a logical level. This proves challenging. How can we accomplish this? Where do we deploy such logical NIDS? Front end, back end or both? What kind of attacks can we detect within our implemented solution? Internal, external and distributed attacks?

- **Loss of control.** Where is my data and who has access to it. The public cloud is essentially accessible and consumed by untrustworthy instances. That is, those who are not part of the organisation's policy.

- **Cloud as attack platform.** Opposed to traditional environments attackers can now exploit the entire VM and use it to attack other VMs or hosts that reside on that VM. In general, the cloud has shown to provide new security challenges [21]. The cloud's novel dynamic characteristics also require a different approach when trying to analyse intrusion possibilities [6].

- **Multi-tenancy - Attacks.** In public clouds it is possible that multiple virtual instances of separate consumers reside on the same physical machine. In this thesis there are two complications produced by multi-tenancy. Firstly, the cloud might yield novel attack scenarios that exploit vulnerabilities in logical separation to attack consumers on the same physical servers. There are already exploits where an attacker has managed to obtain information from neighbouring tenants [41].

- **Multi-tenancy - NIDS.** Secondly, since underlying hardware is shared between customers, cloud service providers will not send network logs to the consumers. For consumers it is not possible to access their own network data at switch level, because these switches are located on the premises of the service provider.

- **Ease of Use.** Ideally, a proposed public cloud NIDS would suit the cloud model. Easy subscribe and unsubscribe from the IDS service, payment for size and complexity and scalability with compliance of load balancing should be incorporated.

- **Traditional NIDS challenges.** We also identify some challenges that are still relevant for traditional network intrusion detection systems. These are probably hard to tackle in clouds, but still relevant. Examples of these challenges are: large amount of false positives, inability to identify false negatives and not being able to deal with encrypted data.

The challenges above will fuel our research and are referred to when establishing requirements of a public cloud NIDS.

Chapter 4

Requirements

In order to determine which NIDS deployment technique would perform best, we first need to define what would be required from a given technique. Since this research is devoted to finding a consumer deployable network intrusion detection system for public clouds, there are three major concerns which we can base requirements on; public clouds, NIDS and consumers.

4.1 Methodology

As described above, there are three major aspects in our research. Each of them has its own requirements. In order to define clear requirements, we address recommendations by authorities in cloud and IDS fields. To answer this research question, we define step-by-step prerequisites for public clouds, NIDS and consumers. Since the final product of this thesis should be a network intrusion detection technique, the requirements are based on the NIDS but tailored to the cloud and its users. First, we define cloud requirements for the NIDS. These are based on cloud characteristics. Secondly, we define NIDS requirements. These are mostly measures and recommendations that apply to traditional NIDSs. Lastly, we define utility requirements that cover management and cost aspects that are relevant for the consumer.

4.2 Public Cloud IaaS Aspects

The novel aspect of this research is migrating an existing solution (NIDS) to a public cloud setting. Consequently, it is desirable to incorporate the characteristics of the cloud in the new NIDS implementation. In section 1.1 it was established that cloud services require five aspects that differentiate them from traditional computing services. Therefore, the public cloud NIDS should also be able to cope with these characteristics.

- **On-Demand Self-Service.** Since consumers can provision any public cloud instance themselves, the IDS should be able to deal with newly deployed or removed instances.
- **Broad Network Access.** Standard access via the network does not demand particular capabilities from a NIDS service. However, it has to be taken into account that access control in the network layer is a vital aspect when abundant client platforms and services should be able to connect to the network. Furthermore, employing NIDS based on anomalous flows or states might be difficult when connecting devices vary a lot.
- **Resource Pooling.** Resource pooling is a partial reason why we cannot deploy NIDS in a traditional way (multiple tenants on the same physical hardware). Therefore, we are researching logical alternatives for a NIDS. However, resource pooling itself yields no additional requirements.

- **Rapid Elasticity.** In cloud environments, consumers should be able to dynamically or automatically scale their infrastructure's capabilities. Simultaneously, the deployed NIDS should be able to scale with the infrastructure, so the throughput can be handled in (near) real time.
- **Measured Service.** No additional relevant aspects.

4.3 NIDS Aspects

Requirements of a NIDS usually vary a lot per organisation. How a NIDS performs based on requirements is also prone to the user's configurations. These characteristics create a difficult platform to formulate requirements for. However, there are aspects that can be measured or ticked off. These measures should be implemented sufficiently for any deployed NIDS solution. The NIST defines the following specialised set of requirements: security capabilities, performance, management and life cycle costs [33]. This section will yield important characteristics that can be measured in order to test a NIDS. Based on these recommendations and best practices suggested by the NIST [33] and CSA [9], we have defined the important aspects of a NIDS.

4.3.1 Security Capabilities

On the subject of security capabilities, we investigate three aspects. These three aspects are aligned with the three components of the NIDS. Firstly, the sensors that collect information. Secondly, the main component that analyses the data. Lastly, the output process of the NIDS for a user or SIEM system.

- **Information Gathering Capabilities.** The first step of monitoring the network is to actually gather data. The way you setup and deploy your sensors is key for a quality NIDS service. Essentially, the sensors have to be configured in a way to effectively gather and filter the data.
- **Detection Capabilities.** Detection capabilities of the NIDS are probably the most important factor under the security heading. There are quite a few measures that one has to take into account when evaluating the IDS's detection capabilities. The most prominent aspects are: listings of protocols that can be analysed, the types of attacks it can discover, the effectiveness of default configurations, how the NIDS copes with known and unknown attacks, detection accuracy and integration with other IDS sources. Furthermore, detection capabilities also rely on whether the NIDS is effective at detecting attacks covered with IDS evasion techniques. Can the NIDS detect adversaries using packet fragmentation, stream segmentation, obfuscation and other evasion mechanisms. However, in this research the detection accuracy of the IDS itself (e.g., Snort) is not the main concern. The focus lies on architectural design; where to place your detection and analysis units in order to, for example, generate the least false positives.
- **Logging Capabilities.** An important aspect of any detection mechanism is to log findings appropriately. For a NIDS there are a few aspects that are required to properly implement the service. Firstly, logs should be available for analysis at all times. Therefore, a proper implementation with central and local storage is required. This way integrity and availability can be upheld. Another essential requirement is that the clocks of different log sources are synchronised properly so events can be correlate appropriately. This can be quite challenging in cloud services as many clock synchronisation tools are not guaranteed to work in virtual machines.

4.3.2 Performance

Performance is dependent on security capabilities and configurations, so it is once again difficult to determine. To the best of our knowledge, there are no open standards for testing the performance

of a NIDS. However, it is possible to stress the NIDS with maximum throughput and stability tests. To do this we consult NSSLabs, an organisation with expertise in testing security solutions. The test that aligns most with NIDS is their next generation intrusion prevention system methodology [37] for network security.

Combining this test methodology with the NIST's recommendations, we can look at the number of packets per second that the NIDS can process while maintaining high level security capabilities. The test can be executed with different protocols, packet sizes and packet amounts. Furthermore, the test is dependent on the deployed technique. For inline NIDS we are interested in the latency of processing typical and extreme workloads. For out of band NIDS we are interested in the time between event occurrence and reporting.

Stability

An important factor of a properly performing NIDS is the stability of the system. In this notion, stability has nothing to do with raw performance. However, a performance test can affect stability, and therefore also reliability, of the NIDS. For example, the NIDS's detection capabilities should remain functional under high loads.

4.4 Utility Aspects

Utility aspects are those factors that are important for the cloud consumer. These are mostly covered by performance and management. Since the cloud usually provides a pay-per-use service, we also take into account life cycle costs.

4.4.1 Management

For process continuity it is vital that the deployment of our IDS solution does not hinder the consumer in a disruptive manner. To deal with this, the NIDS should be implemented in abidance of the following aspects:

- **Failover Process.** When there is an issue within the consumers environment, whether that is in our NIDS or on a server, the entire infrastructure should not be affected. For our NIDS it is important to think about design; are there redundant services or is there a single point of failure?
- **Easy to Deploy.** Since our solution requires implementation at consumer level, it should be relatively easy to configure and maintain. If the NIDS is susceptible to configuration failure, or requires a total overhaul of the network, it will be infeasible to work with.
- **Load Balancing.** NIDS can be a computationally heavy process as it scales with the amount of data traversing the consumer's network. When all the data has to be processed by a single IDS machine, it might get overloaded.
- **NIDS's Security.** The NIDS should not be susceptible to attacks. Therefore, the design should take into account aspects like attack resistance, access control, evasion attempts and data protection.

4.4.2 Life Cycle Costs

One aspect that remains an important factor for organisations is the cost of implementation and maintenance. In a cloud setting you usually pay per use. Requiring a lot of instances or a lot of data-transfer will add up to the monthly costs of the NIDS service.

- **Deployment Costs.** The costs to deploy the solution.

- **Instance Costs.** The costs to keep the solution up and running. In the cloud you need to think about the size of an instance (scalable) and data transfers. The more processing required on an instance, the larger is has to be, the more it costs. The same goes for data storage. Similarly, ingress and egress data transfers usually come with a pay-per-use model.

4.5 Requirements

In the previous section, we identified all the important characteristics a consumer deployable public cloud NIDS comprises. This section maps these characteristics on measurable or arguable requirements.

4.5.1 Cloud Requirements

For the cloud we have five characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. We already defined that measured service and resource pooling provide no additional characteristics. For on-demand self service, we need our NIDS to monitor any new instance deployed by the consumer. For broad network access, we need to be able to monitor any device connecting to any cloud instance. The rapid elasticity characteristic entails that the consumer's environment can scale in real time, this means that our solution should scale with this environment to not get overloaded with data. This provides the following three requirements:

1. The NIDS should monitor newly configured instances.
2. The NIDS should be able to scale with the environment and data throughput.
3. The NIDS should work across the entire (cloud) environment.

4.5.2 NIDS Requirements

For NIDS requirements we consulted guidelines regarding traditional NIDS systems. Guidelines are not direct requirements, as per definition they only provide the consumer direction. Nonetheless, well-founded sources like the NIST are proper foundation as to what a NIDS should abide to. We divided the NIDS characteristics in three subsections; security capabilities, performance and stability. From these categories we extracted the following requirements:

1. The NIDS should be able to process legitimate traffic while under attack.
2. The NIDS should not allow malicious traffic to pass through (false-negative).[1]
3. False-positives should be as low as possible.[1]
4. The NIDS should be able to cope with high throughput without losing speed or security capabilities.
5. Latency of the NIDS should be as low as possible.
6. The NIDS should affect the (other) host(s) as little as possible.
7. The NIDS itself should be resistant against attacks and evasions.[1]

[1] These requirements are dependant on the detection algorithm, which is not in the scope of this thesis

4.5.3 Utility Requirements

Lastly, this research is relevant for consumers whom want to apply security while embracing the benefits of a cloud system. However, this also means we cannot expect a complex approach that involves a lot of manual engineering. The utility characteristics are divided in costs and management requirements:

1. The NIDS should be easy to deploy and configure for a cloud consumer.
2. The NIDS should be easy to integrate without network redesign.
3. Deployment and maintenance costs should be as low as possible.

4.6 Summary

First, we investigated the important characteristics of public clouds, NIDS and the utilities surrounding this setting. Qualitative and quantitative requirements were defined conform those characteristics. The cloud and utility characteristics yielded qualitative requirements. These qualitative requirements can be scored directly while analysing them theoretically. This will be accomplished in chapter 5, which corresponds with research question two of section 1.5.2. The quantitative requirements lead to quantitative questions that need to be measured and verified in a heuristic experiment. The experiment will be conducted based on the best theoretical method from chapter 5. The experiment is described in chapter 6 and corresponds with research question 3 of section 1.5.3.

4.7 Conclusion

In this chapter we answered research question 1: *What are the requirements for deploying a consumer deployable NIDS for public clouds?* We started off defining all important aspects for a public cloud NIDS. Subsequently, formalising requirements for each aspect of a consumer deployable NIDS for public clouds. We identified requirements based on the cloud, the NIDS, and utilisation. The list of requirements was trimmed down in 4.5, as the scope of this research does not include an analysis of the capabilities of the implemented IDS agent (footnote 1). This leaves us with ten requirements as defined in section 4.5.

Chapter 5

Theoretical Analysis of Cloud NIDS Methods

5.1 Methodology

Research question two is dedicated to finding the theoretically best method for performing network intrusion detection in a public cloud setting. We dive into academic literature to find approaches towards public cloud network intrusion detection. To simplify the approach, the research is divided into three sections. Firstly, we look at some intuitive IDS concepts. These are generic implementations that migrate NIDS capabilities that traditionally run on taps or switches to the cloud. Secondly, we really dive into academic papers to find methods for intrusion detection in cloud systems. Each of these methods is evaluated based on the requirements defined in chapter 4. Sections 5.2 and 5.3 describe each method along with key pros and cons. A full enumeration of pros and cons is displayed in tables 5.1, 5.2 and 5.3. Within these tables entries annotated with a plus sign are pros, entries with a minus sign are cons and entries with a plus-minus sign are minor pros, cons or conditional statements. Based on these findings we will conduct an extensive review that allows us to compare the methods and decide on a theoretically best method. Hence, we will conclude this chapter with a quantitative review of the best choices based on what is required from a public cloud NIDS.

5.2 Conceptual Approaches

This section covers conceptual approaches that could be used to implement NIDS on different architectural levels as opposed to a traditional on premises setting. The methods are intuitive and not abstracted from literature on cloud IDSs.

5.2.1 Bump in the Wire

Bump in the wire uses an extra compute instance, usually in form of a virtual machine, that functions as a legitimate man-in-the-middle box between two communicating machines. These connections can be ingress, egress or locally, as long as the routing configurations are correct. The bump in the wire instance is based on layer 3 traffic opposed to layer 1 and 2 in a traditional setting. This requires the user to actively configure IP routing that sends traffic through the bump in the wire instance. A virtual NIC on the box is employed as capture interface allowing it to apply NIDS functionality or to create a full packet capture. This NIC is then configured to forward the packet to its original destination.

Main Pros

1. Suitable for the pay-per-use model

Main Cons

1. Requires network/route reconfigurations
2. Single point of failure
3. Does not scale with the network

5.2.2 Host based network intrusion detection

NIDS sensors are traditionally deployed in network switches or hubs. Since there is no direct access to the network and infrastructure for consumers in public clouds, an alternative would be to deploy NIDS functionality on every host in the network. Since the end user utilises decrypted data and is not part of an encrypted tunnel, this method could compensate for the encryption challenge that NIDS usually faces.

Main Pros

1. Distributed
2. IDS scales with the environment
3. Easy to deploy

Main Cons

1. CPU intensive process on every host
2. Cannot easily correlate data from different servers

5.2.3 Host based network traffic replication

This method is similar to the one above. NIDS functionality would once again be performed on the end-host. However, this time the end-host forwards the data to another dedicated IDS host. Depending on the implementation, these IDS hosts could be rolled out dynamically, creating a scalable structure.

Main Pros

1. Distributed
2. IDS scales with the environment

Main Cons

1. Increase in network traffic
2. Requires secure tunnels or a dedicated environment to transmit packet captures

5.3 State of the Art Approaches

This section cover NIDS methods found in academic research about intrusion detection in public clouds. The methods listed are a subset of cloud NIDS research as we are primarily interested in approaches that have distinct deployment configurations. Within the scope of this section, there might be some overlap with the previously mentioned conceptual approaches.

5.3.1 IDSaaS

Intrusion Detection System as a Service [2] in public clouds is one of the concepts that covers the goal of this thesis. The method provides a highly elastic method for cloud consumers to deploy IDS capabilities in their public cloud environment. This is accomplished by employing the private networking capabilities that public cloud IaaS providers often offer. In this particular case, the research aimed at a construction tailored for Amazon's AWS. The researchers deployed a Virtual Private Cloud (VPC) on top of their AWS public cloud infrastructure. Within this cloud a public and a private subnet were created. All inbound traffic flows through the internet gateway into the public subnet. The public subnet consists of a dynamic amount of identical IDS servers, which are controlled by the IDSaaS manager. The manager service is used as access point to configure other VMs in both public an private subnets. A load balancer makes sure the inbound traffic is distributed over multiple IDS servers. Only if the traffic is clear, it is forwarded to its destination in the private subnet. In the public subnet the consumers protected property, such as web and database servers, is stationed. Any outbound traffic from the private subnet is then routed through network address translation (NAT). The security is enhanced by deploying security groups for each deployed instance in the VPC. In a sense this method is a variant of the bump in the wire concept mentioned in section 5.2.1. However, the method has its own dedicated environment that does not require alteration of the existing network.

Main Pros

1. Dynamic/Elastic amount of instances/compute power
2. Uses native services that (all) major CSPs offer
3. Easy to deploy and configure
4. Complies with pay-per-use

Main Cons

1. IDS functions on the network boundary
2. Does not function in a distributed environment
3. Requires redesign of the overall network architecture

5.3.2 Nested Virtualization

Before introducing the following two NIDS methodologies, it is required to familiarise ourselves with the concept of nested virtualisation. Large CSPs, such as AWS and Google, disallow traffic to be sniffed via a promiscuous interface [5]. The hypervisor, which is inaccessible for consumers, will not deliver any data to instances that the data is not addressed to. An existing implementation of creating an extra virtual layer on top of your cloud is called nested virtualisation. This enables the user to abstract from the underlying cloud infrastructure. The most well known nested virtualisation that works on public clouds is HVX [16]. HVX has many advantages over the underlying public cloud, such as a full layer 2 and layer 3 network architecture. Since HVX has its own manageable hypervisor we can allow port mirroring in software switches via promiscuous mode. This would be a prerequisite of devising any public cloud NIDS method that makes use of software switches or other promiscuous interfaces. In the upcoming two subsections software switching implementations are presented.

5.3.3 Cloud based Intrusion Detection Service (CBIDS)

Cloud based Intrusion Detection Service (CBIDS) [48] is a framework that can be deployed as a service. It employs a user data collector (UDC) that is integrated in the client's cloud as an independent secured proxy-server. The UDC collects data via virtual SPAN ports in virtual switches that connect to end-host virtual machines. Only the mirror ports from virtual switches are allowed to access the proxy. The UDC then tunnels the data over a secure VPN connection to the Cloud IDS. In the separate Cloud IDS a service component is in charge of analysing and validating inbound traffic to find external intrusions before deciding to forward it to the actual IDS component or to delete it. This method might be interesting, because the externally hosted cloud IDS might as well be hosted by the consumer or a third party of their choice.

Main Pros

1. The Cloud IDS can be hosted on premises
2. Detection works similar to on premises

Main Cons

1. Lots of traffic replication
2. Proxy-bottleneck
3. Nested virtualization needed

5.3.4 Cloud Intrusion Detection System Service (CIDSS)

CIDSS [50] is designed around the software as a service model for cloud users. It consists of multiple lightweight IDS agents that are deployed in the user's network. These agents are grouped based on rulesets and thresholds to improve efficiency and protection flexibility. The data can then be collected, for example by virtual taps, as in the CBIDS paper [48]. The group based structure can provide flexibility to support high bandwidth networks. Furthermore, smart packet filtering is applied to reduce the workload of the underlying service components, allowing near real-time intrusion detection.

Main Pros

1. Agent Grouping to distribute load
2. IDS service works across different network segments

Main Cons

1. Complex for large infrastructure
2. Lack of Scalability
3. Nested virtualization needed

5.3.5 Distributed IDS with Mobile Agents

The Distributed IDS with Mobile Agents proposal [10] is a combination of a peer-to-peer based IDS [49] and the DIDMA mobile agents IDS [29]. The method makes use of an agent deployment toolkit that launches IDS instances in VMs. These mobile agents come in two forms. Static agents are deployed in an agency (sandbox) of a VM. Typically these consist of standard static IDS solutions such as Snort. The agencies send out periodical heartbeats to show they are still functioning. The second form is a group of investigative mobile agents. An IDS control center

deploys these mobile agents to aforementioned agencies. The mobile agents collect, correlate and carry the information back to the control center. The control center will perform intrusion matching to generate alarms. Subsequently, it will store new or updated intrusion data in its database. The peer-to-peer aspect creates an IDS environment that covers multiple subnets. Furthermore, the agents promote a symmetrical distribution of the network load and allow for more attack resistibility, as there is no single point of failure within the IDS. Another important aspect is that the system is highly scalable and can easily be relocated between cloud environments.

Main Pros

1. Scalable
2. Sandboxed IDS environment
3. Application specific detection (limiting required computation power)
4. Peer-to-peer model to connect IDSs between subnets

Main Cons

1. No central IDS learning or knowledge
2. Intensive process on every host
3. Heartbeats can be spoofed

5.3.6 NICE

NICE [8] is a multiphase distributed NIDS and NIPS (prevention) framework which was designed for a XEN virtualization environment. Its aim is to capture and inspect cloud traffic that is suspicious. NICE incorporates software switches to quarantine suspicious VMs. It accomplishes this by deploying a NICE-A IDS engine in Dom0 (CSP Reliant) or DomU of each cloud server. Since this research is dedicated to finding a public cloud solution, the Dom0 aspect will be taken out. For the DomU, the engine will sniff a mirroring port on each virtual bridge in the Open vSwitch (a multilayer virtual switch). The IDS capabilities of NICE-A rely on the IDS engine. Furthermore, NICE uses attack graphs [39] to detect and prevent attacks by correlating attack behaviour. Lastly, NICE includes implementation optimisation to minimise resource consumption in comparison to other proxy based NIDS. These features are desirable for a fully consumer reliant NIDS.

Main Pros

1. Minimized resource consumption
2. Distributed

Main Cons

1. Dependant on underlying infrastructure
2. Full implementation requires Dom0 access
3. Benign CSP assumed
4. Centralized network control (unscalable)

5.3.7 H-NIDS

Hybrid network intrusion detection system (H-NIDS) [35] is a technique that incorporates signature and anomaly based detection to improve detection accuracy. The capturing of data happens on the host OS, where inbound and outbound traffic is sniffed and passed to the signature based component. The signature based detection consists of two NIDS components. A captured packet will be fed to Snort and signature apriori based NIDS. The latter will update the database of attack rules that is used as input for Snort. If the packets pass through Snort, they are routed through the anomaly detection phase. Here, the packets are preprocessed for machine learning classifiers. H-NIDS employs three types of classifiers: Bayesian, associative and decision trees. These three classifiers in combination with Snort will score the packet and decide whether to pass it to the cloud resources or to generate an alert. As a final step, malicious scored packets will be used as input for the knowledge database.

Main Pros

1. Distributed
2. Scalable
3. Works similar to a traditional inline IDS/IPS

Main Cons

1. Latency of the traffic
2. Requires nested virtualization

The latter three NIDS methods all possess distributed characteristics. In these descriptions we have seen fully distributed (peer-to-peer) and centralised approaches. Note that there is also a hierarchical principle that sits in between centralised and fully distributed. In this setting the entire infrastructure is based on hierarchical groups that send processed data upwards. In general, a more centralised approach is less scalable than a more distributed approach. On the other hand, the more distributed approaches introduce more communication and processes.

Table 5.1: Revision Cloud Requirements

	On-Demand Self-Service	**Broad Network Access**	**Rapid Elasticity**
Bump in the Wire	- New instances need to be configured in the route config.	+ No complications.	- The bump in the wire box has to be actively configured to map IP forwarding. - New boxes are hard to scale up with the network.
Inline Host	+ IDS resides on the host.	+ No complications.	+ Host based scales easily.
Out of band Host	+ IDS resides on the host.	+ No complications.	+ Host based scales easily.
IDSaaS	+ New instances can be deployed in private subdomains. This does not affect the NIDS.	+ No complications.	+ Load balancing should take care of this, automatically generating more IDS instances.

CBIDS	+ New instances can always be deployed since switches deal with incoming and outgoing data automatically. However, in the cloud virtual switches are by default implemented in the hypervisor. Since the hypervisor is inaccessible in the public cloud we need to configure our own dedicated switch VM.	+ No complications.	+ New endhosts should be able to be deployed as the virtual switches' span ports do not tamper with other infrastructure.
CIDSS	± Paper defines very vaguely how an IDS agent is deployed: "The agent is a lightweight, single purpose equipment (dedicated hardware or software) integrated inside the user network to collect necessary information."	+ No complications.	± New instances will have to be assigned to a rule set or group.
Distributed P2P-MA	+ Discovery of new agencies/instances will be automatic.	+ No complications; should even work between different subnets.	+ New instances will have their own IDS agency.
NICE	+ IDS is applied via a mirroring port in a virtual switch. Deployment of new instances should not affect this.	- Detection based on VM profiles might be hard when connected devices vary a lot. This might also generate false alarms.	± Intuitively the agent system becomes complex when machines are launched or taken down. For example, is the machine taken down, or is it compromised and not sending heartbeats?
H-NIDS	+ Intrusion detection is performed on the host itself via a packet capture.	+ No complications.	+ New IDS scale with the deployment of new instances. Only the central knowledge database is shared.

Table 5.2: Revision NIDS Security Capabilities

	Information Gathering	**Detection (deployment related detection measures)**	**Logging**
Bump in the Wire	- Only able to examine data that is actively routed through the box.	- Cannot detect distributed attacks.	- Extensive process to log all captures routed through the box.
Inline Host	+ All inbound and outbound data is seen. + Possible to tune the IDS per host.	± Attacks traversing the environment can only be detected if they all trigger their own alarm.	+ Logs can be stored in a cloud instance while alerts can be stored in the central alerting system.
Out of band Host	+ All inbound and outbound data is seen. + Easy to filter out irrelevant packets before sending them to the IDS.	+ All captures can be correlated in a central host.	+ Dedicated IDS host can process logs.
IDSaaS	+ All inbound and outbound data is seen and easily analysed due to load balancing. - Only functional on network boundary.	+ If the IDS uses a shared database, distributed attacks can be detected and events can be correlated between IDS cores.	+ Shared event database for all IDS cores.
CBIDS	+ All data flowing through the virtual switches. - Prone to packet drops for high workloads.	+ Uses virtual instances of what is usually deployed on premises. Should be able to detect the same.	± Dependant on the IDS solution in the analysis engine.
CIDSS	+ All data flowing through the virtual switches. - Prone to packet drops for high workloads.	+ Ring buffer to prevent packet loss. - Bandwidth-wise it would only be capable of doing header inspections.	+ Unified detection instance that combines standardised events from all agent groups.
Distributed P2P-MA	+ Every VM has a static agent in its agency checking for intrusions in a host manner. This should see all targeted attacks, while a control center checks all the suspicious traffic provided by the mobile agents.	+ Works well against distributed attacks - No central learning or knowledge.	+ Alerts are mainly generated by the static agents in the agency. These are collected by the mobile agents and sent to a central alerting console that can correlate and transform SA suspicion into actual alarms.
NICE	+ Makes use of system information, configuration, topology and vulnerability information.	+ Works against collaborative attacks. + Effective against external and internal attackers. - Does not work when multiple cloud servers are involved.	+ Alerts from NICE-A are processed in the IDS control center. Logging should be possible after the alert analyzer has correlated the event; minimising false positives. The paper does not go into detail about logging as it performs IPS.
H-NIDS	+ All inbound and outbound data is seen.	+ Effective against external and internal attacks. + Detects distributed attacks with a centralised database.	+ Logs alerts directly and updates a centralised knowledge or behaviour database.

Table 5.3: Revision NIDS Usability

	Performance	**Management**	**Life Cycle Costs**	**Other**
Bump in the Wire	- The inline bump will be the bottleneck of the network.	- Implementing routing tables can become complex, especially for large organisations.	+ Only cost is the bump in the wire instance.	- Single point of failure for the IDS capability.
Inline Host	- Additional CPU overhead on every host. ± Additional latency for all traffic passing the NIDS in IPS mode.	+ IDS comes with deployed hosts.	+ You already have the instance for your host.	+ Possible to use as IPS.
Out of band Host	± Small process on the host to copy data. - A lot of retransmissions.	+ IDS agent comes with deployed hosts. - All agencies need a secure tunnel to the IDS machine to prevent spoofing.	± Depends on whether the out of band IDS resides on the same host (IDS vs IPS mode).	
IDSaaS	+ Performance should be good due to its scalable model. ± Additional NIDS latency might be present in the inline setting; depending on whether the host forwards the packet immediately or works as a prevention unit simultaneously.	+ Easy to deploy with native cloud services.	+ Pay-per-use model. + New instance required for every upscale.	+ No single point of failure. - Does not work with a distributed infrastructure.
CBIDS	- A lot of traffic replication. - Low throughput due to proxy-bottleneck.	- Dedicated switches would have to be self-configured. - Requires HVX.	± Depends how much data you tunnel, but no extra instance costs.	
CIDSS	± Does not work well with high network bandwidth. + Improves performance by using rulesets per group. + Agents perform early filtering to reduce load on the analysis engine.	± Relatively easy to configure, but gets harder for dynamic or expanding networks. - Requires HVX.	- CCSCs instances for every agent group and IDSCs for every client	- Promiscuous mode is not allowed in most CSP's NICs.

Distributed P2P-MA	+ Application specific detection. + Symmetrical distribution of the network load. - Overhead due to requiring an IDS agency on every host. - Alarm latency due to active MA polling of agencies.	± IDS agencies come with the hosts. The deployment of MAs might be more complex.	+ The control center is the only additional instance	+ Sandboxed IDS agencies. + No single point of failure. - The IDS itself might be vulnerable to spoofing the included heartbeat messages.
NICE	± DomU NICE is slower than Dom0 (we cannot use the latter). - Attack graph generation is computationally heavy, as it requires system information, network topology and configurations as well as actively polled vulnerability information (e.g., vulnerability scanning and pentesting).	- Complex to self-configure; would require out of the box machine images.	- Instances required for bridging as well as for VM profiling and attack analysing.	± NICE-A also takes into account countermeasure selection and attack mitigation. This focus is meaningful and qualitative, but also generates more complexity and overhead.
H-NIDS	+ Low computational and communication costs. - Latency for the detection and feedback phase (less relevant if only Snort is used).	+ IDS comes with deployed hosts. - Requires HVX.	+ Only additional costs for centralised storage and knowledge.	

5.4 Revision

The revision of pros and cons in tables 5.1, 5.2 and 5.3 of user deployable network intrusion detection techniques allows us to construct a verdict on each method. In the three sections below, each method is evaluated based on our finding regarding the cloud, NIDS performance, and utility aspects. This will lead to the most feasible NIDS option.

5.4.1 Cloud Revision

In the cloud setting we defined three aspects that are relevant for the NIDS to deal with; on-demand self-service, broad network access and rapid elasticity. For the on-demand self-service we found that most methods should not be affected, as they either come pre-installed on the host or are deployed somewhere in the network and should easily be able to cope with new devices. The only methods that deemed unsuitable are bump in the wire and CIDSS. Bump in the wire requires the user to actively update layer 3 routing information between hosts in the route config. This is possible but tedious and therefore does not suit the cloud model. For CIDSS, deployment

configurations of agents is too abstract for us to define whether it would be easy to deal with newly deployed cloud instances.

In the broad network access scope our overall finding is that whether the instance is used for a single type of device or many different kinds, the NIDS rules should be able to deal with device-based vulnerabilities. While it is possible to create per-device rules, detection based on VM profiles seems rather complex. Therefore the NIDS technique might prove either complex to configure correctly or generate a lot of false positives when an instance suddenly receives a lot of anomalous traffic due to connecting of benign alternative devices.

Lastly, it was revised how each method would cope with upscaling and downscaling of the user network. Rapid elasticity is one of the most attractive aspects for consumers. Therefore it is important that the NIDS is able to scale with the environment as well. A lot of cloud methods have taken this into account. In general we find that all host based methods scale easily. When a new host is deployed, it automatically has its own NIDS agent or service. Furthermore, we found that the IDSaaS solution takes into account the amount of traffic that flows into the user network and spins up or takes down IDS instances based on that information. Lastly, there are some software switching options. These should be able to deal with increasing and decreasing network traffic. However, once the bandwidth is exceeded packets might be dropped as with physical switches.

Overall we found that CIDSS, NICE and Bump in the Wire solutions might prove difficult or infeasible for a public cloud setting. Host based solutions like H-NIDS and the Distributed P2P-MA would perform adequately, as well as creating a dedicated and automatic environment for network intrusion detection such as in IDSaaS. For the cloud based requirements, software switches seem like a decent solution, therefore CBIDS is still a feasible option.

5.4.2 NIDS Revision

One important aspect of NIDS is that it should be able to view all the relevant network traffic. In this area two of the methods were found to perform insufficient. Firstly, bump in the wire would only see traffic that is actively routed through this box. Any traffic going directly to or from other instances will not be inspected. It seems like this NIDS could easily be evaded by attackers. Secondly, IDSaaS uses a public subnet for all IDS capabilities. This works great for detecting malicious activities in traffic coming from outside the network. However, it does not work well for traffic between hosts inside the private network. A work around for this could be to route all traffic through the public IDS network. However, this would generate a lot of additional data transfer. Furthermore, it might be unsecure to route private data through the already existing public IDS subnet.

The second requirement that was looked into is detection capabilities. Review of detection capabilities is limited, since this research does not take into account what IDS appliance is deployed (e.g., Snort), nor any configuration constraints. However, it is possible to define whether the deployed method can detect a certain type of attack due to its implementation design. If an attack is launched against different targets within the user's network, a host based solution that is not relying on a central knowledge or alarm database will most likely not detect the attack. One of the main aspects is whether the NIDS can detect attacks that are distributed throughout the network. Furthermore, in environments using proxy-like behaviour, such as bump in the wire and CIDSS, it is infeasible to look at all the network traffic routed through the box.

The last aspect of a NIDS is its logging capability. Much like the detection capabilities, logging is dependant on the implemented IDS appliance. There are two important aspects here; the way alarms are escalated to the user and the way logs are stored. For all of the solutions, apart from bump in the wire, logging should not be a problem if implemented correctly.

Overall we find that most of the methods are compliant with the general NIDS capabilities as they are defined in our requirements. However, when being punctilious, only H-NIDS and the two host based methods fully comply to all standards.

5.4.3 Utility Revision

The utility revision is based on consumer and performance requirements of the public cloud NIDS technique. For the performance aspect there are a lot of aspects such as maximum throughput, latency and overhead that can only be validated in an experiment. However, we recognise some intuitive behaviour which we will address here. A tunnel based solution where all data has to be routed through a single point will introduce a bottleneck. All host based solutions introduce additional CPU overhead on every host. However, this way there is no additional bottleneck as the load is distributed and no additional network configurations are necessary. Consequently, the entire network is not affected by adding the IDS to the environment, making it a more feasible option. Due to this, the CBIDS and bump in the wire solutions are undesirable for the IDS' performance. The method using mobile agents ships code to the VMs, rather than data to a computation unit. Research by Dastjerdi showed that this implementation is auspicious when the mobile agent has to visit up to six VMs [10]. For larger networks, a client-server approach performed better in relation to the network load.

For the management aspect we looked at how easily consumers can deploy the NIDS and whether it integrates in already existing cloud environments. For consumers that already have (part of) their environment in the cloud, it is undesirable if the implementation affects their current infrastructure. Consequently, implementations using software switches (CBIDS and CIDSS) are unwanted as they cannot be implemented without disrupting the already in place network due to the necessity of HVX. The same goes for a bump in the wire solution, as all routing has to be reconfigured. HNIDS sensors are configured in the hypervisor of the host OS. To accomplish this in a consumers setting, we would need HVX as well. Lastly, the implementation complexity of NICE makes is less desirable.

Besides management and performance, costs are an important feature for consumers. Especially when taking into account the pay-per-use model of the cloud. Most public cloud service providers require payments per instance per month in addition to storage and data transfer fees. If a public cloud NIDS solution introduces a lot of additional resources, it might become infeasible. If the IDS runs on existing instances, extra costs of new instances will be mitigated. Therefore host based IDSes are cost effective. However, out of band solutions do require the packets to be copied to an additional instance generating additional instance and data transfer costs. CIDSS and NICE require a lot of additional instances to be spun up in the virtual environment. The necessity of additional resources is much less in the H-NIDS and distributed P2P-MA implementations, as there is only one auxiliary node.

5.4.4 Summary

Based on the pros and cons found for each of the three main aspects of the public cloud NIDS for consumers, we can now compose a verdict. For the cloud aspect most methods did well, but CIDSS and bump in the wire turned out to be insufficient. As a NIDS, we only found that the bump in the wire construction would perform inadequately. Additionally, in its standard configuration, IDSaaS would only perform on the network boundary, unless internal traffic is routed through the detection units. Lastly, when looking at usability we found very varying results between performance and management. Based on performance NICE, P2P-MA, CBIDS, inline host based methods and bump in the wire are below average. Based on management its CBIDS, CIDSS, NICE, H-NIDS and bump in the wire that perform insufficiently.

When cross referencing the analysis between sections, we find that CBIDS, CIDSS, NICE and bump in the wire do not perform adequately on more than one of the three analysed aspects. This makes them insufficient to be taken into account for implementation. The remaining methods all have their pros and cons on different levels.

To obtain an overview of where methods lack in comparison to our requirements, we will map the remaining methods on the requirements.

5.5 Requirement Analysis

Before implementing the theoretically best method for our public cloud NIDS solution, we can first investigate whether the remaining methods comply with the requirements defined in section 4.6. Most requirements were analysed within the scope of the extensive revision, but not effectively mapped to what functions should be in place. Table 5.4 shows what requirements are fulfilled for each of the three methods that were qualified as solution during the extensive revision.

Table 5.4: Qualitative Requirement Fulfilment

Requirement	Inline	OOB	IDSaaS	P2P-MA	H-NIDS
Can monitor newly configured instances	✓	✓	✓	✓	✓
Able to scale with the environment	✓	✓	✓	✓	✓
Works across the entire (cloud) environment	✓	✓	✗	✓	?
Can process legitimate traffic when under attack	?	✓	✓	?	?
Easy to deploy and configure	✓	✓	✓	✗	✓
Easy to integrate without network redesign	✓	✓	✓	✓	✗

From table 5.4 we find that only out of band host based NIDS fully complies with our requirements. However, the inline method's only deficiency is that we do not know whether the NIDS can process traffic when under heavy load. However, when an attack is targeted to a single victim, we do know that only that particular host's NIDS is affected as there is an instance on every host. Furthermore, whether or not the NIDS is capable of processing legitimate traffic is dependant on what IDS solution is used in the process. Therefore, we decide to let both inline and out of band methods comply with our requirement set.

When analysing the other three methods we find major shortages. IDSaaS is deployed on a different subnet. This entails that IDSaaS is unable to see any attack that originates from within the private subnet. Intuitively, IDSaaS functions much like a next generation firewall. P2P Mobile Agents seem to work well on most cloud and detection based requirements. However, when taking into account the usability of the system, we run into some difficulties. Firstly, every host in the network will need an agency, which needs to be connected via peer-to-peer. Furthermore, the cloud network needs to be supplemented with a mobile agent toolkit, which is not a readily available cloud service and therefore has to be implemented or hosted by a third party. Additionally we found that the model of shipping code instead of data is only efficient for a low number of clients. The last method we took into account was H-NIDS. The method itself works well as it runs on a host just like inline and out of band NIDS. However, data is redirected from the hypervisor, which is inaccessible in the cloud. Therefore, a nested hypervisor would be required. This also entails that the entire infrastructure of the client has to be reconfigured, which is a major deficiency.

5.5.1 Inline vs. Out of Band

We have gathered that there are two methods that comply with our set of qualitative requirements that can be identified theoretically; inline host based NIDS and out of band host based NIDS. To sketch where the differences between these methods surface, we evaluate them with an implementation design.

Inline

In the inline configuration of the host based NIDS, we configure an alerting engine on every host. As with any NIDS our implementation consists out of three aspects. However, the information gathering and alerting happens within the alerting engine of the host. The IDS deployed on the host, for example Snort, pulls packets from the network interface and analyses them. Since every host has its own IDS agent, the ruleset can be tailored for the service running on the corresponding host. This can be desirable as a lot of data will not have to be matched against attack signatures that the host is not vulnerable to. Lastly, any generated alarms will be forwarded to a central

console where an analyst can review the alarms. Furthermore, it is possible to correlate the alarms to detect distributed attacks. Note that this is not the same as correlating events between different hosts. The diagram in figure 5.1 shows what an implementation of the above would look like.

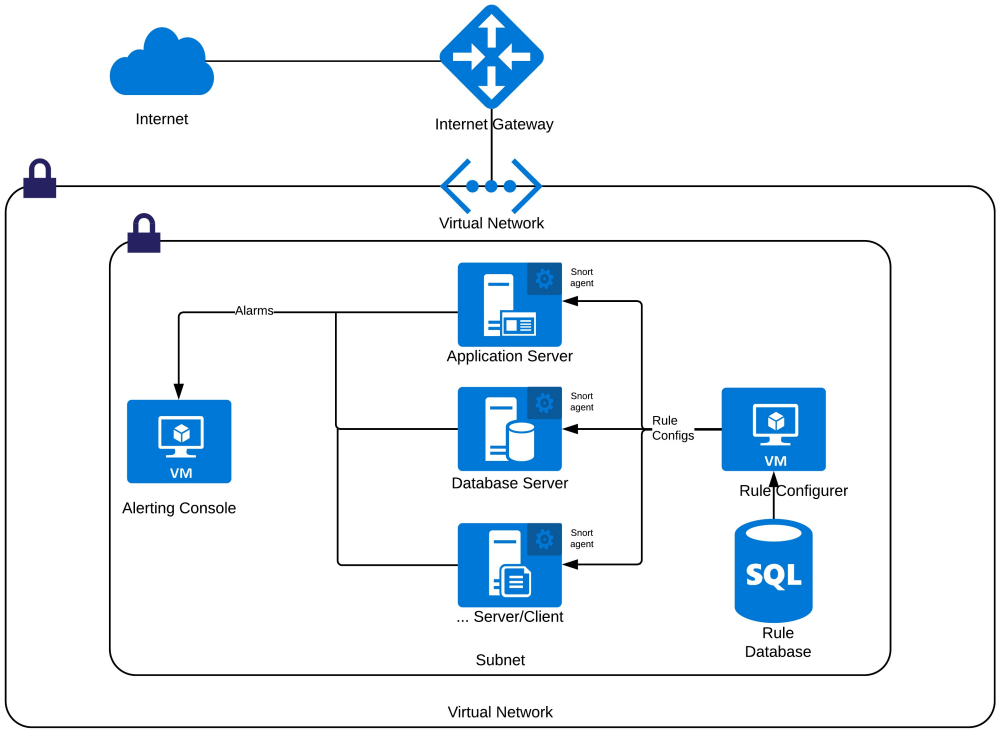

Figure 5.1: Inline Host NIDS

Out of Band

In contrast to the inline variant, the out of band version of this implementation has a central IDS host where all forwarded data is correlated. On every host a packet dump agent is deployed. These packet dumps are forwarded through secure tunnels, so the IDS VM is able to discriminate between different machines. If not, the engine would be vulnerable to spoofed packets. Replicating all the data at the host entails that the IDS does not scale with the environment. Therefore, we place a load balancer before the IDS instance. Consequently, the IDS capabilities can be scaled up or down with the amount of traffic flowing through the balancer. The diagram in figure 5.2 shows what an implementation of the above would look like.

5.5.2 Comparison

When comparing the implementation schemes shown in figures 5.1 and 5.2 the main difference is that the inline implementation impacts the CPU load on every host, while the out of band method mostly impacts the network load. The network load on the inline configuration is little to none, as only alarms need to be forwarded to a central host. The out of band configuration copies nearly all data, doubling the amount of traffic traversing the network. Furthermore, to accomplish this in a secure way, the copied data has to be sent in a secure way, otherwise it can be tampered with or spoofed. This requires additional implementation of secure protocols or tunnels between the hosts and the IDS servers. This could in turn prove to be difficult when adding the load balancing aspect. As for the CPU load, the impact of a packet dump process on a host is rather small, while

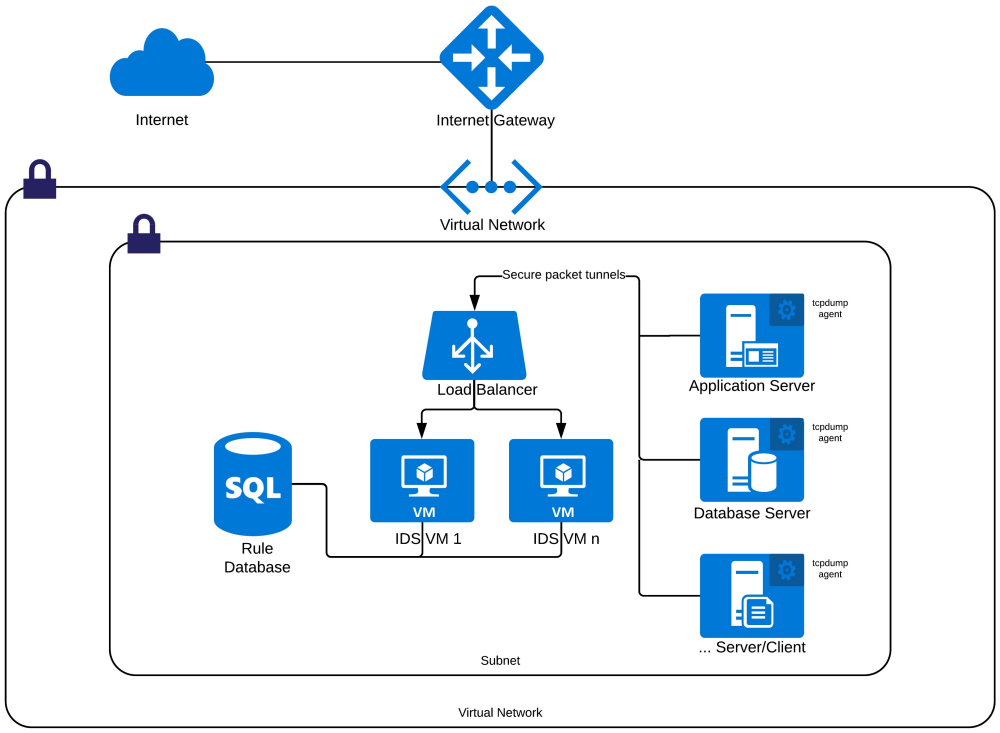

Figure 5.2: Out of band Host NIDS

running an entire IDS agent can consume quite some resources. The main counter-argument for a solution that consumes resources of production systems is that it might impede the processes that run on that particular host. However, we feel like this can be countered quite easily by assigning a limited amount of resources to the IDS agent. Additionally, out of band configurations require additional instances that need to be deployed for the IDS host. Evidently, this will introduce costs as the IDS does not automatically scale with the introduction of new hosts. The above findings have been documented in table 5.5. When comparing the implementations designs, taking into account our comparison, we find that inline host based NIDS is the most promising solution for a consumer deployable NIDS for public clouds.

Table 5.5: Inline and Out of Band Comparison

	Inline	**Out of Band**
CPU Load	High	Low
Network Load	Minimal	High
Correlation	Only on alarms and within one host	Correlation possible on all data
Deployment	Easy	Tunnels between all hosts and the load balances; a lot of network interfaces required
Costs	Only one machine for alerting and configuring	Additional computing instances needed based on amount of traffic

5.6 Conclusion

In this chapter, we answered research question 2: *Based on these requirements, what are the theoretically most promising consumer deployable NIDS methods for public clouds?* After conducting an extensive research on nine possible cloud NIDS methods, we found five methods that seemed feasible. Those five methods were scoped down by mapping them on our set of requirements. This resulted in two possible implementations: inline host based NIDS and out of band host based NIDS. For each of these two methods, we created an implementation design. After analysing the implementation based on their differences, we conclude that inline host based NIDS is the best solution. In chapter 6 we will implement this solution to verify whether inline host based NIDS also complies with the performance standards investigated theoretically.

Chapter 6

Implementation

In the previous chapter we identified that a host based solution would be the best option to accomplish a public cloud NIDS. Two host based solutions complied with the qualitative requirements from section 4.6; inline and out of band. After we constructed implementation schemes for both these methods, we found that the inline variant looks most promising.

6.1 Introduction

While answering the previous research question, we managed to pin down the qualitative requirements of public cloud network intrusion detection. However, the true performance of such a NIDS has yet to be tested. In this section we will implement the inline host based solution and test it based on the quantitative requirements defined in section 4.6:

1. How many packets per second can the NIDS process before dropping security capabilities (e.g., in the form of dropping packets)?
2. How much latency does the NIDS introduce under normal and stressed behaviour?
3. How much impact does the IDS process have on the (other) host(s) in the network?

6.2 Methodology

The aim of our testing methodology is to quantify and test the performance of the NIDS. To accomplish this, we set up a test environment. As platform for the public cloud NIDS implementation, Microsoft's Azure was used. The choice of platform was arbitrary, as our comparison in section 2.1.4 showed that there was very little difference between the popular CSPs. Especially regarding the functionality used within our solution. The host based NIDS does not require any special features and is designed to work in any public cloud setting.

On our Azure platform we created an IPSEC tunnel between our local firewall and the Azure gateway. Within the Azure platform we created a virtual network in which a subnet was placed. The subnet has a virtual gateway with a public IP to be resolved through the IPSEC tunnel, so we can remotely access hosts within that subnet (figure 6.1).

In the network two virtual machines were deployed; one server with our IDS solution and one without. Each of these machines requires its own drive, network interface and security group (figure 6.2). Both machines run on Linux with the Canonical's Ubuntu 16.04 server distribution (the highest available distribution at the time of implementation within Azure, now superseded by 17.04). Ubuntu was chosen as it is the most frequently used Linux distribution in cloud environments at the time of writing [46]. The servers have been allocated with relatively small resources; 2 CPUs and 4GB RAM. Reasoning behind this is twofold. Firstly, we want to ensure the feasibility of our solution for smaller and more often chosen systems; more resources can always

Figure 6.1: Network Setup

be allocated for larger consumers. Secondly, there is no necessity to run large CPU and network extensive programs, because CPU load can be simulated and the amount of network traffic is proportional with the analysis component. Meaning, if on a relatively quiet host our IDS solution will use 20% of the total CPU, it will need a similar percentage on a large system (that system just requires more resources).

The reason we chose for a minimum of two cores, is that we can bind our IDS agent to a single core, where it will take priority over regular processes. This way, other processes running on the server can consume resources of this core when our agent does not require them, but not the other way around. Functioning and workload of the IDS will differ per system, but they will remain proportional for similar systems of different sizes. Both servers share the same security group. For simplicity we have enabled some standard protocols to access the hosts remotely (figure 6.3). This does not serve security issues for our research, as the IPSEC tunnel disallows external connections. Full specifications of the hardware and software can be found in appendix A.

The virtual server hosting the IDS solution runs a Snort [31] agent as detector. Snort is configured to log alerts as unified2, which is much faster compared to, for example, ASCII format. To increase performance and efficiency, we have also deployed Barnyard2 [15], which writes unified2 logs to a mysql database as a separate process, allowing the IDS agent to fully focus on processing network traffic. For our experiment, we installed the default Snort image with all default Talos [45], Emerging Threat [13] and Community rules enabled. We disabled the checksum verification, as this collided with the bigFlows datastream, generating a lot of alarms. By tuning the ruleset and defining analysis components that are relevant for the environment, the consumer could easily double the performance of the agent.

Figure 6.2: Server Resources

Inbound security rules

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
1000	default-allow-ssh	22	TCP	Any	Any	Allow
1010	RDP	3389	Any	Any	Any	Allow
1020	HTTP	80,8080	Any	Any	Any	Allow
1030	HTTPS	443	Any	Any	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInB...	Any	Any	AzureLoadBal...	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Outbound security rules

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Figure 6.3: Security Rules

6.3 Experiment

As described at the beginning of this chapter, our experiment should verify whether our public cloud NIDS solution functions conform performance requirements. Therefore we will test the latency the agent introduces, its security performance, and the impact it has on the (other) host(s).

The network connection to the server is tested with iPerf [27]. To find the limitations of our server we consecutively sent 100 Mbps of TCP and UDP traffic over the interface for 10 seconds. The maximum throughput averaged at 61Mbps of UDP traffic and 52Mbps of TCP traffic. This means that testing with transmissions rates that exceed roughly 50Mbps, the interface will start dropping, rather than our IDS solution.

All three experiments require us to stress the host with different streams of data. For our experiment we will use a clean - malware free - dataset[1] that will be ran through both servers. The dataset is sent over the server's interface with Appneta's tcpreplay [14], which is running software from the University of California, Berkeley and the Lawrence Berkeley Laboratory.

The dataset contains five minutes of realtime data on a busy private network (averaging at 9Mbps), which provides a proper benchmark for a single host. Note that this is data flowing over an entire private network with multiple servers and clients. In our implementation all these servers would only have to perform intrusion detection on the data that flows over their interfaces. However, to stress our system we feed the entire dataset to our server's interface.

[1] http://tcpreplay.appneta.com/wiki/captures.html

6.3.1 Latency Experiment

In this experiment we test how much latency the NIDS introduces under normal and stressed behaviour. The purpose of this experiment is to test how the IDS performs under certain loads. We accomplish this by passing different loads of network traffic over the network to our IDS server. Then, we record the mean latency and the corresponding standard deviation for each test. The idea of this test is to stress test the performance of the IDS with high network performance while maintaining low latency.

6.3.2 Security Experiment

The second experiment is dedicated to finding out how many packets per second the NIDS can process before significantly dropping security capabilities (e.g., in the form of dropping packets). An important aspect of an IDS is that it actually tries to match all traffic for malicious content. If the IDS agent is unable to analyse all packets properly, it will be unreliable. As described in section 6.2, we installed an untuned agent to test the lower bound security performance of our system. To test the default capabilities we run traffic over the monitored interface. This experiment was conducted in a similar manner as the latency experiment (section 6.3.1). First, traffic was sent over the interface in a realtime manner. After this we will increase the transfer rates relative to the initial throughput. We will keep increasing the rate until a substantial amount (10% or more) of packet drops occurs. Note that packet drops is not the same as actual network packet loss (of which you should never want more than 1%). The packet drops we measure in this experiment are packets that are dropped by the NIDS agent, not by the network interface card (NIC). This can occur either because its packet buffer is full or because it cannot process the packet for any other reason (discarding the packet). Meanwhile, the packets are still going over the interface and reaching their actual destination.

6.3.3 Impact Experiment

In this final experiment, we test the impact of the NIDS process on the (other) host(s). In an arbitrary network it is vital that an IDS does not affect other hosts in the network in a significantly negative way. However, this requirement is not applicable in our setup as every server will have its own internal IDS agent. The only way our host could impact the performance of another host is if it would somehow introduce a significant amount of latency. However, this is already being covered in the latency experiment.

Every server has its own IDS agent, which is not allowed to disrupt other processes that run on that particular server. This is accomplished by binding the agent to a single core. All other processes run on another or multiple other cores. However, when there is little traffic over the interface, other processes can consume CPU power from the agent's core, never the other way around. This was accomplished with the cpulimit package [44].

6.4 Results

This section will contain results of the experiment. As with the methodology, we have split up the results into their corresponding subsections. First, we will display the latency related experiment, followed by the security and impact experiments.

6.4.1 Latency Results

During this experiment we tested the latency of both our NIDS server as well as our regular server. The setups of these systems were identical, with exception of the IDS agent. Our IDS agent sniffs this ethernet interface. Evidently, we want to know whether this causes a significant delay to the traffic reaching its destination. We tested the latency with two types of tests. The first test comprised a mimic of a realtime data being pushed onto the servers network interface

card. During the second test we sent traffic out over the interface at a steady rate. The results of both test are displayed in table 6.1. The columns indicated with (r) show the latency when data was sent out at realtime speed. Columns with (s) show the latency when the data was put onto the interface at a static rate. The last two columns show the difference in latency between the server with the IDS agent (NIDS) and the regular server (Reg).

Figure 6.4 shows the total latency of sending the bigFlow packet capture over the interface. The first column shows the latency when data is replayed at its original speed. The latter columns multiply that speed relative to the original transmission rates. The rates have been increased until the interfaces started dropping. From that moment, measuring latency is nonsensical. Figure 6.5 is constructed similarly. The discrepancy being that the data is sent out at static rates, which can be found on the x-axis. Error bars have been left out of the graphic, as the error was too small to visualise (table 6.2). Lastly, we have mapped the differences in latency between the NIDS server and the regular server in figure 6.6. A positive difference means the regular server was faster than the NIDS server. In this figure, we see that in every experiment run the latency in our NIDS server was a higher than the regular server. There was a single exception in the 40Mbps run, where the differences between the means indicate that the NIDS server managed to perform 0.02 seconds faster.

Table 6.1: Results Latency Experiment

Mean NIDS (r)	**Mean Reg (r)**	**Mean NIDS (s)**	**Mean Reg (s)**	**Diff (r)**	**Diff (s)**
307.5	306.54	281.74	279.8	0.96	1.94
156.19	155.09	140.51	138.58	1.10	1.93
104.93	103.94	94.62	93.95	0.99	0.66
78.87	78.67	70.62	70.64	0.19	-0.02
65.05	63.22	57.41	57.28	1.82	0.13

Table 6.2: Standard Deviation Latency Experiment

Stdev NIDS (r)	**Stdev Reg (r)**	**Stdev NIDS (s)**	**Stdev Reg (s)**
0.97	0.68	0.91	1.35
0.72	0.53	0.07	0.22
0.37	0.36	0.33	0.25
0.25	0.28	0.38	0.32
0.13	0.06	0.19	0.08

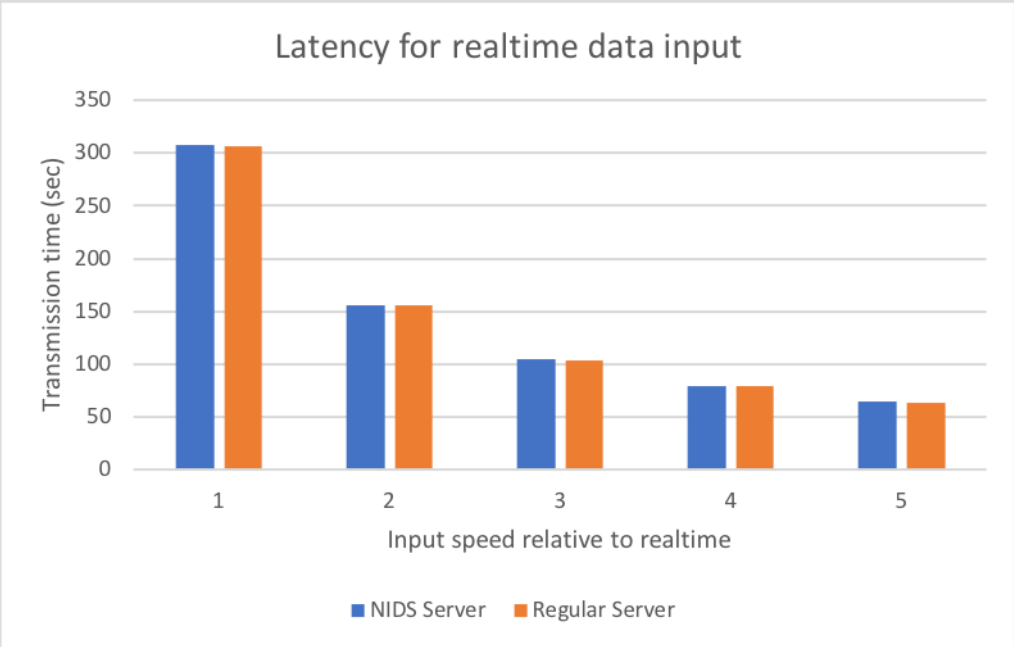

Figure 6.4: Results Realtime Latency Experiment

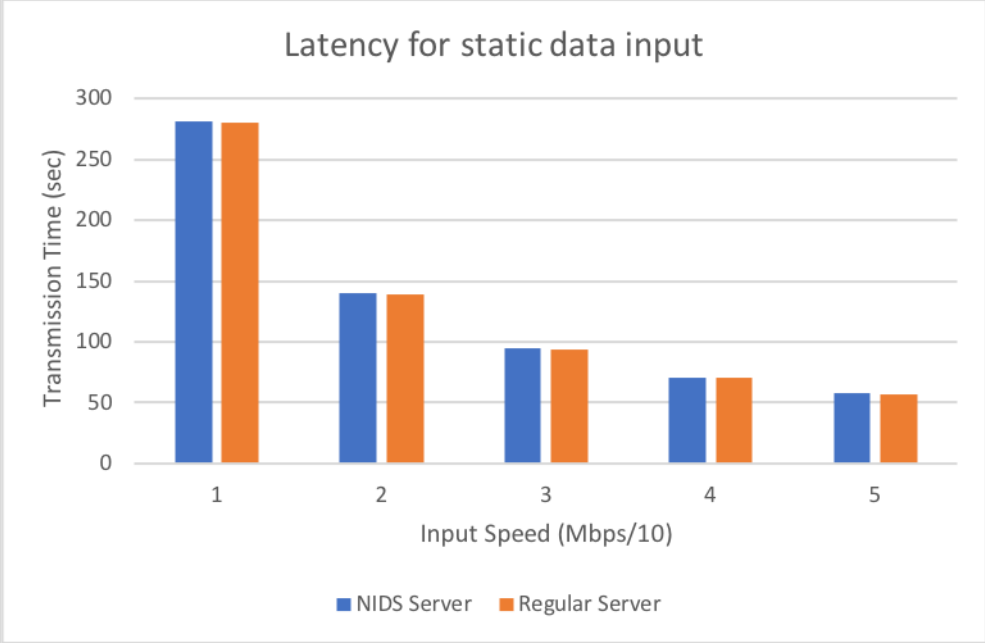

Figure 6.5: Results Static Latency Experiment

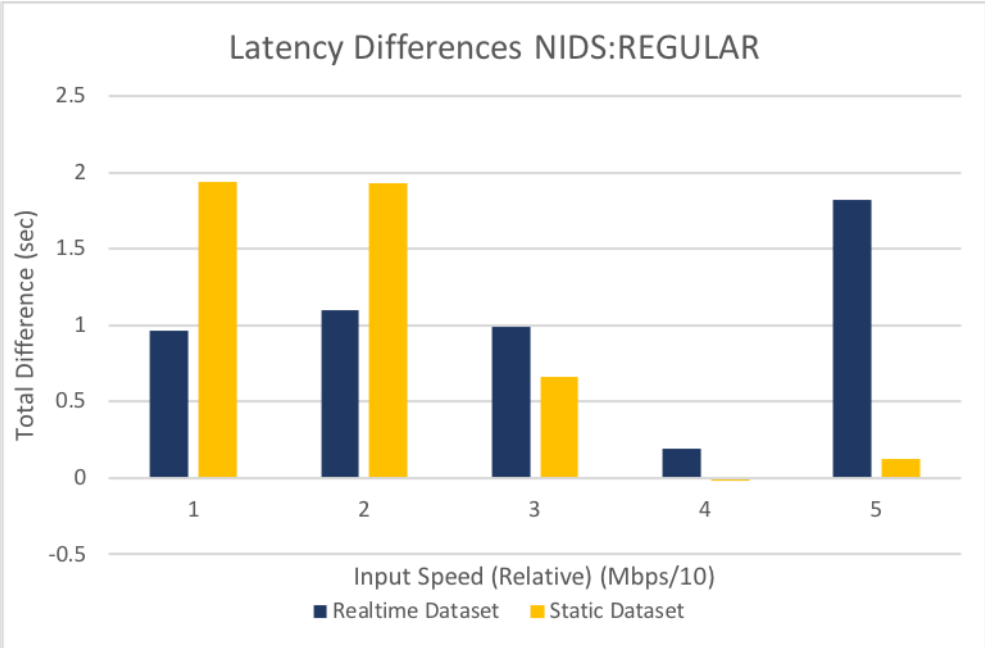

Figure 6.6: Difference in Latency

6.4.2 Security Results

In the security experiment we kept increasing the transfer rates of the bigFlow packet capture. For every test, we replayed the capture three times to check for unexpected outliers. For each of these runs, we calculated the mean percentage of packet drops as well as the corresponding standard deviation. The results of the experiment are displayed in table 6.3 and visualised in figure 6.7. When we increased the transmission rate to five times the original speed, the packet loss grew over 18%, at which point the NIDS was deemed significantly unreliable.

Table 6.3: Mean and Standard Deviation of Packet Drops

PPS	Mean Pktdrop%	Stdev Pktdrop%
2574.33	0.03	0.09
5068.28	0.05	0.08
6631.98	1.86	2.04
10037.38	5.81	1.93
12169.96	18.04	8.17

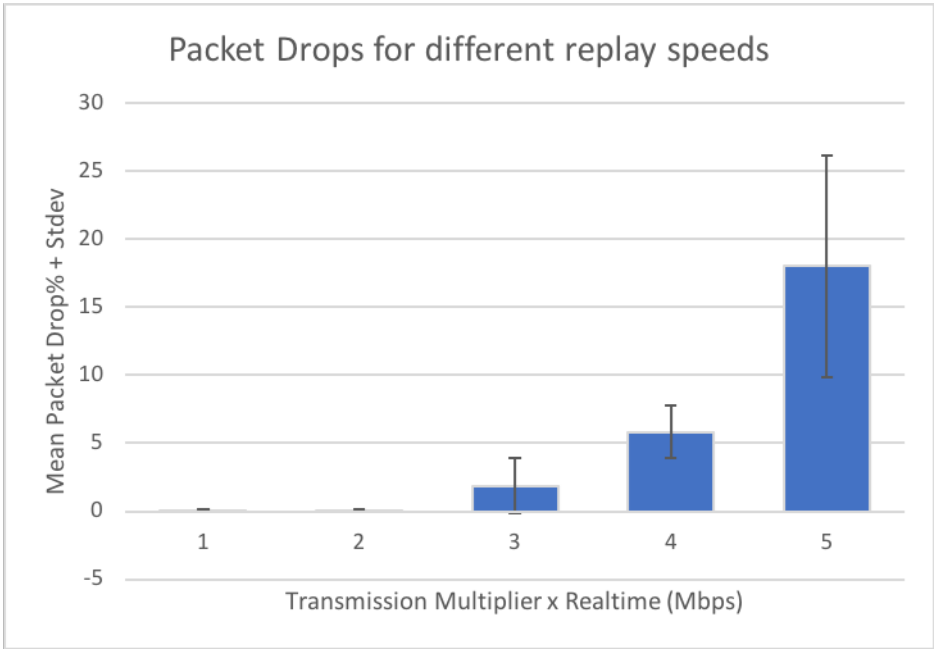

Figure 6.7: Results Security Experiment

6.4.3 Impact Results

As discussed previously, impact comes in two forms. Firstly, we find impact on other hosts, when our NIDS agent would impede on other hosts in the network. This could happen when our IDS server introduces a significant amount of latency. We will not discuss latency any further, as this was already covered in the latency experiment above.

In addition to impediment on other hosts, the NIDS agent running on our NIDS server can also hinder other processes running on the server itself. In our implementation we already assured that the IDS agent cannot occupy more than one CPU core, so the other native services have at least one or more cores to run on. However, CPU power on the NIDS server could very well be used by another process, when our NIDS agent suddenly requires it. This will affect that process at that time.

Table 6.4 shows the results of this experiment. During the experiment a few outliers occurred, where the CPU suddenly read 0%. This behaviour is defined as erroneous as the machine was working continuously. We expect this to be a tooling error. Therefore these outliers were removed. The first row of table 6.4 shows a typical heavy load for a private network. The network throughput is then increased by a factor two every next run. At the fifth run we observed a mean CPU usage of 93.37%, hitting 100% CPU for over a third of the run. At this point the NIDS agent could not obstruct any more processes as it had reached its limitations.

Table 6.4: Mean and Standard Deviation of CPU Percentages

Mean	**Stdev**
26.81	11.94
49.28	20.66
65.18	19.74
80.92	16.66
93.37	5.77

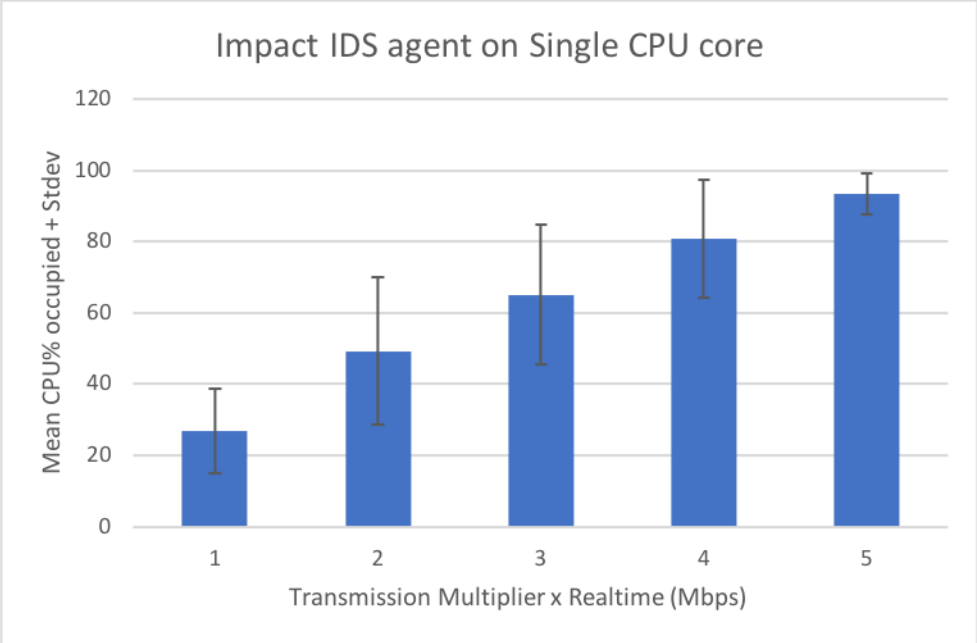

Figure 6.8: Result Impact Experiment

6.5 Analysis

In this section we analyse the results of our experiment. This is done in the same manner as before; each experiment will have an associated analysis section.

6.5.1 Latency Analysis

Across the two experiments, the regular server was faster than the server that hosted our NIDS agent in almost every run. Nevertheless these differences are rather small. The largest difference was measured at 1.94 seconds, while across all experiments the additional latency averages at 0.97 second. Except for the run at five times realtime speed, all the experiments stay within an additional latency of 1%. Each of these runs contained 791615 packets (355417784 bytes), indicating an average additional latency of $0.97sec/791615pkts = 1.22534\ \mu s/pkt$. Since a network, especially one with shared resources, is susceptible to a lot of external factors the additional latency of 1% is ought to be negligible.

6.5.2 Security Analysis

In this experiment we have tested how much traffic a minimal deployment of our IDS agent can handle. From the data in table 6.3, we find a major drop off in packet loss between four and five times the original network speed. This coincides with roughly 10000pps for arbitrary sized packets (total data transfer of 39Mbps). This is a feasible solution for most servers, however the link speed of present-day networks can go over 1Gbps. This means that it would easily be possible to overflow a single server, evidently, making it susceptible to a denial of service attack. Nevertheless, the possibilities for cranking up the throughput of the IDS agent are plenty. The consumer can, for example, choose to allocate more resources (run the IDS on multiple cores), the ruleset and preprocessors can be tuned in order to process more packets per second, and a consumer can balance incoming load. The latter can be accomplished with a service like PFRING (which integrates perfectly with our IDS agent [38]), which enables you to capture packets faster and capturing them more efficiently, preserving CPU cycles.

6.5.3 Impact Analysis

When observing figure 6.8, we find a linear relationship between the data throughput and the CPU usage of our IDS agent. Evidently, the available resources for other processes have a reversed relation. Once again the impact on resources using CPU cycles could be decreased with similar solutions as the ones suggested in our security analysis.

6.6 Conclusion

In this chapter we answered our last research question: *How does the theoretically most promising method perform in a heuristic experiment, taking all requirements into account?*

Most importantly, we wanted to verify that our solution is feasible in terms of performance. This was accomplished with a ternary of experiments. First, we tested additional latency that our IDS solution introduces. From a series of tests this turned out to be 1% additional latency on average. Even though this is a small amount of additional latency, it was observed in almost every test that the NIDS server was indeed processing packets marginally slower.

Secondly, we tested the security capabilities of our solution by transmitting high dataloads over the server's network interfaces. The minimal solution endured properly for an adequate load. It must be noted that for a *minimal* solution, high loads can cause our NIDS agent to discard packets when its queue is full.

Lastly, we tested the relation between load and impact on the host. This is a linear relationship, which proves to be feasible for a small server. Servers that process large loads of data, will require additional resources. However, due to the limitation of the IDS process, the impact on other processes (which is probably the core business) will remain negligible.

Furthermore, we shortly want to touch on the other requirements of chapter 4. We have shown that from a theoretical perspective, our method meets all these requirements. However, we have not touched on whether or not our minimal solution meets the requirements as well.

1. **The NIDS should monitor newly configured instances.** As long as newly configured instances (automatically) also run the installation process of our IDS agent, this requirement should be met. It would even be possible to predefine machine images that already have the agent installed.

2. **The NIDS should be able to scale with he environment and data throughput.** Every host has its agent, so it scales perfectly. This is one of the strongest point of our agent-based solution.

3. **The NIDS should work across the entire (cloud) environment.** Similarly to the previous point, we can easily deploy our solution across all segments of the environment. Different subnets, clouds or even sites have no impact on this.

4. **The NIDS should be easy to deploy and configure for a cloud consumer.** Deployment of the agent is fairly easy when the agent comes with a default installer or image. However, tuning the agent to perform better can be a tedious process requiring expertise.

5. **The NIDS should be easy to integrate without network redesign.** No network redesign is needed as shown in the implementation phase. The only requirement is that the agent should be installed on the server and sniff its network interfaces.

6. **The NIDS should be able to process legitimate traffic while under attack.** Legitimate traffic is always processed as long as the infrastructure can handle it (e.g., the bus, NIC etcetera). However, it has to be noted that even though legitimate traffic is still being analysed, our agent will stop observing every single packet when its queue is filled. Nonetheless, this has been shown to have minimal consequences for substantial data transfers.

7. **Deployment and maintenance costs should be as low as possible.** The deployment costs of our minimal solution are negligible, as only a single additional core is required (which can even be used by other processes). There is no additional traffic traversing out of cloud-boundaries, generating no additional costs.

All in all, we conclude that our out of band host based network intrusion detection solution performs adequately. A minimal deployment of our solution has shown to work properly for normal to large network loads. Additionally, the solution has potential to be scaled up, even for a single host. The only downside perceived is that scaling up per host, without throwing additional resources at it, would require time and expertise. This is something we want to omit in the future.

Chapter 7

Conclusions

This final chapter presents the overall conclusion to the research. In section 7.1 we present the conclusion to our main research question by revisiting the subquestions defined in section 1.5. Finally, we conclude this thesis in section 7.2 with limitations and future work.

7.1 Conclusion

In this thesis we have investigated the best way to approach network intrusion detection in public clouds, without having to rely on cloud service providers or other third parties. The research goal was to find a NIDS method for public clouds that the consumer can deploy by himself. This was formalised with the following research question:

What is the most effective way of accomplishing consumer deployable network intrusion detection in public clouds?

To answer this question properly, we divided our research in three chapters with their corresponding subquestions.

Subquestion 1: What are the requirements for deploying a consumer deployable NIDS for public clouds?

In chapter 4, we identified all important aspects for a consumer-deployable network intrusion detection system for public clouds. These aspects can be categorised in three factors: public cloud, NIDS and the consumers. For each of these factors, we derived requirements that our solution should satisfy. The cloud requirements are based on characteristics of cloud systems:

1. The NIDS should monitor newly configured instances.
2. The NIDS should be able to scale with the environment and data throughput.
3. The NIDS should work across the entire (cloud) environment.

For NIDS requirements we consulted guidelines regarding traditional NIDS systems. We divided the NIDS characteristics in three subsections; security capabilities, performance and stability. From these categories we extracted the following requirements:

1. The NIDS should be able to process legitimate traffic while under attack.
2. The NIDS should be able to cope with high throughput without losing speed or security capabilities.

3. Latency of the NIDS should be as low as possible.
4. The NIDS should affect the (other) host(s) as little as possible.

Lastly, this research is relevant for consumers whom want to apply security while embracing the benefits of a cloud system. The utility characteristics are divided in costs and management requirements:

1. The NIDS should be easy to deploy and configure for a cloud consumer.
2. The NIDS should be easy to integrate without network redesign.
3. Deployment and maintenance costs should be as low as possible.

The defined requirements comprise a minimal set that the cloud NIDS solution should comply with. The requirements do not include capabilities and performance of the analysis engine used within our solution. This is outside the scope of this research.

Subquestion 2: Based on these requirements, what are the theoretically most promising consumer deployable NIDS methods for public clouds?

In the chapter 5, we delved into literature and exhaustively researched intrusion detection techniques that applied to cloud systems. For each of these techniques, we verified that we could, in one way or another, apply them to the initial requirement of being able to use them in a public cloud without requiring CSP interference. For all these methods we conducted a critical revision, marking all pros and cons for each of the important aspects we found in chapter 4. We then mapped this onto our requirements and found two possible contenders. Both methods function as an agent that runs on cloud servers and taps the network interfaces. The difference being that one operates inline; meaning that the IDS functionality runs on the host. The second method operates out of band and ships raw packet captures to a centralised IDS server. For each of these two contenders, we drew implementation schemes and verified feasibility. Based on our findings in section 5.5.2, the inline method shows superiority in terms of security and cost. Its only downside opposed to the out of band option is that it takes some resources away from its host. Therefore, the inline host based NIDS solution was elected as theoretically most promising.

Subquestion 3: How does the theoretically most promising method perform in a heuristic experiment, taking all requirements into account?

Lastly, to verify how our solution performs in practice, we implemented a minimal agent in an Azure Ubuntu server. The server was stressed with a dataset that mimics a realtime busy private network. The security capability was measured in terms of packets that could not be analysed by the agent due to heavy load. The performance of the agent was measured by comparing the added latency and impact on the host to an identical server without the agent. The NIDS service performed well under these circumstances, even when the throughput was increased. We found that performance-wise, the out of band host based NIDS solution sufficed, even when resources are little and the settings are untuned. There are, however, a few deficiencies, which we will touch upon in section 7.2.

Coming back to our main research question, we have found an effective and fairly simple way for cloud consumers to apply network intrusion detection in public cloud settings. This thesis shows that the host based method is the only implementation that meets all the predefined requirements. We have proven that the implementation works in practice and has potential to perform better after tuning. Furthermore, because the NIDS is deployed as an agent, it can easily be altered and scaled.

7.2 Limitations & Future Work

This thesis was based on a lot of theoretical research on papers from third party authors. Evidently, not all of these methodologies had a full blown implementation scheme that we could analyse up to the smallest of detail. Whether or not these methods met the requirements was well founded. However, at the end of our theoretical analysis, we ended up with two host based methods that sufficed with all the requirements; out of band and inline. The inline solution is easier to deploy from a consumer perspective, and it is less easy to tamper with. The security component is kept within a single host, so the data cannot be tampered with. Nonetheless, we feel like in the future it is interesting to investigate whether there is a simple and safe way to deploy multiple sniffing agents on all hosts, while keeping a central analysis engine. The major benefit of a central engine is that it can correlate the different events happening throughout the network. This is much more complicated when the central database only contains alerts, rather than events.

That being said, the correlation on alarms has not been tested, since we only worked with a single NIDS agent during our implementation. For future work, it would definitely be interesting to test whether correlating on alarms is effective.

Furthermore, a limitation of our current implementation of the agent is that its packet buffer fills up faster than the network interface. Meaning that the NIDS capability will drop before the interface starts dropping. Consequently, it is possible to perform denial of service on the agent. Whether or not you can do this, depends on a lot of factors, such as the resources of the NIDS agent, the qualification of the network interface cards and buses, the network speed, etcetera.

Lastly, for our research, we wanted to show that the chosen solution is feasible in a default configuration of a NIDS agent. This means that it is far from optimal and implementation of an optimal agent has to be investigated thoroughly. Some optimisations have already been suggested throughout the thesis, including but not limited to: load balancing, BPF filters, preprocessing, scaleable resource access, and rule set tuning. Especially for the latter of these optimisations, there is an interesting field of research. Namely, how well the NIDS agent performs, is highly dependent on the amount of rules it has to match. A lot of rules are only valid for a certain type of operating system, network environment, usage and more. If the NIDS agent could somehow read these specifications or create a baseline of the environment, it could choose what rules to enable in a heuristic manner. This would create a dynamic agent, which needs little CSP nor consumer interference.

Bibliography

[1] M. Ahmed, R. Pal, H.M. Hossain, M. Bikas, and M.K. Hasan. Nids: A network based approach to intrusion detection and prevention. *Computer Science and Information Technology - Spring Conference*, pages 141–144, 2009. 3

[2] T. Alharkan and P. Martin. IDSaaS: Intrusion detection system as a service in public clouds. *Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012*, pages 686–687, May 2012. 22

[3] Amazon Web Services Inc. Accessed on April 20th 2017 at `https://aws.amazon.com/`. 8

[4] Amazon Web Services Inc. Amazon web services: Overview of security processes. 2016. Accessed on April 20th 2017 at `https://d0.awsstatic.com/aws-answers/VPC_Security_Capabilities.pdf`. 8

[5] Amazon Web Services Inc. VPC security capabilities. February 2016. Accessed on September 25th 2017 at `https://d0.awsstatic.com/whitepapers/aws-security-whitepaper.pdf`. 22

[6] J. Arshad, P. Townend, and J. Xu. A novel intrusion severity analysis approach for clouds. *Future Generation Computer Systems*, pages 416–428, January 2013. 14

[7] Centraal Bureau voor Statistiek. Netherlands in top five users of cloud-based services, July 2016. 1

[8] C.J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang. Nice: Network intrusion detection and countermeasure selection in virtual network systems. *IEEE Transactions on Dependable and Secure Computing*, pages 198–211, January 2013. 24

[9] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing v3.0. 2011. 16

[10] A. V. Dastjerdi, K. A. Bakar, and S. G. H. Tabatabaei. Distributed intrusion detection in clouds using mobile agents. In *2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 175–180, December 2009. 23, 31

[11] Y. Diogenes. Azure security center detection capabilities. 2017. Accessed on April 24th 2017 at `https://docs.microsoft.com/en-us/azure/security-center/security-center-detection-capabilities`. 9

[12] Y. Diogenes, D Shinder, and T. Shinder. Azure network security. 2016. Accessed on April 24th 2017 at `https://www.microsoftpressstore.com/articles/article.aspx?p=2730118`. 9

[13] Emerging Threat. Accessed on January 17th 2018 at `https://rules.emergingthreats.net/`. 37

[14] F. Klassen and the AppNeta Team. Tcpreplay. Accessed on November 27th 2017 at `http://tcpreplay.appneta.com/`. 38

[15] Firns, I. Barnyard2. Accessed on November 22nd 2017 at https://github.com/firnsy/barnyard2. 37

[16] A. Fishman, M. Rapoport, E. Budilovsky, and I. Eidus. HVM: Virtualizing the cloud. *5th USENIX Workshop on Hot Topics in Cloud Computing*, 2012. 22

[17] K. Giannakouris and M. Smihily. Cloud computing - statistics on the use by enterprises. March 2017. 1

[18] Google Inc. Accessed on May 22nd 2017 at https://cloud.google.com/solutions. 9

[19] Google Inc. Google infrastructure security design overview. 2017. Accessed on May 22nd 2017 at https://cloud.google.com/security/security-design/resources/google_infrastructure_whitepaper_fa.pdf. 9

[20] Google Inc. Google security whitepaper. 2017. Accessed on May 22nd 2017 at https://cloud.google.com/security/whitepaper. 9

[21] B. Grobauer, T. Walloschek, and E. Stöcker. Understanding cloud computing vulnerabilities. *IEEE Security and Privacy*, pages 50–57, June 2010. 14

[22] Trusted Computing Group. Cloud computing and security - a natural match. April 2010. 13

[23] S. Hansman and R. Hunt. A taxonomy of network and computer attacks. *Computers & Security*, pages 31–43, February 2005. 10, 11

[24] N. Hoque, M.H. Bhuyan, R.C. Baishya, D.K. Bhattacharyya, and J.K. Kalita. Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, pages 307–324, April 2014. 10

[25] IBM. Client-side attacks - XSS. Accessed on July 15th 2017 at https://www.ibm.com/support/knowledgecenter/SSB2MG_4.6.0/com.ibm.ips.doc/concepts/wap_client_side_attacks.htm. 11

[26] IBM. Cross-site request forgery (CSRF) attacks. Accessed on July 15th 2017 at https://www.ibm.com/support/knowledgecenter/SSB2MG_4.6.0/com.ibm.ips.doc/concepts/wap_cs_request_forgery.htm. 11

[27] iPerf. Accessed on November 27th 2018 at https://iperf.fr/. 38

[28] S. Iqbal, M.L. Mat Kiah, B. Dhaghighi, M. Hussain, S. Khan, M.K. Khan, and K.-K. Raymond Choo. On cloud security attacks: A taxonomy and intrusion detection and prevention as a service. *Journal of Network and Computer Applications*, pages 98–120, October 2016. 10

[29] P. Kannadiga and M. Zulkernine. Didma: A distributed intrusion detection system using mobile agents. pages 238–245, May 2005. 23

[30] L. Leong, R. Bala, C. Lowery, and D. Smith. Magic Quadrant for Cloud Infrastructure as a Service, Worldwide. June 2017. 8

[31] M. Roesch and the Snort Team. Snort. Accessed on September 13th 2017 at https://github.com/firnsy/barnyard2. 37

[32] P. Mell and T. Grance. *The NIST Defenition of Cloud Computing*. U.S. National Institute of Science and Technology, September 2011. 1, 2

[33] P. Mell and K. Scarfone. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. U.S. National Institute of Science and Technology, 2007. 16

[34] Microsoft Azure. Accessed on April 25th 2017 at https://azure.microsoft.com. 9

[35] C. N. Modi and D. Patel. A novel hybrid-network intrusion detection system (h-nids) in cloud computing. In *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 23–30, April 2013. 25

[36] N.A. Noureldien and M.O. Hussein. Block spoofed packets at source (bsps): A method for detecting and preventing all types of spoofed source ip packets and syn flooding packets at source: A theoretical framework. In *2009 Second International Conference on the Applications of Digital Information and Web Technologies*, pages 579–583, August 2009. 11

[37] NSSLabs. Test methodology next generation intrusion prevention system. March 2017. 17

[38] NTOP. Accelerating Snort with PFRING DNA. Accessed on January 25th 2018 at `https://www.ntop.org/pf_ring/accelerating-snort-with-pf_ring-dna/`. 44

[39] X. Ou, W.F. Boyer, and M.A. McQueen. A scalable approach to attack graph generation. *Proceedings of the 13th ACM CCS*, pages 336–345, November 2006. 24

[40] Hasika Pamunuwa, Duminda Wijesekera, and Csilla Farkas. An intrusion detection system for detecting phishing attacks. In Willem Jonker and Milan Petković, editors, *Secure Data Management*, pages 181–192, September 2007. 11

[41] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. pages 199–212, November 2009. 14

[42] H. Seybert and P. Reinecke. Half of Europeans used the internet on the go and a fifth saved files on internet storage space in 2014. 2014. 1

[43] M.P. Souppaya, K. Scarfone, and P. Hoffman. Guide to security for full virtualization technologies. January 2011. 12

[44] Sourceforge. Cpulimit. Accessed on January 19th 2018 at `http://cpulimit.sourceforge.net/`. 39

[45] Talos. Accessed on January 17th 2018 at `https://talosintelligence.com/`. 37

[46] The Cloud Market. Accessed on December 15th 2017 at `http://thecloudmarket.com`. 36

[47] I. Ullah. Detecting lateral movement attacks through smb using bro. November 2016. 12

[48] W. Yassin, N.I. Udzir, Z. Muda, A. Abdullah, and M.T. Abdullah. A cloud-based intrusion detection service framework. In *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, pages 213–218, June 2012. 23

[49] D. Ye, Q. Bai, M. Zhang, and Z. Ye. P2p distributed intrusion detections by using mobile agents. In *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, pages 259–265, May 2008. 23

[50] Am. Zarrabi and Al. Zarrabi. Internet intrusion detection system service in a cloud. *International Journal of Computer Science Issues*, pages 308–315, September 2012. 23

Appendix A

System Specifications

Hardware:

1. Processor: Intel Xeon E5-2673 v4 @ 2.29GHz (2 Cores)
2. Motherboard: Microsoft Virtual Machine v7.0
3. Chipset: Intel 440BX/ZX/DX
4. Memory: 4096MB
5. Disk: 32GB Virtual Disk + 9GB Virtual Disk
6. Graphics: Microsoft Hyper-V virtual VGA

Software:

1. OS: Ubuntu 16.04
2. Kernel: 4.13.0-1005-azure (x86_64)
3. Display Driver: modesetting 1.18.4
4. Compiler: GCC 5.4.0 20160609
5. File-System: ext4
6. Screen Resolution: 1152x864
7. System Layer: Microsoft Hyper-V Server